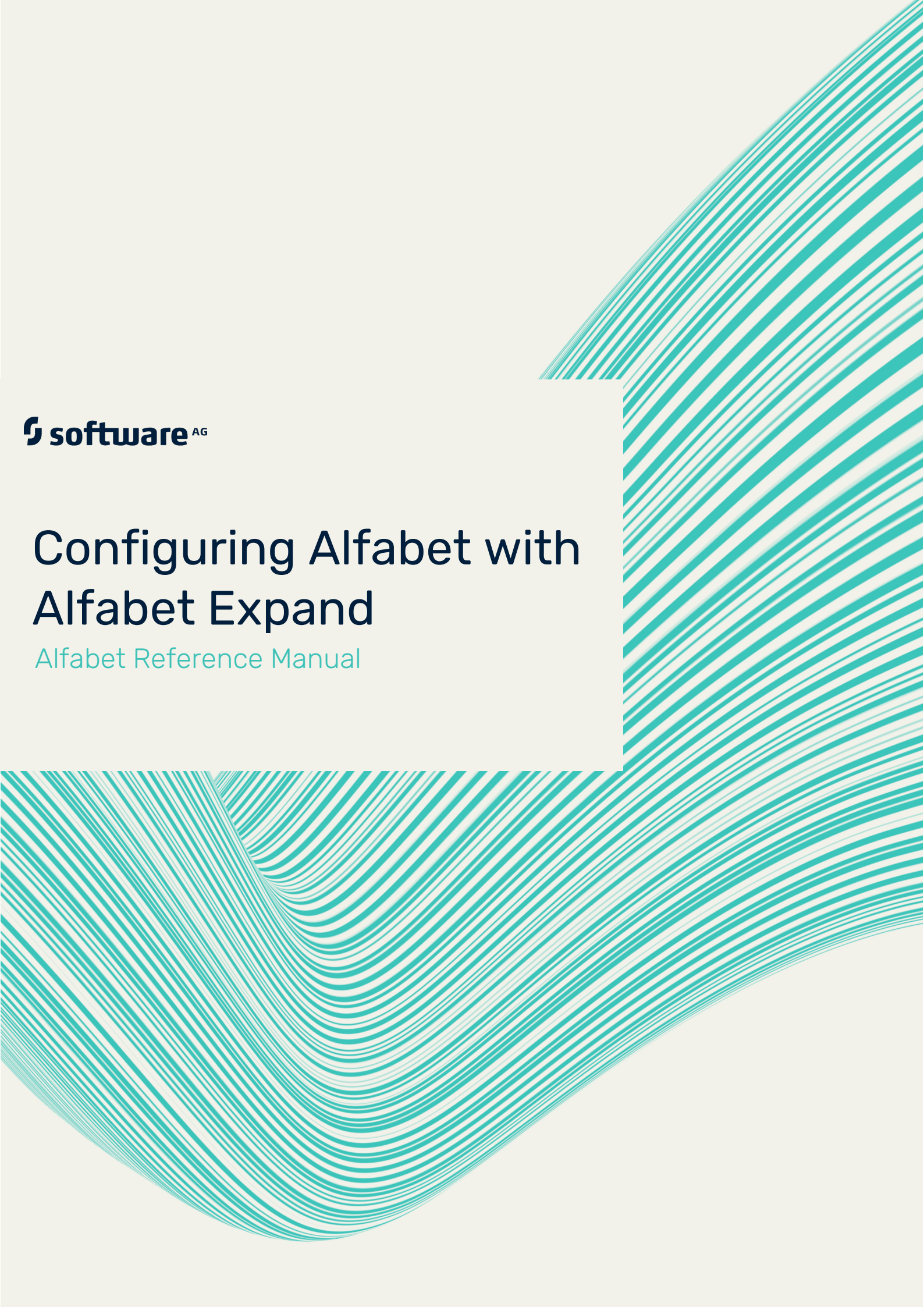


Configuring Alfabet with Alfabet Expand

Alfabet Reference Manual

Documentation Version Alfabet 10.11.0

CONVENTIONS USED IN THE DOCUMENTATION

Convention	Meaning
Bold	Used for all elements displayed in the Alfabet interface including, for example, menu items, tabs, buttons, dialog boxes, page view names, and commands. Example: Click **Finish** when setup is completed.
Italics	Used for emphasis, titles of chapters and manuals. this Example: see the *Administration* reference manual.
Initial Capitals	Used for attribute or property values. Example: The object state Active describes...
All Capitals	Keyboard keys Example: CTRL+SHIFT
File > Open	Used for menu actions that are to be performed by the user. Example: To exit an application, select File > Exit
< >	Variable user input Example: Create a new user and enter <User Name>. (Replace < > with variable data.)
	This is a note providing additional information.
	This is a note providing procedural information.
	This is a note providing an example.
	This is a note providing warning information.

TABLE OF CONTENTS

Chapter 1: Getting Started with Alfabet Expand

Welcome to the configuration tool Alfabet Expand!

This software tool allows you to carry out a number of administrative tasks required to maintain the Alfabet database as well as to configure the Alfabet solution to meet your company's needs.

When you work with the tool Alfabet Expand, you will typically be making changes to the meta-model. The meta-model defines the constructs, rules, and class ontology of Alfabet and constitutes a part of the Alfabet database. Meta-model changes include, for example, the creation of custom properties for object classes, customization of the solution environment, and configuration of user profiles.

You must ensure that Alfabet Expand is only accessible to those individuals in your company that make up the core team of solution designers who are authorized to make changes to the meta-model. Access to Alfabet Expand as well as the availability of specific functionalities in Alfabet Expand is controlled on a per-user basis. Each user must be explicitly specified to have access permission to Alfabet Expand as well as to the individual tabs and menus providing the various functionalities. This ensures that access to the functionalities in Alfabet Expand is explicitly differentiated based on the configuration tasks and responsibilities of the individual user.

The definition of user access permissions to Alfabet Expand is defined by the user administrator in the context of the **Users Administration** functionality that can be accessed in the user interface via an administrative user profile. For more information about specifying access permission for Alfabet Expand, see the section *Creating a User* in the reference manual *User and Solution Administration*.

Please note the following:

- Your licensing agreement with Software AG for Alfabet Expand functionalities will not impact which functionalities are listed in the **Expand Access Options** attribute in the **User** editor. All functionalities are displayed regardless of the content of license agreement. However, regardless the definition of **Expand Access Options** attribute, the licensing agreement will override. The definition of the **Expand Access Options** attribute specifies the visibility in both Alfabet Expand Web and Alfabet Expand Windows.
- If the XML object ***PlatformConfiguration*** is specified in Alfabet Expand, it will override the definition of the **Expand Access Options** attribute in the **User** editor in Alfabet Expand Windows.

The primary tasks that are carried out in Alfabet Expand and described in the remainder of this manual include:

- Executing Administrative Tasks in Alfabet Expand: The most important administrative tasks that you carry out in Alfabet Expand include backing-up and restoring the Alfabet database, updating the configuration, creating user profiles and mandates for the user community, and configuring the visibility of functionality in Alfabet Expand.
- Configuring Access Permissions for Alfabet : Alfabet offers various levels of access control. Access permissions are regulated via user profiles, mandates, user and user group authorization, rule-based access permissions, and object release status definitions. Furthermore, access permissions control the visibility and participation in assignments, workflows, and discussion groups.
- Localization and Multi-Language Support for the Alfabet Interface: Alfabet provides multi-language support as well as localization possibilities to adapt the Alfabet interface to suit the needs and

expectations of the various locales in which your enterprise operates. Information is provided to configure cultures, create custom terminology, and translate the Alfabet interface and online Help.

- Configuring the Class Model: All semantic object classes in Alfabet have a preconfigured set of standard object class properties. With Alfabet Expand, you can create additional properties for a specific class, design the layout of these custom properties in tabulated pages displayed in the relevant editor, and translate the custom properties for the interface languages your company uses. Class keys can be defined for one or more properties available to a class. Additionally, you can define object class stereotype for a specified set of object classes. The release statuses, object states, and lifecycle definitions of objects in an object class are also configurable.
- Configuring Custom Editors: Custom editors can be created for an object class and designed to allow users to enter values for the custom properties configured for your enterprise. Custom editors can be implemented in the context of standard data capture editors, surveys, wizards, and workflows.
- Configuring Custom Selectors and Search Functionalities: Alfabet provides a number of methods to search for and find objects. Object classes and object class stereotypes may be configured to be searchable in the standard Alfabet search functionalities. Additionally, custom selectors can be created and implemented in the standard search functionalities as well as in page views, standard or custom editors, workflows, and configured reports. The solution designer can further configure whether a wildcard <*> should be automatically implemented when a user enters search criteria.
- Configuring Wizards: Wizards provide an efficient means to guide users through a specific set of activities. You can configure multiple wizards per object class, defining the set of page views, their sequence, and prompts that remind a user that a specific field has not been filled out.
- Configuring Standard Business Functions and Custom Explorers: Custom explorers can be configured and assigned to user profiles as Alfabet functionalities.
- Configuring Object Views: A custom object view allows the object profile displayed for an object class to be customized and implemented for a specific user profile. Object cockpits can be further configured for an object view in order to provide the user with various options of viewing specified sets of data in a dashboard form.
- Configuring User Profiles for the User Community: Alfabet functionalities can be combined in a package for various user profiles in your company. With Alfabet Expand, you can define a set of functionalities relevant for each user profile as well as refine the configuration by determining which object classes, object class properties, and page views should be excluded from view from each user profile. Furthermore, you can assign HTML-based guide pages that serve as a start page that guide a user via hyperlinks to the relevant functionalities. Guide pages are configured in the Guide Pages Designer. For more information, see the reference manual *Designing Guide Pages for Alfabet*.
- Configuring Workflows: Workflows can be configured to guide users through a multi-step workflow process. To make the workflow functionality available to the user community, workflow templates with workflow steps must be configured in Alfabet Expand.
- Configuring Alfabet Functionalities Implemented in the Solution Environment: There are a number of functionalities that must be configured before they can be implemented in the Alfabet software. These functionalities include, for example, data capture environments such as the **Capture Applications** functionality as well as the **Project Management**, **Strategy Deduction**, **Compliance Management**, **Capture Enterprise Releases**, **Contract Management** capabilities. Further, a wide variety of aspects in the solution environment can be configured by means of XML objects located in Alfabet Expand. Topics include, for example, the assignment capability, the automatic generation of e-mails, monitors, custom diagram definitions and the default settings for the Alfabet Diagram Designer, and currency units for cost estimation. Please see the overview at the beginning of the chapter for a list of topics available regarding the configuration of the solution environment.

- Configuring the Visualization of the Alfabet Interface: The visualization of the Alfabet interface can be specified in a variety of ways to align with your corporate guidelines. This includes, for example, the font family and colors used in the interface, the inclusion of the company logo in the header, the icons used to visualize indicators in object cockpits, configured reports, custom editors, the images used to visualize enterprise milestones as well as object classes in the interface.
- Configuring Reports: You can configure reports that can be executed by users to search for objects that match customer-specific criteria. In the report, users will be able to navigate to the object in the Alfabet interface. You can either provide reports by defining an Alfabet query or native SQL query to find objects directly in Alfabet, or you can search the Alfabet database with external reporting tools and link to the resulting report from within the Alfabet interface. Report results are either displayed in a tabular report or in a preconfigured graphic report. Many types of graphic representations are available including, for example, treemap reports, layered diagrams, grid reports, matrix reports, HTML reports, portfolio reports, Gantt chart reports, lane reports, and reports to analyze data cubes.
- Defining Queries: Alfabet supports native SQL queries as well as Alfabet queries, a powerful query language that allows a wide variety of information to be retrieved from the Alfabet database. Alfabet queries can be defined for the search functionality, export definitions, custom reports, and form-based data collection as well as for other aspects that can be customized in Alfabet.
- Publishing Data In Microsoft® Word Format: The Alfabet Publication Framework (APF) allows you to publish Microsoft® Word documents about data in the Alfabet database.
- Configuring Surveys for Data Capture Campaigns: The Surveys capability allows for stakeholder surveys or data acquisition campaigns to be configured. Such surveys are typically driven by regulatory or senior management requests. Alfabet Expand provides a capability that allows surveys to be configured so that the data collection process is standardized by a high degree of automation and can be conducted in a timely manner.

Please note that changes to the Alfabet database structure shall only be performed via the official Alfabet components such as Alfabet Expand or Alfabet Administrator. Software AG cannot be held responsible for any damage done to the Alfabet database by changes to the database caused by customers with direct access to the database (for example, via direct execution of SQL commands such as `DROP`, `ADD`, etc. that alter the database structure).

The following information is available:

- Overview of Alfabet Expand Windows and Alfabet Expand Web
- Installation
- Starting and Logging In to Alfabet Expand Windows
- Alfabet Expand Web for Alfabet Cloud Enterprise
- Defining the Language Setting for the Alfabet Expand Inteface
- Understanding the Alfabet Expand Interface
- Overview of the Menus and Toolbar Buttons in Alfabet Expand
- Understanding the Tabs and Explorer Structures in Alfabet Expand
- Meta-Model Tab
- Admin Tab

- Presentation Tab
- Functions Tab
- Workflows Tab
- Reports Tab
- Diagrams Tab
- Publications Tab
- Surveys Tab
- Diagrams Tab
- ADIF Tab
- Working with Configuration Objects
- Using the Show Usage Functionality
- Using the Delete, Cut, Copy, and Paste Functionalities
- Searching for a Configuration Object in the Explorer Tree
- Defining Attributes for Configuration Objects
- Working with XML Objects
- Using Server Variables in Configurations
- Accessing and Editing Objects in Alfabet
- Getting An Overview of the Current Configuration
- Tools and Alfabet Functionalities for Configuration

Overview of Alfabet Expand Windows and Alfabet Expand Web

Alfabet Expand is available either as a Web-based tool (Alfabet Expand Web) or as an application (Alfabet Expand Windows) that allows different users configuring Alfabet solutions:

- The configuration tool Alfabet Expand Windows offers the complete range of configuration functionalities and is the preferred tool for solution designers configuring an Alfabet solution installed at the customer.
- Alfabet Expand Web offers most common configuration functionalities for solution designers using an Alfabet Cloud solution including he tool Guide Pages Designer that is required to design and upload guide pages for the Alfabet interface. For detailed information about the configuration of guide pages, see the reference manual *Designing Guide Pages for Alfabet*.

Technically, multiple instances of both Alfabet Expand Windows and Alfabet Expand Web can connect to the Alfabet database in parallel. When multiple instances of Alfabet Expand are used in parallel with a connection to the same Alfabet database, the following problems occur:

- Changes made to the meta-model by one user might be overwritten by changes made by another user working with another instance of Alfabet Expand if they are attempting to save the same configuration object.
- All instances of Alfabet Expand load a working copy of the meta-model during connection to the meta-model. If the meta-model is changed by another user working with another instance of Alfabet Expand, the changes are not automatically uploaded to the currently active Alfabet Expand sessions. The solution designer will not be aware of changes unless he/she uses the menu option **Meta-Model** > **Reread Meta-Model**.

The following is recommended to prevent overwriting data and inconsistencies in the Alfabet database when using Alfabet Expand:

- Do not work with Alfabet Expand in conjunction with a Alfabet database in a production environment. Configuration should be carried out in a configuration environment and stored in an AMM file. The meta-model of the production environment should only be updated with the AMM file after the configuration environment, or a test environment has been tested and approved.
- Only one instance of Alfabet Expand should be used at a time. If it is necessary to use multiple instances in parallel, only instances of Alfabet Expand Windows that connect by means of a server alias to the Alfabet database should be used. The tracking of meta-model changes should be activated in the server alias configurations of all Alfabet Expand instances.
- A backup of the Alfabet database should always be made before making changes in order to prevent the inadvertent and irrevocable loss of important configuration data.

The following mechanisms are implemented to reduce the risk of overwriting concurrently made changes:

- When a user saves changes to object classes, object class properties and class keys, the connection of all other Alfabet components to the Alfabet database will be closed to inhibit the overwriting of changes by other active Alfabet Expand instances and prevent inconsistencies that may result from users checking in data for which the configuration has changed. This applies to changes made in both Alfabet Expand Windows and in Alfabet Expand Web.
- Alfabet Expand Windows can be configured to detect the changes made by other instances. If the meta-model is changed in one of the instances of Alfabet Expand and the changes are saved to the meta-model, the users of the other Alfabet Expand instances are informed via a message on the Alfabet Expand host that the meta-model has changed. Clicking on the message opens a report in the center pane of Alfabet Expand providing information about the change made to the meta-model and the user performing the change. This mechanism is only valid if the following applies:

- Both instances are Alfabet Expand Windows.
- Both Alfabet Expand Windows instances connect directly to the Alfabet database with a server alias.
- The **Track Meta-Model Changes** parameter is activated in the server alias of the Alfabet Expand instances as described in the reference manual *System Administration*. Please note that meta-model changes cannot be traced for instances of Alfabet Expand Web.
- A different username is used for login to the Alfabet Expand instances.

Access to the Alfabet solution is not defined in Alfabet Expand. Access to the Alfabet solution is defined by a system administrator using the tool Alfabet Administrator or by a user administrator working in the **User Administration** functionality that can be accessed via an administrative user profile. For more information, see the reference manual *System Administration* or the reference manual *User and Solution Administration*.

Installation

Instructions on the setup and installation of Alfabet and the Alfabet components including Alfabet Expand can be found in the reference manual *System Administration*. This manual is available in PDF format and is found on the Alfabet installation CD-ROM.

Please note that your enterprise will be provided with a mechanism that specifies the licensing information regarding the visibility of tabs in Alfabet Expand during software setup. This reference manual provides documentation for all functions available in the configuration tool Alfabet Expand. Therefore, depending on your licensing agreement, you may not see all functions described here.

Starting and Logging in to Alfabet Expand Windows

You must ensure that Alfabet Expand is only accessible to those individuals in your company that make up the core team of solution designers who are authorized to make changes to the metamodel. Access to Alfabet Expand as well as the availability of specific functionalities in Alfabet Expand is controlled on a per-user basis. Each user must be explicitly specified to have access permission to Alfabet Expand as well as to the individual tabs and menus providing the various functionalities. This ensures that access to the functionalities in Alfabet Expand is explicitly differentiated based on the configuration tasks and responsibilities of the individual user.

The definition of user access permissions to Alfabet Expand is defined by the user administrator in the context of the **Users Administration** functionality that can be accessed in the user interface via an administrative user profile. For more information about specifying access permission for Alfabet Expand, see the section *Creating a User* in the reference manual *User and Solution Administration*.

Please note the following:

- Your licensing agreement with Software AG for Alfabet Expand functionalities will not impact which functionalities are listed in the **Expand Access Options** attribute in the **User** editor. All functionalities are displayed regardless of the content of license agreement. However, regardless the definition of **Expand Access Options** attribute, the licensing agreement will override. The definition of the **Expand Access Options** attribute specifies the visibility in both Alfabet Expand Web and Alfabet Expand Windows.
- If the XML object ***PlatformConfiguration*** is specified in Alfabet Expand, it will override the definition of the **Expand Access Options** attribute in the **User** editor in Alfabet Expand Windows.

Alfabet Expand connects directly to the Alfabet database in parallel with a running Alfabet Web Application. Configuration at runtime no longer requires the connection of Alfabet Expand to the Alfabet Server and all

functionality of Alfabet Expand is available at runtime of the Alfabet Web Application. There is no limitation to the number of connections made by Alfabet Expand to the Alfabet database. In other words, multiple instances of Alfabet Expand may connect concurrently to the same database. The following applies when changes are made in Alfabet Expand and saved to the database:

- A change made to the meta-model (in other words, any change made to the class model in the **Meta-Model** tab in Alfabet Expand) will cause a semantic lock to the database. All currently open connections from browser clients and other Alfabet Expand clients will be closed. A message will be displayed to the other Alfabet Expand clients informing them about the lock to the database and that changes can be saved to an AMM file. The meta-model changes will then be updated to the Alfabet database. The Web server must be restarted after an update to the meta-model. Users can then reconnect to the Web server by either implicitly or explicitly logging in. Please note that updating the meta-model via an AMM file is considered a meta-model change regardless of whether the class model has been changed.
- Any change that does not involve a change to the meta-model will be immediately updated to the database with no interruption of the connection to the Web server. The changes made to the database will only be applied after the Web server has been restarted/refreshed.
- When two instances of Alfabet Expand are used in parallel, each instance can scan the Alfabet database for changes performed via other active instances of Alfabet Expand. If a change is saved by another instance of Alfabet Expand, a message will be displayed in the Windows® toolbar. Clicking the link in the message opens a report that provides details about the changes. The system administrator can activate scanning of the Alfabet database for changes via the **Track Meta-Model Changes** parameter in the tab **Expand** of the server alias in the Alfabet Administrator application. For more information, see the section *Configuring Alfabet Expand Windows to Scan the Meta-Model for Changes* in the reference manual *User and Solution Administration*.
- When a change in the meta-model is saved by a running instance of Alfabet Expand, the solution designer of a parallel instance of Alfabet Expand should save the changes to an AMM file by means of the menu option **Meta-Model** > **Reread Meta-Model** in order to load the changed meta-model and update the meta-model with his/her configuration changes to the AMM file. Please note that changes made in Alfabet Expand Web, for example, do not trigger a notification about a change. Therefore, the option **Meta-Model** > **Reread Meta-Model** in the menu should be used prior to each critical action to ensure working with the correct meta-model.
- Because an unambiguous identification of the configuration objects is required to support the parallel connection of Alfabet Expand to the Alfabet database, a unique key constraint has been set for the **Name** attribute of configuration objects in Alfabet Expand. If two existing configuration objects have the same name in the Alfabet meta-model, the name of one of the objects will be changed automatically during migration to the next release. A message about the change will be written to the migration log file.

To start Alfabet Expand:

1) Choose **Start** > **Programs** > **Alfabet** > **Alfabet Expand**.
2) In the **Connect To the Server** field, select a server alias for login. For assistance regarding the correct server alias o to use, contact your system administrator.
3) Enter your user name in the **User Name** field and password in the **Password** field and click **OK**. The tool Alfabet Expand opens.

Alfabet Expand Web for Alfabet Cloud Enterprise

The core functionalities of Alfabet Expand are available in a Web browser for customers using Alfabet Cloud Enterprise. Users can leverage the browsers tabbing feature, printing, and scroll bars and search capability allows to find text in the current view. The following describes the most significant features and user interactions:

- Alfabet Expand Web will be available to all Alfabet Cloud Enterprise customers that have purchased the associated license. Please direct any questions about your license key to Software AG Support.
- The path to enter in the browser address field is: `https:`<URL of the Alfabet Web Application>`/expand.aspx`. The Alfabet Expand login screen will be displayed directly in a browser tab. Upon login, the user will land directly on a start page in the same browser tab. When the solution designer clicks a tile representing a configuration functionality, the selected designer will open in the same browser tab, overlaying the previous view in the tab. Please note that multiple browser tabs are not supported in Alfabet Expand Web. The following tiles are displayed:

- **Class Designer**: Allows the solution designer to configure the class model including enumerations and cultures.
- **Presentation Model Designer**: Allows the solution designer to configure custom editors, class settings, view schemes, object views, wizards, text and monitor templates, HTML templates, selectors, and XML objects.
- **Icon Gallery**: Allows the solution designer to upload icons to the Alfabet database and structure them in icon galleries.
- **Report Designer**: Allows the solution designer to configure reports. Please note that currently the report assistants for Business Chart Report, Portfolio Chart, Matrix Report, Gantt Report, Evaluation Report, and Grid Report can be used in Alfabet Expand Web. If other report assistants are required, the solution designer is advised to use Alfabet Expand Windows.
- **View Designer**: Allows the solution designer to open the designer to configure the visualization of custom editors, object cockpits, selectors, and configured reports. Please note that only one custom editor, object cockpit, selector, or configured report can be configured at a time.
- **Guide Page Designer**: Allows the solution designer to configure start pages for user profiles.
- **ADIF Designer**: Allows the solution designer to view ADIF schemes. Please note that the configuration and import/export of ADIF schemes must be executed in the Alfabet Expand Windows interface (Windows client).
- **Administrator**: Allows the solution designer to configure user profiles and mandates as well as define custom business functions/explorers.
- **Utilities**: Allows the solution designer to export the configuration as an AMM file and upload custom translations (XLS and VOC files) to the Alfabet database.
- Please note that the configuration of workflows, Alfabet publications, and ADIF schemes is only possible in Alfabet Expand Windows interface.

- It is possible to open multiple instances of Alfabet Expand Web in parallel and to use Alfabet Expand Web in parallel with Alfabet Expand applications. Please note however, that it is recommended that only one Alfabet Expand Web session is open at a time. The following problems occur when working with multiple instances of Alfabet Expand:

- If a configuration object is simultaneously being edited by multiple solution designers, the most recent configuration that is saved to the Alfabet database will overwrite any previous configurations saved.
- The solution designers working with Alfabet Expand Web will not be notified if a configuration object is simultaneously being edited or saved via other instances of Alfabet Expand Web or via an instance of Alfabet Expand Windows.
- Alfabet Expand Web is working with a working copy of the meta-model uploaded to the Web Server. If a solution designer is working with Alfabet Expand Web and the meta-model is changed via a simultaneously used instance of Alfabet Expand Web or Alfabet Expand Windows, the changes are not automatically uploaded to the Web Server and the solution designer will not see them in the explorer trees and attributes settings displayed in his/her current session. The solution designer must use the option **Meta-Model** > **Re-Read Meta-Model**.
- A change made to the meta-model (in other words, any change made to the class model in the **Meta-Model** tab in Alfabet Expand) will cause a semantic lock to the Alfabet database. All currently open connections from browser clients and other Alfabet Expand clients will be closed. A message will be displayed to the other Alfabet Expand clients informing them about the lock to the database and that changes can be saved to an AMM file. The meta-model changes will then be updated to the Alfabet database. The Web server must be restarted after an update to the meta-model. Users can then reconnect to the Web server by either implicitly or explicitly logging in. Please note that updating the meta-model via an AMM file is considered a meta-model change regardless of whether the class model has been changed.
- Any change that does not involve a change to the meta-model will be immediately updated to the Alfabet database with no interruption of the connection to the Web server. The changes made to the Alfabet database will only be applied after the Web server has been restarted/refreshed. A Web Server restart can be invoked via the option **Meta-Model** > **Restart Web Application** to make changes visible to users concurrently working with the Alfabet user interface.
- The **Clear Changes** button allows the solution designer to undo all changes that have been made in the current Alfabet Expand Web session since the last time the **Save** button was clicked.
- The options (such as **Create**..., **Design**..., etc.) available to define configuration objects are displayed by clicking the arrow to the right of the object to open a drop-down menu. The right-click action in Alfabet Expand Web is reserved for browser-based actions such as Back, Forward, Save as, etc. Please note however that these actions have no relevance for Alfabet Expand Web, which operates in the same browser tab with the same URL during the entire session.
- The browser's pop-up blocker must be disabled in order to use the **Show Usage** functionality available for most configuration objects.
- All interactions in Alfabet Expand must be explicitly saved via the **Save** button in order to be saved in the Alfabet database.
- The browser Back button is not supported in the context of Alfabet Expand Web. To switch to the start page for Alfabet Expand Web, click the **Home** button. To switch to a different functionality, select an option in the Expand Designers menu available in the standard toolbar. The URL and the browser tab caption will not change when navigating through the Alfabet Expand Web.
- The **Alfabet Query Builder** is not available in Alfabet Expand Web. If the Alfabet query language is used in the configuration instead of native SQL, the Alfabet query must be entered in the **Query as Text** field available for the relevant configuration object.
- The Actipro Syntax Editor Software© technology, which is available in the Alfabet Expand Windows to verify the XML syntax in XML objects, is not available in Alfabet Expand Web. Solution designers

working with Alfabet Expand Web can refer to the default definition in the **XML Def** and **XML XSD** attributes of the XML object. The default definition can be copied to an XML editor, revised as needed using the syntax validation functionality in the editor, and then copied to the **XML Def** attribute in the Alfabet Expand Web.

- The standard documentation (*Configuring Alfabet with Alfabet Expand*) that is available for Alfabet Expand Windows is also available via the Help button for the browser-based version of Alfabet Expand. Please not that although this documentation provides configuration information, it has not been adjusted to reflect the interface of Alfabet Expand Web.
- The interface of Alfabet Expand Web is displayed in English only.

Defining the Language Setting for the Alfabet Expand Interface

The interface of Alfabet Expand is available in the same languages that are supported for the Alfabet interface:

The Alfabet interface is available in the following languages:

Language	Locale ID
Arabic (Saudi Arabia)	1025
German (Germany)	1031
English (United States)	1033
French (France)	1036
Portuguese (Brazil)	1046
Polish (Poland)	1045

To specify the language of the interface for Alfabet Expand, select the language in the drop-down list available in the **Change Language** field in the toolbar in Alfabet Expand. The next time you log in, the selected language will be displayed.

Please note that the language of buttons in system windows (`Yes`, `No`, `OK`, `Cancel`) as well as such headers of system message dialogs (for example, `Warning`) depend on the language settings of your operating system.

Understanding the Alfabet Expand Interface

The following section contains detailed information about the interface elements in Alfabet Expand. You should take some time to familiarize yourself with the various elements in the interface and acquaint yourself with the terminology used in the rest of this manual.

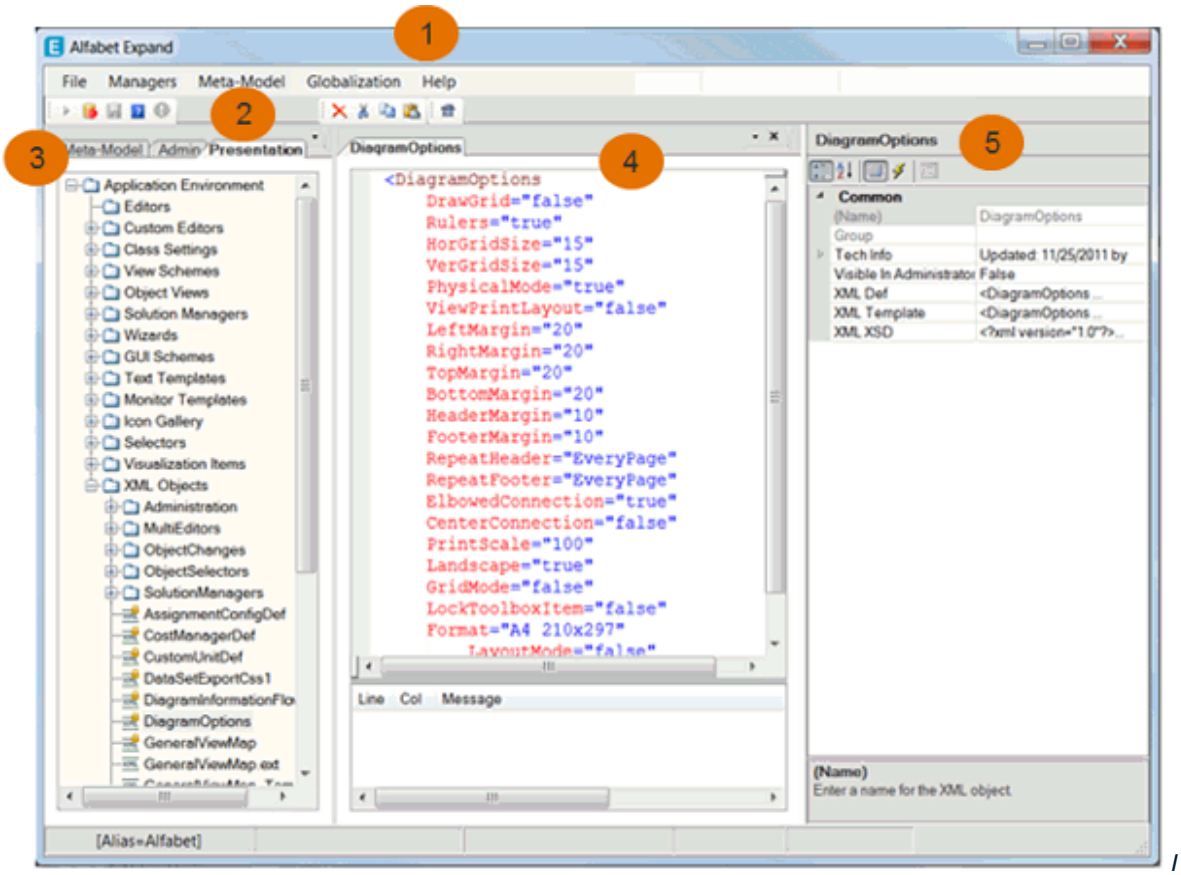


FIGURE: Interface in Alfabet Expand

1. Menus	Menus providing access to options to carry out tasks in Alfabet Expand.
2. Toolbar	The tools displayed may vary depending on the configuration object selected in the explorer or in the center pane.
3. Tabs	Tabs provide access to explorers containing configuration objects.
4. Editor	The center pane displays editors to design or edit configuration objects such as custom editors, object cockpits, and XML objects.
5. Attribute Window	The attribute window contains the attributes that can be defined for the configuration object currently selected in the explorer.

The following interface elements are described in more detail below:

- Overview of the Menus and Toolbar Buttons in Alfabet Expand

- Understanding the Tabs and Explorer Structures in Alfabet Expand
- Meta-Model Tab
- Admin Tab
- Presentation Tab
- Functions Tab
- Workflows Tab
- Reports Tab
- Diagrams Tab
- Publications Tab
- Surveys Tab
- Diagrams Tab
- ADIF Tab

Overview of the Menus and Toolbar Buttons in Alfabet Expand

A standard menu bar and toolbar provides you with a number of basic functions necessary for working with Alfabet Expand. In addition to the standard menu and toolbar, context-sensitive menus and toolbars will be displayed depending on the object you have selected. The following standard menus and menu items are available in Alfabet Expand.

File Menu	
Open Database	Select to open the database.
Close Database	Select to close the database. Saves all non meta-model changes (for example, export definitions and custom reports) to the database.
Save Changes to Database	Select to save meta-model changes (for example, changes to classes or new custom properties), changes to the solution environment (for example, custom editors and wizards) and changes to the business model (for example, business applications) to the database.
Activate Advanced Mode / **Deactivate Advanced Mode**	Select to switch between advanced and normal mode. Advanced mode is only required if custom buttons have been implemented for configured reports by Software AG Support. These buttons are only displayed for further configuration in advanced mode.

Managers Menu

Search Manager	Select to open a full-text search functionality in order to search for objects in the Alfabet database.
Database Manager	Select to either backup the current database in an archive file or restore the current database with another database stored in an archive file located in a local or network folder. **NOTE**: Prior to using the restore functionality, make sure that no Alfabet components are currently connected to the Alfabet database. This applies for example to the Alfabet Web Application, the Alfabet Server (service), Alfabet Expand or batch utilities. The functionality is only available if Alfabet Expand is not currently connected to an Alfabet database.
Object Configuration Manager	Select to recreate, rescan, archive, and restore the object settings that are defined in the **Configure** menu in the clipboard of Alfabet. Object settings configure the presentation of objects.
Context Settings Manager	Select to recreate, rescan, archive, and restore context settings. Context settings include most recent field definitions, filter settings, and views.
ARIS Interoperability	Select to create a user token for the setup of the ARIS - Alfabet Interoperability Interface. For more information, see the reference manual *ARIS - Alfabet Interoperability*.
Guide Page Designer	Select to configure guide pages as start pages for the user interface. The Guide Pages Designer is available in your browser. For more information about designing guide pages and using the Guide Pages Designer, see the reference manual *Designing Guide Pages for Alfabet*.

Meta-Model Menu

Update Meta-Model	Select to overwrite or merge the custom configuration of this Alfabet database with a custom configuration of the meta-model from another database stored in an AMM file. For more information, see the section.
Update Meta-Model Configuration from Master Database	This option is for configuration of Cloud services and for internal use of Software AG Support only. In a customer installation, it is usually deactivated.
Show Update Meta-Model Configuration History	This option is for configuration of Cloud services and for internal use of Software AG Support only. In a customer installation, it is usually not showing any data.

Reread Meta-Model	Select to reload a custom configuration of the meta-model saved in the Alfabet database. Any changes not saved before clicking the **Reread Meta-Model** button will be lost. For more information, see the section About Storage of Changes to the Meta-Model Performed with Alfabet Expand in the Alfabet database .
Clear Meta-Model Changes	Select to remove all changes performed to the meta-model since the last saving of the configuration via the **Save**  button.
Create Configuration Meta-Model Update File	Select to store the customer configuration of the meta-model in an AMM update file. For more information, see the section Saving the Configuration of the Alfabet Solution to an AMM File.
Find Meta-Model Objects for Deployment	Select to store selected objects of the customer configuration of the meta-model into an AMM update file. For more information, see the section Saving the Configuration of the Alfabet Solution to an AMM File.
Set Current Tag	Select to define a tag which will be added to all configuration objects that you generate during the current session to mark them as part of the same configuration. For more information, see Identifying Configuration Objects via Solution Tagging.
Save Meta-Model Objects marked by Current Tag	Select to create an AMM update file containing all configurations that are tagged with the tag defined with the object **Set Current Tag**. Form ore information, see Storing all Objects Tagged With a Selected Tag to an AMM File.
Save Configuration	Select to save the entire configuration as an XML file.
Compare Configurations	Select to compare the current configuration with a configuration stored in an AMM update file or to compare the configurations in two different AMM update files. For more information, see Comparing Database Configurations.
Check Meta-Model for Database Consistency	Select to check whether the current database is in accordance with the current meta-model. for more information, see the section Saving the Configuration of the Alfabet Solution to an AMM File.
Check All Queries	Select to test whether changes to the Alfabet query language require you to adapt existing Alfabet queries in the configuration. For more information, see the section Testing Queries for Compliance with the Current Release.
Compile All Dynamic Assemblies	Select to compile customer specific assemblies delivered by Software AG after upload to the Alfabet database. For more information, see Managing Assemblies

Globalization Menu	
Edit Translation	Select to choose a translation language for the custom editors and custom properties. In the window that opens, you can translate the custom editor captions. **NOTE**: Every caption will be extracted only once in the dictionary. Duplicate parameters entered in the Translation Manager will not be extracted. For more information, see the section Modifying, Translating and Managing the Vocabularies in the chapter Localization and Multi-Language Support for the Alfabet Interface.
Update Guide Pages Translation	Upload the translation of your company's navigation pages. For more information, see the section Modifying, Translating and Managing the Vocabularies in the chapter Localization and Multi-Language Support for the Alfabet Interface.
Help Menu	
F1	Open the Online Help for the node or property grid that is currently selected.
About	View information about the installation of Alfabet Expand.
Toolbar	
	Open database.
	Close database.
	Save meta-model changes to database. You should save frequently when working in Alfabet Expand. Oftentimes you will not be able to select another functionality until you have saved the data.

Understanding the Tabs and Explorer Structures in Alfabet Expand

At the top of the right pane, you will see a row of tabs labeled **Meta-Model**, **Admin**, **Presentation**, **Functions**, **Reports, Publications, Workflow**, **ADIF**, and **Surveys**. Click a tab to view an explorer and its nodes. Each node provides functionalities that allow you to configure your Alfabet solution.

Please note that your enterprise will be provided with a mechanism that specifies the licensing information regarding the visibility of tabs in Alfabet Expand during software setup. This reference manual provides documentation for all functions available in the configuration tool Alfabet Expand. Therefore, depending on your licensing agreement, you may not see all functions described here.

To access a functionality for a node, right-click the node in the explorer. A context menu will open that allows you to select a functionality. If a functionality is greyed out and cannot be selected, click the **Save** button in the toolbar.

The purpose of each tab is explained below.

Meta-Model Tab

The **Meta-Model** tab allows you to view and, to some extent, modify the class model. The primary activity that occurs in the **Meta-Model** tab is the creation and definition of new custom properties required to meet the specific data input needs of your organization.

When you expand the **Classes** node, you may see any of the following:

- A private object class cannot be modified. Custom object class properties cannot be configured for private object classes. Private object classes are displayed for informational purposes and can be configured in Alfabet queries.
- A protected object class can be edited in a limited way. A protected object class typically has private object class properties that cannot be edited, protected object class properties that may be edited (for example, `ResponsibleUser`, `Documents`, `CREATION_DATE`, etc.) but allows for custom object class properties to be created and configured for a protected object class.
- A public object class is a custom class that has typically been created by your enterprise with the help of Software AG Support. All relevant object class properties may be edited and the custom class may be deleted. The attribute **Tech Name** must be explicitly defined for call custom object classes. Custom object class properties can be created and configured for a public object class.
- Some private and protected object class properties are highlighted yellow in the **Class Model** explorer. These are standard object class properties that are recommended by Software AG to be mandatory properties. No enforcement mechanism is implemented if the object class property is undefined. However, if the object class property is included in a class key definition, an error will occur if the object class property is not defined. Therefore, if you include a mandatory object class property in a class key, the object class property may not be hidden in the editor or wizard used to create new objects of the relevant property class.

In Alfabet Expand, object classes are displayed with their names, which means the value of the **Name** attribute of the object class. In some cases, this may differ from the caption used in the standard Alfabet user interface.

When you expand the **Properties** node below an object class, you may see any of the following:

- A private object class property cannot be modified. Private object class properties are displayed to provide guidance in defining new objects.

Please note that the **Hint** attribute cannot be defined for private object class properties . If you need to provide custom tooltips for private object class properties, you can either:

- Display the object class property in an object cockpit and configure a hint for the interface control. In this way, you can define a customized tooltip for the specific context that the object class property is displayed in each relevant object cockpit. For more information, see the section Configuring Object Cockpits for a Custom Object View.
- Define a custom translation to replace the standard tooltip via the translation capability available in Alfabet. The translation of the standard tooltip will be displayed in all instances in Alfabet in which it appears. For inherited object class properties such as `Name` and `ID` which are used in most protected object classes in Alfabet, the custom translations will be displayed for all `Name` and `ID` properties for all relevant object classes. For more information, see the chapter Localization and Multi-Language Support for the Alfabet Interface.

- A protected object class property is a standard property that allows some attributes for the object class property to be edited including, for example, the **Caption** attribute, which is displayed for the object class property in the Alfabet interface, and the **Hint** attribute, which is the text for the tooltip that is displayed in the **Attributes** section of an object view and the editor. Both the **Caption** attribute and **Hint** attribute can be translated for the Alfabet interface. A protected object class property may not be deleted.
- A public object class property is a custom object class property that has typically been created by your enterprise. All relevant attributes may be edited and the custom object class property may be deleted.

Click any object class node in the **Class Model** explorer to display its attributes in the attribute window. These attributes specify the technical data about the object class property. The attributes displayed in the attribute window will depend on the object class property type definition in the **Type** attribute. For more information about configuring object classes and object class properties, see the chapter Configuring the Class Model.

Some object classes have a **Class Key Group** node below the object class which will display any existing class keys. A class key allows for the specification of a unique combination of values for a specified set of standard or custom object class properties. This may be implemented to specify uniqueness constraints for the definition of an object in the object class as well as to create indexes to improve performance. For more information about the configuration of class keys, see the section Configuring Class Keys for Object Classes.

Below the **Class Model** node, you will also see a node labelled **Enums**, which may contain private, protected, and public enumerations. Enumerations allow you to define values that can be selected in combo-box or checked list box interface controls. Once an enumeration is defined, you may assign it to any custom object class property of the type `String`, `Real`, or `Integer` associated with a protected object class. An enumeration can be reused for numerous object class properties for multiple object classes. For more information about the configuration of enumerations, see the section Defining Protected and Custom Enumerations.

You may see any of the following under the **Enums** node:

- Private enumerations  are for reference purposes only and cannot be edited.
- Protected enumerations  are created by Software AG. The enumeration name cannot be changed but you can edit the values specified in the **Sort Enumeration Items** attribute in order to determine which values should be displayed to users in the Alfabet interface.
- Public enumerations  are custom enumerations. All attributes can be edited.

The folder labelled **Cultures**  displays all languages supported for your Alfabet product.

The folder labelled **API Cultures** is only relevant if your company uses the Alfabet RESTful services and wants to define date and number formats for the service calls that differ from the formats used on the Alfabet user interface. No private or protected API cultures are available. For information about the functionality provided via this folder, see the reference manual *Alfabet RESTful API*.

The folder labelled **Database Views** is relevant for performance enhancement and usability enhancement when defining queries for configured reports and other configurations of Alfabet. A database view creates a new table based on a native SQL query that may, for example, collect data from multiple database tables of the Alfabet meta-model. A query in a configured report can then read the data directly from the database view instead of searching for the data in multiple database tables. No private or protected database views are available. Database views are exclusively created by the customer on demand. For more information about the functionality provided via this folder, see the section Creating Database Views To Enhance Performance And Support Search Functionalities.

Admin Tab

The **Admin** tab allows you to create and define user profiles and mandates for the user community.

Please note that changes made to user profiles via the **User Profiles Administration** functionality that can be accessed via the `Admin` user profile are not automatically updated to the user profile configuration in the configuration tool Alfabet Expand that is concurrently in use. The solution designer working in Alfabet Expand must either use the **Rescan Tree** functionality or close and reopen the database to view the updated user profiles.

Presentation Tab

The **Presentation** tab provides functionalities that allow you to configure various aspects of the solution environment. When you click the **Presentation** tab, you will see the **Application Environment** tree structure with the following nodes:

- The **Editors** folder contains editors that have been made available upon request to your enterprise by Software AG Support.
- The **Custom Editors** folder contains the custom editors you have created. You can manually create and design custom editors here.
- The **Class Settings** folder allows you to define class settings that you can assign to view schemes.
- The **View Schemes** folder allows you to define view schemes that you can assign to user profiles.

- The **Object Views** folder allows you to configure custom object views for the user community.
- The **Solution Managers** folder contains XML objects that allow you to configure various solution functionalities.
- The **Wizards** folder allows you to define wizards for the maintenance of data in the Alfabet solution.
- The **GUI Schemes** folder allows you to specify the visualization of the Alfabet user interface.
- The **Text Templates** folder contains all text templates used for automatic e-mails executed via batch processes for assignments, monitors, and workflows. Text templates are predefined for the relevant contexts in Alfabet in which they are used. New text templates should only be created for the workflow functionality. However, you can edit all existing text templates as well as define localized (translated) versions of the existing text templates.
- The **Monitor Templates** folder allows you to edit predefined templates that define which properties may be monitored for a select class by a monitors created in the Alfabet solution.
- The **Icon Gallery** folder allows you to define groups comprising icons that may be defined to represent qualitative or performance information (such as indicators) or associated with functionalities. The icon gallery includes a predefined set of icons. However, custom icons can be uploaded to Alfabet in the **Icons** tab.
- The **Selectors** folder allows you to define custom selectors that can be implemented in Alfabet search functionalities as well as selectors available in editors and page views.
- The **Visualization Items** folder allows you to define HTML templates that can be implemented in the workflow capability.
- The **Widgets** folder allows you to configure widgets to use in the configured widget reports that can be embedded in custom object views and object cockpits.
- The **XML Objects** folder contains XML objects that enable you to configure many aspects of the Alfabet solution environment.

Functions Tab

The **Functions** tab allows you to configure custom explorers. Custom explorers are implemented as functionalities and can be assigned to menu items or navigation pages.

Workflows Tab

The **Workflows** tab allows you to configure workflow templates in order to make Alfabet 's workflow functionality available to your user community.

Reports Tab

In the **Reports** tab, you can configure reports available in the Alfabet solution.

Diagrams Tab

n the **Diagrams** tab, you can configure custom diagram item templates that can be made available in diagrams by means of the Alfabet Diagram Designer.

Publications Tab

In the **Reports** tab, you can configure export of data to Microsoft® Word-based publications.

Surveys Tab

In the **Surveys** tab, you can configure data capture campaigns that consist of the definition of customized object classes, object class properties, class settings, enumerations, custom editors, wizards, workflows, and configured reports to support the data capture process.

Diagrams Tab

In the Diagrams tab, you can configure custom diagram item templates that allow alternative visualizations of the objects displayed in diagrams. They can be designed with a different shape, size, color, graphic icon, attributes and text than the standard diagram item template used for the object.

ADIF Tab

In the **ADIF** tab, you can access the Alfabet Data Integration Framework (ADIF) capability that supports the batch import, export, and manipulation of large amounts of data in the Alfabet database. The ADIF scheme is an involved process and information about ADIF is provided separately in the reference manual *Alfabet Data Integration Framework*.

Working with Configuration Objects

Any new object that you create in Alfabet Expand is a configuration object. This includes, for example, custom editors, custom wizards, custom object views and object cockpits, workflows and workflow steps, configured reports, class settings, etc. Configuration objects are displayed as nodes in the explorers available in Alfabet Expand.

Keep the following in mind when working with configuration objects:

- **Most functionalities in Alfabet Expand are available by right-clicking a node in the explorer.** To access functionalities associated with a selected object (for example, to create new objects or subordinate objects), right-click the object in the explorer tree to open a menu displaying the functionalities.
- To display an attribute window in the right pane in which you can define the objects attributes, click the object in the explorer tree. In the attribute window, click an attribute to display instructional text

at the bottom of the screen providing useful information about the attribute. This information will help you understand and define a value for the attribute.

- Objects in the explorers display icons that indicate the editability of the object:
- A private object  with a red lock is ReadOnly and is displayed to provide guidance in defining new objects.
- A protected object  with an orange lock may have some property values that can be changed but the object may not be deleted.
- A public object  with no lock may have all relevant property values changed and may be deleted.
- Some objects in explorers may displays icons indicating their object state (for example, workflows).To update the icon in the explorer after the **State** attribute has been changed for a configuration object, you must right-click the object in the explorer and select **Rescan Tree**. The following states are displayed in Alfabet Expand:
- An active object  will have a green symbol.
- A plan object  will have an orange symbol.
- A retired object  will have a red symbol.
- To ease the configuration of some configuration objects, an editor pane is available by double-clicking the object in its explorer. For example, the following editors are available:
- A design functionality supports the definition and visual design of custom editors  . The editor allows you to select view controls via buttons in the toolbar and place them in the custom editor. Clicking the view control refreshes the attribute window in the right pane in which the view controls attributes can be defined.
- The XML object editor supports the definition of a selected XML object  . XML elements and attributes can be defined directly in an easy-to-use editor interface as well as the conventional XML format. The new editor provides verification of the XML syntax by means of Microsoft® IntelliSense® technology. Any errors in syntax are reporting with color-coding and can be immediately corrected by the solution designer. For more information about editing an XML object and XML syntax verification, see the section Working with XML Objects.
- You must always click the **Save**  button in the Alfabet Expand toolbar to save your changes to configuration objects.

Using the Show Usage Functionality

The **Show Usage** functionality is available for most configuration objects and allows you to understand where a selected configuration object is used by another configuration object. For example, you may want to clean up the wizards folder and delete wizards that are obsolete and no longer in use. Or perhaps you no longer know where your configured wizards are used. By right-clicking a wizard and selecting the **Show Usage** option, you will see the class settings and workflow templates where the wizard is implemented.

Please note that guide pages/guide views may have configured reports embedded in them. The **Show Usage** functionality does not list any configured reports that are embedded in guide pages/guide views.

Most configuration objects will display the **Show Usage** option when you select the object and open the context menu. When you open **Show Usage** dialog, the information about the selected configuration object's usage is displayed in a matrix. The selected configuration object is displayed on the X-axis and the path to the object referring to the selected configuration object is displayed on the Y-axis. A filter is available that allows you to limit the information to configurations relevant for a specified class.

Using the Delete, Cut, Copy, and Paste Functionalities

Before you use the **Delete** or **Cut** operations, you should first use the **Show Usage** functionality in order to understand where the configuration object you are removing may be used. The **Delete** or **Cut** operations may lead to an object being irrevocably deleted from the Alfabet database, thus causing inconsistencies if the configuration object is referenced by other configuration objects in your solution configuration. For more information about the **Show Usage** functionality, see the section Using the Show Usage Functionality.

- The **Delete** operation allows you to select a configuration object in order to remove it from the solution configuration. The configuration object will be irrevocably deleted from the Alfabet database.
- The **Cut** operation allows you to select a configuration object in order to paste its definition (attributes) to another object. When you cut the source object, it will be removed from the configuration object that you are removing it from. Depending on the type of configuration object that you are cutting, the configuration object may be deleted from the Alfabet database. To cut the selected object, right-click the object and select **Cut** . Confirm the warning message explaining that the object will be deleted.
- The **Copy** operation allows you to select a configuration object in order to paste its definition (attributes) to another object. When you copy the source object, it will not be deleted from the explorer. To copy the selected object, right-click the object and select **Copy** .
- The **Paste** operation allows you to take the cut or copied definition of a source object and use it to replace the definition of a selected target object. When you paste to a target object, you will replace the existing definition of the target object with the definition of the source object you have cut or copied. To paste a cut or copied definition to the selected object, right-click the object and select **Paste** .

  There is no Undo action! Cut and **Paste** actions are not reversible. Therefore, you should be certain that a source object may be deleted as the result of the **Cut** action and the definition of a target object may be overwritten as the result of a **Paste** action.

Searching for a Configuration Object in the Explorer Tree

To search for a specific configuration object in the explorer tree, right-click the selected node and select **Navigation** > **Find Node**. In the editor that opens, enter the name of the node in the **Find What** field. You can use the * symbol as a wildcard if you do not know the node's complete name. Click **Find Next** until you find the relevant node.

Defining Attributes for Configuration Objects

If you click a configurable object in an explorer, an attribute window will be displayed on the right side of the screen. The attribute window displays various attributes for the selected object. Please keep the following in mind when working with the attribute window:

- The attribute fields displayed may depend on the values entered for other fields. For example, when defining a custom property  , the fields available will depend on the **Property Type** attribute that you define for the custom property. In this case, the attribute window will be automatically updated to show the relevant attributes.
- Attributes displayed in black text can be edited. Attributes displayed in grey text cannot be edited.
- Click an attribute to display instructional text at the bottom of the screen providing useful information about the attribute. This information will help you understand and define a value for the attribute.
- To define an attribute in the attribute window, click the cell to the right of the attribute field that you want to edit. Fields can be edited in different ways.
- Click in an attribute's cell and enter text directly in the cell.
- Click the **Drop-Down**  button to select an item in a drop-down menu.
- Click the **Browse**  button to open an editor or dialog box.
- A validation mechanism checks for correct syntax when defining the **Name** attribute for most configuration objects. The technical name of a configuration object may not begin with an empty space nor include any of the following special characters: **\ / * ? " > < | ':**
- If the technical name of a configuration object (such as a custom wizard, custom editor, object view, etc.) is changed, the name will be correctly updated in the class setting referencing the changed object during design time. Please note that the name of a changed configuration object will not be updated in guide pages nor are the guide pages referencing a configuration object listed in the **Show Usage** functionality. If you plan to change the name of a configuration object, the reference in the guide page should be changed prior to changing the configuration object name. For more information about configuring guide pages, see the reference manual *Designing Guide Pages for Alfabet*.
- Because an unambiguous identification of configuration object is required to support the parallel connection of Alfabet Expand to the Alfabet database, a unique key constraint has been set for the `Name` attribute of configuration objects in Alfabet Expand. If two existing configuration objects have the same name in the Alfabet meta-model, the name of one of the objects will be changed automatically during migration to a new version of Alfabet. A message about the change will be written to the migration log file.

- If a configuration object requires an XML definition, the syntax specified in the XML editor will be validated when the XML editor is closed.
- You must always click the **Save** button in the Alfabet Expand toolbar to save your changes.

Working with XML Objects

An XML object is a configuration file made up of XML elements that allows definitions to be specified for diverse objects and functionalities in the solution environment. In Alfabet Expand, you will find a variety of existing preconfigured XML objects that serve as templates to configure the XML objects relevant to your enterprise.

Alfabet Expand provides an XML editor that allows you to edit the XML objects. XML attributes can be defined directly in an easy-to-use editor in the context of Alfabet Expand. The editor provides verification of the XML syntax by means of Microsoft® IntelliSense® technology. Any errors in syntax will be reported with color-coding and can be immediately corrected.

Please note that the XML objects ***CUST_LDAP_PERS_Selector***, ***CUST_SQLDB_PERSON_SelectorDef***, and ***OracleSOAManager*** are relevant for enabling interoperability with external systems. Customers will receive specific instructions from Software AG Support if interoperability is required.

You can edit the configuration to suit the needs of your company or user community. You can use the preconfigured XML object provided by Software AG as a reference for your XML definitions.

For details about the specific definition of the XML objects provided by Software AG, search for the XML object name in the index of the documentation (PDF or online Help) to locate the relevant information describing the meaning of the XML elements.

To configure an XML object:

1) In the **Presentation** tab, expand the **XML Objects** folder and navigate to the relevant XML object that you want to define.
2) Right-click an existing protected XML object that you want to define and select **Edit XML Object**. The XML editor opens in the center pane.

Please keep the following in mind when defining XML elements in an XML object:

- The **XML XSD** attribute displays the XML schema definition relevant for the XML object.
- The **XML Template** attribute displays the standard definition for the XML object. Before you edit the XML object, you can copy and paste the predefined information in a standard text editor as a reference for the initial definition provided by Software AG. At any point, you can replace your configuration with the standard definition provided by Software AG. To do so, right-click the XML object and select **Replace Configuration with Default Values** to view all possible.

- The **XML Definition** attribute is where you must specify the XML object for your solution configuration.
- The **Visible in Administrator** attribute allows you to specify whether the XML object shall be available in the tool Alfabet Administrator (= `True`).
- In some cases, you may need to add additional elements to the XML object. For example, you can configure an unlimited number of XML elements `Stereotype` in the XML object `ValueManager`. To add additional elements, you must copy an existing XML element, paste it correctly in the XML object, and modify the attributes.
- If you enter the name of an object class or property (standard and custom), you must enter the correct value of the **Name** attribute of the object class. In some cases, the name deviates from the caption of the object class displayed in the Alfabet interface. A misspelled name will be ignored and could be a source of error in the solution.
- Boolean values should be written in lower-case letters (`true` / `false`). Please note that errors may occur in the implementation if Boolean values are entered in upper-case letters (`TRUE` / `FALSE`).
- Lists must be comma-separated with no empty spaces. For example, for the XML attribute `MappingClasses`, you would enter: `Domain,BusinessFunction,ITMasterPlanMap,ITStrategyMap`
- If a color is to be defined for an XML attribute, you must enter a Windows, Web, hexadecimal, or RGB color value.
- Special characters should be avoided unless explicitly required for an XML attribute.
- If you want to enter a string that contains special characters (for example, a greater than (>) or lesser than (<) symbol), you must replace the special characters with respective XML compliant code, for example:
  - > for >
  - < for <
  - " for "
  - [for [
  -] for]
- For more information about including queries in XML objects, see the section Defining Filters for Configured Reports and Selectors.

3) In the XML editor, edit the necessary XML attributes, as needed. To do so, edit the value in the apostrophe marks ("XXX").

4) In the toolbar, click the **Save**  button to save your changes. Review the bottom pane to determine if there are any errors in the XML syntax.

- Purple errors represent mistakes in the XML element.
- Red errors represent mistakes in the XML attribute.
- Blue errors represent mistakes in the XML attribute value.
- The order of child elements in an XML element should not be changed.
- Errors are only reported for mistakes in the XML syntax. You must ensure that the value defined for the XML attribute is relevant. (For example, that class names are correctly spelled, that "`true`" or "`false`" is entered for a Boolean property type, that lists are comma-separated with no empty spaces, etc.)

5) Make corrections to the syntax as needed and click the **Save**  button to save your changes and recheck the syntax. If errors continue to exist, continue this procedure until all errors have been resolved, or refer to the **XML XSD** attribute to understand the XML schema definition relevant for the XML object.

Using Server Variables in Configurations

A value for an XML attribute can either be defined as a string directly in the XML element or via a server variable. It is not possible to define a value with a server variable written as part of a string. The complete value of the XML element in the XML object must be substituted with a server variable. In an XML object, the server variable definition must be specified with the equal symbol followed by the value to be assigned to the XML attribute in double quotes. For example: `<XML attribute>="$SQLSERVER"`. The server variable is referenced in the relevant XML attribute as: `$`<server variable name>. For example, a server variable called `SQLSERVER` is referenced as `$SQLSERVER` in the XML object.

Server variables are defined in the Alfabet Administrator. To define a server variable to use in XML objects:

1) In the Alfabet Administrator, click the **Alfabet Aliases** node in the **Administrator** explorer.
2) In the table on the right, select the server alias that you want to define a server variable for and click the **Edit**  button. The alias editor opens.
3) Go to the **Variables** tab and click the **New** button. A dialog box opens.
4) In the **Variable Name** field, enter a unique name for the server variable.

   The server variable name may only contain letters (English alphabet), numbers, and the underscore symbol.

5) In the **Variable Value** field, specify the value to use for the server variable.

   If the variable value contains a special character according to XML standards (for example: &, %, ;,<,>), these characters must be replaced by with their respective XML compliant code (for example: `&` for &)

6) Click **OK** to save your changes. The server variable definition appears in the list of server variables.

To edit or delete the server variable, select the server variable in the table and click the **Edit** or **Delete** button below the table.

7) Click **OK** to save your changes and close the editor. The server variable definition is now available in the server alias configuration and can be used in the relevant XML objects available in the **Integration Solutions** folder in Alfabet Expand.

Accessing and Editing Objects in Alfabet

Sometimes Alfabet objects must be accessed and edited for administrative reasons. For example, it may be necessary to reassign objects to a new authorized user if the previous user who was responsible for those objects is no longer part of the Alfabet user community. Or it may be necessary to correct an object's release status that may have accidentally been set to a non-editable release status.

Normally, this type of object maintenance is carried out by a user administrator accessing the Alfabet solution by means of the `Admin` user profile but it can also be done by accessing the Alfabet solution from Alfabet Expand.

User administrators and users of Alfabet Expand have complete access to all objects in Alfabet regardless of the access permissions defined for such objects. You can make specific changes to the solution environment by working directly in the Alfabet solution that you access in Alfabet Expand.

To access the Alfabet solution from Alfabet Expand:

1) Go to the **Admin** tab and expand the **User Profiles** node.
2) Right-click the user profile that provides access to the object you want to edit and select **Run Application with Selected User Profile**. The Alfabet solution opens.
3) Navigate to the object that must be modified and make the relevant changes.
4) When the changes have been made, close Alfabet. In the toolbar, click the **Save**  button to save your changes. The changes are saved to the Alfabet database.

Getting An Overview of the Current Configuration

Several object classes in the Alfabet meta-model store information about the current structure of the object class model including both standard and customer defined configurations performed on Alfabet object classes as well as the current configuration of the presentation model, like the configuration of object profiles. The information stored in these tables can be used to build configured reports that inform about selected aspects of the configuration.

Each of these object classes has class settings, a simple object profile with an attribute section only and a selector assigned.

Each object in the `ALFA_MM_*` and `ALFA_PM_*` tables has a unique `REFSTR`. References between the tables are provided via properties of the type `Reference`.

The following object classes are available to inform about the meta-model structure:

- ALFA_MM_CLASS_INFO: Each object class in the Alfabet meta-model is stored as an object of the ALFA_MM_CLASS_INFO class. The properties of the ALFA_MM_CLASS_INFO object class return relevant attributes of the object class, like for example the **Tech Name** and **Caption** and whether the object class can have mandates. The table only stores information about object classes that are visible in the Meta-Model tab in Alfabet Expand.

- ALFA_MM_STEREOTYPE_INFO: Each stereotype defined in the XML object in the attribute **Stereotypes** of an object class is stored as an object of the ALFA_MM_STEREOTYPE_INFO class. The properties of the ALFA_MM_STEREOTYPE_INFO class inform about the object class the stereotype is defined for and relevant attributes of the relevant XML element Stereotype.

- ALFA_MM_PROP_INFO: Each object class property in the Alfabet meta-model is stored as an object the ALFA_MM_PROP_INFO class. The properties of the ALFA_MM_PROP_INFO class inform about the object class the object class property belongs to and relevant attributes of the object class property, like for example the data type.

- ALFA_MM_RELATION_INFO: Each relation established via an object class property in the Alfabet meta-model is stored as an object of the object class ALFA_MM_RELATION_INFO. The properties of the ALFA_MM_RELATION_INFO store the information about the object class property establishing the relation, the object classes the relation is established from and pointing to and whether the relation is a one to one, one to multiple, or multiple to multiple relation. If an object class property in the Alfabet meta-model stores references to multiple object classes, a separate object of the object class ALFA_MM_RELATIONS_INFO is created for each target object class.

- ALFA_MM_ENUM_INFO: Each enumeration item created for an enumeration in the Alfabet meta-model is stored as an object of the object class ALFA_MM_ENUM_INFO. The properties of the ALFA_MM_ENUM_INFO object class store information about the enumeration the enumeration item is defined for and the relevant attributes of the enumeration item.

- ALFA_MM_INTEGRITY_INFO: The attribute **Integrity Info** of an object class defines which objects referenced by an object of the current class via defined object class properties are dependent objects. If an object is deleted, all objects that are referenced via one of the properties defined in the **Integrity Info** attribute are also automatically deleted. Each object class property in the Alfabet meta-model that is part of an **Integrity Info** definition is stored as an object of the object class ALFA_MM_INTEGRITY_INFO. The properties of the object class ALFA_MM_INTEGRITY_INFO store information about the master object class, which is the one for that an object is actively deleted, the object class for those referenced objects is automatically deleted as well, and the object class property that is establishing the dependent relation.

  Please note the following about the information stored in the ALFA_MM_INTEGRITY_INFO table.

  - Integrity references are only stored in the ALFA_MM_INTEGRITY_INFO table if both the master class and the referenced object class are visible in the **Meta-Model** tab in Alfabet Expand.

  - The **Integrity Info** attribute is inherited within the hierarchy of object classes. The hierarchical structuring of object classes is not visible in the **Meta-Model** tab in Alfabet Expand. Customers with a license for ADIF can see the object classes within the hierarchical structure in the **Meta-Model** node of the **ADIF** tab explorer. If an integrity reference is not included in the **Integrity Info** attribute of the master object class or the referenced object class but nevertheless stored in the ALFA_MM_INTEGRITY_INFO table, it is inherited from a hierarchical parent class.

- ALFA_MM_CAPABILITY: Each object class of the Alfabet meta-model is assigned to one or multiple Alfabet capabilities. Each combination of object class and capability is stored as an object of the object class ALFA_MM_CAPABILITY.
- ALFA_MM_CULTURE_INFO: Each culture defined in the Alfabet meta-model is stored as an object of the ALFA_MM_CULTURE_INFO object class. The properties of the ALFA_MM_CULTURE_INFO object class return relevant attributes if the culture, like for example the setting for the data translation capability.
- ALFA_PM_OBJECTVIEW_INFO: Stores information about the object views in the Alfabet meta-model. The properties of the ALFA_PM_OBJECTVIEW_INFO object class return relevant attributes of the Object View configuration object, like for example the **Name** and **Caption** and the object class the object view is defined for.
- ALFA_PM_OBJECTVIEWDETAIL_INFO: This object class stores information about which page views, configured reports, and object cockpits are assigned to the object view in the object view configuration of the Alfabet meta-model. The information about the page view and report configuration includes information about the relevant work space. The information is stored as references to the ALFA_PM_*_INFO object classes storing information about the configuration objects assigned to the object cockpit as well as the ALFA_REPORT table for configured reports.
- ALFA_PM_COCKPIT_INFO: This object class stores information about the object cockpit configuration of the current Alfabet meta-model. The properties of the ALFA_PM_COCKPIT_INFO object class return a reference to the relevant object view information in the ALFA_PM_OBJECTVIEW_INFO object class and relevant attributes of the object cockpit configuration object, like for example the **Name** and **Caption**, defined visibility conditions and which kind of help is provided for the object cockpit.
- ALFA_PM_COCKPITDETAIL_INFO: This object class stores information about which page views and configured reports are included in object cockpits. The information includes a reference to the information about the relevant object cockpit information in the ALFA_PM_COCKPIT_INFO object class and references to the relevant information about the standard page view or configured report in the ALFA_PM_PAGEVIEW_INFO object class or ALFA_REPORT object class respectively.
- ALFA_PM_PAGEVIEW_INFO: This object class stores information about the standard page views in the Alfabet meta-model.
- ALFA_PM_WORKSPACE_INFO: This object class stores information about the work spaces in the object profile configuration of the Alfabet meta-model.
- ALFA_PM_EDITOR_INFO: This object class stores information about the editors configured in the Alfabet meta-model.
- ALFA_PM_WIZARD_INFO: This object class stores information about the wizards configured in the Alfabet meta-model.
- ALFA_PM_WIZARDSTEP_INFO: This object class stores information about the wizard steps configured for the wizards in the Alfabet meta-model, including a reference to the information about the relevant wizard in the ALFA_PM_WIZARD_INFO object class.
- ALFA_PM_CONDITION_INFO: This object class stores information about the **Condition** s configured in the **Presentation** tab of Alfabet Expand. ALFA_PM_*_INFO object classes storing information about configuration objects with a visibility condition reference this object class.

For information about the object class properties of the ALFA_MM_* and ALFA_PM_* object classes, see the reference manual *Alfabet Meta-Model*.

The content of these database table will not be updated automatically when a new configuration is added to the meta-model using Alfabet Expand.

To update the information after having performed a change to the configuration of the Alfabet class model:

1) In the menu of Alfabet Expand click **Meta-Model** > **Generate Meta-Model Information** to update the `ALFA_MM_*` object class tables or **Meta-Model** > **Generate Presentation Model Information** to update the `ALFA_PM_*` object class tables.

The information is updated automatically during update of the meta-model from an AMM file or a master database.

The content of the table must not be altered by any other mechanisms.

Tools and Alfabet Functionalities for Configuration

Software AG provides a number of tools to simplify the tasks of configuration. If a specific tool is not described in this documentation, the respective manual describing the tool will be referenced.

The following Alfabet tools and functionalities are described below:

Use the Configuration Tool...	In Order To...
Alfabet Administrator	Configure the Alfabet components and manage the Alfabet database. Configure access rights to Alfabet and track Alfabet usage and audit history. The Alfabet Administrator is described in the reference manual *System Administration*.
Alfabet Expand	Configure the Alfabet solution. Many Alfabet functionalities as well as administrative tasks are based on a specific solution configuration defined in Alfabet Expand. For information about the configuration tool Alfabet Expand, see the reference manual *Configuring Alfabet with Alfabet Expand*. This is available as a PDF document on your installation CD. Alfabet Expand is currently available as either Alfabet Expand Windows or as Alfabet Expand Web. Alfabet Expand Web is available for the Alfabet Cloud Enterprise and is a web-based tool that supports most core configuration tasks. Alfabet Expand Windows is required for more complex configuration tasks including workflow configuration, ADIF, and APF.
Guide Pages Designer	Create customized as start pages for the user profiles in your user community. The Guide Pages Designer is available as a web-based tool. For information about the Guide Pages Designer, see the reference manual *Designing Guide Pages for Alfabet*. This is available as a PDF document on your installation CD.

Use the Functionality in Alfabet...	In Order To...
Administration application	The **Administration** application is accessible via the Alfabet interface by using the `Admin` user profile. For more information, see the reference manual *User and Solution Administration*. The **Administration** application contains the following modules and functionalities: • User Administration • User Profiles Administration • Users Administration • User Groups Administration • Reports Administration • Solution Administration • Monitors • Date Monitors • Consistency Monitors • Notification Monitors • Monitor Management • Workflow Administration • Internal Documents • Archive Manager • Broadcast Messages • Discussion Groups • Risk Management Templates • Technical Environments • Time Series Manager • Publications Manager • Risk Mitigation Templates by Risk Mitigation Category • ADIF Jobs Administration • Email Messages logging • Automated Data Translations • Event Administration

Use the Functionality in Alfabet...	In Order To...
	• Questionnaire Indicators • Configuration • Reference Data • Evaluations and Portfolios • Class Configuration • Diagram Views • Costs Centers • Manage Business Documents • Color Rules Manager • Enterprise Calendars • Integration Solutions Configuration • Search • Simple Search • Browse • Full-Text Search • Glossary • Reports • Publication • Admin Desktops • Organization Admin • Application Group Admin • Component Group Admin • Domain Admin • ICT Object Category Admin • Data • Capture Data • ADIF Jobs Administration

Use the Functionality in Alfabet...	In Order To...
Configuration module	The **Configuration** module is accessible via the Alfabet interface. For more information, see the reference manual *Configuring Evaluation and Reference Data in Alfabet*. The **Configuration** module contains the following functionalities: • Reference Data • Evaluations and Portfolios • Class Configuration • Diagram Views • Costs Centers • Manage Business Documents • Color Rules Manager • Enterprise Calendars • Integration Solutions Configuration

The configuration tools Alfabet Expand and the Guide Pages Designer are optional components. Their availability is regulated by your enterprise's licensing agreement with Software AG. In addition to the configuration tools, a number of small executables for batch job processing are available. Their functionality is described in the reference manual *System Administration* in conjunction with the relevant tasks to be performed.

Chapter 2: Executing Administrative Tasks in Alfabet Expand

This chapter describes the tasks that are not directly related to the configuration of an Alfabet functionality but are required to administrate the configurations performed with Alfabet Expand and to support system administration tasks.

The following administrative tasks can be carried out in Alfabet Expand:

- Anonymizing All Relevant Data in the Alfabet database
- Anonymizing Data of Selected Users
- Creating a Database Archive File Containing Anonymized Data
- Checking Anonymization Actions
- Configuring Default User Settings for the User Community
- Configuring the Use of External Sources with Alfabet
- Configuring Selectors for Data Synchronization with External Sources
- Configuring the Alfabet User Interface to Map Standard Configuration Objects to Custom Configuration Objects
- Configuring the Visibility of Tabs in Alfabet Expand

Applying Configuration Changes to Other Databases

The Alfabet database is the basis of the Alfabet software. Software AG provides mechanisms to archive part or all of the configuration performed in an Alfabet database and restore it to another Alfabet database. These mechanisms enable configuration based on a copy of a production database in a configuration environment as well as the testing of the new configuration prior to implementing it in the production environment.

Software AG provides the proprietary file format AMM for storing the configuration of an Alfabet solution in one Alfabet database and restoring it in another Alfabet database. This is the central mechanism for configuration exchange. For special data types that cannot be included in AMM files, the **Import Data Search** capability can be used to import data from one Alfabet database to another Alfabet database.

The following information is available:

- About the Storage of the Configuration in the Alfabet database
- About Storage of Changes to the Meta-Model Performed with Alfabet Expand in the Alfabet database
- Overview of Administrative Tasks Related to Solution Design
- Taking Over Configurations Performed in Alfabet Expand With AMM Based Mechanisms
- Identifying Configuration Objects via Solution Tagging
- Setting Tags for a Single Configuration Object
- Setting or Removing Tags For Multiple Configuration Objects Simultaneously
- Setting a Default Tag Automatically Applied to New and Changed Objects
- Versioning of Configurations
- Saving the Configuration of the Alfabet Solution to an AMM File
- Saving Complete or Tagged Types of Configuration Objects or the Complete Configuration with Alfabet Expand

- Saving Selected Objects of the Configuration with Alfabet Expand
- Storing all Objects Tagged With a Selected Tag to an AMM File
- Updating the Configuration of the Alfabet Solution Environment with Alfabet Expand
- Direct Import of the Solution Configuration from a Master Database
- Overwriting the Solution Configuration with the Configuration of a Master Database
- Merging the Solution Configuration with the Configuration of a Master Database
- Checking and Repairing the Configuration Updates From a Master Database
- Correcting Issues that Occurred During Meta-Model Update
- Checking the Success of Meta-Model Updates
- Securing and Checking Database Consistency with the Meta-Model
- Comparing Database Configurations
- Exporting Information About the Custom Configuration in an XML File
- Building Custom Reports that Inform About the Current Structure of the Standard And Custom Meta-Model
- Importing Objects of Configuration Relevant Object Classes from a Master Database
- Configuring the Connection Between the Source and Target Database
- Configuring the Source Database to Provide Access to the Relevant Data via the Alfabet RESTful Services
- Configuring the XML Object AlfabetIntegrationConfig of the Target Database
- Creating an Alfabet Database Connection in the Target Database
- Providing Access to the Import Data Search Functionality
- Optional Configuration for Detection of Changes to Configuration Object
- Importing Configuration Objects from a Source Database
- Managing Assemblies
- Uploading Assemblies from One Database to Another Database via AMM File

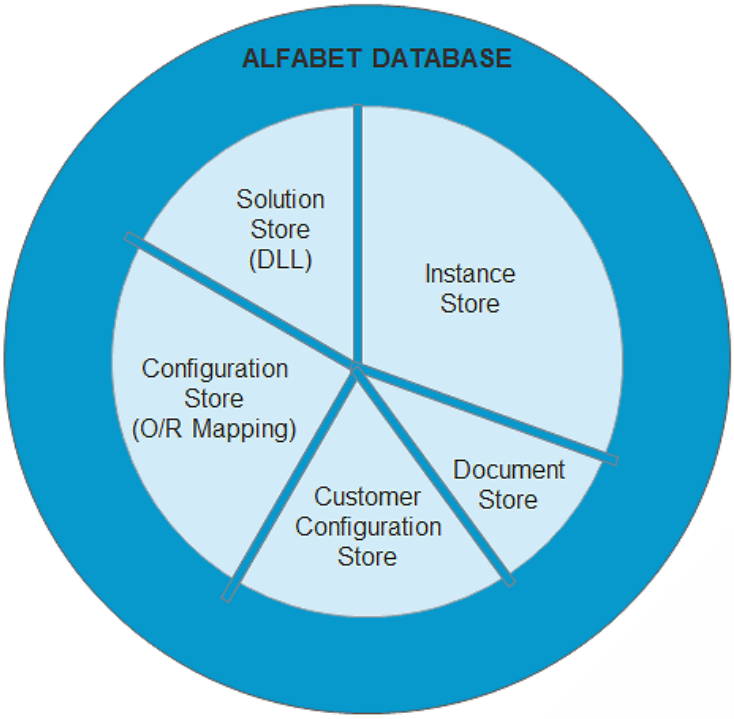
About the Storage of the Configuration in the Alfabet database

The Alfabet database comprises the following information:

- The functionality of the Alfabet solution (solution store). The software functionality is delivered by Software AG as a DLL file. The main solution assembly is typically uploaded to the Alfabet database by Software AG Support only during the upgrade to new Alfabet versions. Special assemblies delivered to the customer upon request can be uploaded using the **Assemblies** functionality in the tool Alfabet Administrator. For more information about managing assemblies, see the section

Managing Assemblies in the reference manual *System Administration*. Assemblies uploaded to one Alfabet database can then be stored to AMM files and transferred to other Alfabet database s.

- The configuration and database structure provided by Software AG for the Alfabet solution (configuration store). This meta-model information is typically updated by Software AG Support only during upgrade to a new Alfabet version. The update is executed via an AMM file that includes all meta-model related information.
- The configuration performed by the customer using Alfabet Expand or Alfabet Administrator (customer configuration store). Customers can save the configuration in AMM files and merge the configuration to another Alfabet database or overwrite the existing configuration with the customer configuration stored in an AMM file. For more information about saving and restoring the configuration store, see Saving the Configuration of the Alfabet Solution to an AMM File.
- The instances in the Alfabet inventory created by users in the Alfabet user interface (instance store). The instance store cannot be stored in AMM files. Nevertheless, the instance can include configuration relevant data such as, for example, reference data that includes indicator types and role types defined via the **Configuration** functionalities in the Alfabet user interface. To transfer data over to another Alfabet database, a direct connection between the source and target database can be configured by the solution designer and the data can then be integrated to the target database via the Alfabet user interface of the target database. Integration can be performed on an object-by-object basis.
- The documents uploaded to the **Internal Document Selector** (document store). The customer-defined guide pages that are created to provide guide pages for the Alfabet user interface are the only part of the document store that can be saved to an AMM file that can then be used to overwrite guide pages of a target database with the version stored in the AMM file. All other documents in the **Internal Document Selector** can only be stored in normal database backup files on the database server level.



About Storage of Changes to the Meta-Model Performed with Alfabet Expand in the Alfabet database

Technically, changes to the meta-model may be performed via Alfabet Expand while at the same time:

- users are currently changing the data in the same database via a running Alfabet Web Application.
- the meta-model is changed by another solution designer in another instance of Alfabet Expand connected to the same database.

It is highly recommended to change the configuration via Alfabet Expand in a solution environment that is not connected to the production database to avoid that configuration changes interfere with users changing the data in the same database.

It is also highly recommended to change a database with one Alfabet Expand instance at a time.

Even if only one solution designer is changing the configuration in a development environment, it is useful to know how and when the configuration changes performed in Alfabet Expand are available in the Alfabet database and to the user connected to the Alfabet database via a running Alfabet Web Application. The solution designer might want to check the configuration by accessing the Alfabet user interface via a running Alfabet Web Application in the development environment.

Alfabet Expand Windows, Alfabet Expand Web and the Alfabet Web Application are loading a working copy of the configuration available in the Alfabet database at startup. The following applies for configuration changes:

- If a solution designer changes the configuration via Alfabet Expand Windows or Web, the changes are instantly applied to the working copy her/his Alfabet Expand instance, but they are only stored into the Alfabet database when the solution designer clicks the **Save** button in the toolbar of Alfabet Expand.
- This option re-loads the configuration from the Alfabet database into the local working copy of the configuration, overwriting any local changes.
- If a solution designer saves the configuration via the **Save** button, the configuration of the Alfabet database is changed, but changes are not instantly visible on an Alfabet user interface that is connected to the Alfabet database via a running Alfabet Web Application. The Alfabet Web Application also uses a working copy that is updated in regular intervals. Changes to the configuration of the Alfabet database might therefore only be visible to the user after several minutes. To ensure that the changes are visible in the user interface, the solution designer should re-start the web server hosting the Alfabet Web Application prior to checking new changes in the user interface.

  The Alfabet Web Application can be configured to work in `Design` mode for testing purposes. One of the options available in design mode is a button **Meta-Model** in the user interface with options to re-read the meta-model or to re-start the web server. For more information, see *Special Configuration of Testing Environments* in the reference manual *System Administration*.

- The working version of the meta-model in Alfabet Expand is not updated automatically. Therefore, if one Alfabet Expand instance saves configuration changes to the database, the working version of the meta-model of all concurrently active Alfabet Expand instances is outdated. To update the working version of Alfabet Expand with the current version of the meta-model configuration in the Alfabet database, click **Meta-Model** > **Reread Meta-Model** in the menu of your Alfabet Expand instance.

- Changes to object classes, like for example introducing a new custom property, are central changes that might interfere with the storage of object data that is stored when a user creates objects of the class. To ensure that the central class model a user is working with is always identical with the class model stored in the Alfabet database, saving a change to the object class model via the **Save** button will cause a semantic lock to the Alfabet database. All currently open connections from Alfabet components will be closed. A message will be displayed to solution designers concurrently working with other Alfabet Expand Windows instances, informing them about the lock to the database. Alfabet users currently working on the same Alfabet database in the Alfabet user interface or with Alfabet Expand Web are informed that their session expired. The meta-model changes will then be updated to the Alfabet database. The Web server must be restarted.

  If the connection to the database is closed while you have already created or changed configurations, you can save your unsaved changes into an AMM update file, reconnect to the Alfabet database, and check in the configuration change via the AMM update file. Please note that this automatically saves the changes. For information about storing and re-storing configurations in AMM files, see Saving the Configuration of the Alfabet Solution to an AMM File. In the functionality to selectively store configuration objects in AMM files a search for unsaved objects is available that eases the storing of unsaved configuration in this context.

- A backup of the Alfabet database should always be made before making changes in order to prevent the inadvertent and irrevocable loss of important configuration data.

Overview of Administrative Tasks Related to Solution Design

It is recommended that all changes to the solution configuration are performed in a development environment and tested in a test environment before the solution configuration is migrated to the production environment. All development and test environments should be based on copies of the production database. An overview of the development, test, and production environments is provided in the section *Best Practice Installation and Workflow* in the reference manual *System Administration*.

Software AG provides mechanism that allows the customer configuration specified in the development environment to be saved and restored in another database independent of other parts of the database. With these mechanisms, the configuration developed by a solution designer in a test environment can be first taken over to a test database and, after successful testing, to a production database without affecting the content of the instance store, which means the data created by users via the Alfabet user interface.

In general, two main mechanisms are available for taking over configurations from one database to another database:

- The customer configuration store and optionally the solution store can be saved to an AMM updater file. The updater file allows the configuration of a target database to either be merged or replaced by the configuration stored in the AMM updater file.
- Configuration relevant objects stored in the instance store of the Alfabet database can be imported to a target database via a direct connection between the source (master) database and the target database. The target database must be configured to connect to the source database prior to using this functionality. The configuration can then be transferred via an administrator that is logged in to the Alfabet user interface and has access to the import functionality.

For best practice solution configuration and database maintenance, it is recommended that a solution designer should be responsible for the development database, whereas any tasks performed on the database

in the production environments should be performed by a system administrator. The tool Alfabet Administrator allows all tasks that are required for database maintenance to be performed by the system administrator. Most of the tasks can also be performed by the solution designer using Alfabet Expand. Depending on your enterprise's policies, the solution designer could optionally carry out system administration tasks on the development and test database independent of the system administrator.

A typical solution design cycle includes the following database maintenance tasks:

- A new configuration is best carried out on an up-to-date copy of the production database. Backup files created during regular backup on the database level can be used to restore a copy of the production database in a test environment.

The following is possible:

- The solution designer can restore the current database and start working with the exact copy of the production database, thus overwriting any configuration steps done in the development environment that have not yet been applied to the production environment
- The solution designer can conserve current changes to the configuration in the development environment. He/she can save the current customized configuration of his/her development database to an AMM file before overwriting the development database with the database backup file of the production database and restore the customized configuration he/she is currently working on from the AMM after database restore.
- After performing the required solution configurations, the solution designer can then save the configuration to an AMM update file. This file is then used to replace the configuration of an up-to-date copy of the production database in a test environment, for example by a system administrator using the tool Alfabet Administrator. Alternatively, a direct connection from the test environment to the development environment can be established that allows either the solution designer or the tester to update the test environment configuration using Alfabet Expand.
- Once tests have indicated that the configuration is correct, the configuration of the production database can be replaced or merged with the new configuration.
- Whether configurations have to be merged or replaced with a new configuration depends on the configuration steps performed. The AMM file includes information about the update mechanism. It can exclusively be generated with the configuration tool Alfabet Expand because the solution designer can best decide on the update requirements to be specified for the AMM file.

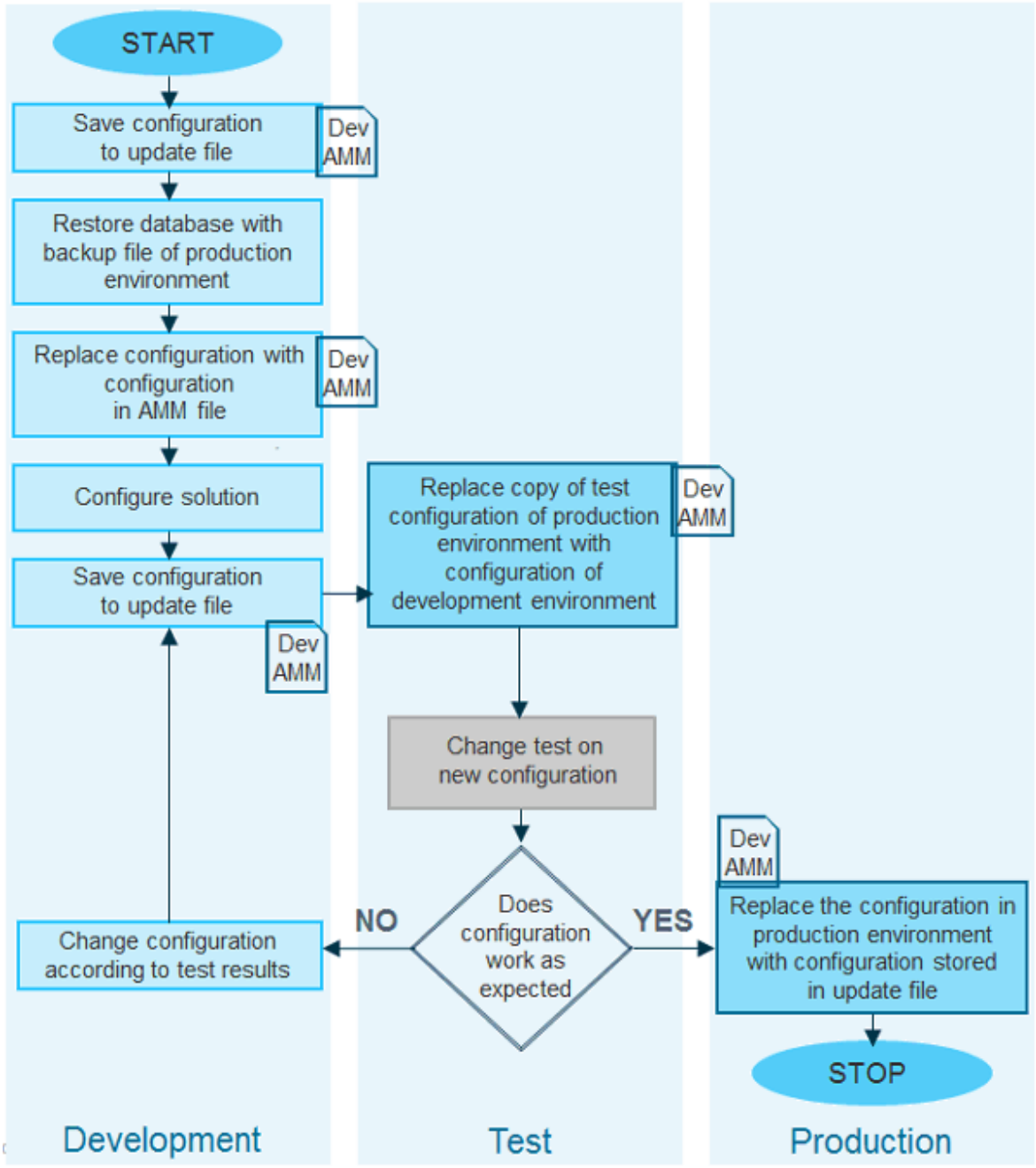


FIGURE: Best practice for the configuration of the Alfabet solution

The figure above does not include the mechanisms for transferring configuration objects via a direct connection between databases. For the update via an AMM file from another Alfabet database in the test and production environment, an AMM based update via a direct connection can alternatively be performed. If required, the import of configuration objects in the instance store shall be executed afterward the update of the configuration via the AMM based mechanisms via the **Import Data Search** functionality.

Taking Over Configurations Performed in Alfabet Expand with AMM Based Mechanisms

Different AMM mechanisms are available that allow to take over the configuration, that means all configuration objects defined in Alfabet Expand and a subset of the reference data defined in the **Configuration** functionalities in the Alfabet user interface.

Some configurations in Alfabet are not stored as part of the meta-model configuration and must be saved to a target database using other mechanisms. This applies to the following configurations:

- **Configuration of some reference data configured in the Configuration module in the Alfabet user interface**: These configurations are saved in the instance store of the database. Only the selected subset of configurations listed above can be included in AMM files. The functionality **Import Data Search** can be used for import of the reference data that cannot be included into the AMM file. For more information, see Importing Objects of Configuration Relevant Object Classes from a Master Database.
- **Mandate configuration**: The configuration of mandates is saved in the instance store of the database. The functionality **Import Data Search** can be used for import of mandate configurations. For more information, see Importing Objects of Configuration Relevant Object Classes from a Master Database.
- **Documents uploaded to the Internal Document Selector**: Documents in the **Internal Document Selector** can only be stored in normal database backup files on the database server level or added to the target database manually. This may impact the images and stylesheet files stored in the **Internal Document Selector** that are to be used in HTML templates implemented in the workflow capability. Although HTML templates can be saved to an AMM file and uploaded to a target database via the meta-model update, the images and stylesheet files must be uploaded to the **Internal Document Selector** of the target database via **Internal Documents** functionality in the Alfabet user interface, which is accessible via the `Admin` user profile.

During the creation of an AMM update file, the solution designer can decide about the scope of the configuration that shall be taken over to the target database. He/she can decide to select:

- The complete configuration.
- All configuration objects of selected types of configuration objects such as all workflows or all diagram view items.
- All configuration objects tagged with a defined tag that is set by the solution designer as part of the configuration process. It is additionally possible to store only tagged objects of selected configuration object types into the AMM file.
- A subset of configuration objects selected by the solution designer from a list of all available configuration objects according to his/her demand prior to creating the AMM file.

During the restore process, the parts of the configuration saved in the AMM file are merged with the corresponding configuration of the target database. The parts of the configuration that are not included in the AMM file are not affected by the merge action. Alternatively, the entire existing configuration in the target database can be replaced with the configuration in the AMM file. If this option is selected, the solution designer must make sure that the complete configuration is stored in the update file.

Alfabet Expand basically offers two mechanisms to take over configurations stored into AMM files to a target database:

- The AMM file is created by a solution designer from the development database in an Alfabet Expand connected to the development database. The solution designer can decide with maximum flexibility about which configuration objects to take over to the target database. The AMM file is then used to update the target database.
- The creation of the AMM file and the update of the configuration are both performed via an Alfabet Expand connected to the target database. In this case, a direct connection between the source and the target database must first be configured in the server alias of the target database. The complete update process including storage of data from the source database in the AMM file and update of the meta-model in the target database is performed on the target database, most likely by someone that did not change the configuration in the source database personally. Therefore, the selection mechanisms for taking over solution objects are restricted for this mechanism and the basic decision about overwriting the configuration in the target database or merging the configurations is defined in the server alias of the Alfabet Expand connecting to the target database.

For all AMM file-based mechanisms, translations provided in the source database for the solution configuration objects stored in the AMM file are automatically also stored in the AMM file and added to the vocabulary of the target database during update of the meta-model with the AMM file. Vocabulary review information like proposed changes and reviewer information is not stored in the AMM file. The following columns of the ***ALFA_SYS_VOCABULARY*** database table are included into the update via AMM file: `ORIGINAL`, `SOURCE`, `ACCESSIBILITY`, `HASHED_ID`, `OBSOLETE`, `CREATE_DATE`, `OBSOLETE_DATE`, and `REPLACES`.

The following information is available:

- Identifying Configuration Objects via Solution Tagging
- Setting Tags for a Single Configuration Object
- Setting or Removing Tags For Multiple Configuration Objects Simultaneously
- Setting a Default Tag Automatically Applied to New and Changed Objects
- Versioning of Configurations
- Saving the Configuration of the Alfabet Solution to an AMM File
- Saving Complete or Tagged Types of Configuration Objects or the Complete Configuration with Alfabet Expand
- Saving Selected Objects of the Configuration with Alfabet Expand
- Searching for Objects in the Find Results Table of the Meta-Model Objects Editor
- Storing all Objects Tagged With a Selected Tag to an AMM File
- Updating the Configuration of the Alfabet Solution Environment with Alfabet Expand
- Direct Import of the Solution Configuration from a Master Database
- Overwriting the Solution Configuration with the Configuration of a Master Database
- Merging the Solution Configuration with the Configuration of a Master Database
- Checking and Repairing the Configuration Updates From a Master Database
- Correcting Issues that Occurred During Meta-Model Update

- Checking the Success of Meta-Model Updates
- Securing and Checking Database Consistency with the Meta-Model
- Comparing Database Configurations
- Exporting Information About the Custom Configuration in an XML File
- Building Custom Reports that Inform About the Current Structure of the Standard And Custom Meta-Model

Identifying Configuration Objects via Solution Tagging

Solution objects can be marked with one or multiple tag names to define which solution configuration project the configuration is relevant for. This is useful to ease the selection of meta-model objects for deployment because you can search for a defined solution tag. The main reason for tagging, however, is to identify the objects that are part of a selected configuration when the configuration must be changed.

For example, a company-specific process requires the maintenance of custom attributes for an object class. The custom attributes defined for the process as well as the custom editor, the custom reports, and the workflow designed for the process are all tagged with the name of the process. After the solution is complete and successfully migrated to the production environment, an additional custom attribute was required as well as modification of the custom editor and the design of a completely different workflow.

During redesign in the development database, the solution designer is able to identify which objects might be affected by searching for the relevant solution tags. After changing the configuration, the production database must be updated. The solution designer selects all configuration objects tagged with the process name to upload to the AMM file.

During update of the production database, these objects will be updated. But in the case of the workflow template, the update is not sufficient because the old workflow template for the solution is not updated but deleted and replaced by a new workflow template with a different name. The solution designer can now configure the AMM file to trigger automatic deletion of all configuration objects tagged with the process name from the target database prior to writing the configuration stored in the AMM file to the database. In this way, the old workflow will be deleted from the target database and the new workflow template will be added.

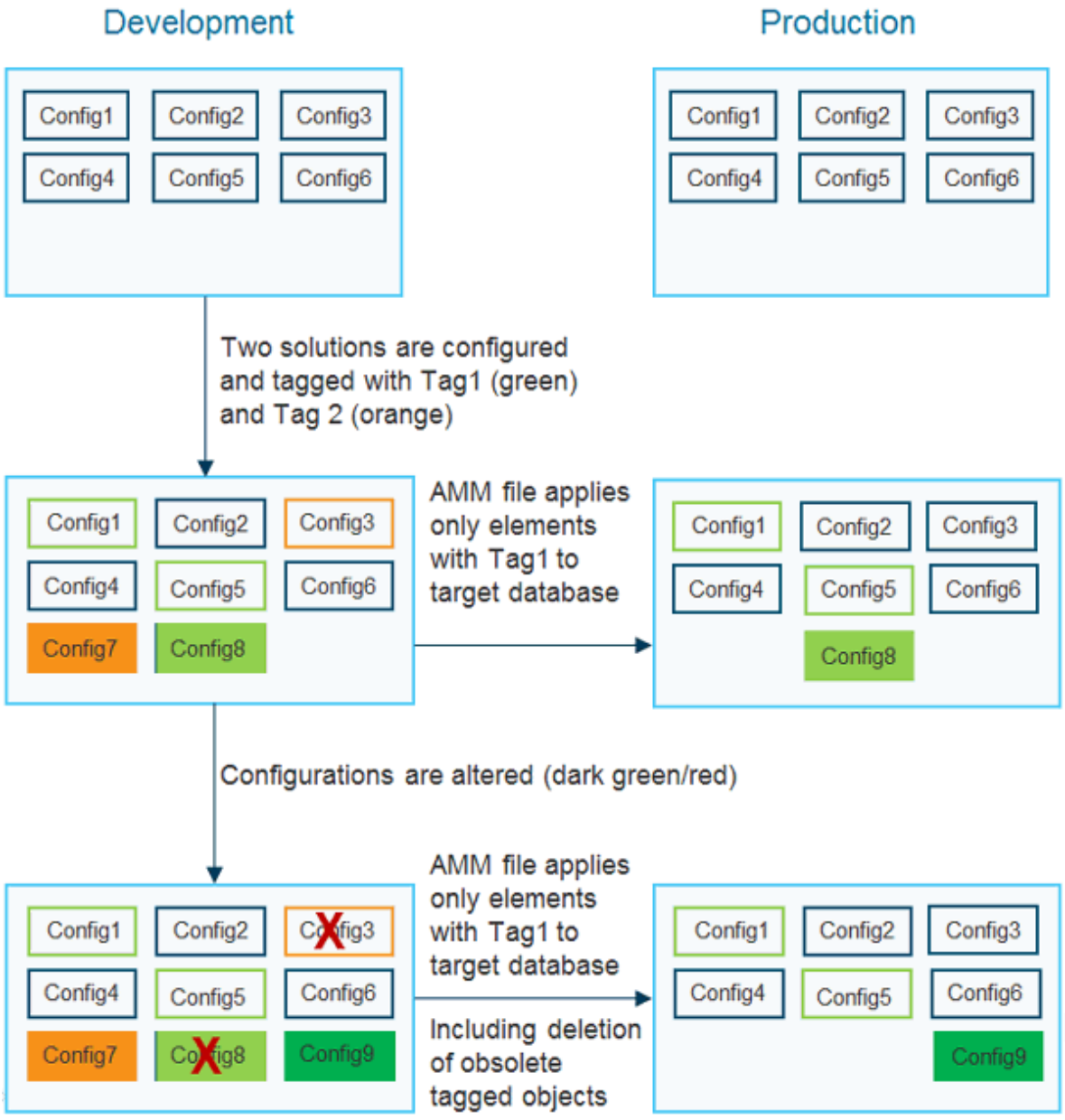


FIGURE: Configuration changes applied to a target database using solution tagging mechanisms in AMM files

In addition to applying a tag to configuration objects, you can also alter the attribute **Version** of an object each time a reconfiguration is done to specify for which version of a feature this configuration is done. The **Version** attribute is part of the **Tech Info** specification for a configuration object.

Solution tags can be added to single objects while the object is configured or to a bundle of configuration objects during selection of the configuration objects for upload to an AMM file.

To avoid errors caused by misspelling solution tag names, the tag name will be defined only once the first time it is applied and is then selected from a selector for all subsequent tagging actions. The names of the solution tags are case-sensitive.

Instead of setting the tags for each individual configuration object or a bundle of configuration objects manually, you can configure Alfabet Expand to mark all new or changed configuration objects with a default tag. A solution designer can set the default tag to the task he/she is currently working with to ensure that all objects are correctly tagged and change the default tag when starting the configuration of the next feature. If you change an existing configuration object that is tagged with a tag different to the selected default tag, the default tag is added to the list of tags for the configuration object. Existing tagging is not overwritten.

The default tag setting is removed when the user logs out from Alfabet Expand and must be re-set if required after re-login.

Once objects are tagged, you can create a AMM file including only the tagged objects via the mechanism described in the section Saving the Configuration of the Alfabet Solution to an AMM File.

If you want to delete obsolete tagged objects in the target database, make sure that the AMM file contains all objects currently tagged with the tag name are included in the AMM file and enter the name of the tag in the field **Provide comma-separated tags to find objects to be removed**. It is recommended that you use this mechanism if you want to only replace (rather than merge) the configuration of reports, workflows or ADIF schemes.

Changes to the class model, that means custom object classes and object class properties, cannot be removed via this mechanism.

Setting Tags for a Single Configuration Object

To tag a single configuration object:

1) Select the configuration object that you want to tag in the explorer.
2) In the attribute window of the configuration object, expand the **Tech Info** section.
3) Click the **Browse**  button in the **Tags** field to open the **Meta-Model Tags** window.
4) Click into the field for the attribute **Tags** to open the **Meta-Model Tags** window.
5) In the toolbar, click the **Rescan Tags**  button to update the display of available tags in the list.
6) If you want to add a new tag to the list, click the **New Tag**  button, enter a name for the tag in the **Name** field of the input dialog window and click **OK**. The names of the solution tags are case-sensitive.
7) Select the one or more checkboxes for the listed tags to apply the tags to the configuration object.

   Selecting and clearing tags in this search field can only be performed via selecting or clearing tags from the **Meta-Model Tags** window. If you try to write content directly to the field or delete the field content, an error message will be displayed informing you that the field content is invalid. To correct this error, click **Cancel** in the error message. The old content will be displayed and can be altered via the **Meta-Model Tags** window.

8) Click **OK** to close the dialog.

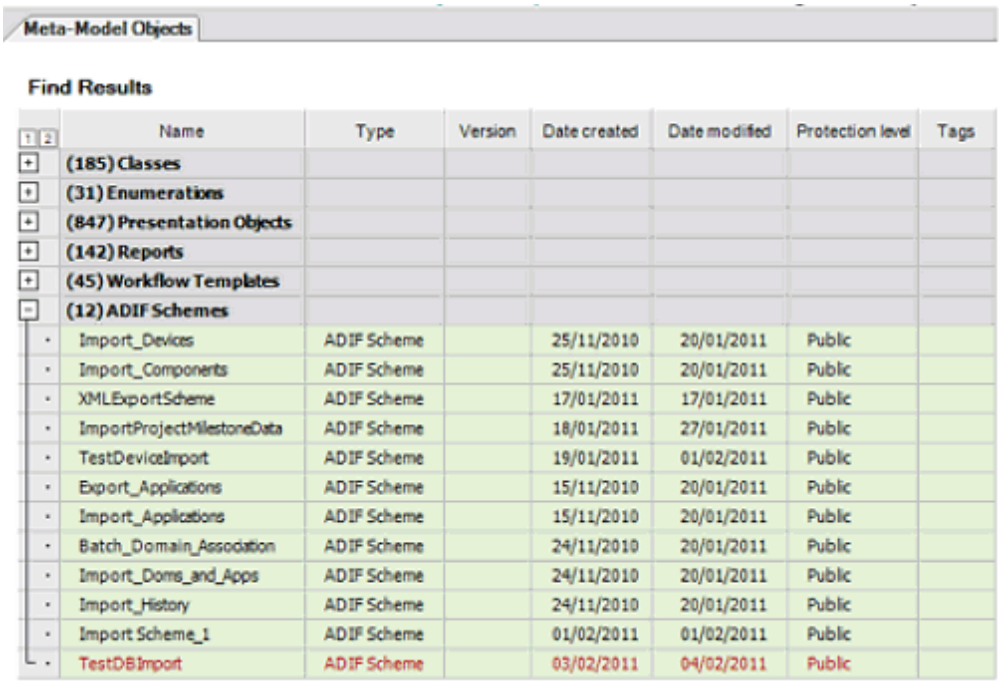
Setting or Removing Tags For Multiple Configuration Objects Simultaneously

To simultaneously set a tag or remove tags from multiple configuration objects:

1) In the Alfabet Expand menu bar, select **Meta-Model** > **Find Meta-Model Objects for Deployment**. A new **Meta-Model Objects** editor opens in the center pane between the explorer window and the attribute window.

2) In the toolbar, click the **Refresh Find Results** button.

3) All customer configured configuration objects are displayed in the **Meta-Model Objects** editor, sorted by category. The number of selectable objects in each category is displayed in front of the category name. To view the objects in a category, expand the category by clicking the sign in front of the category header. Alternatively, you can expand all categories by clicking the in the first column header.



4) Select the objects that you want to alter the tag setting for in the **Find Results** table. You can select several objects in the table at the same time by holding down the CTRL key while selecting.

> Optionally, you can use the search functionality of the **Meta-Model Objects** editor to limit the display of objects in the table to objects matching a defined search condition.
>
> For more information, see Saving Selected Objects of the Configuration with Alfabet Expand.

5) Do one of the following to alter the tag setting of the selected objects:

- To tag all selected objects, click the **Set Tag(s)** option in the toolbar to open the **Meta-Model Tags** window and click the **Rescan Tags** button to update the display of available tags in the list.

If you want to add a new tag to the list, click the **New Tag** button, enter a name for the tag in the **Name** field of the input dialog window and click **OK**. The names of the solution tags are case-sensitive.

Select one or more of the listed tags to apply the tags to the selected configuration objects.

- To remove selected tags from the selected objects, click the **Select Tags To Be Removed** button in the toolbar of the **Find Results** table. In the **Meta-Model Tags** window that opens, select the tags that you want to delete from the selected objects and click **OK** to save your changes.
- To remove all tags from the selected objects, click the **Remove All Tags from Selected Objects** button in the toolbar of the **Find Results** table.

Setting a Default Tag Automatically Applied to New and Changed Objects

To set a default tag:

1) In the toolbar of Alfabet Expand select **Meta-Model** > **Set Current Tag**.
2) If you want to add a new tag to the list, click the **New Tag** button, enter a name for the tag in the **Name** field of the input dialog window and click **OK**. The names of the solution tags are case-sensitive.
3) In the list of tags, click the link that you want to set as default tag to all new and changed configuration objects.

   The selection is removed when the user logs out of Alfabet Expand. If the selection shall be used again in the new session, the selection has to be repeated.

   Within the current session, you can revert to working without a default tag by selecting `None` from the list.
4) Click **OK** to close the window.

Versioning of Configurations

Optionally the solution designer can provide a name and version number for the configuration defined for the company. The configuration name and version defined in Alfabet Expand is written to all AMM files generated from the database and will overwrite a configuration name and version in the target database on update of the meta-model with the AMM files. the system administrator updating a database configuration with the AMM file can view the configuration name and version in the summary or the AMM content.

If a configuration name and version is defined, it will be displayed to users clicking **Help** > **About Alfabet** in the main menu of the Alfabet user interface on top of the Alfabet version information:

To define a configuration name and version:

1) In Alfabet Expand, go to the **Admin** tab.
2) In the explorer, click the **Environment** node.
3) In the attribute window, define the name and version for the configuration in the **Configuration Name** and **Configuration Version** attributes.
4) Click the **Save** button to save your changes.

Two new **Configuration Name** and **Configuration Version** attributes have been added to the **Utilities** node in Alfabet Expand Web. These attributes have already been available in Alfabet Expand Windows in the **Environment** node in the **Admin** tab. The name and version defined with these attributes will be written to all AMM files generated from the database and will be overwritten in the target database with the values in the AMM file on update of the meta-model. Solution designers can optionally use these attributes to manage the version history of changes applied to a production environment. The **Configuration Name** and **Configuration Version** values are shown in the **About Alfabet** screen.

Saving the Configuration of the Alfabet Solution to an AMM File

You can save the customized meta-model configuration of the Alfabet solution to AMM update files. An AMM file can then be used to replace or modify the configuration in an existing database. Not only is information about the database configuration contained in the AMM files. The AMM files also control how the information is updated in the target database. The decision to replace or merge the configuration with that of the target database or to migrate workflows during the configuration update is made when the AMM file is created.

AMM is a proprietary file format and therefore not recognized by Web browsers during download. If you want to store AMM files in a Web-based content management system or intranet for download, you must create a ZIP file containing the AMM file and store the ZIP file so that you can download the file via a Web browser to your local file system.

Usually, the system administrator will restore a custom configuration in the test and production environments using the Alfabet Administrator. Nevertheless, the functionality is also available in Alfabet Expand to allow the solution designer to restore a configuration.

Alfabet Expand offers different mechanisms for creation of an AMM file. The mechanisms differ in the way objects are selected for upload to the AMM file and in the availability of selection whether the configuration in the AMM file overwrites or changes the configuration in the target database.

Mechanism	Selection of objects	Overwriting or Changing the Configuration of the Target Database
Create Configuration Meta-Model Update File For more information, see Saving Complete or Tagged Types of Configuration	• Complete configuration • Selected types of configuration objects only, for example only all workflows.	Merge or replace

Mechanism	Selection of objects	Overwriting or Changing the Configuration of the Target Database
Objects or the Complete Configuration with Alfabet Expand.	• Only configuration objects tagged with a defined tag and belonging to selected configuration object types.	
Find Meta-Model Objects for Deployment For more information, see Saving Selected Objects of the Configuration with Alfabet Expand.	The solution designer can choose any combination of available configuration objects for upload to the AMM file. For configuration object types for that no objects are selected individually for upload, either all or none of the available configuration objects can be uploaded. Solution tagging can be taken into account.	Merge only
Save Meta-Model Objects Marked by Current Tag For more information, see Storing all Objects Tagged With a Selected Tag to an AMM File.	If all configuration objects that shall be merged into a target database configuration are tagged with the same tag, the tag can be set as current tag and an AMM file can be created that includes only the tagged objects.	Merge only

Saving Complete or Tagged Types of Configuration Objects or the Complete Configuration with Alfabet Expand

To save a custom configuration to an AMM file:

1) In the menu bar in Alfabet Expand, select **Meta-Model** > **Create Configuration Meta-Model Update File**. The **Create Configuration Meta-Model Update** editor opens:
2) In the **General** tab, select the location to store the information in the **Select Output File** field.
3) Provide a meaningful name and description for the update performed by the AMM file when applied to the target database in the fields **Name** and **Description**. The name and description are displayed in the **Update Meta-Model** dialog box when the AMM file is used to update the configuration in a database. It allows the person performing the update to be informed about the content of the configuration in the AMM file.
4) In the **Meta-Model Content** tab, select the configuration that you want to add to the AMM file:

- **Save Configuration**: Select the checkbox to save custom object class properties, custom enumerations and all customer defined configuration objects in the **Presentation**, **Functions**, and **Surveys** tabs in Alfabet Expand.

- **Save Reports**: Select the checkbox to save the content of the **Reports** root node of the explorer in the **Reports** tab in Alfabet Expand. If the report structure in the target database will change with the update, bookmarks linking to configured reports will be automatically updated when the configuration is merged.

  If a configured report that exists both in the target database and in the AMM file is updated, and a user or solution designer has restricted the display of columns of the configured report via the **Configure** button, new columns added to the configured reports will be hidden because they are not selected in the visibility setting.

  For information about hiding columns in configured reports, see the section Defining the Visibility of Page Views/Configured Reports Available at the Root Node of an Explorer in the chapter Configuring User Profiles for the User Community.

- **Save Workflow Templates**: Select the checkbox to save the content of the explorer of the **Workflow** tab of Alfabet Expand.
- **Save ADIF Schemes**: Select the checkbox to save the content of the explorer of the `ADIF` tab in Alfabet Expand.

  After the target database is updated by means of the AMM file, all ADIF import schemes in the target database that have the **Auto-Run** attribute set to `True` will be automatically executed. For more information about automatic execution of ADIF schemes, see *Configuring ADIF Schemes to be Automatically Executed on Update of the Meta-Model* in the reference manual *Alfabet Data Integration Framework*.

- **Save Event Templates**: Select the checkbox to save the content of the explorer of the **Events** tab in Alfabet Expand.
- **Save Diagrams**: Select the checkbox to save the custom diagram definitions and custom diagram item templates defined in the **Diagrams** tab in Alfabet Expand.

  Icons are not automatically saved if they are included in a diagram item template and the diagram item template is stored in the AMM file. If your configuration includes icons that are not available in the target database, make sure to also select **Save Icons**.

- **Save Publications**: Select the checkbox to save the content of the **Publications** tab in Alfabet Expand.
- **Save Database Views**: Select the checkbox to save the database views defined in the **Meta-Model** tab in Alfabet Expand.

  When the configuration is restored to the target database, all database views in the target database that are created based on a **Database View** configuration object will be recreated, even if **Save Database Views** is not checked. The solution designer should then check whether the creation of database views was successful. The required procedure is described in the section Saving the Configuration of the Alfabet Solution to an AMM File in the reference manual *Configuring Alfabet with Alfabet Expand*.

- **Save Icons**: Select the checkbox to save the icons uploaded to the Alfabet database.

Please note that the **Save Icons** checkbox in the **Meta-Model Content** tab must be selected in order to include any images included in the configuration of guide views.

- **Save Culture Settings**: Select the checkbox to save the content of the **Cultures** and **API Cultures** explorer nodes of the **Meta-Model** tab in Alfabet Expand.
- **Save Translation**: Select the checkbox to save the custom translation of strings displayed in the Alfabet user interface that are customized via the **Translation Editor** in Alfabet Expand.

When the translation includes the caption and description of configured reports, the **Update Translation** functionality available in the **Report** root node in Alfabet Expand must be executed on the target database after update with the AMM file.

5) Select the following options, if applicable for your configuration:

- **Replace Entire Existing Configuration**: Select the checkbox if you want the configuration in the AMM update file to overwrite an existing configuration in the target database during the **Update Meta-Model** action. If the checkbox is not selected, the configuration in the AMM update file is merged with the configuration in the target database.

If the **Replace Entire Existing Configuration** checkbox is selected, the configuration of the target database will be deleted prior to saving the configuration in the AMM file to the target database. All customer configurations that are part of the configuration of the target database but not part of the configuration saved in the AMM file will be lost. The replace mechanism deletes all parts of the configuration including the parts that are not selected to be part of the AMM update. The **Replace Entire Existing Configuration** checkbox should only be selected for AMM files when the parts of the configuration to be replaced have been selected.

Please note that reference data, assemblies and guide pages are not deleted prior to applying the configuration stored in the AMM file.

Cultures in the target database are removed with the exception of the primary culture (en-US), the default culture and the configuration culture. Any settings in the AMM file about the default culture or the configuration culture are ignored and the settings in the target database are maintained.

Configured reports that are automatically generated for the faceted semantic search functionality of the AlfaBot will not be deleted from the target database. For more information about automatically generated reports, see Managing Automatically Generated Reports.

If you want to replace only part of the configuration (for example, the configuration of workflows only), it is recommended that you use the solution tagging option. For more information about solution tagging, see Identifying Configuration Objects via Solution Tagging in the reference manual *Configuring Alfabet with Alfabet Expand*.

- **Automatically Migrate Affected Workflows**: Select the checkbox if you want the workflow templates to be automatically migrated to the target database. Migration of workflows is a complex process that is described in detail in the section Creating a Migration Definition to Update Running Workflows in the chapter Configuring Workflows.
- **Only Include Objects with the Following Tags**: The box lists all solution tags available in the current configuration. If you do not select any checkboxes, all configuration parts selected in the **Meta-Model** tab are saved without taking solution tagging into account. If you select one or multiple checkboxes, only solution objects which are both marked with a selected tag and are of a configuration object type selected in the Meta-Model tab are included into the AMM file. If solution tagging is not available for a

configuration object type that is selected, all objects of that type will be saved independent of selected tags. For more information about solution tagging, see the section Identifying Configuration Objects via Solution Tagging in the reference manual *Configuring Alfabet with Alfabet Expand*.

For example, the configuration includes 20 configured reports and 5 publications.

You have tagged two configured reports and two publications with a tag **PublicationReports** and select this tag in the **Only Include Objects with the Following Tags** field.

In addition, you have selected the **Save Reports** checkbox, but you have not selected the for **Save Publications** checkbox.

The AMM file will include the two tagged configured reports. It will not include any other configured reports because they are not tagged. It will not include an tagged or untagged publications because publications were not selected for upload.

- **Provide comma-separated tags to find objects to be removed**: Enter the name of a solution tag or a comma-separated list of multiple solution tag names to delete all public (customer-defined) configuration objects tagged with the specified tag name(s) from the target database prior. This option is useful if a tagged configuration has been changed in a way that includes the deletion of objects. The obsolete objects can be removed form the target database as well. For more information about solution tagging, see the section Identifying Configuration Objects via Solution Tagging in the reference manual *Configuring Alfabet with Alfabet Expand*.

Make sure that all objects required for the tagged solution are available in the AMM file when setting this option. All objects with the solution tag not included in the AMM file will be deleted from the target database.

If the tag name(s) that shall be used to remove objects is also used in the database you are currently connected with to tag configuration objects, you can select the tag name(s) from a multi-select combo-box instead of typing them in the field. Click the button on the right of the field to open the multi-select combo-box and select the checkbox of all tags in the current configuration for that all configuration objects shall be deleted in the target database prior to import of the configuration objects in the AMM file.

6) The **Guide Pages** tab displays all guide page projects in the current database. Select the checkbox for all guide page projects that should be merged to the existing guide page configuration of the target database. Guide pages are merged to the target database independent of the setting of the **Replace Entire Existing Configuration** field in the **Meta-Model** tab. Optionally, you can select the **Remove All Guide Pages from Target Database Before Updating** checkbox to overwrite the complete guide page configuration of the target database with the guide pages in the AMM file. If the checkbox is not selected, guide pages in the AMM file will be added to existing guide pages in the target database, thus overwriting guide pages with the same name.

Please note that the **Save Icons** checkbox in the **Meta-Model Content** tab must be selected in order to include any images included in the configuration of guide views.

The styles configured in the guide pages stored in the AMM file will overwrite the styles of the guide pages in the target database. You should ensure that the styles relevant for your enterprise are correctly configured in the guide pages before importing them via an AMM file to the production environment. For more information about the configuration of

guide pages and their styles, see the section *Formatting and Designing the Guide Pages* in the reference manual *Designing Guide Pages for Alfabet*.

7) In the **Reference Data** tab, user profiles and a specified subset of configurations performed in the **Configuration** functionalities in the Alfabet user interface can be optionally uploaded to the AMM file. These objects are stored in the instance store of the database. In contrast to the other configuration objects, they can be created and edited by users via the Alfabet user interface. Independent of the setting of the **Replace Entire Existing Configuration** field in the **Meta-Model** tab, they are always merged with the existing configuration in the target database.

Note the following about the update of user profiles and reference data via AMM files:

- **User profile configuration**: User profiles in the AMM file are added to the existing user profile configuration in the target database. User profiles with the same name will be overwritten in the target database.

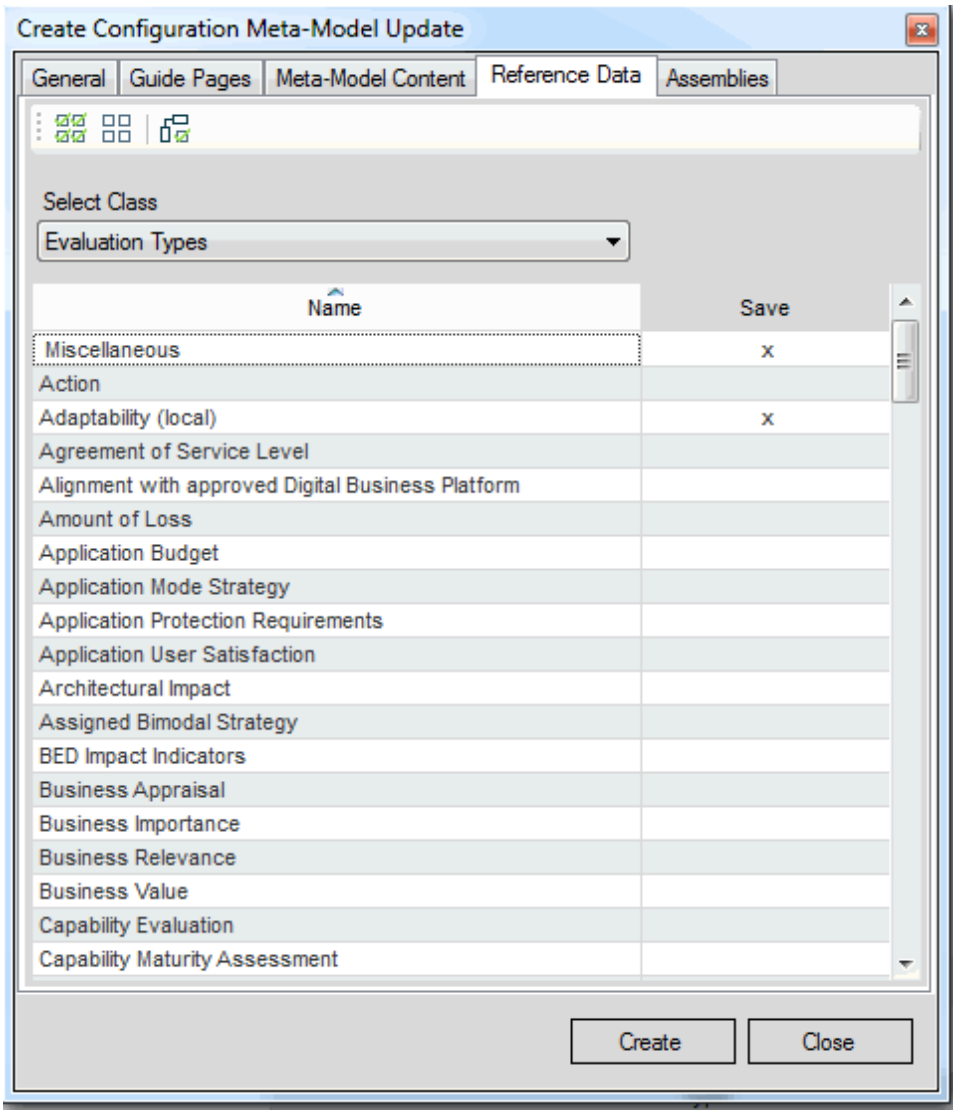
  If a guide page/guide view is defined as the start page for a user profile, you must explicitly include the guide page/guide view in the AMM update. If the guide page/guide view is not included, then the user will see an empty start page.

- **Configuration of reference data and evaluations**: Existing objects in the Alfabet database are overwritten if the key property for identification of the object is identical. The following table lists the configured objects that can be migrated via AMM, the key used to identify the objects, and whether the objects can be added to the AMM file by direct selection of objects or as sub-objects of a selectable parent object:

Object Class	Key	Selection
`EvaluationType`	Name	Directly
`IndicatorType`	Evaluation Type, Name	Included with selection of the evaluation type they are assigned to.
`RoleType`	Name	Directly
`PrioritizationScheme`	Name	Directly
`ITPortfolio`	Name	Directly
`DiagramView`	Name	Directly

DiagramView-Item	Name	Included with selection of the diagram view they are assigned to.
ColorRuleGroup	Name	Directly
ColorRule	Name	Directly or included with selection of the color rule group they are assigned to.
ClassConfiguration	Class Name	Directly

A table lists all configuration objects available for the type of configuration object selected in the **Select Class** field above the table. Select the type of configuration object in the **Select Class** field and click the cell in the **Save** column of the table for all configuration objects of the selected type that are to be saved to the AMM file:



To save all configuration objects or user profiles currently displayed in the table to the AMM file, click the **Check All** button.

When you select **Class Configuration** in the **Select Class** field and stereotypes are defined for an object class, each stereotype is listed in a separate row of the table with the **Name** defined as "Name (Stereotype)".

Color rules that are assigned to a color rule group cannot be included separately. Select **Color Rule Groups** in the **Select Class** field and select a color rule group to include the color rule group and all color rules that are assigned to the color rule group to the AMM update file. When you select Color Rules in the Select Class field, the table only displays color rules that are not assigned to a color rule group. These color rules can be selected separately.

Select the **Check Dependent Objects** button to select objects that depend on other objects that you have currently selected in the list. When you click the button, the dependent objects will be automatically selected:

- For each evaluation type currently selected in the list, the following dependent objects are also selected:
- all prioritization schemes to which the evaluation type or any of the indicator types assigned to the evaluation type are assigned
- all IT portfolios to which the evaluation type or any of the indicator types assigned to the evaluation type are assigned
- all diagram views to which any of the indicator types assigned to the evaluation type are assigned
- class configurations to which the evaluation type is assigned
- For each prioritization scheme currently selected in the list, the following dependent objects are also selected:
- all IT portfolios to which the prioritization scheme is assigned
- class configurations to which the prioritization scheme is assigned
- For each IT Portfolio currently selected in the list, the following dependent objects are also selected:
- class configurations to which the IT portfolio is assigned.

8) In the **Assemblies** tab, assemblies can be optionally uploaded to the AMM file if the configuration has been specified via a DLL file delivered by Software AG. If you need to include assemblies in the AMM file, see the section Managing Assemblies for detailed information.

9) Click the **Create** button to generate the AMM update file. A message will be displayed once the AMM has been successfully created.

Saving Selected Objects of the Configuration with Alfabet Expand

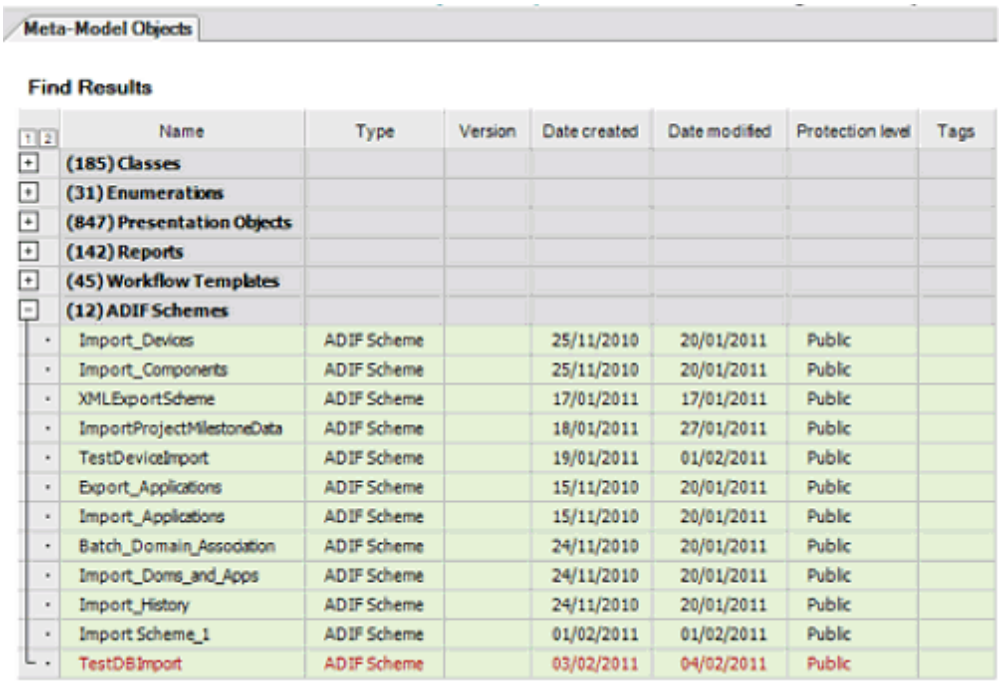
Alfabet Expand provides a mechanism that allows you to select parts of the configuration of a database to be migrated via an AMM file to a target database. Only the selected elements in the configuration will be saved to the AMM file and merged to the configuration of the target database. Substitution of the complete configuration cannot be performed with AMM files that are created from a selection of available

configuration objects, with the exception of selected object class configurations. Whether the configuration of an object class is merged or overwritten in the target database can be selected individually for each object class added to the AMM file.

The selected elements in the configuration can be tagged with a solution name that makes it possible to categorize the elements according to configuration projects, identify the elements via search mechanisms, and delete the elements from the target database. For more information about solution tagging, see Identifying Configuration Objects via Solution Tagging.

To upload selected configuration objects to an AMM file.

1) In the menu of Alfabet Expand, select **Meta-Model** > **Reread Meta-Model** to ensure that the locally cached version of the Meta-Model is consistent with the Meta-Model saved in the Alfabet database.
2) In the menu of Alfabet Expand, select **Meta-Model** > **Find Meta-Model Objects for Deployment**. A new pane **Meta-Model Objects** opens between the explorer window and the attribute window.
3) In the toolbar, click the **Refresh Find Results** button. All configuration objects are displayed and sorted according to category in the **Find Results** table.



Note the following about working with the **Find Results** table:

- The number of selectable objects in each category is displayed in parentheses behind the category name.
- To view the objects in a category, expand the category by clicking the ⊞ sign in front of the category header. Alternatively, you can expand all categories by clicking the 2 in the first column header.
- The table displays the following information about the configuration object:

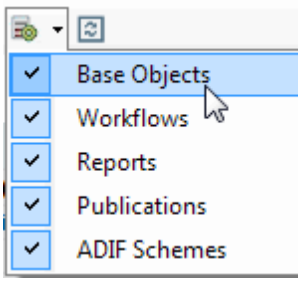
- **Name**: The value of the **Name** attribute of the configuration object.

- **Type**: The type of configuration object in the Alfabet meta-model. When exporting the information about the meta-model to an XML file with the functionality **Meta-Model** > **Save Configuration** of Alfabet Expand, information about the object is stored in an XML element with the name given as **Type** here.
- **Version**: The value of the **Tech Info** > **Version** attribute of the configuration object.
- **Date created**: The value of the **Tech Info** > **Creation Date** attribute of the configuration object.
- **Date modified**: The value of the **Tech Info** > **Last Update** attribute of the configuration object.
- **Protection Level**: `Public` objects are created by the customer, `Protected` objects are standard configuration objects of the Alfabet meta-model that may have been modified by the customer.
- **Tags**: The value of the **Tech Info** > **Tags** attribute of the configuration object. Solution tagging is designed to mark all configuration objects that are required for a specific functionality with a tag to ease the identification of relevant objects. For more information, see Identifying Configuration Objects via Solution Tagging.
- **Notes**: The value of the **Tech Info** > **Implementation Notes** attribute of the configuration object.

4) Select the objects that you want to upload to the AMM file in the **Find Results** table. You can select several objects in the table at the same time by holding down the CTRL key while selecting.

The following functionalities can help you to select the relevant objects:

- You can restrict the number of categories displayed in the table by selecting or clearing areas of configuration in the sub-menu of the **Restrict Configuration Object Types To Be Searched For** button in the toolbar of the **Find Results** table:



- Optionally, you can use the search functionality of the **Meta-Model Objects** pane to limit the display of objects in the table to objects matching a defined search condition.
- Optionally, you can use solution tagging to mark all objects that are relevant for a specific configuration task with the same tag and search for the object in the table via the search functionality. For more information, see Identifying Configuration Objects via Solution Tagging.
- To make sure that all relevant configuration objects required to merge a configured feature into a target database are added to the selection container,

you can use the **Show Usage for Selected Objects** button in the toolbar of the **Find Results** table to see a report that informs about the usage of the selected objects within the configuration. The button is also available in the toolbar of the selection container to view a report about the usage of all configuration objects already added to the selection container.

The report displays the selected objects on the x axis and the path to the location in the configuration referring to the selected objects on the y axis.

On top of the report, a filter allows you to limit the information to configuration relevant for a defined object class.

- The results of the search displayed in the table can be exported to a Microsoft® Excel® file to provide the information about selection of objects to persons that do not have access to Alfabet Expand or to archive the content of an AMM file in a readable format. To save the content of the **Find Results** table, click the **Export to Excel** button in the toolbar of the table.

5) The **Selection Container** table is displayed below the **Find Results** table. In the toolbar above the **Selection Container** table, click the **Add Selected Objects from Find Results to Selection Container** button. The objects which you selected in the table are added to the selection container. To remove a subset of the selected objects from the selection container, select the objects in the selection container and click the **Remove Selected Objects from Selection Container** button in the toolbar of the selection container. To remove all selected objects from the selection container, click the **Remove all objects from the Selection Container** button in the toolbar of the selection container.

It is recommended that you tag the selected objects with a solution tag prior to adding them to the selection container. Tagging allows objects in both the source and the target database to be identified.

You can save the configuration in the selection container in a Microsoft® Excel® file or in an XML file. This feature may be useful to generate reports about the configuration update, compare different stages of configuration, or to archive different stages of configuration. In the toolbar of the selection container, click the **Export to Excel** button and select **Create XML File** to save the information about the configuration elements in the selection container as XML output. Click the **Export to Excel** button and select **Create XML File** to save the information about the configuration elements in the selection container as XML output.

6) If the selection includes object classes, you can optionally change the update mode in the target database for each object class from `Merge` to `Replace`. Select the required update mode in the drop-down list in the **UpdateMode** column.

If you merge a class configuration, new custom properties are added to the object class in the target database and changes to object class properties are applied, but custom object class properties that have been removed in the configuration saved to the AMM file are not deleted in the configuration of the target database.

7) In the toolbar of the selection container, click the **Create Configuration Update File**  button and select **Create AMM File**. The **Create Configuration Meta-Model Update** editor opens.

8) Select the location for storing the information in the field **Select Output File**.

9) Provide a meaningful name and description for the update performed by the AMM file when applied to the target database in the fields **Name** and **Description** in the section **General**. The name and description are displayed in the **Update Meta-Model** dialog box when the AMM file is used to update the configuration in a database. It allows the person performing the update to be informed about the content of the configuration in the `AMM` file.

10) The objects that were added to the selection container are already added to the AMM file. Optionally, you can select additional configuration parts that you want to add to the AMM file in the **Meta-Model Content** tab:

- **Save Icons**: Select the checkbox to save the icons uploaded to the Alfabet database. If the checkbox is not selected, the configuration is saved without icons. If icons have been added to the selection container, the option is deactivated.
- **Save Culture Settings**: Select the checkbox to save the content of the **Cultures** and **API Cultures** explorer nodes of the **Meta-Model** tab of Alfabet Expand. If the checkbox is not selected, the configuration is saved without the configured culture settings.
- **Save Translation**: Select the checkbox to save the custom translation of strings displayed on the Alfabet user interface that are customized via the **Translation Editor** in Alfabet Expand. If the checkbox is not selected, the configuration is saved without translations.

11) Select the following options, if applicable for your configuration:

- **Automatically Migrate Affected Workflows**: Select the checkbox in the **Meta-Model Content** tab if you have included workflow template configuration in the `AMM` file and want the workflow templates to be automatically migrated to the target database.

Migration of workflows is a complex process that is described in detail in the section Creating a Migration Definition to Update Running Workflows in the chapter Configuring Workflows.

- **Provide comma-separated tags to find objects to be removed**: Enter the name of a solution tag or a comma-separated list of multiple solution tag names to delete all public (customer-defined) configuration objects tagged with the specified name(s) from the target database prior. This option is useful if a tagged configuration has been changed in a way that includes the deletion of objects. The obsolete objects can be removed form the target database as well. For more information about solution tagging, see the section Identifying Configuration Objects via Solution Tagging.

Make sure that all objects required for the tagged solution are available in the AMM file when setting this option. All objects with the solution tag not included in the AMM file will be deleted from the target database.

12) The **Guide Pages** tab displays all guide page projects in the current database. Select the checkbox for all guide page projects that should be merged to the existing guide page configuration of the target database. Guide pages are merged to the target database independent of the setting of the **Replace Entire Existing Configuration** field in the **Meta-Model** tab. Optionally, you can select the **Remove All Guide Pages from Target Database Before Updating** checkbox to overwrite the complete guide page configuration of the target database with the guide pages in the AMM file. If

the checkbox is not selected, guide pages in the AMM file will be added to existing guide pages in the target database, thus overwriting guide pages with the same name.

Please note that the **Save Icons** checkbox in the **Meta-Model Content** tab must be selected in order to include any images included in the configuration of guide views.

The styles configured in the guide pages stored in the AMM file will overwrite the styles of the guide pages in the target database. You should ensure that the styles relevant for your enterprise are correctly configured in the guide pages before importing them via an AMM file to the production environment. For more information about the configuration of guide pages and their styles, see the section *Formatting and Designing the Guide Pages* in the reference manual *Designing Guide Pages for Alfabet*.

13) In the **Reference Data** tab, user profiles and a specified subset of configurations performed in the **Configuration** functionalities in the Alfabet user interface can be optionally uploaded to the AMM file. These objects are stored in the instance store of the database. In contrast to the other configuration objects, they can be created and edited by users via the Alfabet user interface. Independent of the setting of the **Replace Entire Existing Configuration** field in the **Meta-Model** tab, they are always merged with the existing configuration in the target database.

Note the following about the update of user profiles and reference data via AMM files:

- **User profile configuration**: User profiles in the AMM file are added to the existing user profile configuration in the target database. User profiles with the same name will be overwritten in the target database.

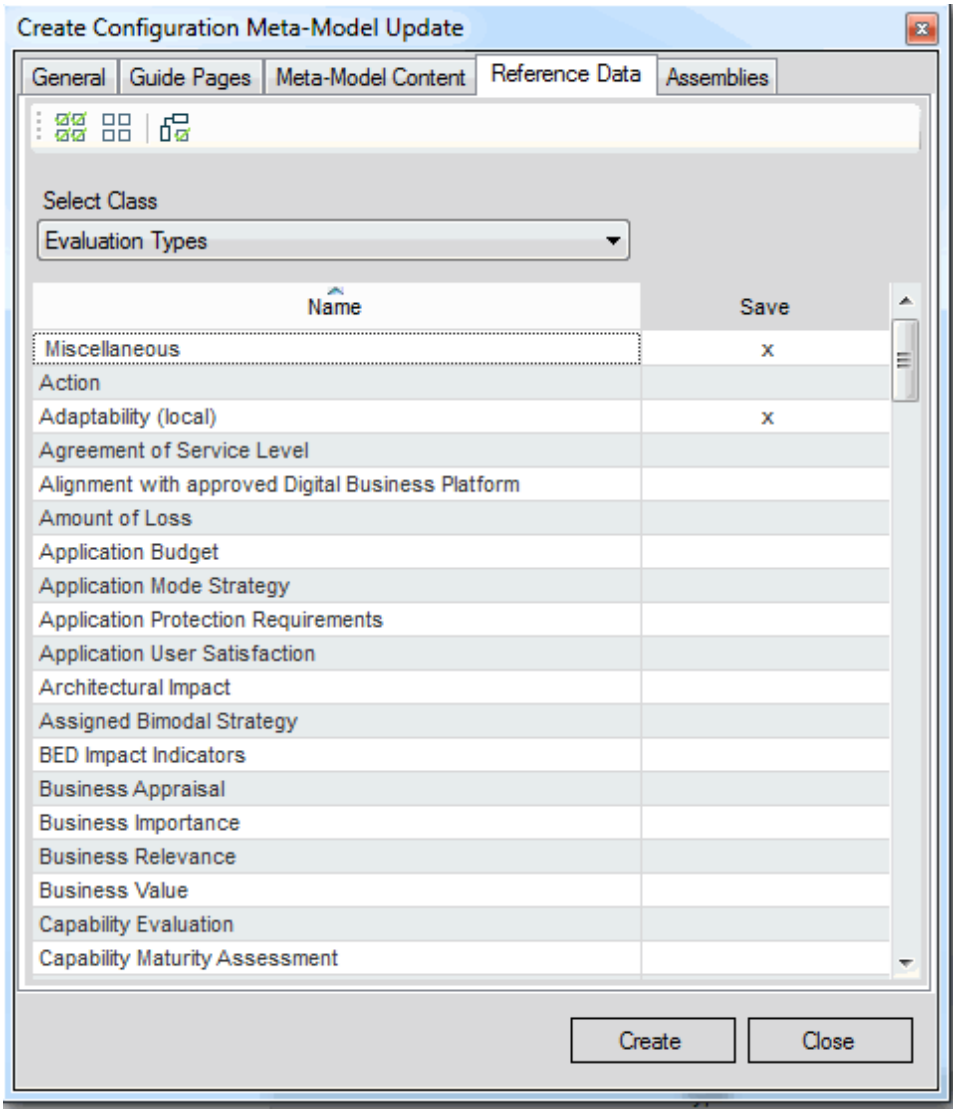
  If a guide page/guide view is defined as the start page for a user profile, you must explicitly include the guide page/guide view in the AMM update. If the guide page/guide view is not included, then the user will see an empty start page.

- **Configuration of reference data and evaluations**: Existing objects in the Alfabet database are overwritten if the key property for identification of the object is identical. The following table lists the configured objects that can be migrated via AMM, the key used to identify the objects, and whether the objects can be added to the AMM file by direct selection of objects or as sub-objects of a selectable parent object:

Object Class	Key	Selection
`EvaluationType`	Name	Directly

IndicatorType	Evaluation Type, Name	Included with selection of the evaluation type they are assigned to.
RoleType	Name	Directly
PrioritizationScheme	Name	Directly
ITPortfolio	Name	Directly
DiagramView	Name	Directly
DiagramViewItem	Name	Included with selection of the diagram view they are assigned to.
ColorRuleGroup	Name	Directly
ColorRule	Name	Directly or included with selection of the color rule group they are assigned to.
ClassConfiguration	Class Name	Directly

A table lists all configuration objects available for the type of configuration object selected in the **Select Class** field above the table. Select the type of configuration object in the **Select Class** field and click the cell in the **Save** column of the table for all configuration objects of the selected type that are to be saved to the AMM file:



To save all configuration objects or user profiles currently displayed in the table to the AMM file, click the **Check All** button.

When you select **Class Configuration** in the **Select Class** field and stereotypes are defined for an object class, each stereotype is listed in a separate row of the table with the **Name** defined as "Name (Stereotype)".

Color rules that are assigned to a color rule group cannot be included separately. Select **Color Rule Groups** in the **Select Class** field and select a color rule group to include the color rule group and all color rules that are assigned to the color rule group to the AMM update file. When you select Color Rules in the Select Class field, the table only displays color rules that are not assigned to a color rule group. These color rules can be selected separately.

Select the **Check Dependent Objects** button to select objects that depend on other objects that you have currently selected in the list. When you click the button, the dependent objects will be automatically selected:

- For each evaluation type currently selected in the list, the following dependent objects are also selected:

- all prioritization schemes to which the evaluation type or any of the indicator types assigned to the evaluation type are assigned
- all IT portfolios to which the evaluation type or any of the indicator types assigned to the evaluation type are assigned
- all diagram views to which any of the indicator types assigned to the evaluation type are assigned
- class configurations to which the evaluation type is assigned
- For each prioritization scheme currently selected in the list, the following dependent objects are also selected:
- all IT portfolios to which the prioritization scheme is assigned
- class configurations to which the prioritization scheme is assigned
- For each IT Portfolio currently selected in the list, the following dependent objects are also selected:
- class configurations to which the IT portfolio is assigned.

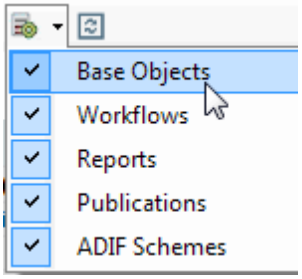
14) In the **Assemblies** tab, assemblies can be optionally uploaded to the AMM file if the configuration has been specified via a DLL file delivered by Software AG. If you need to include assemblies in the AMM file, see the section Managing Assemblies for detailed information.

15) Click the **Create** button to generate the AMM update file. A message will be displayed once the AMM has been successfully created.

Searching for Objects in the Find Results Table of the Meta-Model Objects Editor

A customer configuration may have a high number of configuration objects. To ease the selection of objects in the **Find Results** table, a number of sorting and search options are available:

- The **Find Results** table lists objects via categories. To view the objects in a category, expand the category by clicking the ⊞ sign in front of the category header.
- The columns in the **Find Results** table provide information about relevant attributes of the configuration object. If you click a table header, the table is sorted by the property displayed in the column.
- You can restrict the number of categories displayed in the table by selecting or clearing areas of configuration in the sub-menu of the **Restrict Configuration Object Types To Be Searched For** button in the toolbar of the **Find Results** table:



- A search mechanism allows objects to be displayed in the **Find Results** table to be limited to objects that match defined search criteria. The search mechanism is described below.

The search functionality is available when you currently navigate in the **Find Results** table:

1) Click anywhere inside the **Find Results** table to open the attribute window of the **Find Results** table.
2) In the attribute window, define your search condition by setting any or a combination of the following attributes:

- **Creator**: Enter the user name of a Alfabet user to find all objects that the user created. The user logged in to Alfabet Expand when the configuration object was created is stored as creator.
- **Creation Date After**: Select a date from the calendar to find all objects created after that date.
- **Creation Date Before**: Select a date from the calendar to find all objects created before that date.
- **Last Update After**: Select a date from the calendar to find all objects last updated after that date.
- **Last Update Before**: Select a date from the calendar to find all objects last updated before that date.
- **Last Update User**: Enter the user name of a Alfabet user to find all configuration objects last updated by the user. The user logged in to Alfabet Expand when the configuration object was saved the last time is stored as last update user.
- **Name**: Enter the name or part of the name of a configuration object to find all objects with a **Name** attribute matching the defined search string. An asterisk can be used as wildcard.
- **Version**: Enter the version or part of the version information to find all objects with a **Version** attribute matching the defined string. An asterisk can be used as wildcard.
- **Tags**: Click the **Browse** button in the **Tags** attribute field to open the **Meta-Model Tags** window. In the toolbar, click the **Rescan Tags** button to update the display of available tags in the list. Select one or more of the listed tags to find all objects that are tagged with at least one of the selected tag names. For more information about solution tagging, see the section Identifying Configuration Objects via Solution Tagging.

  Selecting and also clearing tags in this search field can only be performed via selecting or clearing tags from the **Meta-Model Tags** window. If you try to write content directly to the field or delete the field content, an error message will be displayed that informs you that the field content is invalid. To correct this error, click **Cancel** in the error message. The old content is displayed and can be altered via the **Meta-Model Tags** window.

- **Survey**: Select a survey from the drop-down list of available survey configurations to find all objects configured for the survey. If you add a configuration object belonging to a survey configuration to an AMM file, the survey will be created in a target database updated with the AMM file, but it will only contain the added configuration object. Make sure to add all configuration objects belonging to the survey to the AMM file to take over the correct survey definition.
- **Unsaved Objects**: Select `True` to find only configuration objects that have not currently been saved in the current configuration and therefore not been saved to the Alfabet database. Select `False` to view both saved and unsaved configuration objects.

3) In the toolbar, click the **Refresh Find Results** button. The display of objects in the **Find Results** table is limited to the objects matching the defined search criteria.

Storing all Objects Tagged With a Selected Tag to an AMM File

If you have tagged all objects that you want to merge with a selected database with the same tag without having tagged other objects with this tag, you can create an AMM file only containing the selected objects in a simple action.

Please note the following about this mechanism:

- Workflows, Guide Pages, Reference Data and Assemblies cannot be added to the AMM file created via this mechanism.
- The configuration is always merged into the target database.

To create an AMM file containing tagged objects only:

1) In the menu of Alfabet Expand, select **Meta-Model** > **Reread Meta-Model** to ensure that the locally cached version of the Meta-Model is consistent with the Meta-Model saved in the Alfabet database.
2) In the toolbar of Alfabet Expand select **Meta-Model** > **Set Current Tag**.
3) In the list of tags, click the link that you want to set as default tag to all new and changed configuration objects.

   The selection is removed when the user logs out of Alfabet Expand. If the selection shall be used again in the new session, the selection has to be repeated.

4) In the menu of Alfabet Expand, select **Meta-Model** > **Save Meta-Model Objects Marked by Current Tag**. An explorer window opens.
5) Select a location for storage of the AMM file on the local file system. Optionally you can alter the default name of the file, that is concatenated from the tag name and a time stamp. The file extension must be `.amm`.
6) Click **Save** to save the AMM file in the selected location.

Updating the Configuration of the Alfabet Solution Environment with Alfabet Expand

Please note the following about the process to update the meta-model with AMM files:

- During update of the meta-model with an AMM file the database is run in a restricted mode. That means that all connections to the database except for the one required to update the meta-model are closed and new connections cannot be established. The restricted mode ends automatically when the update process is finished. If the restricted mode persists after the update is finished, it can be manually released via the Alfabet Administrator. For more information about manually releasing the restricted mode, see *Releasing the Restricted Mode* in the reference manual *System Administration*.
- Shut down all other Alfabet components except the database server hosting the Alfabet database prior to update of the meta-model. For more information about planned shutdown of the Alfabet components, see *Planned Outages of the Alfabet Components* in the reference manual *System Administration*.
- Prior to update of the meta-model, database replication mechanisms targeting the Alfabet database must be shut down.

- Login to the Alfabet database on the database server for database update is not performed with the database user defined in the server alias configuration, but rather by means of the database user `AlfaRuntimeUser` that is automatically generated for the process by the system. This database user has Read/Write access to the database during the update of the meta-model. It is not possible to access the Alfabet database with a user `AlfaRuntimeUser` in other contexts.
- A number of mechanisms are available that ensure that the translation settings of the target database are not corrupted via a meta-model update with an AMM file:

- Information about the configuration language of the source databse is stored in the AMM file. Update of the meta-model via the AMM file will fail if the settings for the configuration language are different in the AMM file and the target database.
- On update of the meta-model with an AMM file which is configured to replace the culture configuration of the target database, the cultures in the target database are removed with the exception of the primary culture (en-US), the default culture and the configuration culture. Any settings in the AMM file about the default culture or the configuration culture are ignored and the settings in the target database are maintained.

- The case sensitivity setting from the collation of the source database is stored in the AMM file. Update of the meta-model via the AMM file will fail if the case sensitivity setting in the AMM file differs from the case sensitivity setting of the target database.
- Length limitation for database table columns on an Oracle® database server is more restrictive than on a Microsoft® SQL Server®. The allowed length on Oracle® database servers for the **Tech Name** attribute of custom object class properties in Alfabet depends on the configuration of data translation. The maximum length of a technical name may not exceed 25 characters for properties that have the **Enable Data Translation** attribute set to `True`. For properties that are not translatable, the maximum length of a technical name may not exceed 30 characters. To avoid problems on migration of the database from a Microsoft® SQL Server® to an Oracle® database server because of **Tech Name** attributes of custom object class properties not matching the restrictions, a check for database server compatibility is available. A read-only **Database Platform Compatibility** attribute is available. The attribute is set to `Oracle` if the database is hosted on an Oracle® database server. If the database is hosted on a Microsoft® SQL Server®, the **Database Platform Compatibility** attribute is set to `SqlServer` if the database contains **Tech Name** attributes that violate the size restriction and will be set to `Common` if the database does not violate the size restriction. On creation of an AMM file, the **Database Platform Compatibility** attribute of the current database is written to the AMM file. On update of the meta-model with the AMM file, the **Database Platform Compatibility** attribute value in the target database is compared with the value in the AMM file. Updating the meta-model will fail if the Alfabet database is hosted on an Oracle® database server and the AMM file has the **Database Platform Compatibility** value set to `SqlServer`. The database will remain unchanged and offending **Tech Name** values will be written to a log file.

Before you update a production database with an AMM file created with an Update Solution option, you must backup the target database for security reasons. Solution tagging is a sensitive process that can lead to data loss if not carefully planned and executed. It is strongly recommended that you perform solution tagging with the help of your consultant from Alfabet only. The update of a database via an AMM file created with an **Update Solution** option should be thoroughly tested on a test database before the production environment is updated. For more information about solution tagging, see the section Identifying Configuration Objects via Solution Tagging.

If you merge the solution configuration in the AMM file with the configuration of a target database, only the configuration objects that are stored in the AMM file will be overwritten.

Overwriting configuration objects is especially important for the update of configuration objects with sub-objects. For example:

- Updating an object class with information in the AMM file will delete all custom object class properties and numeration information for the class in the target database and substitute it with the information in the AMM file.
- Updating a report folder will delete all reports from the report folder in the target database and substitute it with the reports available in the AMM file configuration.

Therefore, prior to creating an AMM update file, you must ensure that the configuration in the development database is identical to the configuration in the target database (with the exception of the desired configuration change).

Update of the meta-model with an AMM update file changes the configuration of the meta-model in your database. Please note the following about the restore of the configuration:

- **Clarify any concerns regarding your specific customization before initiating the Update Meta-Model functionality!** Some customizable features in Alfabet may not be part of the meta-model configuration. They are stored in the instance store and could be lost/damaged when a configuration is saved and restored to another database with an AMM file. This applies to the following configurations:

- **Export definitions**: To save and restore the export definitions, the export definitions must be saved and restored separately using the tool Alfabet Expand. In Alfabet Expand, the context menu of the root explorer nodes for the respective configurations offer a **Save as** functionality that allows you to save the specific part of the configuration as an XML file and a **Read from File** functionality and **Merge from File(s)** functionality to restore the specific configuration.
- **Mandate configuration**: The configuration of mandates is saved in the instance store of the database and can only be saved and read/merge to another database by using the **Import Data Search** functionality or by manually recreating the configuration in the target database using Alfabet Expand or the Alfabet Administrator. For more information about the activation and use of the **Import Data Search** functionality, see the section Importing Objects of Configuration Relevant Object Classes from a Master Database in the reference manual *Configuring Alfabet with Alfabet Expand*.
- **Most reference and evaluation data configured in the Configuration functionalities in the Alfabet user interface**: For example, reference data such as role types, cost types, indicator types, or portfolios are configured in Alfabet configuration functionalities in the Alfabet user interface. These configurations are saved in the instance store of the database. Only the following subset of these configurations can be included in the AMM file:
- Evaluation Types
- Portfolios
- Prioritization Schemes
- Diagram Views
- Color Rule Groups
- Color Rules
- Class Configurations
- Roles

Other configuration data can only be saved and read/merge to another database by using the **Import Data Search** functionality or by manually recreating the configuration in the target database using Alfabet Expand or the Alfabet Administrator. For more information about the activation and use of the **Import Data Search** functionality, see the section Importing Objects of Configuration Relevant Object Classes from a Master Database in the reference manual *Configuring Alfabet with Alfabet Expand*.

- **Documents uploaded to the Internal Document Selector**: Documents in the **Internal Document Selector** can only be stored in normal database backup files on the database server level or added to the target database manually. This may impact the images and style sheet files uploaded to be used for visualization items (configured HTML templates for the workflows functionality). While the visualization item can be saved to an AMM file and uploaded to a target database via meta-model update, the images and style sheet files must be uploaded to the **Internal Document Selector** of the target database via Alfabet Expand or the Alfabet user interface.
- All database views in the target database are recreated after a meta-model update and need to be checked for consistency afterwards as described in the section Checking the Consistency of the Database View With the Meta-Model. The recreation ensures that new database views added to the configuration via the meta-model update are created and that database views that are not compatible with the changes to the meta-model applied via meta-model update are removed.
- AMM files can be configured to either replace or merge the configuration stored in the AMM file with the configuration in the target database:
- **Replace Configuration**: The whole customer configuration in the target database will be deleted prior to writing the content of the AMM file to the target database. Configurations that have no corresponding object in the AMM file will be deleted.

  Configured reports that are automatically generated for the faceted semantic search functionality of the AlfaBot will not be deleted from the target database. For more information about automatically generated reports, see Managing Automatically Generated Reports.

- **Merge Configuration**: All objects in the configuration will overwrite corresponding objects in the database. Database objects that have no corresponding object in the AMM will remain unchanged. Objects that are only available in the AMM will be added.

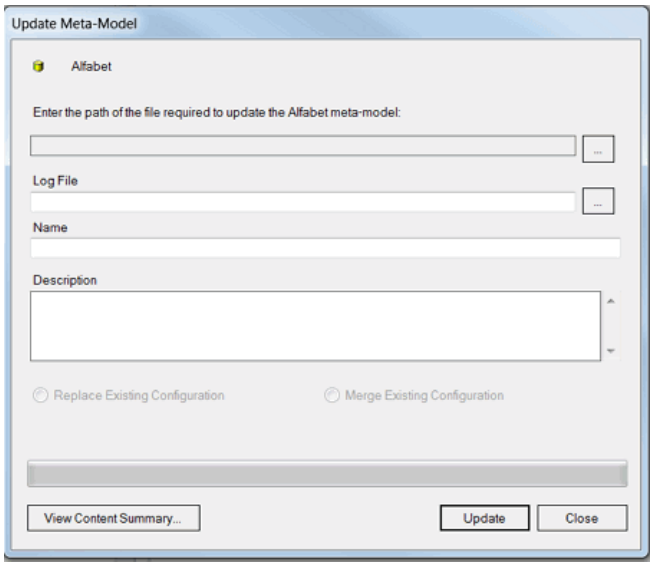
Alfabet provides mechanism to check the content of an AMM file prior to update of the meta-model and to control the success of the update action:

- When updating the meta-model using Alfabet Expand or the Alfabet Administrator, a window for selection of the AMM file that shall be used for update opens. After having selected an AMM file, you can click the button **View Content Summary** to see a summary of the content of the AMM file.
- You can use the **Compare Configurations** functionality of Alfabet Expand to view the differences between the configuration in the AMM file with the configuration of the target database that you are planning to replace or merge. For more information, see *Comparing Database Configurations* in the reference manual *Configuring Alfabet with Alfabet Expand*.
- A log file is created during update of the meta-model. The log file is written to the directory that contains the AMM file and has the name <Name of the AMM file>_<Time Stamp>.log. You can consult the log file to view any issues that occurred during update.
- After update, you should check the consistency of the update with the standard meta-model. A consistency check can be performed via Alfabet Expand. For more information, see in the reference manual *Configuring Alfabet with Alfabet Expand*.

If the functionality for update of the meta-model is not working, please check the alias configuration of Alfabet Expand. The option **File-Based Update Permitted** must be activated in the **Expand** tab of the alias configuration editor. This option is activated by default.

Prior to updating the meta-model, ensure that no Alfabet components are currently connected to the Alfabet database and that replication mechanisms that target the Alfabet database are shut down.

1) Open the tool Alfabet Expand without logging in to a database.
2) In the menu bar in Alfabet Expand, select **Meta-Model** > **Update Meta-Model**. A new window opens.
3) Select a server alias for connection to the Alfabet database that you want to update and click **OK**. A window for login to the database opens
4) Enter the user name and password defined for login to the Alfabet database or, if Windows sign on is configured for connections to the Alfabet database, click **OK** without entering a user name and password.
5) Click **OK**. A new window opens:



6) Click the **Browse** button next to the field **Enter the path of the file required to update the Alfabet meta-model**. An explorer opens.
7) Navigate to the folder where the AMM file with your saved configuration is located and click **Open**. The path to the update file is displayed.

   The following information about the selected file is displayed after the file was selected:

- If a name was specified during the creation of the update file, it will be displayed in the **Operation** field.
- If a description was specified during creation of the update file, it will be displayed in the **Description** field.
- The relevant field of the options **Replace Existing Configuration** or **Merge Existing Configuration** is automatically selected depending on whether a merge or replace action is being executed.

- To view a summary of the content of the file, click **View Content Summary**. A new window opens that provides information about the options selected during creation of the file. Verify that the correct update file is being used and click **Close**.

8) Optionally, you can change the location for the log file created during the update process in the field **Log File**. By default, the log file is stored in the directory of the AMM file used for the update. The default log file is automatically specified in the field **Log File**.

Changes in the vocabularies that might affect custom translations are logged separately in a Microsoft Excel file. The log file for the update process includes a link to the translation log.

9) Click **Update**. The configuration stored in the update file replaces or merges with the configuration of your current database.

After the update of the configuration, all ADIF import schemes that are available in the target database after the update and have the attribute **Auto-Run** set to `True` will be automatically executed in the order specified via the context menu of the root node of the ADIF explorer in Alfabet Expand. For more information about automatic execution of ADIF schemes, see *Configuring ADIF Schemes to be Automatically Executed on Update of the Meta-Model* in the reference manual *Alfabet Data Integration Framework*

If the automated data translation feature is implemented, update of the meta-model will take significantly longer than without the feature being implemented.

10) A message is displayed informing you about the success of the update action and advising you to check the log file. Confirm the message and open the log file to see whether the changes performed via the update conflict with any of the existing objects or configurations in your database. The messages also inform about translation issues caused by changes of strings during update.

After applying a configuration to an Alfabet database, it is recommended that you review the Alfabet database for consistency with the meta-model as described in the section Securing and Checking Database Consistency with the Meta-Model.

Direct Import of the Solution Configuration from a Master Database

This functionality is only available if the server alias used to connect with Alfabet Expand to the target database is configured to establish a connection to the source database.

The server alias configuration also includes a setting that decides about the update method. Configurations can either be taken over completely, which means that the configuration of the target database is overwritten, or the configuration of the source database can be merged into the configuration of the target database. In the latter case, the user triggering the configuration update with Alfabet Expand on the target database can decide about which part of the configuration shall be included in the update.

For the update of the solution configuration from a master database, a restore mechanism is available. The history of performed updates can be displayed and logging information can be checked for each import. If the import has failed, the configuration of any point in the update history can be restored in the target database.

Please note the following about the process to update the meta-model from a master database:

- During update of the meta-model the database is run in a restricted mode. That means that all connections to the database except for the one required to update the meta-model are closed and new connections cannot be established. The restricted mode ends automatically when the update process is finished. If the restricted mode persists after the update is finished, it can be manually released via the Alfabet Administrator. For more information about manually releasing the restricted mode, see *Releasing the Restricted Mode* in the reference manual *System Administration*.

Before you update a production database, you must backup the target database for security reasons.

Update of the meta-model from a master database changes the configuration of the meta-model in your database. Please note the following about the restore of the configuration:

- **Clarify any concerns regarding your specific customization before initiating the update from master database functionality!** Some customizable features in Alfabet may not be part of the meta-model configuration. They are stored in the instance store and could be lost/damaged when a configuration is taken over via master database update. This applies to the following configurations:

- **Export definitions**: To save and restore the export definitions, the export definitions must be saved and restored separately using the tool Alfabet Expand. In Alfabet Expand, the context menu of the root explorer nodes for the respective configurations offer a **Save as** functionality that allows you to save the specific part of the configuration as an XML file and a **Read from File** functionality and **Merge from File(s)** functionality to restore the specific configuration.

- **Mandate configuration**: The configuration of mandates is saved in the instance store of the database and can only be saved and read/merge into another database by using the **Import Data Search** functionality or by manually recreating the configuration in the target database using Alfabet Expand or the Alfabet Administrator. For more information about the activation and use of the **Import Data Search** functionality, see the section Importing Objects of Configuration Relevant Object Classes from a Master Database in the reference manual *Configuring Alfabet with Alfabet Expand*.

- **Most reference and evaluation data configured in the Configuration functionalities in the Alfabet user interface**: For example, reference data like role types and cost types or indicator types and portfolios are configured in Alfabet configuration functionalities in the Alfabet user interface. These configurations are saved in the instance store of the database. Only the following subset of these configurations can be included into the AMM file:

- Evaluation Types
- Portfolios
- Prioritization Schemes
- Diagram Views
- Color Rule Groups
- Color Rules
- Class Configurations
- Roles

Other configuration data can only be saved and read/merge into another database by using the **Import Data Search** functionality or by manually recreating the configuration in the target database using Alfabet Expand or the Alfabet Administrator. For more information about the activation and use of the **Import Data**

Search functionality, see the section Importing Objects of Configuration Relevant Object Classes from a Master Database in the reference manual *Configuring Alfabet with Alfabet Expand*.

- **Documents uploaded to the Internal Document Selector**: Documents in the **Internal Document Selector** can only be stored in normal database backup files on the database server level or added to the target database manually. This may impact the images and style sheet files uploaded to be used for visualization items (configured HTML templates for the workflows functionality). While the visualization item can be saved to an AMM file and uploaded to a target database via meta-model update, the images and style sheet files must be uploaded to the **Internal Document Selector** of the target database via Alfabet Expand or the Alfabet user interface.
- All database views in the target database are recreated after a meta-model update and need to be checked for consistency afterwards as described in the section Checking the Consistency of the Database View With the Meta-Model. The recreation ensures that new database views added to the configuration via the meta-model update are created and that database views that are not compatible with the changes to the meta-model applied via meta-model update are removed.

The following information is available:

- Overwriting the Solution Configuration with the Configuration of a Master Database
- Merging the Solution Configuration with the Configuration of a Master Database
- Checking and Repairing the Configuration Updates From a Master Database

Overwriting the Solution Configuration with the Configuration of a Master Database

Complete updates will be performed if the update of the configuration from a master database is triggered with a server alias of Alfabet Expand that is configured to perform complete updates.

The method is recommended to take over configurations from a development to a test environment after complete performance of a configuration. The tester can take over the configuration without any knowledge about configuration details.

Do the following to overwrite the configuration in your current database with the configuration in a master database:

1) In the toolbar, select **Meta-Model** > **Update Meta-Model Configuration from Master Database**. The **Update from Master Database <Database Name>** dialog opens.
2) In the **General** tab, define the following fields:

- **Name**: Enter a meaningful name for the update. The name is used in the update history to identify the update.
- **Description**: Optionally a description that is displayed in the update history to provide more information about the update.

3) Optionally, go to the **Reference Data** tab and select user profiles and a specified subset of configurations performed in the **Configuration** module in Alfabet to be uploaded. These objects are stored in the instance store of the database. In contrast to the other configuration objects, they can be created and edited by users via the Alfabet user interface. In contrast to solution configurations performed in Alfabet Expand, solution objects stored in the instance store are not overwriting the instance store tables in the target database. Instead, the configuration is merged.

Note the following about the update of user profiles and reference data via AMM files:

- **User profile configuration**: User profiles in the AMM file are added to the existing user profile configuration in the target database. User profiles with the same name will be overwritten in the target database.

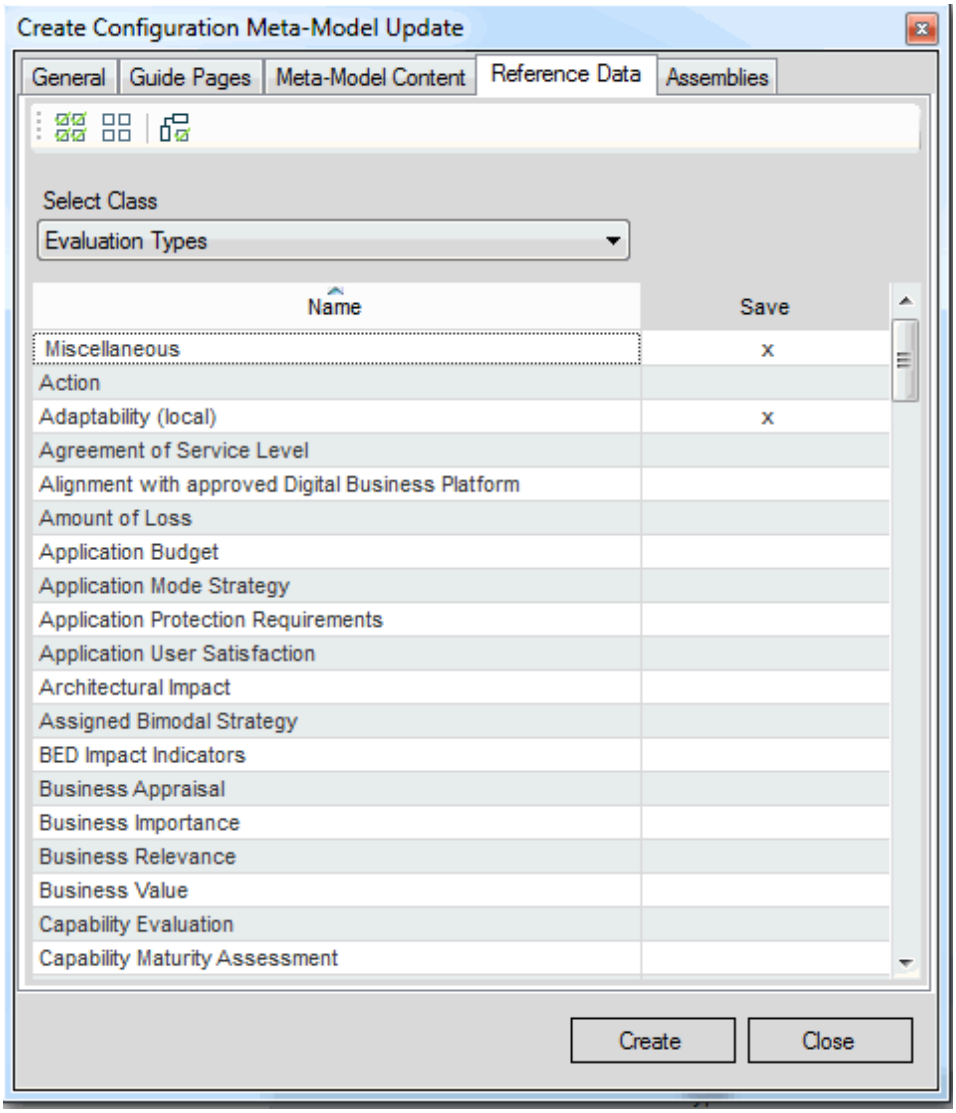
  

  If a guide page/guide view is defined as the start page for a user profile, you must explicitly include the guide page/guide view in the AMM update. If the guide page/guide view is not included, then the user will see an empty start page.

- **Configuration of reference data and evaluations**: Existing objects in the Alfabet database are overwritten if the key property for identification of the object is identical. The following table lists the configured objects that can be migrated via AMM, the key used to identify the objects, and whether the objects can be added to the AMM file by direct selection of objects or as sub-objects of a selectable parent object:

Object Class	Key	Selection
EvaluationType	Name	Directly
IndicatorType	Evaluation Type, Name	Included with selection of the evaluation type they are assigned to.
RoleType	Name	Directly
PrioritizationScheme	Name	Directly
ITPortfolio	Name	Directly
DiagramView	Name	Directly
DiagramViewItem	Name	Included with selection of the diagram view they are assigned to.
ColorRuleGroup	Name	Directly
ColorRule	Name	Directly or included with selection of the color rule group they are assigned to.
ClassConfiguration	Class Name	Directly

A table lists all configuration objects available for the type of configuration object selected in the **Select Class** field above the table. Select the type of configuration object in the **Select Class** field and click the cell in the **Save** column of the table for all configuration objects of the selected type that are to be saved to the AMM file:



To save all configuration objects or user profiles currently displayed in the table to the AMM file, click the **Check All** button.

When you select **Class Configuration** in the **Select Class** field and stereotypes are defined for an object class, each stereotype is listed in a separate row of the table with the **Name** defined as "Name (Stereotype)".

Color rules that are assigned to a color rule group cannot be included separately. Select **Color Rule Groups** in the **Select Class** field and select a color rule group to include the color rule group and all color rules that are assigned to the color rule group into the AMM update file. When you select Color Rules in the Select Class field, the table only displays color rules that are not assigned to a color rule group. These color rules can be selected separately.

Select the **Check Dependent Objects** button to select objects that depend on other objects that you have currently selected in the list. When you click the button, the dependent objects will be automatically selected:

- For each evaluation type currently selected in the list, the following dependent objects are also selected:
- all prioritization schemes to which the evaluation type or any of the indicator types assigned to the evaluation type are assigned
- all IT portfolios to which the evaluation type or any of the indicator types assigned to the evaluation type are assigned
- all diagram views to which any of the indicator types assigned to the evaluation type are assigned
- class configurations to which the evaluation type is assigned
- For each prioritization scheme currently selected in the list, the following dependent objects are also selected:
- all IT portfolios to which the prioritization scheme is assigned
- class configurations to which the prioritization scheme is assigned
- For each IT Portfolio currently selected in the list, the following dependent objects are also selected:
- class configurations to which the IT portfolio is assigned.

4) In the tab **Assemblies** tab, assemblies can be optionally uploaded to the AMM file if the configuration has been specified via a DLL file delivered by Software AG. The functionality on this tab is limited to upload of assemblies already available in the master database.

For an overview about assemblies and their role in configuration, see the section Managing Assemblies.

5) Confirm the message by clicking **OK**. All connections to the Alfabet database are closed during update.

Merging the Solution Configuration with the Configuration of a Master Database

Updates of selected parts of the configuration will be performed if the update of the configuration from a master database is triggered with a server alias of Alfabet Expand that is configured to perform selective updates.

The method is recommended to take over defined parts of the configurations from a development to a test environment. A solution designer can for example tag all configurations for a defined feature with a specific test and the tester can be advised to take over all configuration objects with that tag. Additional, incomplete configurations that are currently available in the master database are not taken over to the test environment during the selective update, because they are tagged differently.

Selective updates are highly flexible. During the update, the person performing the update decides whether to take over

- the complete configuration,
- only selected types of configuration objects, like for example configured reports only, or

- only objects tagged with a defined tag.

It is also both possible to overwrite the existing configuration or to merge the configuration from the master database into the existing configuration of the target database.

Do the following to take over selected parts of the configuration in a master database to your current database:

1) In the toolbar, select **Meta-Model** > **Update Meta-Model Configuration from Master Database**. The **Update from Master Database <Database Name>** dialog opens.
2) In the **General** tab, define the following fields:

- **Name**: Enter a meaningful name for the update. The name is used in the update history to identify the update.
- **Description**: Optionally a description that is displayed in the update history to provide more information about the update.

3) In the **Meta-Model Content** tab, select the configuration that you want to add to the AMM file:

- **Save Configuration**: Select the checkbox to save custom object class properties, custom enumerations and all customer defined configuration objects in the **Presentation**, **Functions**, and **Surveys** tabs in Alfabet Expand.
- **Save Reports**: Select the checkbox to save the content of the **Reports** root node of the explorer in the **Reports** tab in Alfabet Expand. If the report structure in the target database will change with the update, bookmarks linking to configured reports will be automatically updated when the configuration is merged.

  If a configured report that exists both in the target database and in the AMM file is updated, and a user or solution designer has restricted the display of columns of the configured report via the **Configure** button, new columns added to the configured reports will be hidden because they are not selected in the visibility setting.

  For information about hiding columns in configured reports, see the section Defining the Visibility of Page Views/Configured Reports Available at the Root Node of an Explorer in the chapter Configuring User Profiles for the User Community.

- **Save Workflow Templates**: Select the checkbox to save the content of the explorer of the **Workflow** tab of Alfabet Expand.
- **Save ADIF Schemes**: Select the checkbox to save the content of the explorer of the `ADIF` tab in Alfabet Expand.

  After the target database is updated by means of the AMM file, all ADIF import schemes in the target database that have the **Auto-Run** attribute set to `True` will be automatically executed. For more information about automatic execution of ADIF schemes, see *Configuring ADIF Schemes to be Automatically Executed on Update of the Meta-Model* in the reference manual *Alfabet Data Integration Framework*.

- **Save Event Templates**: Select the checkbox to save the content of the explorer of the **Events** tab in Alfabet Expand.
- **Save Diagrams**: Select the checkbox to save the custom diagram definitions and custom diagram item templates defined in the **Diagrams** tab in Alfabet Expand.

Icons are not automatically saved if they are included in a diagram item template and the diagram item template is stored in the AMM file. If your configuration includes icons that are not available in the target database, make sure to also select **Save Icons**.

- **Save Publications**: Select the checkbox to save the content of the **Publications** tab in Alfabet Expand.
- **Save Database Views**: Select the checkbox to save the database views defined in the **Meta-Model** tab in Alfabet Expand.

When the configuration is restored to the target database, all database views in the target database that are created based on a **Database View** configuration object will be recreated, even if **Save Database Views** is not checked. The solution designer should then check whether the creation of database views was successful. The required procedure is described in the section Saving the Configuration of the Alfabet Solution to an AMM File in the reference manual *Configuring Alfabet with Alfabet Expand*.

- **Save Icons**: Select the checkbox to save the icons uploaded to the Alfabet database.

Please note that the **Save Icons** checkbox in the **Meta-Model Content** tab must be selected in order to include any images included in the configuration of guide views.

- **Save Culture Settings**: Select the checkbox to save the content of the **Cultures** and **API Cultures** explorer nodes of the **Meta-Model** tab in Alfabet Expand.
- **Save Translation**: Select the checkbox to save the custom translation of strings displayed in the Alfabet user interface that are customized via the **Translation Editor** in Alfabet Expand.

When the translation includes the caption and description of configured reports, the **Update Translation** functionality available in the **Report** root node in Alfabet Expand must be executed on the target database after update with the AMM file.

4) Select the following options, if applicable for your configuration:

- **Replace Entire Existing Configuration**: Select the checkbox if you want the configuration in the AMM update file to overwrite an existing configuration in the target database during the **Update Meta-Model** action. If the checkbox is not selected, the configuration in the AMM update file is merged with the configuration in the target database.

If the **Replace Entire Existing Configuration** checkbox is selected, the configuration of the target database will be deleted prior to saving the configuration in the AMM file to the target database. All customer configurations that are part of the configuration of the target database but not part of the configuration saved in the AMM file will be lost. The replace mechanism deletes all parts of the configuration including the parts that are not selected to be part of the AMM update. The **Replace Entire Existing Configuration** checkbox should only be selected for AMM files when the parts of the configuration to be replaced have been selected.

Please note that reference data, assemblies and guide pages are not deleted prior to applying the configuration stored in the AMM file.

Cultures in the target database are removed with the exception of the primary culture (en-US), the default culture and the configuration culture. Any settings in the

AMM file about the default culture or the configuration culture are ignored and the settings in the target database are maintained.

Configured reports that are automatically generated for the faceted semantic search functionality of the AlfaBot will not be deleted from the target database. For more information about automatically generated reports, see Managing Automatically Generated Reports.

If you want to replace only part of the configuration (for example, the configuration of workflows only), it is recommended that you use the solution tagging option. For more information about solution tagging, see Identifying Configuration Objects via Solution Tagging in the reference manual *Configuring Alfabet with Alfabet Expand*.

- **Automatically Migrate Affected Workflows**: Select the checkbox if you want the workflow templates to be automatically migrated to the target database. Migration of workflows is a complex process that is described in detail in the section Creating a Migration Definition to Update Running Workflows in the chapter Configuring Workflows.
- **Only Include Objects with the Following Tags**: The box lists all solution tags available in the current configuration. If you do not select any checkboxes, all configuration parts selected in the **Meta-Model** tab are is saved without taking solution tagging into account. If you select one or multiple checkboxes, only solution objects which are both marked with a selected tag and are of a configuration object type selected in the Meta-Model tab are included into the AMM file. If solution tagging is not available for a configuration object type that is selected, all objects of that type will be saved independent of selected tags. For more information about solution tagging, see the section Identifying Configuration Objects via Solution Tagging in the reference manual *Configuring Alfabet with Alfabet Expand*.

  For example, the configuration includes 20 configured reports and 5 publications.

  You have tagged two configured reports and two publications with a tag **PublicationReports** and select this tag in the **Only Include Objects with the Following Tags** field.

  In addition, you have selected the **Save Reports** checkbox, but you have not selected the for **Save Publications** checkbox.

  The AMM file will include the two tagged configured reports. It will not include any other configured reports because they are not tagged. It will not include an tagged or untagged publications because publications were not selected for upload.

- **Provide comma-separated tags to find objects to be removed**: Enter the name of a solution tag or a comma-separated list of multiple solution tag names to delete all public (customer-defined) configuration objects tagged with the specified tag name(s) from the target database prior. This option is useful if a tagged configuration has been changed in a way that includes the deletion of objects. The obsolete objects can be removed form the target database as well. For more information about solution tagging, see the section Identifying Configuration Objects via Solution Tagging in the reference manual *Configuring Alfabet with Alfabet Expand*.

  Make sure that all objects required for the tagged solution are available in the AMM file when setting this option. All objects with the solution tag not included in the AMM file will be deleted from the target database.

  If the tag name(s) that shall be used to remove objects is also used in the database you are currently connected with to tag configuration objects, you can select the tag name(s) from a multi-select combo-box instead of typing them in the field. Click the button on the right of the field to open the multi-select combo-box and select

the checkbox of all tags in the current configuration for that all configuration objects shall be deleted in the target database prior to import of the configuration objects in the AMM file.

5) The **Guide Pages** tab displays all guide page projects in the current database. Select the checkbox for all guide page projects that should be merged to the existing guide page configuration of the target database. Guide pages are merged to the target database independent of the setting of the **Replace Entire Existing Configuration** field in the **Meta-Model** tab. Optionally, you can select the **Remove All Guide Pages from Target Database Before Updating** checkbox to overwrite the complete guide page configuration of the target database with the guide pages in the AMM file. If the checkbox is not selected, guide pages in the AMM file will be added to existing guide pages in the target database, thus overwriting guide pages with the same name.

Please note that the **Save Icons** checkbox in the **Meta-Model Content** tab must be selected in order to include any images included in the configuration of guide views.

The styles configured in the guide pages stored in the AMM file will overwrite the styles of the guide pages in the target database. You should ensure that the styles relevant for your enterprise are correctly configured in the guide pages before importing them via an AMM file to the production environment. For more information about the configuration of guide pages and their styles, see the section *Formatting and Designing the Guide Pages* in the reference manual *Designing Guide Pages for Alfabet*.

6) In the **Reference Data** tab, user profiles and a specified subset of configurations performed in the **Configuration** functionalities in the Alfabet user interface can be optionally uploaded to the AMM file. These objects are stored in the instance store of the database. In contrast to the other configuration objects, they can be created and edited by users via the Alfabet user interface. Independent of the setting of the **Replace Entire Existing Configuration** field in the **Meta-Model** tab, they are always merged with the existing configuration in the target database.

Note the following about the update of user profiles and reference data via AMM files:

- **User profile configuration**: User profiles in the AMM file are added to the existing user profile configuration in the target database. User profiles with the same name will be overwritten in the target database.

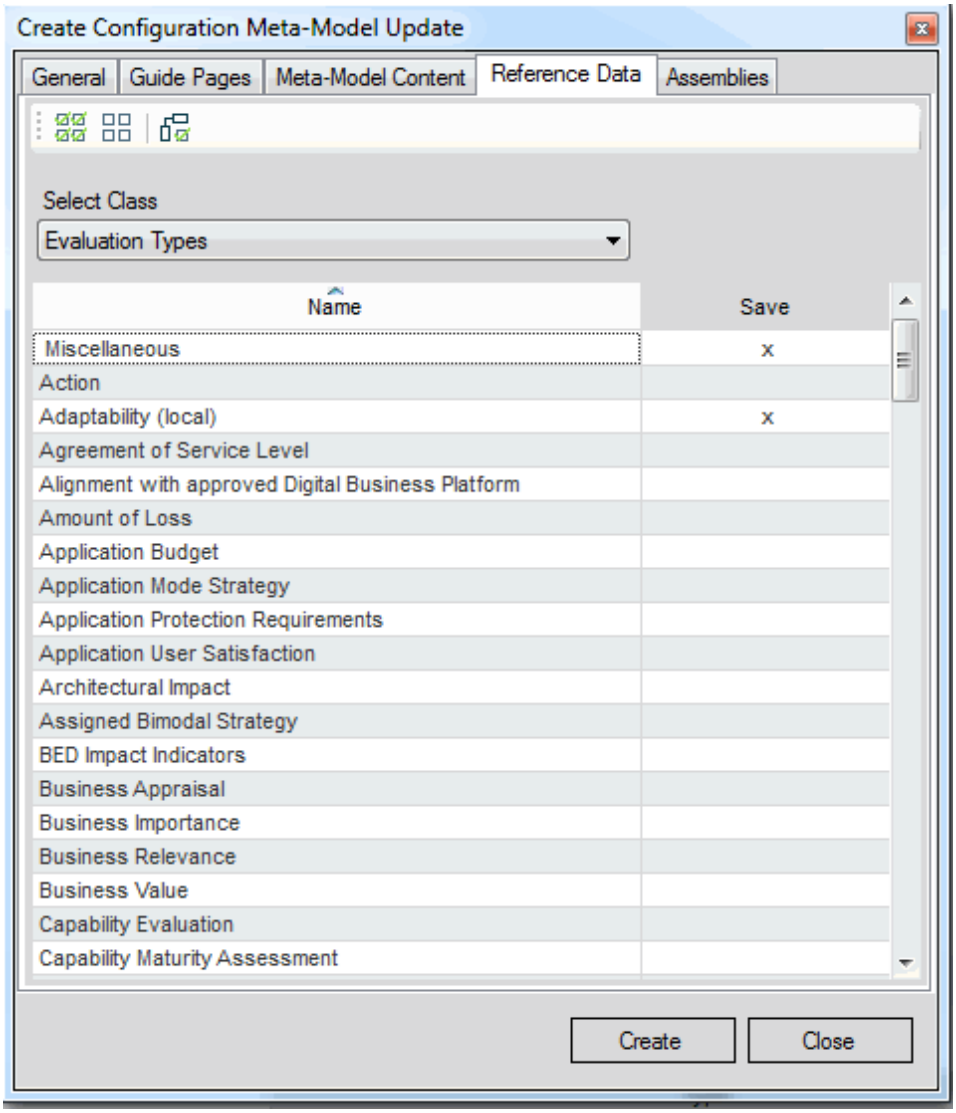
  

  If a guide page/guide view is defined as the start page for a user profile, you must explicitly include the guide page/guide view in the AMM update. If the guide page/guide view is not included, then the user will see an empty start page.

- **Configuration of reference data and evaluations**: Existing objects in the Alfabet database are overwritten if the key property for identification of the object is identical. The following table lists the configured objects that can be migrated via AMM, the key used to identify the objects, and whether the objects can be added to the AMM file by direct selection of objects or as sub-objects of a selectable parent object:

Object Class	Key	Selection
EvaluationType	Name	Directly
IndicatorType	Evaluation Type, Name	Included with selection of the evaluation type they are assigned to.
RoleType	Name	Directly
PrioritizationScheme	Name	Directly
ITPortfolio	Name	Directly
DiagramView	Name	Directly
DiagramViewItem	Name	Included with selection of the diagram view they are assigned to.
ColorRuleGroup	Name	Directly
ColorRule	Name	Directly or included with selection of the color rule group they are assigned to.
ClassConfiguration	Class Name	Directly

A table lists all configuration objects available for the type of configuration object selected in the **Select Class** field above the table. Select the type of configuration object in the **Select Class** field and click the cell in the **Save** column of the table for all configuration objects of the selected type that are to be saved to the AMM file:



To save all configuration objects or user profiles currently displayed in the table to the AMM file, click the **Check All** button.

When you select **Class Configuration** in the **Select Class** field and stereotypes are defined for an object class, each stereotype is listed in a separate row of the table with the **Name** defined as "Name (Stereotype)".

Color rules that are assigned to a color rule group cannot be included separately. Select **Color Rule Groups** in the **Select Class** field and select a color rule group to include the color rule group and all color rules that are assigned to the color rule group to the AMM update file. When you select Color Rules in the Select Class field, the table only displays color rules that are not assigned to a color rule group. These color rules can be selected separately.

Select the **Check Dependent Objects** button to select objects that depend on other objects that you have currently selected in the list. When you click the button, the dependent objects will be automatically selected:

- For each evaluation type currently selected in the list, the following dependent objects are also selected:

- all prioritization schemes to which the evaluation type or any of the indicator types assigned to the evaluation type are assigned
- all IT portfolios to which the evaluation type or any of the indicator types assigned to the evaluation type are assigned
- all diagram views to which any of the indicator types assigned to the evaluation type are assigned
- class configurations to which the evaluation type is assigned
- For each prioritization scheme currently selected in the list, the following dependent objects are also selected:
- all IT portfolios to which the prioritization scheme is assigned
- class configurations to which the prioritization scheme is assigned
- For each IT Portfolio currently selected in the list, the following dependent objects are also selected:
- class configurations to which the IT portfolio is assigned.

7) In the **Assemblies** tab, assemblies can be optionally uploaded to the AMM file if the configuration has been specified via a DLL file delivered by Software AG. If you need to include assemblies in the AMM file, see the section Managing Assemblies for detailed information.
8) Confirm the message by clicking **OK**. All connections to the Alfabet database are closed during update.

Checking and Repairing the Configuration Updates From a Master Database

A history is available for configuration updates from a master database. The history also provides access to log information for the update file and a summary of configuration objects updated.

The history only lists meta-model updates from a master database and data anonymization actions. File based updates of the meta-model are not included in the history.

Every entry in the history can serve as a restore point to re-establish the configuration prior to the selected update in the history.

If update from a master database is not enabled, the history is nevertheless displayed to inform about the anonymization actions. Restore of the configuration from a restore-point is then deactivated.

Do the following to check and optionally restore the configuration:

1) In the menu of Alfabet Expand, select **Meta-Model** > **Show Meta-Model Configuration History**. The history opens in the middle pane.
2) The history shows the following information:

You might need to scroll from left to right or enlarge the window to see all relevant information.

- **Name**: The name of the update defined during triggering of the update in the update dialog window.

- **Description**: The name of the update defined during triggering of the update in the update dialog window.
- **Update Time**: The date and time when the update was performed.
- **Executor**: The first name and name of the user that triggered the update process.
- **Product Version**: The version of the software that was imported from the source database. Please note that the version must match the current version of the target database. You can for example not restore the configuration from an update with a different version number than the current one.
- **Update Status**: The update status should be `Finished`. If it is `Error`, the update failed and you should check the log file for more information about the error that occurred. If the update process led to problems in your configuration, because only partial updates were performed, you can restore the configuration prior to the update by clicking the update in the table and clicking **Restore Configuration** in the toolbar. If the update was not performed at all, the log file is empty.

3) You can do any of the following:

- **To view the content summary of the AMM file** an update is based on, select the update in the table and click **Show Summary** in the toolbar.
- **To view the log file of the update process**, select the update in the table and click **Show Update Log** in the toolbar. The log file includes the content summary of the AMM file and informs about the success of the update steps.

Log messages are displayed in the language currently set for the master database.

- **To export the content of the history table** click **Export** in the toolbar. Select the output format from the drop-down list.
- `To restore the configuration prior to an update`, click the update that you want to undo, and click **Restore Configuration** in the toolbar. Confirm the warning by clicking **Yes**. The restore may take some time. All connections to the database are closed during update.

Please note the following:

- The current version of the Alfabet database must match the version number of the restore point to perform the restore.
- Reference data configuration in the instance store, like for example cost types, indicator types or user profiles, is not restored via this mechanism.

Correcting Issues that Occurred During Meta-Model Update

After update of the meta-model, check whether issues occurred during update. The following mechanism should be used to find and correct meta-model update issues:

- Checking the Success of Meta-Model Updates
- Securing and Checking Database Consistency with the Meta-Model

Checking the Success of Meta-Model Updates

If issues occurred during update of the meta-model via AMM file, a log file is created during update of the meta-model. The log file is written to the directory that contains the AMM file and has the name <Name of the AMM file>_<Time Stamp>.log. Consult the log file to view any issues that occurred during update.

Securing and Checking Database Consistency with the Meta-Model

The structure and content of the Alfabet database tables must be in accordance with the meta-model of the current Alfabet release.

To ensure database consistency with the meta-model, use only Alfabet software and interfaces for external access when performing changes to the Alfabet database. Any changes performed directly on the database with third-party mechanisms can lead to database corruption and data loss.

To make sure that the current database is in accordance with the current meta-model, Alfabet Expand provides a test mechanism that checks consistency issues.

To check the database consistency:

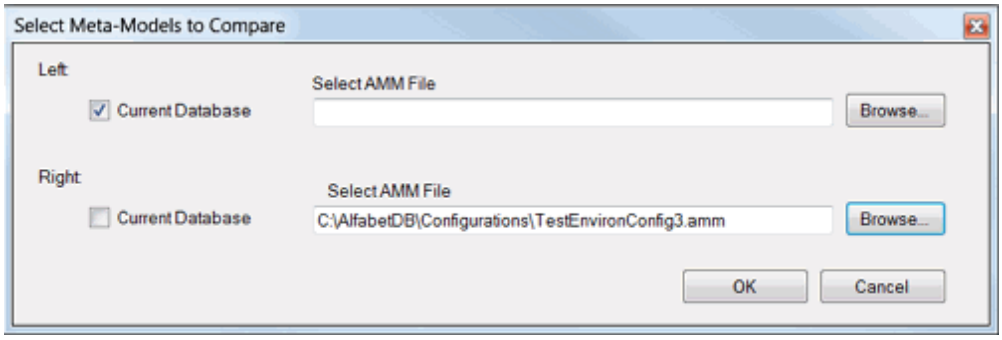
1) Open the tool Alfabet Expand with a server alias.
2) In the menu, select **Meta-Model** > **Check Meta-Model for Database Consistency**. The check will be performed, and a message will display information about the results. If an inconsistency is found (for example, a database table with an invalid name), an error message will be displayed that providing information about the error and advising you to contact Software AG Support.

Comparing Database Configurations

Alfabet Expand provides a mechanism to compare Alfabet configurations. You can either compare the configuration of the current Alfabet database with the configuration stored in an AMM file or the configuration stored in one AMM file with the configuration stored in another AMM file.

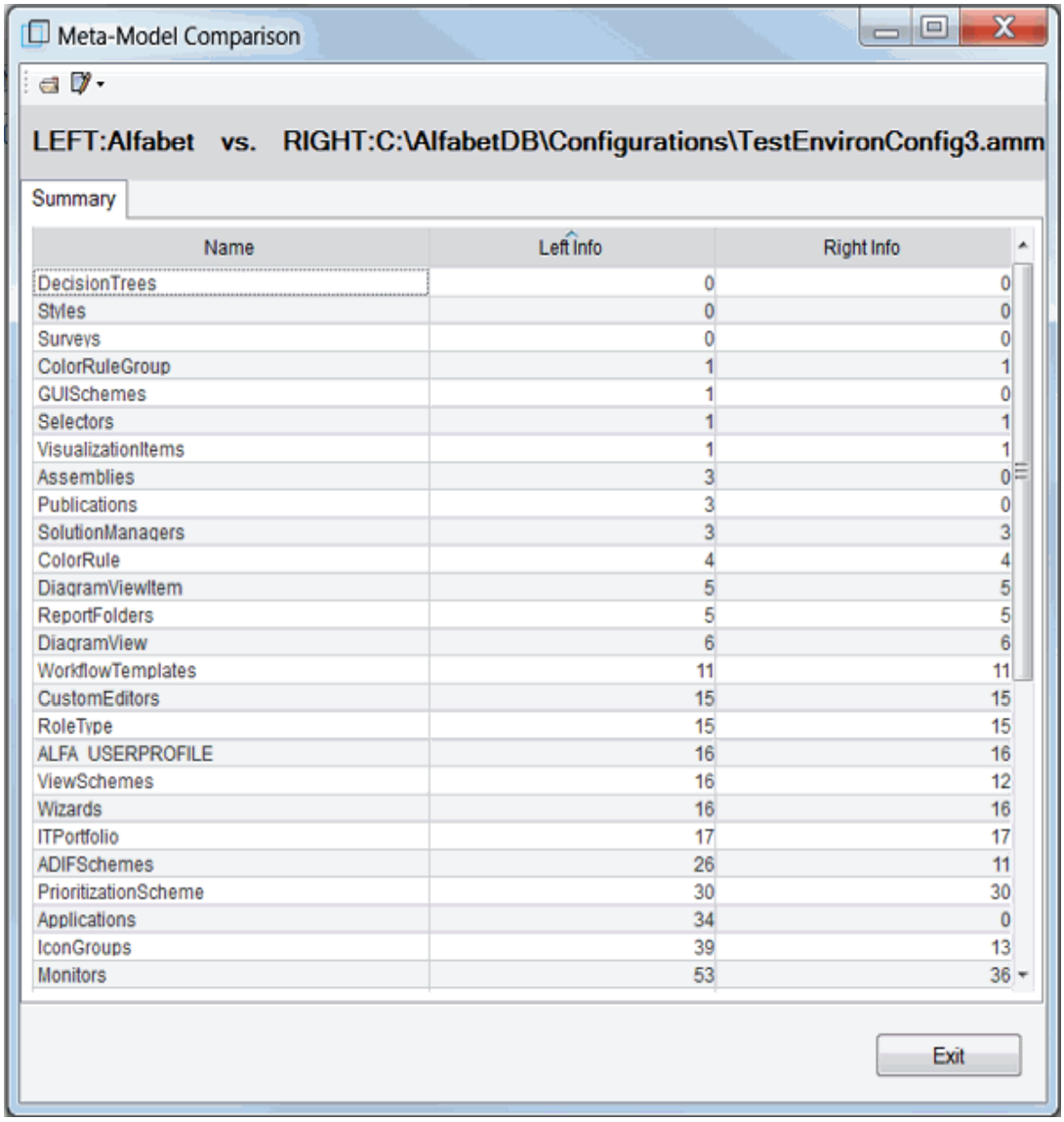
To compare two configurations:

1) In the menu of Alfabet Expand, select **Meta-Model** > **Compare Configurations**. The **Meta-Model Comparison** window opens.
2) In the toolbar of the **Meta-Model Comparison** window, click the **Select Meta-Models**  button to open the meta-model editor.
3) Select the meta-models that shall be displayed on the left side of the report and the right side of the report of the meta-model comparison:



- To select the meta-model configuration of the Alfabet database you are currently working with, activate the **Current Database** checkbox.
- To select a meta-model configuration stored in an AMM file, click the **Browse** button, and select the file from the local file system.

4) Click **OK**. The comparison is executed. This may take some time. The summary of the comparison is displayed in the **Meta-Model Comparison** window:

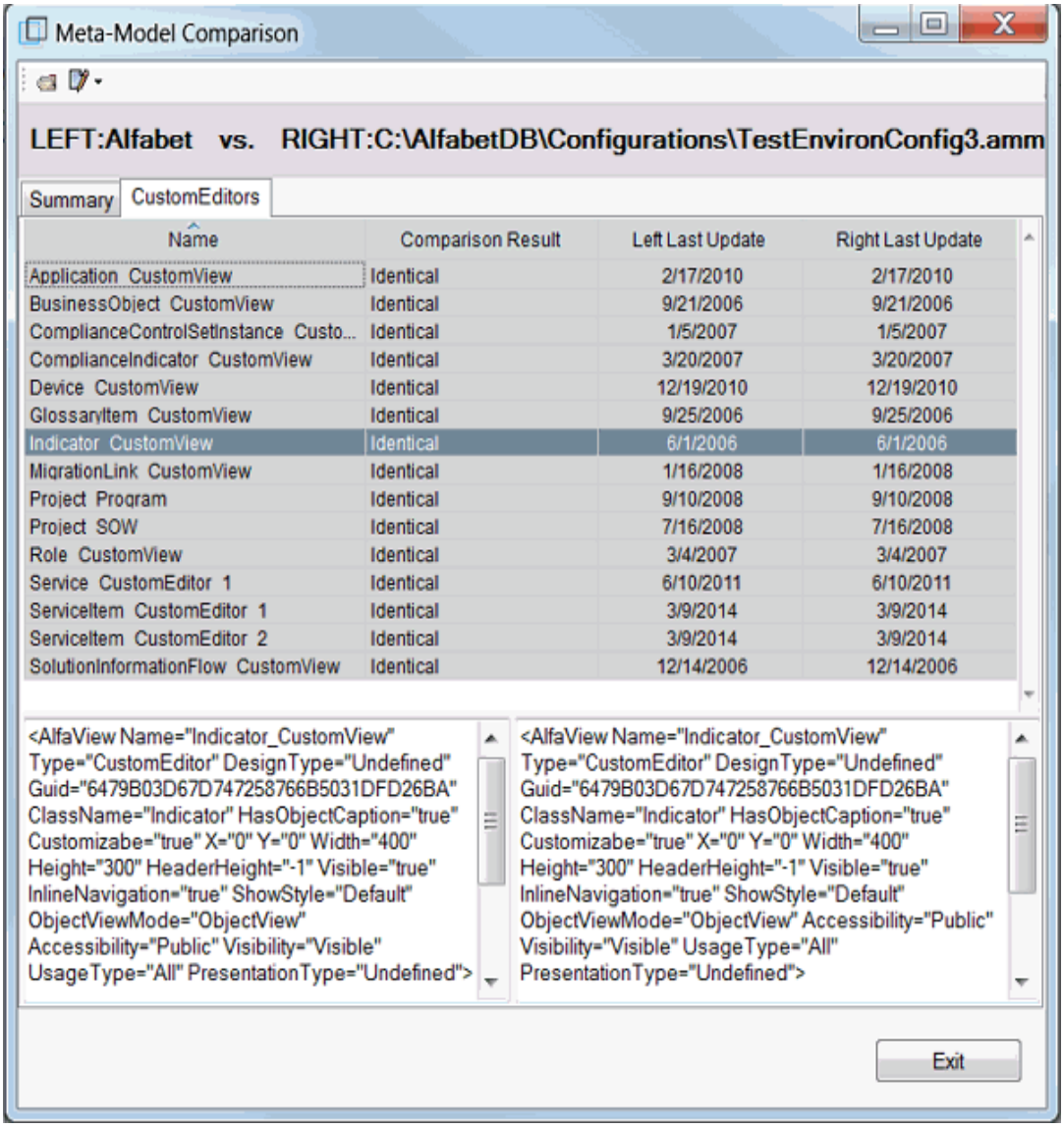


Name	Left Info	Right Info
DecisionTrees	0	0
Styles	0	0
Surveys	0	0
ColorRuleGroup	1	1
GUISchemes	1	0
Selectors	1	1
VisualizationItems	1	1
Assemblies	3	0
Publications	3	0
SolutionManagers	3	3
ColorRule	4	4
DiagramViewItem	5	5
ReportFolders	5	5
DiagramView	6	6
WorkflowTemplates	11	11
CustomEditors	15	15
RoleType	15	15
ALFA_USERPROFILE	16	16
ViewSchemes	16	12
Wizards	16	16
ITPortfolio	17	17
ADIFSchemes	26	11
PrioritizationScheme	30	30
Applications	34	0
IconGroups	39	13
Monitors	53	36

- On top of the **Summary** tab, you can see which meta-model configurations are selected for the left and right side of the comparison.
- The **Name** column lists the configuration object types that are part of the Alfabet meta-model.
- For each configuration object type, the number of configuration objects available in the respective meta-model is given in the **Left Info** and **Right Info** column.

  If the number of objects is identical in the **Left Info** and **Right Info** column, that does not indicate that the configurations are identical. If configuration objects are substituted by other configuration objects or changed, this does not change the overall number of configuration objects. To view the actual changes in the configuration, open a detail view:

5) Double-click a configuration object type in the table. A new tab opens that displays the differences between the two selected configurations for the selected configuration object type:



For each configuration object of the selected type that is part of either of the compared configurations, the following information is displayed

- **Name**: The name of the configuration object.

- **Comparison Result**: Indicates, whether the configuration object is identical or different for the compared meta-model configurations:
- **Identical**: The object exists in both configurations and is identical.
- **Different**: The object exists in both configurations and the configuration changed. The respective rows are displayed in red.
- **Left Only**: The object exists in the configuration represented by the left column only. The respective rows are displayed in yellow.
- **Right Only**: The object exists in the configuration represented by the right column only. The respective rows are displayed in yellow.
- **Left Last Update**: The date when the configuration of the object was last changed in the meta-model selected for the left column.
- **Right Last Update**: The date when the configuration of the object was last changed in the meta-model selected for the right column.

6) If you click a row in the table, the configuration of the object in the left and right column are displayed as XML. You can check in the XML which changes have been applied.
7) To limit the display to the relevant information about objects that have been changed, you can change the view by hiding objects with a selected comparison result value from the list. In the toolbar, click the **Hide** button and select one of the options in the sub menu to hide either objects identical in both databases or objects that are specific for one of the configurations and are not available in the other one. You can repeat the action to select another of the hide options. Both hide options are then applied. Clicking an already selected option will deselect the option.

Exporting Information About the Custom Configuration in an XML File

The **Meta-Model** > **Save Configuration** functionality in Alfabet Expand saves the customer configuration of the meta-model including custom properties, enumerations, culture settings, custom object views, custom explorers, data-set configurations, wizards, monitor settings, XML objects, text templates, workflow templates, configured reports, and custom translations to an XML file. This feature may be useful to generate reports about the configuration, compare different stages of configuration, archive different stages of configuration, or to be able to access the configuration information while the server is running and standalone access to Alfabet Expand is not available.

To save the configuration of the Alfabet solution environment:

1) Open the tool Alfabet Expand with a server alias.
2) In the menu, select **Meta-Model** > **Save Configuration**. A dialog window opens.
3) If necessary, you can change the location and/or file name of the default output file displayed in the dialog box.
4) Select that part of the configuration that you want to save. You can select all or some of the following options:

- **Save Entire Configuration** to save all customer specific configurations including, for example, view schemes, custom object views, and class settings.

Custom reports, publications, and workflow templates configured by your enterprise will not be saved by selecting this option!

- **Save All Reports** to save the custom reports configured by your enterprise.
- **Save All Workflow Templates** to save the workflow templates configured by your enterprise.
- **Save All Publications** to save the publication configuration of the Alfabet Publication Framework.

5) Click **Save**. The configuration XML file is saved.

Building Custom Reports that Inform About the Current Structure of the Standard And Custom Meta-Model

Importing Objects of Configuration Relevant Object Classes from a Master Database

Configurations may require the creation of objects for object classes in the Alfabet meta-model. For example:

- User profiles
- Mandates
- Reference data configured in the Configuration functionalities in the Alfabet user interface, for example, role types or cost types.
- Monitors
- Configured Reports

The AMM update mechanism only includes a subset of the objects stored in the instance store.

For import of other parts of the configuration objects stored as objects of object classes in the Alfabet meta-model, a unidirectional, direct connection can be established between two Alfabet database s for data update in the target database with data in the source database. Once the target database is configured to connect to the source database, a user can perform the import of new configuration relevant meta-model objects via the functionality **Import Data Search** of the Alfabet user interface. The import mechanism informs the user whether objects were added, and new objects can then be selected for import.

Optionally, the import mechanism can be configured to also detect changes to given configuration objects. This option is only available for a subset of the relevant classes.

The import mechanism has been developed for configuration objects only and shall not be used for import of any other object classes. The following limitations apply:

- The import mechanism matches the objects in the source and target database exclusively on basis of the **Image Properties** defined for the object class in the relevant class settings. Image properties must be carefully set to ensure that they can be used to unambiguously identify single objects. Source database objects with image properties not identical to the image properties of an object in the target database are regarded as new objects.

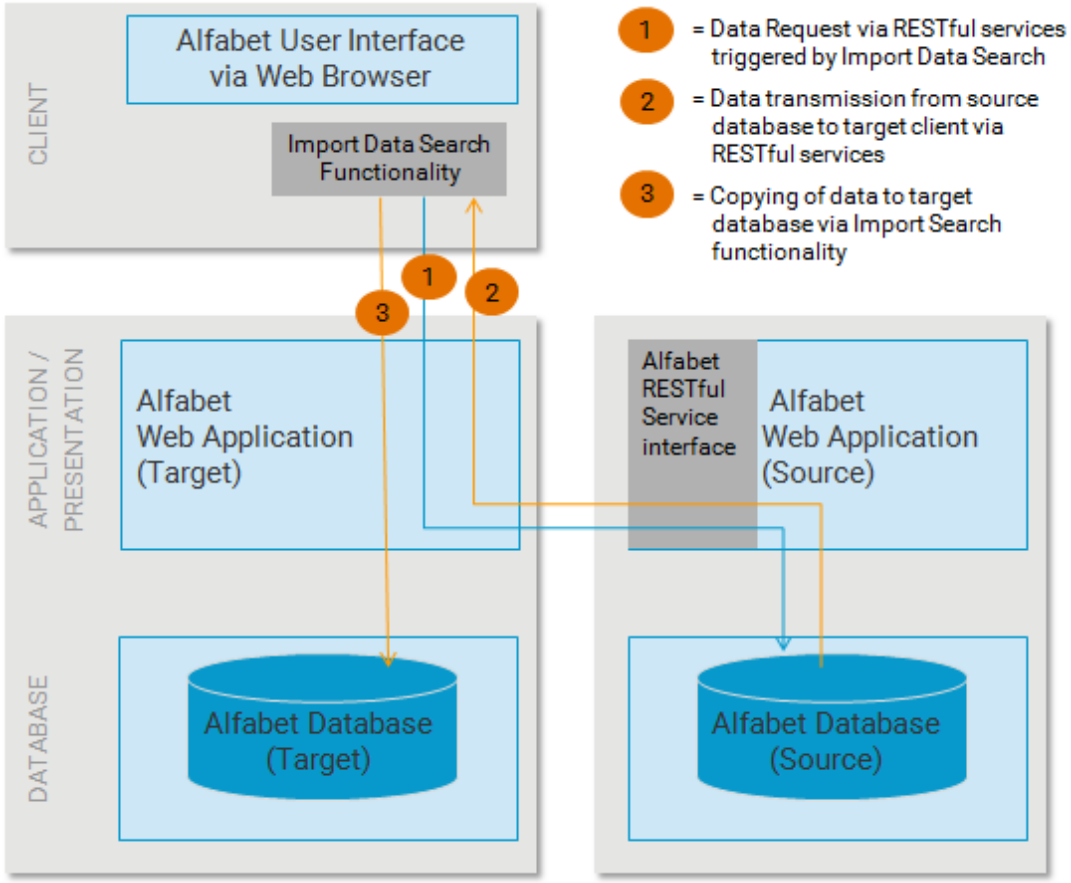
- The import mechanism optionally detects changed objects by comparing the `LAST_UPDATE` property of the object in the source and target database. If the `LAST_UPDATE` date of the object in the source database is younger than the `LAST_UPDATE` date of object in the target database, the object is marked as changed. Please note the following:
  - The comparison is based on the date without time. Therefore, if an object is for example imported and then changed in the source database on the same day, the object is not regarded as changed when performing the next update.
  - The way the object has been changed is not considered by this mechanism.
  - This feature is not available for all relevant object classes and may require additional configuration of the object class.
- During import, the object is created in the target database or, if an object with identical values for the image properties already exists in the target database, the object in the target database is overwritten. A merging process is not available.
- Values for object class properties of the type `ReferenceArray` are ignored during import.
- Values for object class properties of the type `Reference` are only imported if the referenced object already exists in the target database. If you would for example import a hierarchy of cost types, you must first import all root type cost types and then all cost types of the second level of the hierarchy followed by all cost types of the third level of the hierarchy. If you import a cost type on the second level of the hierarchy without prior import of the parent cost type, the cost type will become a root level cost type in the target database.
- Values for object class properties of the **Type** `String` that are based on an enumeration, that means that the **Type Info** attribute of the property is set to the name of an enumeration, are only imported if the specified enumeration item is available in the target database. It is recommended to import the enumeration configuration of the source database into the target database via AMM file based meta-model update prior to using **Import Data Search**.

The following information is available:

- Configuring the Connection Between the Source and Target Database
- Configuring the Source Database to Provide Access to the Relevant Data via the Alfabet RESTful Services
- Configuring the XML Object AlfabetIntegrationConfig of the Target Database
- Creating an Alfabet Database Connection in the Target Database
- Providing Access to the Import Data Search Functionality
- Optional Configuration for Detection of Changes to Configuration Object
- Importing Configuration Objects from a Source Database

Configuring the Connection Between the Source and Target Database

The connection to the source database is based on the Alfabet RESTful services. Establishing the connection requires that two Alfabet Web Application s are running, one with a connection to the source and one with a connection to the target database. The release version of the Alfabet components must be identical for source and target database. When a user triggers the data import mechanism on the Alfabet user interface, the Alfabet Web Application of the target database sends RESTful service requests to the Alfabet RESTful API of the Alfabet Web Application of the source database to request the data.



The configuration includes the following:

- Configuration of the Alfabet Web Application of the source database to activate the Alfabet RESTful API and set the required access permissions for access to the relevant data in the source database.
- Configuration of the Alfabet Web Application of the target database to act as a RESTful client. This configuration is done via Alfabet Expand in the XML object `AlfabetIntegrationConfig`. The configuration also includes a definition of the object classes that shall be imported.
- Configuration of the user access to the functionality in the target database. For this purpose, an object of the object class Alfabet Database Connection must be created via the Alfabet user interface and the access to the Alfabet database connection as well as to objects of the relevant object class must be granted to a user via the user profile assigned to the user.

The definition of object classes for that import is performed is typically further limited with each configuration step:

- Typically, only one database connection is configured in the XML object ***AlfabetIntegrationConfig*** for the connection to the source database. The configuration includes the definition of all object classes that shall be imported from the source database.
- Multiple different Alfabet database connections can then be configured for the same connection to the source database. For each Alfabet database connection, one or a subset of the object classes that can be imported via the connection can be selected as allowed object classes.
- In the **Import Data Search** functionality, the user first selects an Alfabet database connection. He/she then selects one of the object classes configured to be allowed object classes for the Alfabet database connection. Optionally, a full text search filter can be used to further restrict the import to a subset of the selected object class. As a result, import is limited to a subset of objects of one object class at a time. This is also relevant for performance, because the transmission of object data is limited to the objects found by the filtering.

The configuration includes a number of access permission settings to ensure that import can be restricted to a defined user for a defined object class.

The following configuration steps are required to activate the functionality:

- Configuring the Source Database to Provide Access to the Relevant Data via the Alfabet RESTful Services
- Configuring the XML Object AlfabetIntegrationConfig of the Target Database
- Creating an Alfabet Database Connection in the Target Database
- Providing Access to the Import Data Search Functionality
- Optional Configuration for Detection of Changes to Configuration Object

Configuring the Source Database to Provide Access to the Relevant Data via the Alfabet RESTful Services

The source database must be configured to allow access to the endpoint `objects` and `metamodel` of the Alfabet RESTful services. The required configuration is described below in a short overview. For a detailed description of the required configuration of the see the Alfabet RESTful API reference manual *Alfabet RESTful API*.

- A license for the Alfabet Data Integration Framework (ADIF) must be active.
- Configure the `web.config` file of the Alfabet Web Application to enable Alfabet RESTful services.
- Configure the Alfabet Web Application on the Web Server to enable access to the Alfabet RESTful services.
- Configure the server alias of the Alfabet Web Application to enable access to the Alfabet RESTful services.
- Create a user for access via the Alfabet RESTful services, with **Has GetObjectsByFilters Access** and **Has Meta-Model Access** permissions to the Alfabet RESTful services and generate a user password for the user.
- For each object class that shall be imported, define a class setting with the attribute **Allow Read via Rest API** set to `true`.

- Assign the relevant class settings to a user profile via a view scheme and assign the user profile to your access user.
- If your company is using Alfabet 's mandate concept, make sure that the mandate settings for your access user allow viewing all relevant objects.

In addition to the configuration relevant for the Alfabet RESTful services, the object classes that shall be imported into the target database may be configured to have an object class property `LAST_UPDATE` that enables to detect whether a configuration object that exists in both the source and target object has been changed in the source database after the last change to the object in the target database. For more information about the complete configuration to activate this feature, see Optional Configuration for Detection of Changes to Configuration Object.

Configuring the XML Object AlfabetIntegrationConfig of the Target Database

The Alfabet Web Application of the target database acts as RESTful client, connecting to the RESTful API of the Alfabet Web Application of the source database. The connection parameters must be configured in the configuration of the target database using Alfabet Expand:

1) Go the **Presentation** tab of Alfabet Expand and expand the explorer node **XML Objects** > **Integration Solutions**.
2) Right click the XML object `AlfabetIntegrationConfig` and select **Edit XML** from the context menu. The XML object opens in the middle pane.
3) Edit the XML as described below.
4) Click the **Save**button to save your changes.

The XML object must contain the following XML structure:

```
<AlfabetIntegrationConfig>
   <Connection
      name = "My Test Import"
      service="http://localhost/ALFABET1"
      active="true"
      data_portion="80"
      search_limit="500"
      emptyValues="true"
      user="DAME"
      psw="H7GLVUGZWQETKJFSX7HY6OK2CUB4WRGK"
      profile="MASTER">
         <ImplementedClass parent ="IncomeType" />
         <ImplementedClass parent ="ObjectMonitor" />
   </Connection>
</AlfabetIntegrationConfig>
```

The following XML elements and their attributes are part of the specification:

Element (Bold) / Attribute	Allowed Values	Mandatory/Optional	Configuration Requirements
Alfabet-Integration-Config		Mandatory	Root node of the configuration
Connection		Mandatory	The configuration parameters for sending the RESTful service call and request the data from the source database. This XML element can be added multiple times to define multiple source databases.
`name`	String	Mandatory	Enter a unique name for the connection configuration. The name is used to identify the connection configuration for example in editors on the Alfabet user interface.
`service`	URL	Mandatory	Enter the URL of the Alfabet Web Application for access to the source database.
`active`	`true`/`false`	Optional, the default is `true`	This attribute can be defined to deactivate (`false`) or activate (`true`) the connection. If the attribute is set to `false`, the connection will not be displayed in the drop-down list for the connection specification in the editor **Alfabet Database Connection** on the Alfabet user interface. In the **Import Data Search** functionality, all Alfabet database connections already defined for the connection definition are not available for triggering data import.
`data_portion`	Integer	Optional, the default is 100.	Enter the maximum number of results to be returned in a single request. If the maximum number of search results exceeds the `data_portion` value, multiple requests are sent to the Alfabet RESTful API of the Alfabet Web Application connected to the source database, each returning a subsequent subset of the data, until the maximum overall number of search results defined with the attribute `search_limit` is reached.
`search_limit`	Integer	Optional, the default is 300.	Enter the maximum number of objects for that the response of the call shall return data. This value may be set to limit data return from databases containing a high number of objects. **NOTE**: Import of a very high number of objects may lead to performance issues. It is recommended to restrict the maximum number of objects to be returned and use the filters on the **Import Data Search View** to limit the

Element (Bold) / Attribute	Allowed Values	Mandatory/Optional	Configuration Requirements
			number of imported data to a subset of the data available in the source database.
`emptyValues`	`true/false`	Optional	If set to `true`, all object class properties for an object are included into the response call, even if the property value is `NULL`. If set to `false`, only object class properties that are set in the source database are transmitted. It is recommended to set the parameter to `false` for transmission of high amounts of data. In the target database, values for a property are set to `NULL` for an object if the response call does not contain a value for a property or if it contains an empty field for the property.
`user`		Mandatory	Enter the user name of the user configured in the source database to grant access to the functionality. For more information about the configuration of the user, see .
`psw`		Mandatory	Enter the REST API password of the user configured in the source database to grant access to the functionality. For more information about the configuration of the user, see .
`profile`		Mandatory	Enter the user profile configured in the source database to grant the correct access permissions to the relevant object classes.
ImplementedClass		Mandatory	Defines which classes can be imported from the source database. Multiple XML elements `ImplementedClass` can be added to the XML object to define import from all relevant classes within one `Connection` definition.
`parent`		Mandatory	Enter the name of the object class to be imported or the name of the parent class in the class hierarchy of the Alfabet meta-model. Import will be allowed for the specified object class and all object classes that are subordinate of the specified object class in the Alfabet meta-model.

The definition of the object classes for import can be done via the parent object class to ease configuration. The hierarchy in the Alfabet meta-model is not visible in the Meta-Model tab in Alfabet Expand. If you have a valid license for ADIF, you can see the class hierarchy in the explorer of the ADIF tab in Alfabet Expand.

The following table lists all object classes that are relevant for import and the parent object class.

The table also informs whether the `LAST_UPDATE` property required for the optional data update functionality is available for the object class and whether it is available by default or requires configuration.

If the object class has standard object class properties of the type Reference or ReferenceArray, this information is also given in the list. Properties of the type ReferenceArray are ignored during import and properties of the type Reference are only imported if the target object for the reference is already available in the target database. If you import for example configured reports structured in report folders, you must allow import of both classes and report folders must be imported first.

If you define a parent object class, you might technically be able to import data for a high number of object classes. Nevertheless, object classes not listed here should not be imported via the **Import Data Search** functionality.

Class caption	Object class name	Parent object class name	Update functionality	Import restrictions
Currency	Currency	ITClass	No	Currencies are hierarchically structured. The currencies must be imported from top level first to the bottom level last.
Cost Type	CostType	ITClass	Requires configuration	Cost types are hierarchically structured. The cost types must be imported from top level first to the bottom level last.
Income Type	IncomeType	ITClass	Requires configuration	Income types are hierarchically structured. The income types must be imported from top level first to the bottom level last.
Connection Type	ConnectionType	ITClass	Requires configuration	
Connection Method	ConnectionMethod	ITClass	Requires configuration	
Connection Frequency	ConnectionFrequency	ITClass	Requires configuration	
Connection Data Format	ConnectionDataFormat	ITClass	Requires configuration	

Class caption	Object class name	Parent object class name	Update functionality	Import restrictions
Role Type	RoleType	ITClass	Requires configuration	
Skill	Skill	ITClass	Requires configuration	
Data Retention Policy	DataRetentionPolicy	ITClass	Requires configuration	
Indicator Time Series	IndicatorTimeSeries	ITClass	Requires configuration	
Generic Reference Data	GenericReferenceData	Artifact	Available by default	
Alfabet Database Connection	Alfabet_DBConnection	IntegrationConnection	Available by default	Assignment of authorized user groups is based on a ReferenceArray property and will not be included into the import. For security reasons, the development and production environment should use a different user configuration and authorized access shall not be imported but configured in the target database.
API Gateway Database Connection	APIGateway_DBConnection	IntegrationConnection	Available by default	Assignment of authorized user groups is based on a ReferenceArray property and will not be included into the import. For security reasons, the development and production environment should use a different user configuration and authorized access shall not be imported but configured in the target database.
API Portal Database Connection	APIPortal_DBConnection	IntegrationConnection	Available by default	Assignment of authorized user groups is based on a ReferenceArray property and will not be included into the import. For security reasons, the development and production environment should use a different user configuration and authorized

Class caption	Object class name	Parent object class name	Update functionality	Import restrictions
				access shall not be imported but configured in the target database.
ARIS Database Connection	ARIS_DBConnection	IntegrationConnection	Available by default	Assignment of authorized user groups is based on a ReferenceArray property and will not be included into the import. For security reasons, the development and production environment should use a different user configuration and authorized access shall not be imported but configured in the target database.
CentraSite Connection	CentraSite_DBConnection	IntegrationConnection	Available by default	Assignment of authorized user groups is based on a ReferenceArray property and will not be included into the import. For security reasons, the development and production environment should use a different user configuration and authorized access shall not be imported but configured in the target database.
Report Folder	ALFA_REPORTFOLDER	ALFA_REPORTBASE	Available by default	Assignment of authorized user groups is based on a ReferenceArray property and will not be included into the import. For security reasons, the development and production environment should use a different user configuration and authorized access shall not be imported but configured in the target database.
Configured Report	ALFA_REPORT	ALFA_REPORTBASE	Available by default	If configured reports are structured in report folders, the report folder must be imported prior to importing the configured reports. Access permission configuration will not be included into the import. For security reasons, the development and production environment should use a different user configuration and access permissions shall not be imported but configured in the target database.
Consistency Monitor	ConsistencyMonitor	ObjectMonitor	No	The listeners of the monitor are stored as ReferenceArray and are not included into the import. For security reasons, the development and production environment should use a different user configuration

Class caption	Object class name	Parent object class name	Update functionality	Import restrictions
				and listeners are then anyway not meaningful to import.
Notification Monitor	NotificationMonitor	ObjectMonitor	No	The listeners of the monitor are stored as ReferenceArray and are not included into the import. For security reasons, the development and production environment should use a different user configuration and listeners are then anyway not meaningful to import.
System Date Monitor	SystemDateMonitor	ObjectMonitor	No	The listeners of the monitor are stored as ReferenceArray and are not included into the import. For security reasons, the development and production environment should use a different user configuration and listeners are then anyway not meaningful to import.
Mandate	ALFA_MANDATE	SYSTEMCLASS	No	

Creating an Alfabet Database Connection in the Target Database

Configuration is done in the **Integration Solutions Configuration** functionality on the Alfabet user interface.

1) Go to the **Integration Solutions Configuration** functionality and click the **Alfabet Database Connection** node in the **Integration Solutions Configuration** explorer.
2) In the view, click **New** > **Create Alfabet Database Connection**.
3) In the **Alfabet Database Connection** editor, define the following fields as needed:

   Basic Data tab:

- **ID**: Alfabet assigns a unique identification number to each ARIS database connection. This number cannot be edited.
- **Name**: Enter a unique name for the Alfabet database connection. The name is displayed in the **Import Data Search** functionality in the drop-down list for selection of the master database for import of Alfabet configuration data.
- **Release Status**: Select the Alfabet database connection's current release status.

The set of release statuses available for an object class are configured by your solution designer in the configuration tool Alfabet Expand. For more information, see the section Configuring Release Status Definitions for Object Classes in the reference manual *Configuring Alfabet with Alfabet Expand*. For general information about release statuses, see the section *Understanding Release Statuses* in the reference manual *Getting Started with Alfabet*.

- **Description**: Enter a meaningful description that will clarify the purpose of the Alfabet database connection.

Authorized Access tab:

- **Authorized User**: Click the **Search** icon to assign an authorized user to the selected ARIS database connection. The authorized user will have Read/Write access permissions for the object and is responsible for the maintenance of the object.
- **Authorized User Groups**: Select one or more checkboxes to assign Read/Write access permissions to all users in the selected user group(s).

Connection tab:

- **Alfabet Connection**: Select the connection to the relevant Alfabet database that is configured in the XML element ***Connection*** in the XML object ***AlfabetIntegrationConfig*** in Alfabet Expand.
- **Allowed Classes**: Select one or multiple object classes for the users authorized to import data via this Alfabet database connection shall be able to import data via the **Import Data Search** functionality. All classes that may be imported according to the definition in the XML element ***Connection*** in the XML object ***AlfabetIntegrationConfig*** in Alfabet Expand are listed in the drop-down list of the field.

After you have selected object classes for import, you will see your selection in the field **Selected Classes**. The user will see the selected classes in the drop-down list of the object class filter in the **Import Data Search** functionality after having selected the Alfabet database connection.

4) Click **Test Alfabet Database Connection**. If your settings are correct, a message "The connection is valid" is displayed. Otherwise, an error message is displayed.

Providing Access to the Import Data Search Functionality

In the target database, the user that shall import the data via the **Import Data Search** functionality must log in with a user profile that is configured to grant access to the functionality per object class. The following must be configured for the user profile:

- The user must be able to access the **Import Data Search** functionality (`ImportDataSearch`) either via a menu item or a guide view or guide page.

For information about how to add a functionality to a user profile, see Making Functionalities Accessible to a User Profile.

- The attribute **Allow Update from External Reference Data Service** must be set to `true` for all class settings of all object classes for those objects shall be imported. The relevant class settings must be assigned via the view scheme configuration to the user profile the user logs in with.

For information about how to configure class settings for an object class, see Configuring Class Settings for Object Classes and Object Class Stereotypes.

For information about how to assign class settings to the view scheme of a user profile, see Configuring a View Scheme for a User Profile.

Optional Configuration for Detection of Changes to Configuration Object

Optionally, the import mechanism can be configured to detect whether a configuration object that exists in both the source and target object has been changed in the source database after the last change to the object in the target database. This mechanism is exclusively based on the value of the `LAST_UPDATE` property of the object.

Some of the configuration objects already have a standard property called `LAST_UPDATE`. For these object classes the functionality is available by default. For other object classes, the functionality can be implemented by adding a custom property `LAST_UPDATE` to the object class in both the source and the target database. A custom property with the Name attribute set to `LAST_UPDATE` will automatically be set to the current data by the system whenever the object is changed.

If a `LAST_UPDATE` property is available for the object class in both source and target database the search functionality of **Import Data Search** automatically compares the dates stores in the custom or standard properties `LAST_UPDATE` of the object if the source and the object with the same image property values in the target database. No further configuration is required.

If the `LAST_UPDATE` date of the target database is older than the one in the source database, the object is marked as changed. If the object in the source database is changed on the same day as the object in the target database, the change cannot be detected, because the LAST_UPDATE property only safes the date and not the date and time of the update.

For a subset of object classes relevant for import a `LAST_UPDATE` property is neither available by default nor can it be added via configuration. A complete overview of the availability of the `LAST_UPDATE` property is given in the list of relevant object classes provided in the section Configuring the XML Object AlfabetIntegrationConfig of the Target Database.

To create a custom `LAST_UPDATE` property, you must perform the following action in Alfabet Expand:

1) In the **Meta-Model** tab, expand the **Classes** node.
2) In the explorer, right-click on the object class for that you want to define a `LAST_UPDATE` property and select **Add New Property** from the context menu.
3) In the window that opens, enter `LAST_UPDATE` into both the **Property Name** and **Technical Name** field. `LAST_UPDATE` must be written in capital letters.
4) Click **OK**. The new property is added to the explorer tree and the attribute window shows the attributes of the new property.
5) In the **Property Type** attribute, select `Date` from the drop-down list.
6) Click the **Save** button to save your changes.

Importing Configuration Objects from a Source Database

The import of configuration relevant objects from a source to a target database can be performed under the following preconditions:

- The connection and data access permissions are fully configured as described in the section above.
- Both databases are currently available via a running Alfabet Web Application.

Import is done in the functionality **Import Data Search** and is a performed in two steps:

- First a comparison between the data in the source database and your current database is triggered. The result is displayed in a table.
- Import can then be performed by selecting one or multiple objects in the table and triggering import via a button in the toolbar.

Prior to starting an import, please carefully consider the restrictions that apply to data import that are listed in the section Importing Objects of Configuration Relevant Object Classes from a Master Database!

Do the following to import objects from the source database in the functionality **Import Data Search**:

1) Set the following filters:

- **Master Connection**: Select the Alfabet database connection that connects to the source database. You will see all Alfabet database connections that have been configured by your solution designer in the functionality **Integration Solutions Configuration**.
- **Search for**: Select the object class for that you want to import objects. The drop-down list contains all object classes that are configured in the Alfabet database connection as allowed object classes and for that you have import permissions granted via the user profile you are logged in with.
- **in**: After selection of an object class in the **Search for** field, the drop-down list of the field **in** is automatically filled with all searchable properties of the class. These are all object class properties with the **Property Type** set to `String` or `Text`. You can deselect properties from the drop-down list to search in a sub-set of the searchable properties only. Please note that for classes that are not visible in the **Classes** explorer in the **Meta-Model** tab of Alfabet Expand, search is limited to finding objects with a defined name.
- **Search Pattern**: Enter the string to find in the object class properties selected in the **in** field. The search finds objects in both source and target database. An asterisk can be user as wildcard in the search pattern.

2) Click the **Search** button. Results are listed in the table. The table lists all objects matching your filter definition with their name, description and, if applicable, their short name. The column **Data Source** and the color coding for the rows in the table informs whether the objects are found in the target database or in the source database only:

Object available in current (target) database	Object available in source database	Value for Data Source	Row color
yes	no	object is not included into the table	
yes	yes, LAST_UPDATE date identical or prior to LAST_UPDATE date of the object in target database or objects do not have a LAST_UPDATE property	Target	no background color
yes	yes, LAST_UPDATE data younger than LAST_UPDATE date of the object in the target database	Target to Update	blue
no	yes	Master	green

3) Select one or multiple new or changed objects in the table and click **Create as Copy** to copy the object into your database.

If the Alfabet user interface is rendered in a secondary language and data translation is enabled for the currently used interface language, the search results are displaying the translated object class properties, if provided. For objects for that no translation is provided, the string is displayed in the standard language (English, en-US). Please note the following for search and import of data:

- The search refers to the object class property values as displayed. That means if a translated value is displayed, the search finds the object by the translated value. If the English value is displayed, the user must enter the English string to find the object.
- The data import compares the values provided in the standard language, which is English (en-US). If for example objects are identified by name and the translation of the name changes while the name in the original language is not changed, the object is detected as changed (if this feature is implemented for the class) and not as a new object.
- If automated data translation capability is implemented, all automated translations for objects that have be imported are handled as automated translations in the target database.

Managing Assemblies

If a DLL file is developed by Software AG for your specific requirements, the file must be uploaded to the Alfabet database. This is typically done by a system administrator using the tool Alfabet Administrator. How the upload is performed will depend on how the file is delivered by Software AG:

- When the assemblies are delivered as files, they are uploaded via the assembly management functionality of the Alfabet Administrator. The upload of assemblies to a database using the Alfabet Administrator is described in the section *Managing Assemblies* in the reference manual *System Administration*.
- When the assemblies are delivered in a AMM file, the AMM file must be applied to the target database with the **Update Meta-Model** functionality. For more information, see the section *Uploading Assemblies via the *.amm Update File to Another Database* in the reference manual *System Administration*.

It is recommended that assemblies are uploaded first to a test environment to test the effects on the existing database before they are implemented in the production environment. If the assembly impacts the solution configuration, it typically must first be uploaded to the database of a development environment and then reviewed in the test environment before being applied to the production environment.

The tool Alfabet Expand provides a mechanism to upload both the customized configuration and the assembly files into an AMM updater file. This file can then be used to update the test and production environment to include both the DLL files and the required configuration changes. The AMM file can also trigger the execution of scripts during the upload of assemblies, if required.

The functionality to take over the configuration of a master database to a target database can be used in the test or production environment to take over the DLL files and the required configuration changes that have been made in the development environment. If the execution of a script is required, this must be performed separately by a system administrator.

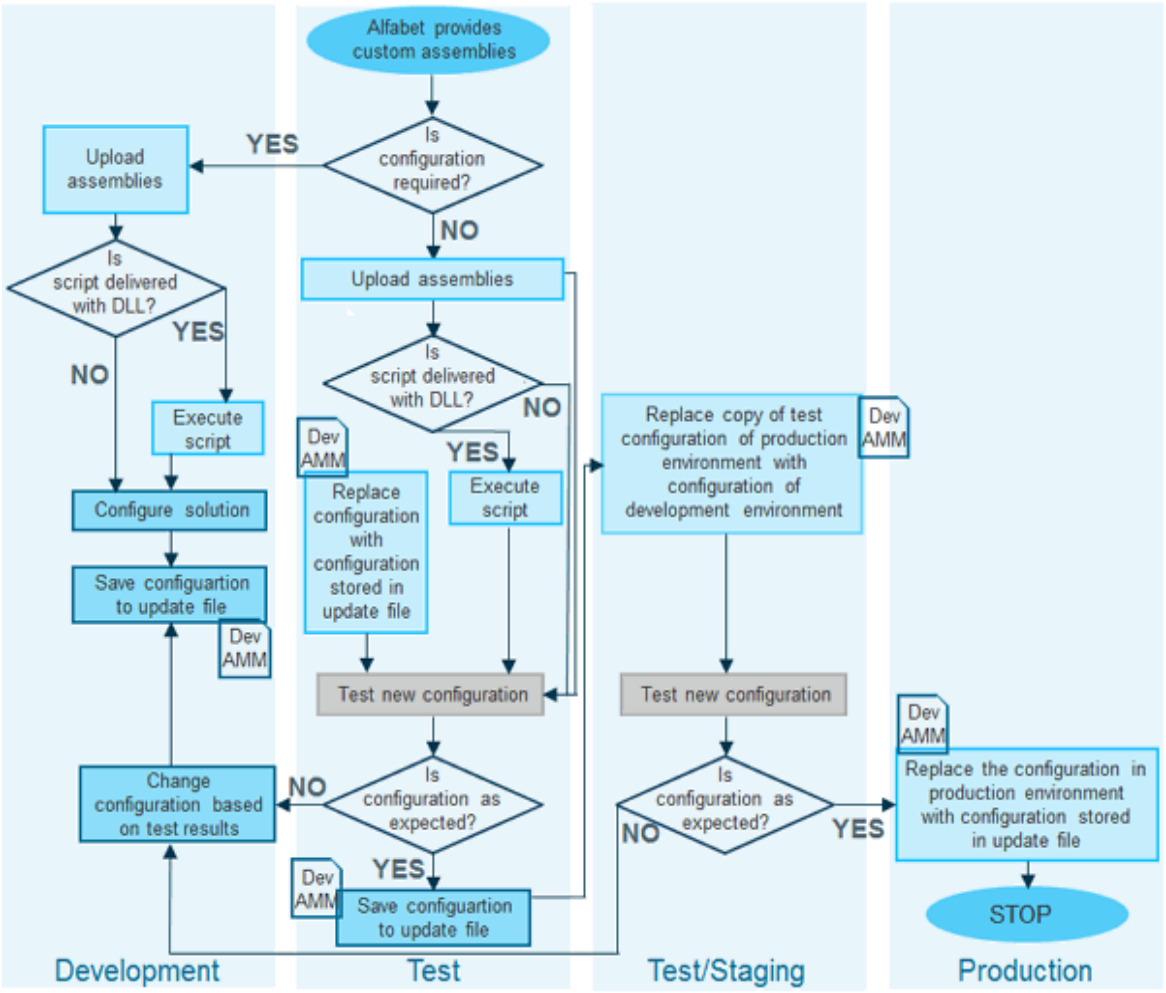


FIGURE: Example for a best practice workflow for the upload of custom assemblies to the database

Uploading Assemblies from One Database to Another Database via AMM File

The AMM based update mechanism that allows you to update the configuration of a Alfabet database with the configuration of another Alfabet database can also be used to streamline the upload of assemblies to a production database. The assemblies delivered by Software AG can be uploaded to an AMM file. Optionally, the execution of scripts and the merge or replace of the configuration of the target database with a configuration optimized for use of the assembly can be configured in the AMM file. The AMM file can then be used to upload the assemblies to a target database together with all scripting and configuration changes required for the assembly.

To save assemblies (and optionally, the custom configuration) to an AMM file.

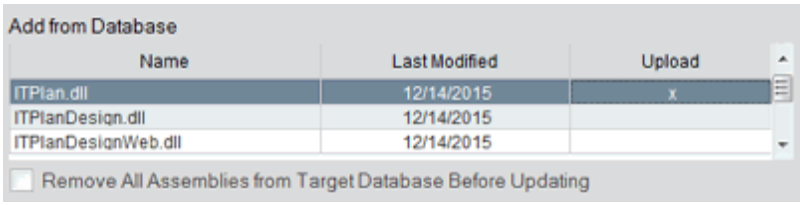
1) In the Alfabet Expand menu bar, select **Meta-Model** > **Create Configuration Meta-Model Update File**. An editor window opens:

2) Click the **Browse** button next to the **Select Output File** field and select the location and file name for storing the AMM file in the browser that opens.

3) In the **Name** field and **Description** field n the **General** tab, provide a meaningful name and description for the update performed by the AMM file when applied to the target database. The name and description are displayed in the **Update Meta-Model** dialog box when the AMM file is

used to update the configuration in a database. It allows the person performing the update to be informed about the content of the configuration in the AMM file.

4) In the **Assemblies** tab, all assemblies in the current database are listed in the **Add from File** table. Click the corresponding cell in the **Upload** column of the DLL to include the assembly in the AMM file. An X is displayed in the cell of each selected DLL. Click the cell again to remove the X and deselect the assembly.



5) In the **Assemblies** tab, select the following options related to the upload of the assembly, if applicable for your configuration:

- **Remove All Assemblies from Target Database Before Updating**: This button is deactivated. The functionality is currently not available.
- **Update Script**: This section allows to define the execution of a script during upload of an assembly. Instructions on how to fill in the fields in the section are provided by Software AG on delivery of the assembly.

> The execution of scripts is limited to scripts delivered by Software AG Support for various tasks such as maintenance procedures. Executing scripts that have not been provided by Software AG can cause severe damage to your Alfabet database.

6) Optionally, you can select all or part of the configuration of the current database to be uploaded to the AMM file together with the assembly by selecting the affected parts of the configuration in these **Meta-Model Content** tab and **Guide Pages** tab. For more information, see Saving Complete or Tagged Types of Configuration Objects or the Complete Configuration with Alfabet Expand.

> If you want to upload only single configured elements (for example, only one workflow instead of all of the workflows), you can define the AMM file with the **Find Meta-Model Objects for Deployment** functionality instead of using the **Create Configuration Meta-Model Update File** option. For more information, see Saving Selected Objects of the Configuration with Alfabet Expand.

7) Click the **Create** button. The AMM update file is created and an information about the successful creation is displayed.

To upload the assemblies stored in an AMM update file to a target database, follow the procedure described in the section Updating the Configuration of the Alfabet Solution Environment with Alfabet Expand.

Anonymizing Data

The data anonymization capability ensures data transparency and accountability across the enterprise as well as meet requirements of the General Data Protection Regulation (GDPR), and security requirements for business operations. For example, by means of pseudonymization, user data can be replaced with an artificial identifier to ensure anonymity if the user leaves the enterprise, or data about the architecture elements in IT can be used in the production environment but all sensitive data can be replaced with artificial identifiers in the development or test environment.

Anonymization can be performed on data of the type `String`, `Text`, `URL` and `Picture`. During anonymization, the original data can either be set to NULL, or be replaced with a random string or the REFSTR of the current object. Replacing the data with the REFSTR of the current object ensures that there is an unambiguous assignment of the data to the object. Therefore, the anonymization of key values, like the Name of an object, the User Name property for Alfabet users, or of object class properties defined as unique keys for a property is restricted to this method. Instead of the Name property of an object class, the solution administrator can decide during configuration of the feature to specify another property as key property that will then be restricted to being replaced with the REFSTR of the object during anonymization.

Anonymization requires configuration of individual object classes and properties thereof to be included into anonymization. Triggering anonymization via one of the available mechanisms then leads to anonymization of all data of all object classes for that anonymization is configured. The data is anonymized in the database table for the object class as well as in the audit history table for the object class. If user data is anonymized and the User Name of the user is included in anonymization, the User Name is also anonymized in all audit history tables of all object classes and in the information about the creation user and last update user available for configuration objects in Alfabet Expand.

For user data an additional method for anonymization is available. Data can be anonymized for individual users only. In addition, individual users can be excluded for anonymization. If data is anonymized in general, these users are not anonymized, which will for example ensure that administrators are still able to log in to Alfabet with their user name while all other user names were changed during anonymization.

Anonymization is performed via the Alfabet Server the Alfabet Web Application is configured to connect to. Prior to performing anonymization, a solution designer must pre-configure the anonymization capability as described in the following.

The following information is available:

- Activating Anonymization for Object Class Properties
- Changing Key Property Settings for Object Classes
- Configuring the Object Class to be Anonymized
- Configuring Anonymization of Data of Single Users Only
- Excluding Users from Anonymization
- Anonymizing Data
- Anonymizing All Relevant Data in the Alfabet database
- Anonymizing Data of Selected Users
- Creating a Database Archive File Containing Anonymized Data
- Checking Anonymization Actions

Activating Anonymization for Object Class Properties

The following steps are involved in the activation and configuration of anonymization:

- Changing Key Property Settings for Object Classes
- Configuring the Object Class to be Anonymized

Changing Key Property Settings for Object Classes

It is crucial to maintain data integrity during anonymization to be able to work with a database that contains anonymized data without any restrictions in usability.

A number of mechanisms are implemented that ensure data integrity:

- If an object class property of the type `String` is based on an enumeration, it is excluded from anonymization.
- The following object class properties are either based on values defined for example in an XML object or are crucial for implemented functionalities and are therefore excluded from anonymization:
- The object class properties `Stereotype`, `ObjectState`, `State`, `Status` and `Status History` for all object classes for which they are available.
- The object class properties `CheckInStatus` and `CheckInProtocol` of the object class `Project`.
- The object class properties `LevelID` and `LevelIDNum` of the object classes `Business Process` and `Domain`.
- For the object class `PERSON`, the object class property values of single objects can be excluded from anonymization. A new Boolean property `ExcludedFromAnonymization` has been added to the object class `PERSON`. A new option **Exclude From Anonymization** has been added to the **User** editor to set this property. If the checkbox is selected, the data for the user cannot be anonymized.
- If the object class property is mandatory, the anonymization method that sets the value to `NULL` is not available.
- If the object class property is included into the specification of a Class Key with a uniqueness constraint for the object class, the object class property can only be substituted with the REFSTR of the object during anonymization to ensure that the uniqueness constraint is observed.
- Independent from Class Key specifications for individual object classes, the `Name` property of an object class is by default regarded as key property and the object class property can only be substituted with the REFSTR of the object during anonymization. This rule can be overwritten by the solution designer in the XML object `AnonymizationKeyManager`. By default, the XML object `AnonymizationKeyManager` contains a definition for the object class `Person`. For the object class `Person`, the object class properties `USER_NAME` and `TECH_NAME` are set as key properties by default. This ensures that the audit history information and the information about the creator and last update user for configuration objects in Alfabet Expand, that is based on the `USER_NAME` or `TECH_NAME` property, is not corrupted.

  The server alias of the Alfabet Web Application can be configured with the attribute **Server Settings** > **General** > **Update History User Name** to use the `TECH_NAME` instead

of the `USER_NAME` of the user for writing information about the `CREATION_USER` and `LAST_UPDATE_USER` into the audit history tables.

If you would like to change the key property settings in the XML object `AnonymizationKeyManager` in Alfabet Expand, you should do the following prior to activating anonymization for object class properties in the Alfabet Expand meta-model:

1) In the **Presentation** tab, expand the node **XML Objects**.
2) Right-click the sub-node `AnonymizationKeyManager` and select **Edit XML** from the context menu. The XML object opens in the middle pane.
3) For each object class for that you want one or multiple object class properties to be used as key properties instead of the Name property only, add an XML element `ClassAnonymizationKeyDef` to the root XML element `AnonymizationKeyManager`.
4) Set the following XML attributes for the new XML element `ClassAnonymizationKeyDef`:

- `ClassName`: Write the name of the object class that the key shall be defined for into the attribute.
- `AnonymizationKeyProperties`: Write the name of the object class property that shall be the key property into the attribute. If multiple object class properties shall be defined as key, the object class property names must be written comma separated into the attribute.

5) Click the **Save**button to save your changes.

The following specification changes the key for the object class Business Function to both the Name and Level ID instead of the Name only:

```
<AnonymizationKeyManager>
   <ClassAnonymizationKeyDef ClassName="BusinessFunction"
   AnonymizationKeyProperties="Name,LevelID"/>
</AnonymizationKeyManager>
```

Configuring the Object Class to be Anonymized

Data of an object class is only anonymized if the object class is configured to be anonymized. By default, anonymization is deactivated in the configuration of all object classes.

The activation settings that are required can only be performed on a subset of classes and object class properties:

- Both the object class and the object class property must be protected or public.
- The object class property must be of the type:
- `String`, not based on an enumeration
- `Text`
- `URL`
- `Picture`
- The property is not explicitly excluded from anonymization. Currently, the following object class properties cannot be anonymized:

- The object class properties `Stereotype`, `ObjectState`, `State`, `Status` and `Status History` for all object classes for which they are available.
- The object class properties `CheckInStatus` and `CheckInProtocol` of the object class `Project`.
- The object class properties `LevelID` and `LevelIDNum` of the object classes `Business Process` and `Domain`.

Do the following to activate anonymization for an object class:

1) In the Meta-Model tab of Alfabet Expand, expand the node **Class Model** > **Classes**.
2) Click the object class that you want to activate anonymization for.

If the object class does not have any object class properties that can be anonymized, the attribute is deactivated.

3) Click into the attribute **Property Anonymization** to open the **Anonymization Rules for Class Properties** table. The table lists all object class properties of the current object class that might be anonymized. If the table does not open, the object class does not have any object class properties that can be anonymized.
4) In the column **Anonymization Type** of the table, select one of the following anonymization methods for each property to be anonymized:

- `ToBeLeftUnchanged`: The values for the object class property are not changed. Anonymization is not applied.
- `ToBeNullified`: The values for the object class property are set to `NULL`.
- `ToBeRandomized`: The values for the object class property are substituted with a random string. The length of the string is identical to the length of the original string. If auditing is activated, identical values are replaced with the same string in the audit tables. This method is only available for properties of the type `String` or `Text`.

The following randomization mechanisms are implemented for strings and texts:

- Properties of the type `String`: A random string is created using the Generate Random Password function of the .NET library with the length of the original string as input. Random strings do not contain special characters.
- Properties of the type `Text`: A random text is generated from the following base text:

```
lorem ipsum dolor sit amet consectetur adipisicing elit
sed do eiusmod tempor incididunt ut labore et dolore
magna aliqua ut enim ad minim veniam quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea
commodo consequat duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu
fugiat nulla pariatur excepteur sint occaecat cupidatat
non proident sunt in culpa qui officia deserunt mollit
anim id est laborum
```

This randomization mechanism is creating random texts from this base text using the variable integer parameters `minWords`, `maxWords`, `minSentences`, `maxSentences`, `numLines`, and the variable boolean parameter `randomSize`. These parameters are a function of the data in the database that shall be randomized

2.000 random texts are created during an anonymization run and subsequently used to replacing the original text from the class property. This allows performance characteristics to be retained after anonymization of data.

- `ToBeReplacedByKey`: The values for the object class property are substituted with the `REFSTR` value of the current object. This method keeps the information about which object the anonymized data belongs to. This might be useful for example in case of user data. This method is only available for properties of the type `String`.

  The columns in the table provide information about the object class property that should be considered when choosing the anonymization method:

  - **Property Name**: The name of the object class property to identify the data that can be anonymized.
  - **Property Type**: The data type the object class property. The data type determines the availability of methods to be selected. Only strings can be replaced with a key and only string and text properties can be randomized.
  - **Is Mandatory**: Some protected object class properties are mandatory. A value must be provided for these properties to ensure operability of all features implemented in Alfabet. If `True` is displayed in this column, the object class property cannot be set to `NULL` during anonymization.
  - **Is Anonymization Key**: If `True` is displayed in this column, the object class property is either defined as key for the object class in the XML object `AnonymizationKeyManager` or there is no definition for the object class in the `AnonymizationKeyManager` and the object class property name is `Name`. This property can neither be set to NULL nor be randomized during anonymization.
  - **In Unique Index**: If `True` is displayed in this column, the object class property is part of a unique class key definition of the object class. This property can neither be set to NULL nor be randomized during anonymization.
  - **Has Validator**: If `True` is displayed, the values for this object class property are validated against a validator defined in the attribute **Validator** of the object class property. The availability of methods is not restricted if a validator is defined but depending on the selected anonymization methods the validation might fail after anonymization. You should check the object class property validator settings prior to selecting a validation method.

Configuring Anonymization of Data of Single Users Only

For the object class `PERSON`, anonymization cannot only be performed on a per class but also on a per object basis. That means that next to anonymizing data for all users that are currently stored in the Alfabet database, you can also anonymize data for one or multiple selected users. anonymization is triggered via Alfabet Expand Windows or on the Alfabet user interface and executed via the Alfabet Server. Prior to triggering user data anyonymization, anonymization must be configured for the object class `PERSON` as described in the section Configuring the Object Class to be Anonymized.

Excluding Users from Anonymization

The object class property values of single users can be excluded from anonymization. A new boolean property `ExcludedFromAnonymization` has been added to the object class `PERSON` for that purpose.

The property is set in the **Users Administration** functionality in the Alfabet user interface:

1) Navigate to the **Users Administration** functionality.
2) In the table, select the user that you want to exclude from anonymization.
3) In the toolbar, click **Edit**  . The **User** editor opens.
4) Select the checkbox **Exclude From Anonymization**.
5) Click **OK** to save your changes.

Anonymizing Data

If anonymization is triggered by one of the methods described below, the anonymization is applied to all data for an object class property if the object class is configured to be anonymized and an anonymization method is specified for the object class property.

For information about the required configuration to enable anonymization, see Activating Anonymization for Object Class Properties.

Anonymization changes the following object class property values in the database:

- Values stored in the object class table of the object class in the Alfabet meta-model. If data translation is enabled, translated values are also anonymized.
- Values stored in the history table <CLASSNAME>`_AU` of the object class.
- Anonymization will be applied to the values in the columns `AUDIT_USER`, `CREATION_USER`, `LAST_UPDATE_USER` and `DELETE_USER` of all audit history database tables (<CLASSNAME>`_AU` and `RELATIONS_AU`), if one of the following applies:
  - If the object class property `USER_NAME` of the object class `PERSON` is anonymized, and the server alias for connection to the Alfabet database is configured to use the `USER_NAME` for auditing.
  - If the object class property `TECH_NAME` of the object class `PERSON` is anonymized, and the server alias for connection to the Alfabet database is configured to use the `TECH_NAME` for auditing.
  - If the object class property `USER_NAME` of the object class `PERSON` is anonymized with the method `ToBeReplacedByKey`, anonymization will also be applied to the **Last Update User** and **Creator** attributes in the **Tech Info** section of the configuration objects in Alfabet Expand. For all other anonymization methods these attributes are left unchanged.

If the anonymized user is the **Last Update User** or **Creator** of any configuration object subordinate to the **Classes** explorer node, the connections of all currently running Alfabet components with the Alfabet database will be terminated and the database will be locked during the anonymization process. The Alfabet components need to be restarted afterwards.

There are three ways to anonymize data:

- The **Anonymize Data** functionality anonymizes all values for all object class properties configured to be anonymized in the Alfabet Meta-Model of the current Alfabet database.
- The **Anonymize User Data** functionality anonymizes data for one or multiple selected users, which means for one or multiple selected objects of the object class PERSON, if the configuration of the object class PERSON specifies that the respective object class property shall be anonymized.
- The **Archive Current Database with Anonymized Data** functionality anonymizes all values for all object class properties configured to be anonymized in the Alfabet Meta-Model in a database archive file (ADBZ) during creation of the archive file. The data in the original Alfabet database that is archived is not affected.

Anonymization can be triggered using other Alfabet components:

- All data anonymization methods are available via Alfabet Expand Windows.
- In Alfabet Expand Web, the method **Anonymize Data** is available in the context menu of **Utilities** > **Meta-Model Configuration**.
- In the Alfabet Administrator, all anonymization methods are available as options in the context menu of the server alias. For more information, see *Anonymizing Data* in the reference manual *System Administration*.
- In the **Users Administration** functionality on the Alfabet user interface, the user administrator can select one or multiple users in the table and select **Action** > **Anonymize User** in the toolbar to trigger anonymization for the selected user(s). For more information, see *Anonymizing User Data* in the reference manual *User and Solution Administration*.
- Anonymization can be triggered via a service call to the new endpoint `anonymizeuser` of the Alfabet RESTful service to anonymize data for users that are found via specification of the user's `REFSTR` in the REST API request. For more information, see *Anonymizing User Data For Selected Users* in the reference manual *Alfabet RESTful API*.
- The console application `AlfaAdministratorConsole.exe` can be used to anonymize all object class property values configured to be anonymized in an Alfabet database or data for one or multiple selected users. For more information, see *Anonymizing Data* in the reference manual *System Administration*.

The following information is available:

- Anonymizing All Relevant Data in the Alfabet database
- Anonymizing Data of Selected Users
- Creating a Database Archive File Containing Anonymized Data

Anonymizing All Relevant Data in the Alfabet database

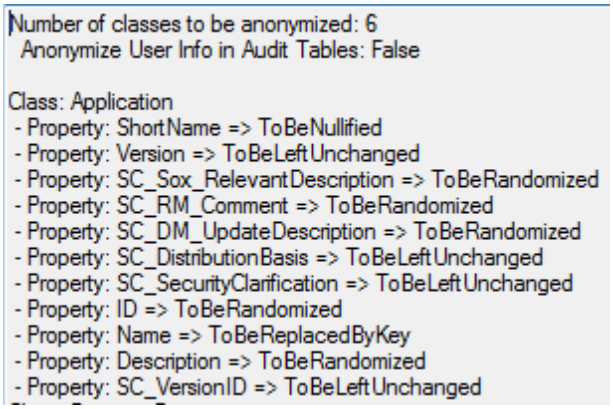
The **Anonymize Data** functionality anonymizes all values for all object class properties configured to be anonymized in the Alfabet Meta-Model of the current Alfabet database.

Anonymizing data is a sensible process that might disrupt database integrity. It cannot be reverted! **Always back up the Alfabet database prior to triggering data anonymization!**

Please note that the connection to the Alfabet database is closed during the anonymization process. Re-login of the current user is performed automatically with no need to fill in a login screen with the user name and password of the last login prior to anonymization. If the object class property `USER_NAME` is anonymized for the object class `PERSON`, a re-login to Alfabet Expand is not possible after anonymization. To avoid problems with re-login, the user performing anonymization can be excluded from anonymization. Alternatively, `ToBeReplacedByKey` may be used as anonymization method, which replaces the user name with the `REFSTR` of the respective user. Automatic re-login will then fail, but you can re-login via the login screen with the `REFSTR` as user name. The `REFSTR` can for example be written from a configured report prior to performing anonymization.

To trigger data anonymization in Alfabet Expand:

1) In the menu, select **Managers** > **Database Manager** > **Anonymize Data**.
2) In the window that opens, define the location for the log file that shall be used to log information about the anonymization process. If you select an already existing log file, the standard message of the selector window informs you that the file will be overwritten. Nevertheless, the log information is appended to the existing content of the selected file.
3) The following information is displayed:



- **Number of classes to be anonymized**: The overall number of object classes for that the attribute **Anonymize** is set to `True`.
- **Anonymize User Info in Audit Tables**: Informs about the anonymization in audit history tables (<CLASSNAME>`_AU` and `RELATIONS_AU`) in the Alfabet database and in the attributes **Creator** and **Last Update User** in the **Tech Info** section of the configuration objects in Alfabet Expand. The attribute **Anonymize** must be set to `True` for the object class `Person` and a method other than `ToBeLeftUnchanged` must be selected for the object class property `USER_NAME` and/or for the object class property `TECH_NAME` of the object class `Person`.
- `True`: The user name will then be anonymized in all audit history tables and in the attributes **Creator** and **Last Update User** in the **Tech Info** section of the configuration objects in Alfabet Expand.
- `True (Technical Info Not Anonymized)`: The user name will only be anonymized in the audit history tables. The attributes **Creator** and **Last Update User** in the **Tech Info** section of the configuration objects in Alfabet Expand is not anonymized. This means that the `TECH_NAME` is

anonymized, while the `USER_NAME` is left unchanged and the server alias configuration specifies that the `TECH_NAME` is written into the audit history.

- `False (Technical Info Anonymized)`: The user name will only be anonymized in the attributes **Creator** and **Last Update User** in the **Tech Info** section of the configuration objects in Alfabet Expand. The user information in the audit history tables is not anonymized. This means that the `USER_NAME` is anonymized, while the `TECH_NAME` is left unchanged and the server alias configuration specifies that the `TECH_NAME` is written into the audit history.
- `False`: Audit history tables and the attributes **Creator** and **Last Update User** in the **Tech Info** section of the configuration objects in Alfabet Expand are not anonymized.
- **Class**: For each object class for that data will be anonymized, all object class properties that may be subject to anonymization are listed with information about the configured anonymization method.

4) Click **Anonymize**.

Anonymizing Data of Selected Users

The **Anonymize User Data** functionality is available to anonymize data for one or multiple selected users, which means for one or multiple selected object of the object class `Person`.

Data for a selected user is only anonymized if anonymization is enabled in the configuration of the object class `Person` and if the user is not explicitly excluded from anonymization.

Anonymizing data is a sensible process that might disrupt database integrity. It cannot be reverted! **Always back up the Alfabet database prior to triggering data anonymization!**

If the anonymized user is the **Last Update User** or **Creator** of any configuration object subordinate to the **Classes** explorer node, the connections of all currently running Alfabet components with the Alfabet database will be terminated and the database will be locked during the anonymization process. The Alfabet components need to be restarted afterwards.

To trigger anonymization of the data of selected users in Alfabet Expand:

1) In the menu, select **Managers** > **Database Manager** > **Anonymize User Data**.
2) In the window that opens, select one or multiple users to be anonymized in the table. You can set the following filters and click **Update** to search for specific users:

- **Search Pattern**: Enter a search pattern to search for in either all standard attributes or in the attribute selected in the drop-down list of the field on the right of the **Search Pattern** field.
- **Profile**: Select a user profile in the drop-down list to limit the display in the table to users assigned to the selected user profile.
- **Show Alfabet-managed Users Without Password**: Select the checkbox to limit the display in the table to users that are logging in with a user name and password managed in Alfabet and that currently have no password assigned.

3) In the field **Log File**, define the location for the log file that shall be used to log information about the anonymization process. If you select an already existing log file, the standard message of the selector window informs you that the file will be overwritten. Nevertheless, the log information is appended to the existing content of the selected file.

4) Click **Anonymize**.

Creating a Database Archive File Containing Anonymized Data

The **Archive Current Database with Anonymized Data** functionality is available to anonymize all values for all object class properties configured to be anonymized in the Alfabet Meta-Model in a database archive file (ADBZ) during creation of the archive file. The data in the original Alfabet database that is archived is not affected. This method is useful to archive the content of a productive database in an anonymized form to implement it in a test or development environment without information about sensitive data.

To archive the non-anonymized data in the current database and apply anonymization to the data in the archive file, do the following in Alfabet Expand:

1) In the menu, select **Managers** > **Database Manager** > **Archive Current Database with Anonymized Data**. The **Alfabet Archive Manager (Anonymized Data)** opens.
2) In the window that opens, click the **Browse**  button on the right of the **The file in which you want to archive the Alfabet database** field to define the location for the database archive file on the local file system. The default name of database archive file contains the string `Anonymized` to distinguish between anonymized and non-anonymized archives which shall be used to log information about the anonymization process.
3) Select the checkbox **Squeeze Audit Tables** if you would like the audit tables to be cleaned prior to archiving the database. Any entries that were generated during, for example, a batch update of data via batch utilities without documenting any audit relevant changes will be deleted from the audit tables prior to archiving of the database.
4) Select the checkbox **Include User Settings** if you want user settings defined for the current database to be added to the archive.
5) Click **Archive**.

To overwrite the content of a database with the data from a database archive file, use the **Restore Database Archive** option in the context menu of the tool Alfabet Administrator. For more information, see *Using the Context Menu Available via the Alias Explorer Node* in the reference manual *System Administration*.

Anonymizing data is a sensible process that might disrupt database integrity. It cannot be reverted! **Always back up the Alfabet database prior to restoring it with an anonymized archive file!**

Checking Anonymization Actions

Anonymization actions on the current database are logged. Information is written into a log file that can be specified in a dialog that opens for triggering the anonymization from Alfabet Expand Windows, the Alfabet Administrator or the Alfabet user interface. When selecting an already existing log file, the log information in the file is appended, although a message in the file selector states that the file is overwritten.

The console applications write information into the standard log files for Alfabet console applications if defined in the command line.

The table provides information about the kind of anonymization action performed, the status of the anonymization, the user performing the anonymization and the update time. In addition, the version of the software that was implemented when the anonymization action was performed is displayed.

Configuring Default User Settings for the User Community

The XML object ***UserPersonalSettings*** allows you to define the default settings for various aspects of Alfabet including, for example, the default user profile displayed in the Login screen, the interface language that Alfabet opens with, or the automatic execution of validation rules for object definition.

The settings configuration in the XML object ***UserPersonalSettings*** will be applied to a named user upon activation in Alfabet. In other words, the default values you define here will be applied to any user that has not already defined his/her user settings.

The user can then modify the default values, as needed. Any changes you later make to the default values in the XML object ***UserPersonalSettings*** will not impact a user who has already defined his/her user settings.

To edit the XML object ***UserPersonalSettings***:

1) Go to the **Presentation** tab, expand the **XML Objects** folder, and expand the **Administration** folder.
2) Right-click the XML object ***UserPersonalSettings*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) Edit the XML attributes, as needed. The table below displays the XML attributes that can be edited for the XML object ***UserPersonalSettings***:

XML Element (bold) / XML Attribute	Explanation
UserPersonalSettings	
`DefaultProfile`	Enter the default user profile that will be selected per default in the **User Profile** field in the Login screen. The user may select a different user profile, as needed.
`DefaultCulture`	Enter the culture code for the default language that should be displayed the first time a user logs in to Alfabet. The Alfabet interface is available in the following languages:

Language	Locale ID
Arabic (Saudi Arabia)	1025

XML Element (bold) / XML Attribute	Explanation
	German (Germany) 1031 English (United States) 1033 French (France) 1036 Portuguese (Brazil) 1046 Polish (Poland) 1045 Users can change the interface language via the < **UserName** > menu in the Alfabet interface. For more information about changing the interface language, see the section *Defining the Language of the User Interface* in the reference manual *Getting Started with Alfabet*. The default culture can also be defined in the server alias configuration. For more information, see the section *Creating a Server Alias for the Alfabet Web Application* in the reference manual *System Administration*.
`BookmarkLayout`	Define the default view for bookmarks available in the **My Bookmarks** functionality. Enter "`Details`" to display the bookmarks in a table structure. Enter "`Icons`" to display the bookmarks as icons on the desktop. For more information about working with bookmarks, see the section *Creating, Managing, and Accessing Your Bookmarks* in the reference manual *Getting Started with Alfabet*.
`ShowRetiredObjects`	Enter "`true`" to display objects that have reached a retired object state in search results as well as standard and configured reports. Enter "`false`" if objects that have reached a retired object state are not to be displayed in search results and standard and configured reports.
`ShowEmptyProfileValues`	Enter "`true`" to display properties that have no value in the **Attributes** section of the object profile. Enter "`false`" if properties must have a value defined to be displayed in the **Attributes** section of the object profile.
`ValidateObjectsByAccess`	Validation rules are based on the post-conditions defined for wizard steps. Any object in the object class that a post-condition is associated with may be validated when a user accesses the object in the object profile. Please note that executing validation rules based on

XML Element (bold) / XML Attribute	Explanation
	post-conditions for wizard steps may significantly impact performance since each time the object is visited and hence refreshed from the database this setting will trigger the execution of all validation rules pertaining to the associated wizard. If the validation of a property value is required, it is recommended that a check entry is configured for the property in the context of a custom object cockpit. Enter "`No`" if validation rules should not be executed. Enter "`Yes`" to execute all validation rules that have been configured via post-conditions on wizard steps. Violations will be displayed in the object profile.  Please note that users can individually decide whether to execute validation rules via the **Execute Object Validation on Access** field in the **User Settings** editor. The user setting in the **User Settings** editor will override the configuration in the XML attribute `ValidateObjectsByAccess` in the XML object ***UserPersonalSettings***. If the user selects **Default** in the **Execute Object Validation on Access** field in the **User Settings** editor, the definition configured in the XML attribute `ValidateObjectsByAccess` will be used. For more information about the definition of post-conditions for wizard steps, see the section Defining a Post-Condition for a Wizard Step. For more information about the configuration of a check entry in the custom object cockpit, see the section Adding a Check Entry to the Object Cockpit.
`IncludeFilter-Summary`	Define the default value for the **Include Filter Summary** field in the **User Settings** editor in Alfabet. The **Include Filter Summary** setting specifies whether a summary of the filter settings defined for a page view or configured report shall be exported to the relevant export file formats available in the page view or report. For more information, see the section *Exporting Data* in the reference manual *Getting Started with Alfabet*.
`WFGuiVersion`	Specify the view that opens when the **Workflow Activities** link defined in a navigation page is clicked or when the **Open Workflow Activities** link in an object profile is clicked. Enter "`Version_1`" if the standard **My Workflow Activities** views should open per default. Enter "`Version_2`" if the configured **Workflow Activities Explorer** view should open per default. For more information about the configuration of workflows, see the chapter Configuring Workflows.
`Collaboration`	Enter "`Yes`" if the collaboration functionality should be enabled. The **Show Collaboration Panel** button will be displayed in object views and page views. Enter "`No`" if the collaboration functionality should not be enabled. For more information about the working with

XML Element (bold) / XML Attribute	Explanation
	the collaboration capability, see the section *Communicating with Your Colleagues via the Alfabet Internal Collaboration Functionality* in the reference manual *Getting Started with Alfabet*.
`MaxDataset-CellTextLength`	Specify the number of characters to display in columns in datasets in order to improve the handling of data in columns in the Alfabet interface. Enter "`-1`" or leave the value empty to specify an unlimited number of characters, or enter an integer to limit the number of characters to display in columns. Any characters after the specified number will be truncated. Users can point to the cell to display a tooltip with the complete text or access the full information about the object by using the preview functionality.
`ValidateIFlowDates`	Specify whether information flows must be defined within the start and end dates of their source and target objects. Enter "`Yes`" if the start and end dates of information flows shall be validated. In this case, an error message will be displayed stating that information flows may not be created if their start and end dates are outside of the period of the start and end dates of the source and/or target objects. Enter "`No`" if the start and end dates of information flows shall not be validated based on the start and end dates of their source and target objects. In this case, information flows may be created with conflicting dates but they will be highlighted red in views to indicate that a confect exists. The XML attribute `ValidateIFlowDates` is set to "`Yes`" per default.
`ShowDialogWhen-PrintingDatasets`	Enter "`true`" if the user prompt " **Please use the Print capability in your browser to print this view** " should be displayed each time the user prints an Alfabet view. Enter "`false`" if the user prompt should be suppressed and not displayed each time the user prints an Alfabet view.
`PowerPoiontExport-Format`	Specify the slide size for export to Microsoft® PowerPoint®. The selected option corresponds to the slide size in the generated PowerPoint. Select `Default` if the slide size should be determined by the default settings specified by your enterprise. It is assumed Full HD resolution is supported. The following size formats are available: • **PPT Widescreen 16:9**: 720 x 405 Points • **PPT Standard 4:3**: 720 x 540 Point • **PPT Widescreen x4**: 1440 x 810 Points. This format is equal to the slide size for current Microsoft® PowerPoint® export features. • **Enable Splash Screen**: • **Enable Assistant**:

XML Element (bold) / XML Attribute	Explanation
`EnableSplashScreen`	Enter "`true`" if the **Edit Splash Screen** option should be available in the **Bookmark** menu. Enter "`false`" if the **Edit Splash Screen** option should not be available in the **Bookmark** menu. For more information about the splash screen capability, see the chapter *Creating a Splash Screen As Your Start Page* in the reference manual *Getting Started with Alfabet*.
`EnableAutoHelp`	Enter "`true`" if the configured automated assistants shall be available. Enter "`false`" to disable all configured automated assistants. For more information about the help provided by assistants, see the section *Using the Automated Help Assistant* in the reference manual *Getting Started with Alfabet*. For more information about configuring the automated assistant capability, see the section Providing Custom Online Help to the User Community.
`CaptureTranslatableContentInCurrentUILanguage`	Enter "`true`" if per default object data in editors and wizards shall be shown and edited in the current language of the user interface. Enter "`false`" if per default object data in editors and wizards shall be shown and edited in the primary language specified for the enterprise or for the object in the editor/wizard. This checkbox must be selected to capture data in a secondary language or statutory language. For more information about capturing data in a secondary language, see the sections Allowing Data To Be Captured in a Non-Primary Language and Specifying a Statutory Language for the Enterprise.
`AnimatedVisualizations`	Enter "`true`" if the graphics in standard views and configured reports displaying portfolios, Gantt charts, branching diagrams, sunray diagrams, circular roadmaps, etc. shall be built dynamically when the view is opened. Enter "`false`" if the graphics shall not be visualized dynamically when a relevant view is opened.
`EnableWebUIPolling`	Web polling allows the user interface to poll the server for the completion status of background processes for which completion shall be announced via an Event Feedback message. Enter "`true`" to enable user interface polling. Additional configuration is required to use the Web polling mechanism. For more information, see the section Configuring the Web Polling Mechanism.
`EnableFeedbackBot`	Enter "`true`" if the Feedback Bot should be displayed. The Feedback Bot allows users to provide feedback for a view, configured report, object cockpit, guide view, etc. For more information about the configuration of the Feedback Bot, see the section Configuring the Feedback Bot.
`EnableCheckFeedbackSameObject`	Select the checkbox to activate the **Feedback for Current View** capability. This is relevant for user profiles responsible for reviewing and

XML Element (bold) / XML Attribute	Explanation
	responding to feedback provided via the Feedback Bot. Feedback that has been provided for a view or report via the Feedback Bot can be displayed in the Alfabet user interface in a secondary view for those users responsible for reviewing and responding to the feedback. This allows the responsible users to navigate the Alfabet user interface and see the feedback for the relevant view where they currently are. A secondary view with the caption **Feedback for Current View** will be displayed with a link if feedback has been provided for the view, configured report, object cockpit, guide view, etc. Clicking the link will open the *Feedback Review Functionality* in a new browser tab which displays all feedback in detail for the view.
`EnableHelpBot`	Enter "`true`" if the FAQ Bot should be displayed.

4) In the toolbar, click the **Save** button to save the XML definition. You must completely close Alfabet Expand to activate any changes you have made to the XML object ***UserPersonalSettings***.

Configuring the Use of External Sources with Alfabet

External data access (for example, redirecting of object selectors) must be configured for your Alfabet solution by Software AG Support. You will be able to configure your individual interface settings only after the basic configuration of the Alfabet solution has been defined.

The configuration of external sources is a complex process and is described in detail in the section *Integrating Data from External Sources* in the reference manual *System Administration*. It is recommended that you edit the XML object ***ExternalSourceConfiguration*** via the assistant provided in the tool Alfabet Administrator only.

If synchronization with an external data source is to be performed on access of objects via the Alfabet interface, the relevant selectors must be configured to allow selection of data from the external source instead of from the Alfabet database. The interface must then be configured to use the external source selectors instead of the standard selectors.

The following must be carried out by Software AG Support in the tool Alfabet Expand in order to configure the redirection of object selectors:

- Create a selector definition to access the external data source. This requires the configuration of an XML object ***SelectorDef***.
- Configure Alfabet to use the external object selector instead of the default object selector in the XML object ***GeneralViewMap***.

The selector definitions in XML objects ***SelectorDef*** cannot be created by the customer, but the XML objects provided by Software AG are visible and editable in Alfabet Expand. Nevertheless, it is recommended that you NOT edit selector definitions or the XML object ***GeneralViewMap*** for external sources without support from Software AG Support.

The following information is available:

- Configuring Selectors for Data Synchronization with External Sources
- Configuring the Alfabet User Interface to Map Standard Configuration Objects to Custom Configuration Objects

Configuring Selectors for Data Synchronization with External Sources

If the data of a specific object class in the Alfabet database is synchronized with the data of an external source, the object selectors in the Alfabet interface will display the data from the external source instead of the data from the Alfabet database.

The following must be carried out by Software AG Support in the tool Alfabet Expand in order to configure the redirection of object selectors:

- Create a selector definition to access the external data source. This requires the configuration of an XML object ***SelectorDef***.

- Configure Alfabet to use the external object selector instead of the default object selector in the XML object ***GeneralViewMap***.

The selector definitions in XML objects ***SelectorDef*** cannot be created by the customer, but the XML objects provided by Software AG are visible and editable in Alfabet Expand. Nevertheless, it is recommended that you NOT edit selector definitions for external sources without support from Software AG Support.

To view the XML objects ***SelectorDef*** defined by Software AG Support for your external source configuration:

1) In Alfabet Expand, go to the **Presentation** tab, expand the **XML Objects** node, and then expand the **ObjectSelectors** folder.
2) Double-click the relevant selector definition. The XML object configuration opens in the center pane. The following table displays the XML elements and XML attributes of the XML object ***SelectorDef*** in order to help you to understand the predefined configuration:

XML Element (Bold) / XML Attribute	Description
ObjectSelectorDef	
`Name`	Specifies the name of the selector definition. This name is used in the XML object ***SelectorDef*** to identify the object selector.
`Pages`	For selector definitions in the scope of external source synchronization, the XML attribute `Pages` must be set to "`SimplePage`". "`Browse`" and "`FullTextSearch`" are not available for selection of data from an external source.
`ExternalSource`	Specifies the external data source from which data is provided in the selector. The external source is defined by the name of the external source configuration in the XML object ***ExternalSourceConfiguration*** that defines access to and data mapping for the external source.
Class	
`Name`	Specifies the object class the selector definition is valid for. The class is defined by the **Name** attribute of the object class in the Alfabet meta-model.

Configuring the Alfabet User Interface to Map Standard Configuration Objects to Custom Configuration Objects

The XML object ***GeneralViewMap*** allows you to map standard configuration objects (such as selectors, editors, and object class icons] to custom configuration objects. For example, you could map Alfabet object selectors to external data sources. If the data of a specific object class in the Alfabet database is synchronized with the data of an external source, the object selectors in the Alfabet interface will display the data from the external source instead of the data from the Alfabet database. For detailed information about the configuration of data synchronization with external sources, see the section *Integrating Data from External Sources* in the reference manual *System Administration*.

To view the XML object ***GeneralViewMap***:

1) Go to the ***Presentation*** tab and expand the node **XML Objects**.
2) Double-click the XML object ***GeneralViewMap***. The XML object configuration opens in the middle pane.
3) Right-click the XML object ***GeneralViewMap*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
4) Define the XML attributes, as needed. The table below displays the XML elements and XML attributes that can be edited for the XML object ***GeneralViewMap***:

XML Element (Bold) / XML Attribute	Description
AlfaViewMap	
`Name`	Displays `Map`.
MapEntry	
`Type`	Enter the type of configuration object to map. For example: `Selector`, `Editor`, `Picture`.
`Source`	Enter the technical name of the source configuration object that shall be substituted.
`Target`	Enter the technical name of the target configuration object that shall substitute the source configuration object.

5) In the toolbar, click the **Save** button to save the XML definition.

Configuring the Visibility of Tabs in Alfabet Expand

You must ensure that Alfabet Expand is only accessible to those individuals in your company that make up the core team of solution designers who are authorized to make changes to the meta-model. Access to Alfabet Expand as well as the availability of specific functionalities in Alfabet Expand is controlled on a per-user basis. Each user must be explicitly specified to have access permission to Alfabet Expand as well as to the individual tabs and menus providing the various functionalities. This ensures that access to the functionalities in Alfabet Expand is explicitly differentiated based on the configuration tasks and responsibilities of the individual user.

The definition of user access permissions to Alfabet Expand is defined by the user administrator in the context of the **Users Administration** functionality that can be accessed in the user interface via an administrative user profile. For more information about specifying access permission for Alfabet Expand, see the section *Creating a User* in the reference manual *User and Solution Administration*.

Please note the following:

- Your licensing agreement with Software AG for Alfabet Expand functionalities will not impact which functionalities are listed in the **Expand Access Options** attribute in the **User** editor. All functionalities are displayed regardless of the content of license agreement. However, regardless the definition of **Expand Access Options** attribute, the licensing agreement will override. The definition of the **Expand Access Options** attribute specifies the visibility in both Alfabet Expand Web and Alfabet Expand Windows.
- If the XML object ***PlatformConfiguration*** is specified in Alfabet Expand, it will override the definition of the **Expand Access Options** attribute in the **User** editor in Alfabet Expand Windows.

To the XML object ***PlatformConfiguration***:

1) Go to the **Presentation** tab, expand the **XML Objects** folder, and expand the **Administration** folder.
2) Right-click the XML object ***PlatformConfiguration*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) Define the following XML attributes:

- `ShowProfileMenu`: Set to "`true`" to display the **Change User Profile** option in the < **Alfabet User** r menu in the user interface Alfabet.
- `ShowProfileSelfAdminMenu`: Set to "`true`" to display the **Assign User Profile** option in the < **Alfabet User** r menu in the user interface Alfabet.
- `ShowPersonalInfoMenu`: Set to "`true`" to display the **Personal Info** option in the < **Alfabet User** r menu in the user interface Alfabet.
- `ShowUserSettingsMenu`: Set to "`true`" to display the **User Settings** option in the < **Alfabet User** > menu in the user interface Alfabet.
- `Expand_ClassModeel`: Set to "`true`" if users shall be allowed to access the **ClassModel** tab n the configuration tool Alfabet Expand.
- `Expand_Admin`: Set to "`true`" if users shall be allowed to access the **Admin** tab n the configuration tool Alfabet Expand.

- `Expand_Functions`: Set to "`true`" if users shall be allowed to access the **Functions** tab n the configuration tool Alfabet Expand.
- `Expand_Reports`: Set to "`true`" if users shall be allowed to access the **Reports** tab n the configuration tool Alfabet Expand.
- `Expand_Workflows`: Set to "`true`" if users shall be allowed to access the **Workflows** tab n the configuration tool Alfabet Expand.
- `Expand_ADIF`: Set to "`true`" if users shall be allowed to access the **ADIF** tab n the configuration tool Alfabet Expand.
- `Expand_Diagram`: Set to "`true`" if users shall be allowed to access the **Diagrams** tab in the configuration tool Alfabet Expand.

4) n the toolbar, click the **Save**  button to save the XML definition. You must completely close Alfabet Expand to activate any changes you have made to the XML object ***PlatformConfiguration***.

Chapter 3: Configuring Access Permissions for Alfabet

Alfabet offers various levels of access control:

- **Access permission for the Alfabet interface**

The access to the Alfabet interface requires login either by means of a user name and password, or by means of a Windows Sign-On mechanism. Windows Sign-On uses the user's Microsoft Windows domain login information as the Alfabet login and opens the interface automatically with the settings defined for the user without requiring the manual input of a user name and password.

It is also possible to configure the Interface with external login mechanisms for, for example, the login to a company portal or login triggered by data from an external LDAP server.

The login mechanism and additional login conditions such as an expiration period for user passwords can be specified in the configuration of the Alfabet Server and the Alfabet Windows Client with the tool Alfabet Administrator.

Which users are allowed to log in to Alfabet must be specified in the **User Administration** functionality that can be accessed in the Alfabet user interface via the `Admin` user profile.

For a detailed description of how to configure login to Alfabet, see the section *Configuring User Authentication* in the reference manual *System Administration*.

- **Access permission for Alfabet functionalities**

Alfabet allows the solution designer to configure the range of functionality that is visible to a user accessing the Alfabet interface. Functionalities are bundled in user profiles and the user profiles are assigned to Alfabet users. When a user logs in to Alfabet, the user profile that he/she logs in with will determine which functionalities are available. A user profile can be assigned to multiple users and one user can have multiple user profiles assigned, allowing him/her to work with different functionalities in different contexts.

For example, a user profile Data Maintenance displays an explorer with all applications that the user is responsible for. The functionality allows the user to view and edit data for the applications. Another user profile Project Planning bundles project planning functionalities and allows the user to create projects, define the architecture relevant for a project, and edit project data.

For a detailed description about configuring user profiles, see the section Configuring User Profiles for the User Community.

- **Access permissions for objects in the Alfabet database**

It is possible that a user logged in to the Alfabet interface may not be able to edit all objects in the database. Access permissions can be granted per object or per object class, depending on the definition of the various mechanisms available to control user access.

This chapter describes the mechanisms to configure access to object data in the Alfabet database and the configuration steps required to activate and customize access permissions. Access permissions can be controlled in different ways, depending on your enterprise's security policies and requirements. Access to objects can be controlled on the level of object classes or on the level of individual objects within object classes. It is possible to hide objects completely from the Alfabet interface or to restrict access permissions to ReadOnly access. Alfabet allows your enterprise to use multiple mechanisms in parallel to control access permissions. A predetermined hierarchy of access concepts specifies which access configuration has

precedence. An overview of the available mechanisms and the order of their execution is provided in order to help you configure access permissions to objects.

The following information is available about the configuration of access permissions to access objects:

- Overview of Access Permissions for Objects
- Access Modes for Objects
- Overview of Access Permission Concepts
- Rules Governing Access Permissions
- *Implementing the Mandate Capability for a Federated Architecture*
- *Creating Mandates for Your Enterprise*
- *Activating the Mandate Capability for the Enterprise*
- *Activating the Mandate Capability for Relevant Object Classes and Object Class Stereotypes*
- *Assigning Mandates to Users*
- Configuring Master Users and Master User Groups
- Configuring Permission Rules for Access to Objects
- Activating the Mandate Capability for the Enterprise

Overview of Access Permissions for Objects

This section provides an overview of the mechanisms available to control access to objects in the Alfabet database and the required configuration steps to implement the individual mechanisms. This overview is followed by an explanation of the interactions between the mechanisms.

The following overview distinguishes between access permissions to hide objects from view and access permissions that disable the editability of objects.

- Access Modes for Objects
- Overview of Access Permission Concepts
- Access Permissions for a User Profile
- Access Permissions in a Federated Architecture
- Access Permissions for Master Users
- Rule-Based Access Permissions
- Access Permissions for Authorized Users
- Access Permissions for Authorized User Groups
- Access Permissions for Deputies
- Access Permissions for Members of Discussion Groups

- Access Permissions for Users Responsible for Assignments
- Access Permissions for Users Responsible for a Workflow Step
- Access Permissions Based on an Object's Release Status
- Purpose of Roles
- Rules Governing Access Permissions

Access Modes for Objects

Access permissions in Alfabet can define whether data about an object:

- is not displayed at all
- is displayed but cannot be edited
- is displayed and can be edited.

Access Concepts Hiding Objects

By means of a federated architecture, it is possible to hide individual objects in an object class while other objects in the same object class are displayed in the Alfabet interface. For more information, see the section *Implementing the Mandate Capability for a Federated Architecture*.

Access Concepts Granting Read/Write Access

By means of user profiles, it is possible to define basic access permissions regarding visibility and editability for objects in Alfabet.

The  symbol is displayed in the object profile of an object that cannot be edited by the current user. If the user profile grants ReadOnly access permissions, the user can only view the visible objects when logging in to Alfabet. If the user profile grants Read/Write access permissions, the user can edit the objects if one of the following permission concepts applies:

- A configured rule-based access permission finds the object. For more information, see the section Rule-Based Access Permissions.
- A configured default permission rule finds the object. For more information, see the section Rule-Based Access Permissions.
- The user is the authorized user of the object. For more information, see the section Access Permissions for Authorized Users.
- The user is a member of an authorized user group defined for the object. For more information, see the section Access Permissions for Authorized User Groups.
- The user is a deputy of the object. For more information, see the section Access Permissions for Deputies.
- The user is a member of a discussion group about the object. For more information, see the section Access Permissions for Members of Discussion Groups.
- The user is responsible to process an assignment for the objects. For more information, see the section Access Permissions for Users Responsible for a Workflow Step.

- The user is responsible to perform a workflow activity for the object. For more information, see the section Access Permissions for Users Responsible for a Workflow Step.

AND

- The release status grants editing of the object. For more information, see the section Access Permissions Based on an Object's Release Status.

The various permission concepts available for Alfabet are evaluated in a given order and some permission concepts are configured to exclude other permission concepts from the evaluation of access permissions. The execution order of access permission concepts is described in the section Rules Governing Access Permissions.

If the user profile is marked as administrative user profile, the user has Read/Write access permissions to all objects that he/she can access in the functionalities available in the user profile. The visibility implemented by the federated architecture concept via mandates also applies in administrative user profiles.

Overview of Access Permission Concepts

This section describes the possibilities available in Alfabet to control access to objects. An overview of the configuration steps required is included in the description. The following information is available:

- Access Permissions for a User Profile
- Access Permissions in a Federated Architecture
- Access Permissions for Master Users
- Rule-Based Access Permissions
- Access Permissions for Authorized Users
- Access Permissions for Authorized User Groups
- Access Permissions for Deputies
- Access Permissions for Members of Discussion Groups
- Access Permissions for Users Responsible for Assignments
- Access Permissions for Users Responsible for a Workflow Step
- Access Permissions Based on an Object's Release Status
- Purpose of Roles

Access Permissions for a User Profile

User profiles are the basis of user administration in Alfabet. The user profile determines the functionalities available to a user. The user profile is the entry point when accessing Alfabet and every user must log in with a user profile that has been assigned to him/her, for example by a user administrator. Therefore, all users accessing Alfabet must be assigned at least one user profile. However, users may possess multiple user profiles in accordance with their responsibilities in the Alfabet user community and in the enterprise as a whole. A user can switch to another user profile at any point during a Alfabet session.

User profiles can be configured to grant either ReadOnly access permissions or Read/Write access permissions.

If the user profile is a ReadOnly profile:

- The user can view information about all objects that are available in the scope of the functionalities assigned to the user profile.
- The user may not edit or delete objects.
- Individual objects may be excluded from view if the mandate capability to manage a federated architecture has been implemented.
- The user will not be able to edit objects in the context of a workflow.

If the user profile is a Read/Write profile:

- The editability of an object will depend on all other access permission concepts described below. Therefore, some objects may be displayed in ReadOnly mode while the user may be able to edit or delete other objects. For an overview of the various concepts governing access, see the section Access Modes for Objects.

If the user profile is marked as administrative user profile, the user has Read/Write access permissions to all objects that he/she can access in the functionalities available in the user profile. The visibility implemented by the federated architecture concept via mandates also applies in administrative user profiles.

Access Permissions in a Federated Architecture

Some enterprises have a federated architecture. If this is the case in your enterprise, then mandates may be implemented to control the visibility of Alfabet objects.

A mandate is a means to organize and structure the federated architecture of a holding company. The assignment of mandates to objects allows the holding company to structure the objects in the enterprise architecture in order to regulate visibility to objects across some or all subsidiaries. Only users explicitly assigned to a mandate will see objects with that mandate definition. An object that has not been assigned to a mandate is considered not to be owned by a mandate and is thus visible throughout the holding company to users with relevant access permissions.

The use of mandates in the Alfabet solution is optional. If mandates are implemented in an enterprise, the visibility of an object with a mandate assignment will take precedence over other concepts of access permissions in Alfabet. For example, an authorized user of an object must be assigned the relevant mandate to access the object that he/she is the owner of.

Within the context of mandates, the conventional rules governing access permissions apply. Thus, a user assigned to a mandate will only have Read/Write access permission to the objects made visible by the mandate if he/she has authorized access to the object as an authorized user, deputy, member of an authorized user group or discussion group or via rule-based access permissions, workflow contributor or assignee of an assignment.

Keep the following in mind when working with mandates:

- The object classes for which mandates are available are preconfigured by Software AG. For more information about whether mandates are available for a particular object class, see *Overview of*

Configurable Features for Object Classes in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

- A user will typically have only one mandate associated with his/her user name. When the user is logged in with a mandate, he/she will have access only to those objects that are assigned to that mandate as well as to all objects that are not assigned to a mandate. In other words, the user will not see objects that have a different mandate assigned.
- Objects that do not have a mandate assigned are visible to anyone with relevant access permissions.
- If an object has a mandate assignment different than the user's mandate assignment, the user will still be able to view basic preview information as well as the reference information about the dependent objects associated with the object.
- If objects at the root level in an explorer structure are controlled by a mandate, then they will only be visible to users logged in with that mandate. Any root object in the explorer structure that has a different mandate than the mandate that the user is logged in with will not be visible in the explorer. If the user has accessibility to an object at the root level in the explorer structure, then all subordinate objects will be displayed in the explorer even if the subordinate objects have a different mandate than the mandate assigned to the user. However, the user will only see the basic preview information for such objects if he/she attempts to select the object.
- Dependent objects that are not controlled by a mandate are accessible to anyone that has access permission to the ascendant object. For example, a business support is a dependent object associated with an application, organization/market product, or business process/domain. Therefore, the business support can be viewed and possibly edited by anyone with a mandate owning the associated application, organization/market product, or business process/domain. Likewise, an information flow can be viewed and possibly edited by anyone with a mandate owning the source or target application.
- When a user with a mandate creates an object, the object is automatically assigned to the user's mandate.

  Please note that the assignment of mandates to information flows deviates from this rule. Mandates for information flows are implicitly derived from the source and target applications/components that they connect. Information flows are assigned to all mandates that the source and target objects are assigned to.

- The mandate owning an object may change over the lifecycle of an object. For example, an application may be owned by one mandate during the pilot phase, but another mandate may become the owning mandate once the application is in a productive status.
- If a user receives an assignment associated with an object owned by a mandate different than the mandate that the user is logged in with, then, in order to view or edit the object, the user must re-login with the mandate that matches the mandate owning the object.

The following steps are necessary in order to implement the mandate capability:

- Mandates must be created.
- The mandate capability must be generally activated.
- The mandate capability must be activated for each object class that will be included in the federated architecture.

These procedures are carried out in Alfabet Expand. For more information, see the section *Implementing the Mandate Capability for a Federated Architecture*.

- Each user must be assigned a mandate. Mandates can be assigned to users in the **User Management** functionality accessible via the `Admin` user profile. For more information, see either the section *Assigning a Mandate to a User* in the chapter *Defining and Managing Users* in the reference manual *User and Solution Administration*.
- When configuring the interface for the user community, you should consider whether the *Mandates Page View* (`ObjectMandates`) must be accessible to the relevant user profile configuration. When an object is created, it will automatically inherit the mandate of the user creating the object. The mandate assignment for an object can later be changed in the *Mandates Page View* for the relevant object.

Access Permissions for Master Users

A master user is a named user specified to have access permissions to all Alfabet objects regardless of the access permissions defined for the object. Likewise, a master user group can also be specified so that all users in the master user group have access permissions to all objects regardless of the access permissions defined for the object. Please note the following:

- The access permissions defined for a master user or users in the master user group will have precedence over access established via the authorized user and authorized user group definition as well as query-based access permission rules.
- The access permissions defined for a master user or users in the master user group do not have precedence over access denied due to a non-editable release status or mandate definitions.

Master users and master user groups are defined in the XML object ***RightsManager*** in Alfabet Expand. For more information, see the section Configuring Master Users and Master User Groups.

Rule-Based Access Permissions

Rule-based access permissions are a form of permission rules based on an Alfabet query or native SQL query that finds an object. Read/Write access to objects as well as the permission to delete objects can be granted by query-based permission rules.

For example, a permission rule can define that a user has Read/Write access to all information flows that have a source or target application for which he/she is the authorized user.

Or

A permission rule can define that a user has Read/Write access to all ICT objects for which he/she has the role Architect defined.

There are two types of permission rules that differ in execution mode and time:

- **Global permission rules** are executed when the user logs in to Alfabet and on first access of an object of a defined class. The evaluation is then repeated in regular intervals. The user has Read/Write access permissions for all objects returned by the query defined for the rule.

Additionally, default permission rules can be defined. Default permission rules can specify Read/Write or ReadOnly access permissions. Default permission rules are not based on an Alfabet query or native SQL query but rather are explicitly defined for an object class. If a default permission rule is defined for an object class and an object of the object class is not found via a query-based permission rule of the type `Global`, the access permission defined in the default permission rule will apply.

If no default permission rule is specified for the object class and an object of the object class is not found via a query-based permission rule, access permissions will be regulated via the concept of authorized user/authorized user groups.

- **Local permission rules** are executed whenever the user accesses an object. The resulting dataset from the query defined for the rule is executed in combination with the XML attribute `ResultType` which can be either `Positive` or `Negative`. When the query returns a dataset with at least one result and the XML attribute `ResultType` is `Positive`, or when the query returns an empty dataset and the XML attribute `ResultType` is `Negative`, Read/Write access permissions will be granted. If the conditions for granting the defined access permissions are not met, the XML attribute `ExecutionType` of the permission rule defines the further processing of the access conditions. If theXML attribute `ExecutionType` is set to `Exclusive`, the user will have ReadOnly access to the object. If the XML attribute `ExecutionType` is set to `Standard`, the access permission concepts defined in addition to the local permission rule (such as conventional access permissions) will be evaluated.

Global permission rules can be applied to one object class and local permission rules can be applied to a different object class, but only one type of permission rule can be defined per object class.

Permission rules are defined in the XML object ***RightsManager*** in Alfabet Expand. For more information, see the section Configuring Permission Rules for Access to Objects.

Access Permissions for Authorized Users

When an object is created, the creator of the object is automatically designated the authorized user. The authorized user is assigned to an object in the **Authorized Access** tab in the respective object's editor. The authorized user has Read/Write access to the object. After an object is created, the authorized user may be changed at any time. Every object can have at most one authorized user. However, it is also possible that no authorized user is defined for an object.

Access permissions for authorized users, deputies and authorized user groups are only valid if:

- global permission rules are defined for the object class, whereby no default permission rule is defined and the object is not included in the dataset returned by any of the global permission rules.

or

- the local permission rules defined for the object class do not grant Read/Write access permission to the object and the XML attribute `ExecutionType` of the local permission rule is set to `Standard`.

If permission rules are defined that do not include the authorized user as a permitted user, the authorized user may only edit the object in the context of the user session in which the object is being created. Once the user session ends, the authorized user will no longer be able to edit the object.

To guarantee that authorized users, deputies and authorized user groups can access objects they are responsible for even if permission rules are specified, a permission rule should be specified for the relevant object class that specifies Read/Write access permissions for users that are authorized user, deputy or member of an authorized user group for the object.

Please note that some functionalities use explorers that are based on the concept of authorized access and therefore display only objects that the current user is the authorized user of or objects that are available to the current user via an authorized user group affiliation. Such explorer nodes are typically named "My <ObjectClassName>" (For example, the **Document Application** functionality has an explorer with the caption **My Applications**). Because a permission rule may be configured so that a user may not have ReadWrite access permission to objects for which he/she is the authorized user, it is recommended that these functionalities are not implemented when permission rules are specified.

You should carefully consider whether the **Authorized User Objects** page view (`USER_PersonalItems`) in the **Personal Info** functionality, which displays all objects that a user is responsible for, should be included in the user profile configuration.

Access Permissions for Authorized User Groups

A user can also be assigned authorized access to an object via his/her membership in a user group if this user group is defined as the authorized user group for the object. In this case, all members of the user group will automatically have Read/Write access to the object. The authorized user group is assigned to an object in the **Authorized Access** tab in the respective object's editor.

The solution designer can also configure rules for the inheritance and/or propagation of a user group's access permissions to all ascendant and/or descendant user groups in the XML object ***RightsManager***. The following can be configured regarding inheritance and propagation:

- Propagation: All user groups in the user group hierarchy that are ascendant to a user group with permissions to an object should also have the same permissions to that object.
- Inheritance: All user groups in the user group hierarchy that are descendant to a user group with permissions to an object should also have the same permissions to that object.

Access permissions for authorized users, deputies and authorized user groups are only valid if:

- global permission rules are defined for the object class, whereby no default permission rule is defined and the object is not included in the dataset returned by any of the global permission rules.

or

- the local permission rules defined for the object class do not grant Read/Write access permission to the object and the XML attribute `ExecutionType` of the local permission rule is set to `Standard`.

If permission rules are defined that do not include the authorized user as a permitted user, the authorized user may only edit the object in the context of the user session in which the object is

being created. Once the user session ends, the authorized user will no longer be able to edit the object.

To guarantee that authorized users, deputies and authorized user groups can access objects they are responsible for even if permission rules are specified, a permission rule should be specified for the relevant object class that specifies Read/Write access permissions for users that are authorized user, deputy or member of an authorized user group for the object.

Please note that some functionalities use explorers that are based on the concept of authorized access and therefore display only objects that the current user is the authorized user of or objects that are available to the current user via an authorized user group affiliation. Such explorer nodes are typically named "My <ObjectClassName>" (For example, the **Document Application** functionality has an explorer with the caption **My Applications**). Because a permission rule may be configured so that a user may not have ReadWrite access permission to objects for which he/she is the authorized user, it is recommended that these functionalities are not implemented when permission rules are specified.

The following configuration is necessary in order to implement the authorized user group concept:

- User groups must be created in the **User Group Administration** functionality that can be accessed in the Alfabet user interface via the `Admin` user profile and users must be assigned to the user group. For more information, see the section *Defining and Managing User Groups* in the reference manual *User and Solution Administration*.
- Rules of inheritance and/or propagation can be defined in the XML object ***RightsManager*** in Alfabet Expand. For more information, see the section Configuring the Propagation/Inheritance of User Group Rights.
- You should consider whether the *Authorized User Groups Page View* (`ObjectUserGrousOnly`), which allows authorized user groups to be assigned to an object, should be included in the **Personal Info** functionality in the user profile configuration.

Access Permissions for Deputies

The authorized user of an object can assign one or more deputies to an object. Each deputy has Read/Write access permissions to the object and thus can capture, edit, and delete the data for the object when the authorized user is unable to.

Access permissions for authorized users, deputies and authorized user groups are only valid if:

- global permission rules are defined for the object class, whereby no default permission rule is defined and the object is not included in the dataset returned by any of the global permission rules.

or

- the local permission rules defined for the object class do not grant Read/Write access permission to the object and the XML attribute `ExecutionType` of the local permission rule is set to `Standard`.

If permission rules are defined that do not include the authorized user as a permitted user, the authorized user may only edit the object in the context of the user session in which the object is

being created. Once the user session ends, the authorized user will no longer be able to edit the object.

To guarantee that authorized users, deputies, and authorized user groups can access objects they are responsible for even if permission rules are specified, a permission rule should be specified for the relevant object class that specifies Read/Write access permissions for users that are authorized user, deputy or member of an authorized user group for the object.

Please note that some functionalities use explorers that are based on the concept of authorized access and therefore display only objects that the current user is the authorized user of or objects that are available to the current user via an authorized user group affiliation. Such explorer nodes are typically named "My <ObjectClassName>" (For example, the **Document Application** functionality has an explorer with the caption **My Applications**). Because a permission rule may be configured so that a user may not have ReadWrite access permission to objects for which he/she is the authorized user, it is recommended that these functionalities are not implemented when permission rules are specified.

When configuring the interface for the user community, you must take into account that the following functionalities may be necessary and must therefore be accessible to users:

- The **Authorized Deputies** page view (`ObjectDeputiesOnly`) should be available in the object views where users with Read/Write access permissions to an object may define a deputy for respective object.
- You should consider whether the **Deputy Objects** page view (`USER_DeputyItems`), which allows persons to be assigned as deputes to an authorized user's object, should be included in the **Personal Info** functionality in the user profile configuration.

Access Permissions for Members of Discussion Groups

Alfabet provides a discussion functionality that supports collaborative discussions about an object by discussion groups that have been configured for your enterprise. If Read/Write access has been configured for a discussion group, the members of that discussion group will also be able to access and edit the object that is under discussion.

Discussion groups and the access permissions of their members are configured in the **Discussion Groups** functionality that can be accessed in the Alfabet user interface via the `Admin` user profile. For more information, see the section *Defining Discussion Groups for Collaborative Discussions* in the reference manual *User and Solution Administration*.

When configuring the interface for the user community, you must take into account that the following functionalities may be necessary and must therefore be accessible to users:

- The **My Discussions** (`DSC_PersonDiscussions`) functionality should be available so that members of a discussion group can access the objects that are under discussion. For a detailed description about the discussion process, see the section *Initiating, Contributing to, and Managing a Discussion About Objects* in the reference manual *Getting Started with Alfabet*.
- The **Discussions** page view (`DSC_Items`) should be available in the object views where users can initiate a discussion about an object as well as contribute to and track the discussion about that object.

Access Permissions for Users Responsible for Assignments

Alfabet provides an assignment capability that allows users to define tasks for objects and assign the tasks to users that are responsible to perform that task.

Any user that has been assigned an assignment associated with an object has Read/Write access to that object for the period of the assignment.

When configuring the interface for the user community, you must take into account that the following functionalities may be necessary and must therefore be accessible to users:

- The **Assignments** page view (`ObjectAssignments`) should be available int he object views where users with edit permissions to an object can create assignments for the object.
- The **My Assignments** (`Home_Assignments`) functionality should be available so that a user can see all assignments that he/she is responsible for.
- The **Sent Assignments** (`Home_SentAssignments`) functionality should be available so that a user can see all assignments that he/she has created. For a detailed description about the assignment process, see the section *Sending and Receiving Assignments for Alfabet Objects* in the reference manual *Getting Started with Alfabet*.

Access Permissions for Users Responsible for a Workflow Step

A workflow is a collaborative process of workflow steps that are typically carried out by one or more users. One or more user(s) that are currently responsible for a workflow step associated with an object will have Read/Write access to that object in the context of the workflow step for as long as he/she is responsible for the workflow step.

If an object targeted has ReadOnly access permissions, the user(s) responsible for performing the workflow step will nevertheless have Read/Write permissions to the targeted object. All other users (including authorized users) will continue to have ReadOnly access permissions to the object for the period that the workflow step is active.

For more information about the configuration required to use the workflow capability, see the chapter Configuring Workflows in this reference manual.

Access Permissions Based on an Object's Release Status

Release status definitions are customizable and can be configured for a preconfigured set of object classes and, if applicable, object class stereotypes. Typical statuses could be Draft, Approved, Retired, etc. The release status is assigned to an object via the object editor when creating or editing the object or via the **Change Status** button in the toolbar of the object profile.

As part of the configuration of release statuses, it is possible to specify which release statues are non-editable. For all objects with release statuses that are configured as non-editable, users will neither be able to edit the attributes in the object's editor nor edit the object's relationships via the page views available in the object view.

Release status definitions are configured in the XML object ***ReleaseStatusDefs*** in Alfabet Expand. For information about configuring release statuses for object classes, see the section Configuring Release Status Definitions for Object Classes in the chapter Configuring the Class Model.

Purpose of Roles

In Alfabet, you may associate roles with a user or organization. A role describes a functional relationship that the user or organization has to an object (for example, as a Controller, Stakeholder, or Manager). The definition of a role does not affect access permissions in any way unless specifically implemented through a configured access permission rule. Roles are for the purpose of documentation and analysis only and do not have any impact on the access permissions associated with the object.

- It is possible to grant Read/Write access permissions to users with a specific role for the object by defining an access permission rule. For more information, see Configuring Permission Rules for Access to Objects.
- Roles are based on configured role types. Role types must first be created in the **Reference Data** functionality and assigned to the relevant object class in the **Class Configuration** module. For more information, see the chapter *Configuring Role Types to Define Roles in the Responsibilities Page View* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.

Rules Governing Access Permissions

Access permissions defined for objects are evaluated in a specified order.

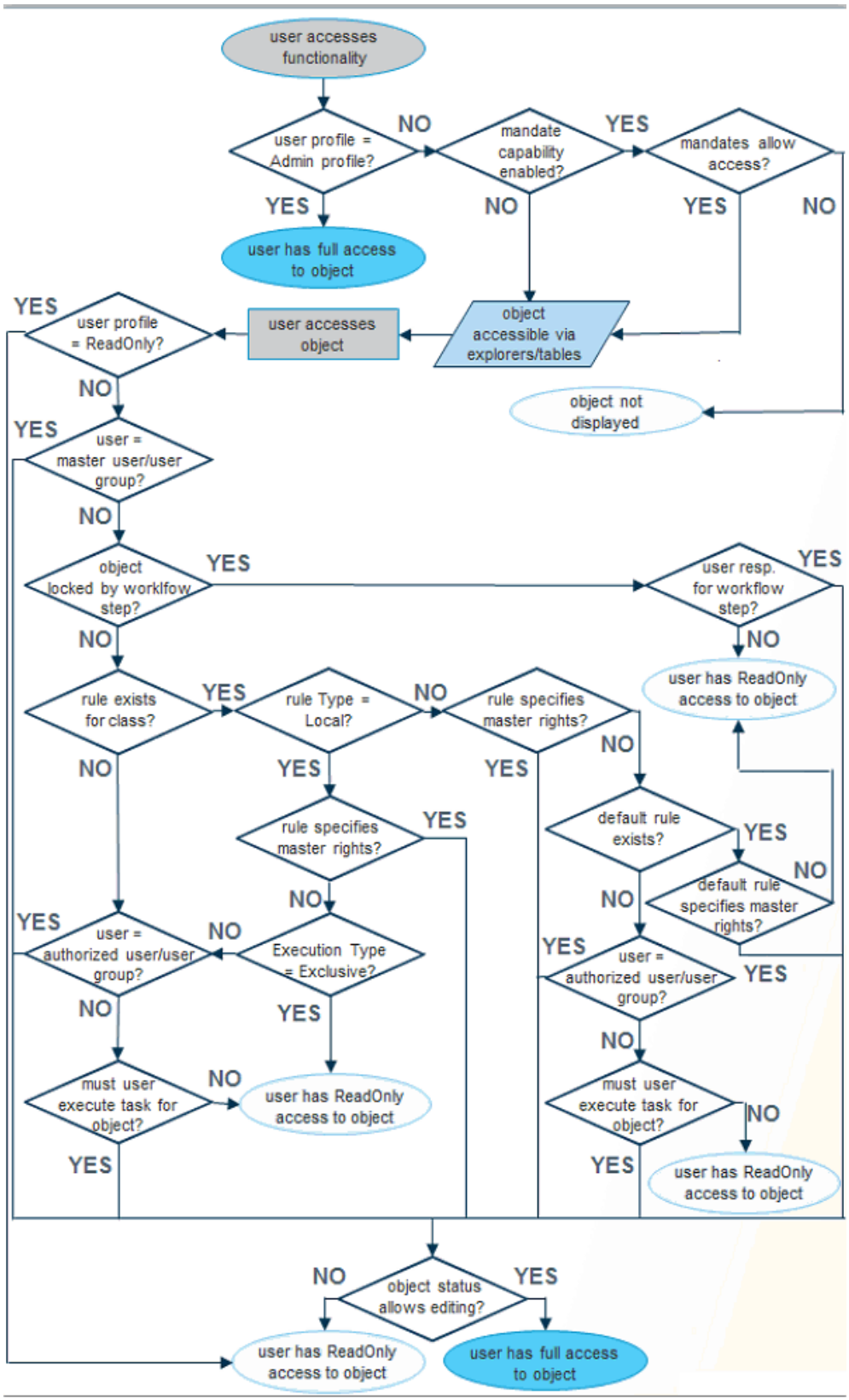


FIGURE: Order of evaluation of access permissions

- Access permissions are not evaluated at all if the user logs in with the `Admin` user profile. Users logged in with the `Admin` user profile can edit any object in the Alfabet database independent of the access permission concepts described below.
- Access permissions governing general visibility of objects take precedence over all other access permission concepts.

If your enterprise has a federated architecture and implements mandates, then users may potentially view and access only those objects that are owned by a mandate that matches the mandate the user is currently logged in with. Other objects will not be visible to the user in explorers and views. However, a preview of an object with a different mandate will be displayed if the object is referenced by an object accessible to the user's mandate.

An object that is not owned by a mandate is visible to anyone, regardless of the mandate assignment.

- For visible objects, the editability of objects is then evaluated as follows:

- If the user profile is a ReadOnly user profile, the user cannot edit any object regardless of the access permissions granted by other access permission concepts.
- If the user is a master user, Read/Write access permissions are granted regardless of the following access concepts if the release status is configured to grant Read/Write access.
- If the object is targeted by an active workflow step that restricts editing of the target object, only the user responsible for performing the workflow step may edit the object. All other users including authorized users will have ReadOnly access permissions to the object for the period that the workflow step is active. This ensures that other users not involved in the current workflow step do not edit the data of the object that is being currently processed in the workflow step.
- If the user profile is a Read/Write user profile, objects will be editable if the user has Read/Write access to the object due to a configured permission rule. A configured permission rule is a query-based rule specified by your enterprise to define the conditions to find the objects that a user may access with Read/Write access conditions.
- Permission rules have not been specified for the object class stereotype or object class.
- If global permission rules are defined for the object class and no default permission rule is defined for the object class, and the object does not match any of the defined permission rules.
- If local permission rules are defined and the XML attribute `ExecutionType` of the last defined rule for the object class is set to Standard.
- The authorized user and members of the relevant authorized user groups can edit the object if the permission rule configuration allows evaluation of other access permission concepts.
- If the user is responsible to perform a task for that object in the context of an assignment, workflow step, or discussion, the user will be able to edit the object as long as he/she is the responsible for the task. The user will be able to edit the object even if no access permissions are granted via the authorized user concept. Configured permission rules supersede the concept of task-based access permissions.
- If the object has a release status that is configured as a non-editable status, the user will have ReadOnly access permissions to the object regardless of the access permissions described above.

For some objects, special conditions apply independent of the general permission concepts:

- When a user creates an object, he/she has access permissions to that object during the active session, even if a configured permission rule based on attributes of the object does not allow the user to edit the object. The same applies when the user changes the properties of the object that define access permissions (for example, the object's mandate setting or authorized use is changed in such a way that would prevent him from seeing or editing the object). Changes to the access permissions are only taken into account for new sessions.
- Although you may have Read/Write permissions to objects that are considered reference data (for example, cost types, user groups, evaluation types), these objects can only be edited in the context of the **Configuration** module or, as the case may be, by accessing Alfabet with the `Admin` user profile.
- In a few cases, access permissions are not directly defined for objects that belong to a few object classes (for example, object classes like Master Plan Folder). These object classes automatically inherit the access information from their ascendant object (for example, the master plan folder inherits the access information from the ascendant master plan).

For an overview of which object classes inherit permission permissions, see *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

Implementing the Mandate Capability for a Federated Architecture

Some enterprises have a federated architecture. If this is the case in your enterprise, mandates may be implemented to control the visibility of Alfabet objects.

A mandate is a means to organize and structure the federated architecture of a holding company. The assignment of mandates to objects allows the holding company to structure the objects in the enterprise architecture in order to regulate visibility to objects across some or all subsidiaries. Only users explicitly assigned to a mandate will see objects with that mandate definition. An object that has not been assigned to a mandate is considered not to be owned by a mandate and is thus visible throughout the holding company to users with relevant access permissions.

The use of mandates in the Alfabet solution is optional. If mandates are implemented in an enterprise, the visibility of an object with a mandate assignment will take precedence over other concepts of access permissions in Alfabet. For example, an authorized user of an object must be assigned the relevant mandate to access the object that he/she is the owner of.

Within the context of mandates, the conventional rules governing access permissions apply. Thus, a user assigned to a mandate will only have Read/Write access permission to the objects made visible by the mandate if he/she has authorized access to the object as an authorized user, deputy, member of an authorized user group or discussion group or via rule-based access permissions, workflow contributor or assignee of an assignment.

For information about which object classes support the mandate capability, see *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

Keep the following in mind when working with mandates:

- A user will typically have only one mandate associated with his/her user name. When the user is logged in with a mandate, he/she will have access only to those objects that are assigned to that mandate as well as to all objects that are not assigned to a mandate. In other words, the user will not see objects that have a different mandate assigned.
- Objects that do not have a mandate assigned are visible to anyone with relevant access permissions.
- If an object has a mandate assignment different than the user's mandate assignment, the user will still be able to view basic preview information as well as the reference information about the dependent objects associated with the object. However, any custom object class properties defined for the object are not visible in the object's preview or object profile to the user with the mandate different than the object's mandate.
- If objects at the root level in an explorer structure are controlled by a mandate, then they will only be visible to users logged in with that mandate. Any root object in the explorer structure that has a different mandate than the mandate that the user is logged in with will not be visible in the explorer. If the user has accessibility to an object at the root level in the explorer structure, then all subordinate objects will be displayed in the explorer even if the subordinate objects have a different mandate than the mandate assigned to the user. However, the user will only see the basic preview information for such objects if he/she attempts to select the object.
- Dependent objects that are not controlled by a mandate are accessible to anyone that has access permission to the ascendant object. For example, a business support is a dependent object associated with an application, organization/market product, or business process/domain. Therefore, the business support can be viewed and possibly edited by anyone with a mandate owning the associated application, organization/market product, or business process/domain. Likewise, an information flow can be viewed and possibly edited by anyone with a mandate owning the source or target application.
- When a user with a mandate creates an object, the object is automatically assigned to the user's mandate.

  Please note that the assignment of mandates to information flows deviates from this rule. Mandates for information flows are implicitly derived from the source and target applications/components that they connect. Information flows are assigned to all mandates that the source and target objects are assigned to.

- When a user switches to another mandate, the access permission of the new mandate is immediately applied, and the user is redirected to the start page.
- The mandate owning an object may change over the lifecycle of an object. For example, an application may be owned by one mandate during the pilot phase, but another mandate may become the owning mandate once the application is in a productive status.
- If a user receives an assignment associated with an object owned by a mandate different than the mandate that the user is logged in with, then, in order to view or edit the object, the user must re-login with the mandate that matches the mandate owning the object.
- The object classes for which mandates are available are configured by Software AG. For an overview of the object classes that support the mandate capability, see *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

Mandates are ignored in the context of workflows. Regardless of the mandate assignments to the user and the object managed in a workflow step, the user assigned responsibility for a workflow step has all access permissions required to execute the workflow step within the context of the

My Workflow Activities functionalities (`WF_UserWorkflowActivities`, `WF_UserWorkflowActivitiesExt`, and `WF_UserWorkflowActivitiesCommon`) and/or the **Workflow Activities Explorer** (`WFS_Explorer`).

Therefore, if only users with a specific mandate should be permitted to perform a workflow step, the workflow designer must ensure that mandate assignment is taken into consideration via a query configured for the workflow step.

The following steps are required to implement the mandate capability:

- You must first create mandates. For more information, see *Creating Mandates for Your Enterprise*.
- You must activate the mandate capability in order to implement it. The mandate capability must be activated in the XML object ***RightsManager*** in order for the activation on the level of the object class to take effect. For more information, see *Activating the Mandate Capability for the Enterprise*
- You must ensure that the mandate capability is activated for each relevant object class. For more information, see *Activating the Mandate Capability for Relevant Object Classes and Object Class Stereotypes*.
- If the object class has object class stereotypes defined, you must configure the mandate capability for the object class stereotypes. For more information, see the section *Activating the Mandate Capability for Relevant Object Classes and Object Class Stereotypes*. For the special case of the implementation of the mandate capability for a domain model, see the section Implementing Mandates for the Domain Model.
- Assign the mandates to users. For more information, see *Assigning Mandates to Users*.

Creating Mandates for Your Enterprise

An enterprise may configure up to 30 mandates. The implementation of the mandate capability and the definition of mandates should only be done with the support of a consultant from Alfabet.

To create mandates:

1) Go to the **Admin** tab and right-click the **Mandates** node  and select **Show All Mandates**. The **Mandate Management** editor opens in the center pane.
2) In the toolbar, click the **New**  button and select **Create Mandate**. A dialog box opens.
3) Define the name and, if necessary, provide a description clarifying the purpose of the mandate.
4) Click **OK** to save the mandate. The new mandate  is displayed below the **Mandates** node in the explorer.

Activating the Mandate Capability for the Enterprise

The XML object ***RightsManager*** allows you to configure whether the mandate capability should be implemented in your enterprise's Alfabet software. The XML object ***RightsManager*** allows you to activate or deactivate the mandate capability as a whole.

To implement the mandate capability, you must activate the mandate capability in the XML object ***RightsManager*** AND explicitly define the mandate capability for the relevant object classes that should be controlled in your enterprise's federated architecture as described in the section Activating the Mandate Capability for Relevant Object Classes and Object Class Stereotypes.

To edit the XML object ***RightsManager***:

1) Go to the **Presentation** tab, expand the **XML Objects** folder, and then expand the **Administration** folder by clicking the + symbol. You will see all XML objects that can be edited.
2) Right-click the XML object ***RightsManager*** and select **Edit XML...**. The XML editor is displayed in the center pane.

For general information about editing XML objects in Alfabet Expand, see Working with XML Objects. All attributes that are described in the following are permissible attributes in the XML object. If they are not automatically displayed in the XML editor with a default value when opening the XML object, you can write them to the appropriate position in the editor. The example provided with this documentation shows the correct position of the attribute within the XML code.

3) In the XML attribute `MandateMode`, enter "`true`" to activate the mandates defined for the relevant object classes. Any objects that do not have a mandate assigned are visible to any user that has relevant access permissions. Enter "`false`" if the mandate capability should not be activated. In this case, the mandate definitions will not be applied to the objects.

The following example shows an XML object ***RightsManager*** for which the XML attribute `MandateMode` is set to "`true`":

```
<?xml version="1.0" encoding="utf-8" ?>
<AlfaRightsManager
    Inheritance="false"
    Propagation="true"
    MandateMode="true"
    AssemblyName="ITPlan"
    AssemblyClass="ITPlanSolution.SolutionRightsManager"
>
<AlfaMethod Type="OnGetObjectRights" Name =
"OnGetObjectRights" />
</AlfaRightsManager>
```

4) In the toolbar, click the **Save**  button to save the XML definition.

Activating the Mandate Capability for Relevant Object Classes and Object Class Stereotypes

To implement the mandate capability, you must ensure that the mandate capability is explicitly activated for each object class that should be controlled in your enterprise's federated architecture. The implementation of the mandate capability for an object class is defined via the **Can Have Mandates** attribute available for the object class. If the object class has object class stereotypes configured, you can implement the mandate capability for one, some, or all of the object class stereotypes. The mandate configuration may differ for each object class stereotype defined for an object class via the XML attribute `HasMandates` that can be defined for the **Stereotypes** attribute for the object class.

Please note however that this is not the case for the object class `Domain`. The object class `Domain` requires that the configuration of the XML attribute `HasMandates` that can be defined for the **Stereotypes** attribute for the object class `Domain` is the same for all domain stereotypes in a domain hierarchy. For more information about the configuration of the mandate capability for the object class `Domain`, see the section Implementing Mandates for the Domain Model.

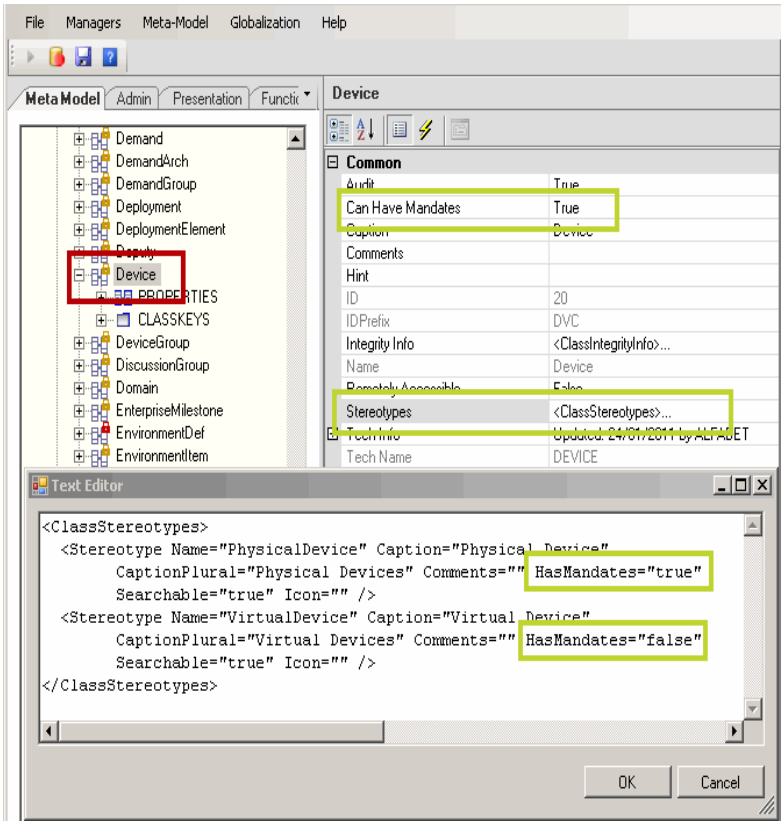
Per default, the XML attribute `HasMandates` is defined as "`false`" for all object class stereotypes. This means that the default setting specifies that objects based on a stereotype will be visible to users with any mandate implemented in your enterprise. Therefore, you must explicitly define the mandate capability for each object class stereotype if visibility should be controlled for the objects of an object class stereotype.

To implement the mandate capability for your Alfabet implementation, you must activate the mandate capability in the XML object ***RightsManager*** as described in the section Activating the Mandate Capability for the Enterprise AND explicitly define the mandate capability for the relevant object classes that should be controlled in your enterprise's federated architecture.

To make the mandate capability active for an object class:

1) Go to the **Meta-Model** tab, expand the **Class Model** folder, then expand and the **Classes** node and navigate to the relevant object class.

2) Click the relevant object class  to open the attribute window.



3) In the attribute window, select `True` in the **Can Have Mandates** field if the mandate capability should be available for the selected class. Select `False` if the mandate capability should not be available.

4) If object class stereotypes have been defined for the object class, you can explicitly define the mandates capability for each stereotype. In the **Stereotypes** field, click the **Browse** button to open the editor and specify the XML definition.

5) To implement the mandate capability for an object class stereotype, the XML attribute `HasMandates` must be defined as "`true`". If you do not want to implement the mandate capability for an object class stereotype, the XML attribute `HasMandates` must be defined as "`false`". If the XML attribute `HasMandates` is defined as "`false`" for an object class stereotype, objects based on the stereotype will be visible to permissible users with any mandate implemented in your enterprise.

6) Close the editor by clicking **OK**.

7) In the toolbar, click the **Save** button to save your changes.

Assigning Mandates to Users

Mandates can be assigned to users via the **User Management** functionality accessible in the Alfabet interface via the `Admin` user profile or via the tool Alfabet Administrator. One or more mandates can be assigned to a user although typically a user will be assigned only one mandate. The user will be able to access any objects that have been assigned to his/her mandate. The access to objects via the assignment of a mandate overrides any other assignment of access permissions.

Please note that a user can be specified as a mandate master in the **User** editor accessible via the **User Management** functionality in the Alfabet interface. The mandate master can log in to Alfabet and view any object regardless of its mandate. The visibility of objects will be based on the user profile that the mandate master is logged in with. For more information, see either the section *Assigning a Mandate to a User* in the reference manual *User and Solution Administration*.

Configuring Permission Rules for Access to Objects

The XML object ***RightsManager*** allows you to configure permission rules that define access permissions to objects in the Alfabet database.

Permission rules identify objects in the Alfabet database by means of an Alfabet query or SQL query and assign Read/Write access permissions to the user for these objects. The user can then edit and delete the object.

There are two types of permission rules that differ in the execution mode and execution time:

Permission Rule Type	Execution Time	Permission Evaluation Mode	Permission Evaluation If Rule Does not Apply
Global	The permission rule is executed when the user logs in and on first access of an object of a class. Execution is repeated in a regular interval configurable in the XML object ***RightsManager*** to consider changes performed to database objects during a user session.	A query is defined for the rule that returns a dataset of objects for that the user shall have Read/Write access permissions. The user can access all objects that are included in the result data set until the next evaluation of the permission rule.	Default permission rules can be defined per object class or object class stereotype that specify the access permissions for objects of a class or stereotype that do not apply to a query based global permission rule. If no default permission rule is defined for the object class, all other access permissions, like access for authorized users or due to an assignment for the object apply.
Local	The permission rule is executed each time a user accesses an object. Please note that this may lower the performance because queries must be executed whenever the user accesses an object.	Access permissions depend on the query defined for the permission rule and the setting of the XML attribute `ResultType`. If the query returns a dataset with at least one result and the XML attribute `ResultType` is set to `Positive`, or if the query returns an empty dataset and the XML attribute `ResultType` is set to `Negative`, Read/Write access permissions will be granted.	The XML attribute `ExecutionType` defines further processing. If the XML attribute `ExecutionType` is set to `Exclusive`, access will be ReadOnly if the rule does not apply. If the XML attribute `ExecutionType` is set to `Standard`, all other access permissions will apply (such as access for authorized users or access due to an assignment for the object).

Local and global permission rules cannot be defined in parallel for an object class. You can define local permission rules for objects of one object class and global permission rules for objects of another object class, but you must make sure that only one type of permission rules is defined per object class.

Access permissions for authorized users, deputies and authorized user groups are only valid if:

- global permission rules are defined for the object class, whereby no default permission rule is defined and the object is not included in the dataset returned by any of the global permission rules.

or

- the local permission rules defined for the object class do not grant Read/Write access permission to the object and the XML attribute `ExecutionType` of the local permission rule is set to `Standard`.

If permission rules are defined that do not include the authorized user as a permitted user, the authorized user may only edit the object in the context of the user session in which the object is being created. Once the user session ends, the authorized user will no longer be able to edit the object.

To guarantee that authorized users, deputies and authorized user groups can access objects they are responsible for even if permission rules are specified, a permission rule should be specified for the relevant object class that specifies Read/Write access permissions for users that are authorized user, deputy or member of an authorized user group for the object.

Please note that some functionalities use explorers that are based on the concept of authorized access and therefore display only objects that the current user is the authorized user of or objects that are available to the current user via an authorized user group affiliation. Such explorer nodes are typically named "My <ObjectClassName>" (For example, the **Document Application** functionality has an explorer with the caption **My Applications**). Because a permission rule may be configured so that a user may not have ReadWrite access permission to objects for which he/she is the authorized user, it is recommended that these functionalities are not implemented when permission rules are specified.

Configuring Global Permission Rules

Global permission rules are query based. A user has ReadWrite access to an object if the query of any of the defined global permission rules for the object class returns that object.

The global permission rules are evaluated whenever a user logs in and on first access of an object of an object class during the current user session. Access permissions for the object class are then stored in the memory and read from the memory when the user accesses another object of the same class. The evaluation of permission rules is repeated during the user session in regular intervals of typically 300 seconds to ensure that changes in the database (such as the creation of new objects or new relations between objects) are taken into account. The interval to evaluate permission rules is configurable in Alfabet Expand.

For example, if a rule specifies that a user can edit all objects for which he/she has the role Architect defined, and the authorized user changes the person that is responsible for the role Architect to another user, the user who formerly had the role Architect assigned will have

ReadOnly permissions to the object as soon as the permission rules are re-executed during the active user session.

Independent of the evaluation of permission rules, the user will have Read/Write access to an object that he/she has created in a session as long as the session is active.

Please note that changes to the database or the user settings during the current session are only evaluated in the given time interval. For example, if a permission rule is based on the user profile the user is currently logged in with and the user session starts with a user profile granting Read/Write access to an object, the user will still have Read/Write access permissions to the object even if he/she changes to another user profile granting ReadOnly access to the object as long as the permission rules are not re-executed.

Additionally, default permission rules can be defined per object class or object class stereotype. These permission rules do not have a query specified. The default permission rule defines the access permissions for objects in a specified object class that are not found via a query-based permission rule.

If no default permission rule is defined for an object class or object class stereotype and the object does not match any of the global permission rules defined, the alternative permission concepts like access-based on authorized user permissions will be evaluated.

Global permission rules are defined in the XML object ***RightsManager***:

1) Go to the **Presentation** tab, expand the **XML Objects** folder, and then expand the **Administration** folder by clicking the + symbol. You will see all XML objects that can be edited.
2) Right-click the XML object ***RightsManager*** and select **Edit XML**. The XML editor is displayed in the center pane.

For general information about editing XML objects in Alfabet Expand, see Working with XML Objects. All attributes that are described in the following are permissible attributes in the XML object. If they are not automatically displayed in the XML editor with a default value when opening the XML object, you can write them to the appropriate position in the editor. The example provided with this documentation shows the correct position of the attribute within the XML code.

3) Optionally, specify the interval to evaluate permission rules. The configured permission rules will be evaluated whenever a user logs in or tries to access an object. The evaluation of permission rules is then repeated during the user session in regular intervals of typically 300 seconds to ensure that changes in the database (such as the creation of new objects or new relations between objects) are taken into account.

   To change the interval, the XML attribute `UpdatePeriod` must be added to the XML element ***AlfaRightsManager***. The value of the XML attribute specifies the update interval in seconds.

The following example shows the configuration of permission rule evaluation every 120 seconds in the XML attribute `UpdatePeriod`. The element includes other settings for the configuration of other access permission concepts:

```
<AlfaRightsManager
Inheritance="false"
Propagation="false"
MandateMode="true"
AssemblyName="ITPlan"
```



```
AssemblyClass="ITPlanSolution.SolutionRightsManager"
UpdatePeriod="120"
>
    [.....]
</AlfaRightsManager>
```

4) Define the permission rules and default permission rules as specified below.

5) In the toolbar, click the **Save**  button to save the XML definition.

A global permission rule is specified in the XML element ***RightsRule***, which is a child element of the XML element ***AlfaRightsManager***. The XML element ***AlfaRightsManager*** may contain multiple ***RightsRule*** elements.

The following example shows the complete configuration for permission rules including the definition of one permission rule that allows the user to access all market products for which the user is assigned to the object via the role Architect:

```
<AlfaRightsManager
[...]
AssemblyName="ITPlan"
AssemblyClass="ITPlanSolution.SolutionRightsManager"
UpdatePeriod="120"
></AlfaRightsManager>
   <AlfaMethod Type="OnGetObjectRights" Name = "OnGetObjectRights"
   />
   <RightsRule Name="Market Product Access for Architects"
   ClassName="MarketProduct"
   Rights="Master"
   Query="ALFABET_QUERY_500 FIND MarketProduct
   InnerJoin Role ON Role.Object = MarketProduct.REFSTR
   InnerJoin RoleType ON RoleType.REFSTR = Role.RoleType
   InnerJoin Person ON Role.Responsible = Person.REFSTR
   WHERE (AND RoleType.Name = 'Architect' Person.REFSTR
   =:CURRENT_USER)"/>
```

The following attributes can be defined for the XML element ***RightsRule*** to configure a global permission rule:

XML Attribute	Allowed Value Types or Allowed Values	Mandatory/Optional	Purpose
`Name`	`String`	Mandatory	Defines a name for the permission rule. The name can be any string and is for internal XML element identification by the editor of the rules.
`Rights`	`Master` `Read`	Mandatory	The XML attribute `Rights` must be set to `Master` to specify a valid permission rule. For default permission rules, the XML attribute `Rights` can be set to `Master` to grant Read/Write access permissions, or to `Read` to grant ReadOnly access permissions.
`Query`	`Alfabet query` `SQL query`	Optional	Defines the objects that the user shall be able to edit. Objects are specified by an Alfabet query in Alfabet query language 500 or by an SQL-based query. If no query is defined, the permission rule is a default permission rule and defines the default access permissions for the object class. **NOTE**: The query must find objects of the object class defined with the XML attribute `ClassName`.
`ClassName`	`String`	Mandatory for stereotype specific rules	If the permission rule is defined for an object class stereotype, the object class and object class stereotype for which the permission is specified must be defined with the XML attribute `ClassName`. The XML attribute `ClassName` must be defined as `<class name>:<stereotype name>`, for example: `Application:BusinessApplication`

A permission rule assigns edit permissions to the objects of the object class defined in the XML attribute `ClassName` that match the conditions of the query defined in the XML attribute `Query`. Default permission rules define access permissions for objects of the object class defined in the XML attribute `ClassName` that do not match the conditions of any query defined in the permission rule for the object class.

Permission rules can be defined per object class or per object class stereotype. To define a permission rule for an object class stereotype, you must define the `RightsRule` attribute as follows:

- The XML attribute `ClassName` must be defined as `<class name>:<stereotype name>`
- The query defined in the XML attribute `Query` must return only objects of the selected stereotype.

It is possible to specify more than one permission rule for an object class. The user will then have edit permissions for all objects for which one of the specified rules apply.

When permission rules are defined for both an object class and stereotypes of that object class, the access permissions for an object of a given stereotype are evaluated by first applying the rules for the stereotype and, if the object does not match the criteria defined in any of these rules, the rules for the object class.

When a user creates an object, he/she has access permissions to that object during the active session, even if a permission rule based on attributes of the objects does not allow editing of the object by the user.

Please consider the following when specifying an Alfabet query for a permission rule:

- Edit permissions are specified for the base object class of the Alfabet query only. Object classes can be added with a `JOIN` to add attributes of other classes to `WHERE` statements. The access permissions of the object classes added with a `JOIN` are not affected by the permission rule.
- `SHOW` and `SORT` property specifications are not allowed in a permission rule.
- Alfabet query language parameters can be used to refer to the current environment. For example, the parameter `CURRENT_USER` can be used to refer to the user that is currently logged in to the Alfabet user interface. The parameter `BASE` that refers to the current object cannot be used for global permission rules because the permission rule is not evaluated for a single base object.

Please note that changes to the database or the user settings during the current session are only evaluated in the given time interval. For example if a permission rule refers to the user profile the user is currently logged in with with the parameter `CURRENT_PROFILE`, and the user session starts with a user profile granting Read/Write access to an object, the user will still have Read/Write access permissions to the object even if he/she changes to another user profile granting ReadOnly access to the object as long as the permission rules are not re-executed.

For more information about Alfabet query language parameters, see Referring to the Current Alfabet Context in a WHERE Condition in the chapter Defining Queries.

- If the Alfabet query contains special characters (for example, a greater then (>) or lesser then (<) symbol) in the `WHERE` clause, you must replace it with the HTML compatible strings:

- `>` for >
- `<` for <
- `&quot;` for "
- `&#091;` for [
- `&#093;` for]

- For basic information about the Alfabet query language, see the chapter Defining Queries.

Please consider the following when specifying an SQL-based query for a permission rule:

- To define an SQL-based query for a permission rule, native SQL can be written directly into the XML attribute `Query`.
- Edit permission are specified for objects of the object class specified with the XML attribute `ClassName` of the permission rule. The SQL-based query is only used to define conditions that an object of the object class specified with the XML attribute `ClassName` must meet in order for the permission rule to apply.

- The SQL-based query must return the values of the `REFSTR` property of the relevant objects.
- Alfabet query language parameters can be used to refer to the current environment. For example, the parameter `CURRENT_USER` can be used to refer to the user that is currently logged in to the Alfabet user interface. The parameter `BASE` that refers to the current object cannot be used for global permission rules, because the permission rule is not evaluated for a single base object. For more information about Alfabet query language parameters, see Referring to the Current Alfabet Context in a WHERE Condition and Using Alfabet Parameters in the chapter Defining Queries.
- If the SQL-based query contains special characters (for example, a greater then (>) or lesser then (<) symbol), you must replace it with the HTML compatible strings:

- `>` for >
- `<` for <
- `&quot;` for "
- `&#091;` for [
- `&#093;` for]

Configuring Local Permission Rules

Independent of the evaluation of permission rules, the user will have Read/Write access to an object that he/she has created in a session as long as the session is active.

Local permission rules are defined in the XML object ***RightsManager***:

1) Go to the **Presentation** tab, expand the **XML Objects** folder, and then expand the **Administration** folder by clicking the + symbol. You will see all XML objects that can be edited.
2) Right-click the XML object ***RightsManager*** and select **Edit XML**. The XML editor is displayed in the center pane.

   For general information about editing XML objects in Alfabet Expand, see Working with XML Objects. All attributes that are described in the following are permissible attributes in the XML object. If they are not automatically displayed in the XML editor with a default value when opening the XML object, you can write them to the appropriate position in the editor. The example provided with this documentation shows the correct position of the attribute within the XML code.

3) Define the XML elements and XML attributes for the permission rules and default permission rules as specified below.
4) In the toolbar, click the **Save**  button to save the XML definition.

A local permission rule is specified in the XML element ***RightsRule***, which is a child element of the XML element ***AlfaRightsManager***. The XML element ***AlfaRightsManager*** may contain multiple ***RightsRule*** XML elements.

The following example shows the configuration of a local permission rule that allows the user to access all market products for which the user is assigned to the object via the role Architect:

```
<AlfaRightsManager
[...]
AssemblyName="ITPlan"
AssemblyClass="ITPlanSolution.SolutionRightsManager"
UpdatePeriod="120"
></AlfaRightsManager>
    <AlfaMethod Type="OnGetObjectRights" Name = "OnGetObjectRights"
    />
    <RightsRule Name="Local Market Product Access for Architects"
    Rights="Master"
    RuleType="Local"
    ExecutionType="Standard"
    ResultType="Positive"
    ClassName="MarketProduct"
    Query="ALFABET_QUERY_500 FIND MarketProduct
    InnerJoin Role ON Role.Object = MarketProduct.REFSTR
    InnerJoin RoleType ON RoleType.REFSTR = Role.RoleType
    InnerJoin Person ON Role.Responsible = Person.REFSTR
    WHERE (AND RoleType.Name = 'Architect' Person.REFSTR
    =:CURRENT_USER MarketProduct.REFSTR =:BASE)"/>
```

The following attributes can be defined for the XML element ***RightsRule*** to configure a global permission rule:

Attribute	Allowed Value Types or Allowed Values	Mandatory/Optional	Purpose
`Name`	`String`	Mandatory	Defines a name for the permission rule. The name can be any string and is for internal identification by the editor of the rules.
`RuleType`	`Local`	Mandatory	The XML attribute `RuleType` must be set to `Local` to specify a local permission rule.
`Rights`	`Master`	Mandatory	The XML attribute `Rights` must be set to `Master` to specify a valid permission rule.

Attribute	Allowed Value Types or Allowed Values	Mandatory/Optional	Purpose
ClassName	String	Mandatory for stereotype-specific rules	If the permission rule is defined for an object class stereotype, the object class and object class stereotype for which the permission is specified must be defined with the XML attribute ClassName The XML attribute ClassName must be defined as <class name>:<stereotype name>, for example: Application.BusinessApplication
Query	Alfabet query SQL query	Mandatory	Defines a query that refers to the current object and may or may not return a dataset. The user has Read/Write access to an object if the query returns a dataset and the XML attribute ResultType is set to Positive or if the query returns no dataset and the XML attribute ResultType is set to Negative.
ResultType	Positive Negative	Mandatory	The user has Read/Write access to an object if the query defined with the attribute Query returns a dataset and the XML attribute ResultType is set to Positive or if the query returns no dataset and the XML attribute ResultType is set to Negative.
ExecutionType	Standard Exclusive	Mandatory	Defines whether other access permission concepts are evaluated when the condition defined via the XML attributes ResultType and Query are not fulfilled. If the XML attribute ExecutionType is set to Exclusive, the user has ReadOnly access when the permission rule does not grant ReadWrite access permissions. If the XML attribute ExecutionType is set to Standard, the access permissions for the object are evaluated according to other configured permission concepts, like the authorized user concept.

It is possible to specify more than one permission rule for an object class. The user will then have edit permissions for an object if one of the specified rules apply.

Note the following when defining more than one local permission rule:

- The permission rules are evaluated in the order they are written in the XML object ***RightsManager*** until the object matches one of the defined criteria.
- When the XML attribute ExecutionType of a rule is set to Exclusive, the following rules will not be evaluated. The XML attribute ExecutionType of the last defined rule specifies, whether other access concepts are evaluated when the object does not apply to any of the defined local rules.
- When permission rules are defined for both an object class and stereotypes of that object class, the access permissions for an object of a given stereotype are evaluated by first applying the rules for the stereotype and, if the object does not match the criteria defined in any of these rules, the rules for the object class.

- When a user creates an object, he has access permissions to that object during the active session, even if a permission rule based on attributes of the objects does not allow editing of the object by the user.

Please consider the following when specifying an Alfabet query for a permission rule:

- Edit permissions are specified for the base object class of the Alfabet query only. Object classes can be added with a `JOIN` to add attributes of other object classes to `WHERE` statements. The access permissions of the object classes added with a `JOIN` are not affected be the permission rule.
- `SHOW` and `SORT` property specifications are not allowed in a permission rule.
- The query must refer to the current object with the Alfabet query language parameter `BASE` that returns the `REFSTR` value of the object the user currently accesses. Other Alfabet query language parameters can also be used to refer to the current environment. For example the parameter `CURRENT_USER` can be used to refer to the user that is currently logged in to the Alfabet user interface.

For more information about Alfabet query language parameters, see Referring to the Current Alfabet Context in a WHERE Condition in the chapter Defining Queries.

- If the Alfabet query contains special characters (for example, a greater then (>) or lesser then (<) symbol) in the `WHERE` clause, you have to replace it with the HTML compatible strings:

- `>` for >
- `<` for <
- `&quot;` for "
- `&#091;` for [
- `&#093;` for]

- For basic information about the Alfabet query language, see the chapter Defining Queries.

Please consider the following when specifying an SQL-based query for a permission rule:

- To define an SQL-based query for a permission rule, native SQL can be written directly into the attribute `Query`.
- Edit permissions are specified for objects of the object class specified with the attribute `ClassName` of the permission rule. The SQL-based query is only used to define conditions that an object of the object class specified with the attribute `ClassName` must meet in order for the permission rule to apply.
- The SQL-based query must return the values of the `REFSTR` property of the relevant objects.
- The query must refer to the current object with the Alfabet query language parameter `BASE` that returns the `REFSTR` value of the object the user currently accesses. Other Alfabet query language parameters can also be used to refer to the current environment. For example, the parameter `CURRENT_USER` can be used to refer to the user that is currently logged in to the Alfabet user interface.

For more information about Alfabet query language parameters, see Referring to the Current Alfabet Context in a WHERE Condition and Using Alfabet Parameters in the chapter Defining Queries.

- If the SQL-based query contains special characters (for example, a greater then (>) or lesser then (<) symbol), you have to replace it with the HTML compatible strings:

- `>` for >
- `<` for <
- `&quot;` for "
- `&#091;` for [
- `&#093;` for]

Configuring Master Users and Master User Groups

In the XML object ***RightsManager***, the XML elements ***MasterUser*** and ***MasterUserGroup*** allow a named user or user group to have access permissions to all Alfabet objects regardless of the access permissions defined for the object.

Please note the following:

- The access permissions defined for a master user or users in the master user group will have precedence over access established via the authorized user and authorized user group definition as well as query-based access permission rules.
- The access permissions defined for a master user or users in the master user group do not have precedence over access denied due to a non-editable release status or mandate definitions.
- The named user or users in the user group must access Alfabet with a user profile that has ReadWrite access permissions defined.
- The XML object ***RightsManager*** is handed over to the server process at start of the application. Any changes to the XML object ***RightsManager*** only become valid after server restart. If you want to test your configuration by accessing the Alfabet user interface via Alfabet Expand, you must restart Alfabet Expand after making any change to the XML object ***RightsManager*** and prior to opening the Alfabet user interface.

To edit the XML object ***RightsManager***:

1) Go to the **Presentation** tab, expand the **XML Objects** folder, and then expand the **Administration** folder by clicking the + symbol. You will see all XML objects that can be edited.
2) Right-click the XML object ***RightsManager*** and select **Edit XML**. The XML editor is displayed in the center pane.

   For general information about editing XML objects in Alfabet Expand, see Working with XML Objects. All attributes that are described in the following are permissible attributes in the XML object. If they are not automatically displayed in the XML editor with a default value when opening the XML object, you can write them to the appropriate position in the editor. The example provided with this documentation shows the correct position of the attribute within the XML code.

3) To define a master user, add a child XML element ***MasterUser*** to the XML element ***AlfaRightsManager***. TheXML element ***MasterUser*** must have an XML attribute `Name` that defines the user name of the master user.

The name entered in theXML attribute `Name` of the XML elements ***MasterUser*** and ***MasterUserGroup*** must be capitalized as it is in the Alfabet database table. Any user name defined in the **User** editor (`USER_Editor`) in the context of the **Users Administration** functionality is saved to the Alfabet database in all capital letters, regardless of how it is entered in the editor. For more information about defining users in Alfabet, see the section *Defining and Managing Users* in the reference manual *User and Solution Administration*.

4) To define a master user group, add a child XML element ***MasterUserGroup*** to the XML element ***AlfaRightsManager***. The XML element ***MasterUserGroup***t must have an XML attribute `Name` that defines the name of the user group.

Below is an example of the definition of a master user and a master user group in the XML object ***RightsManager***:

```
<AlfaRightsManager
[...]
AssemblyName="ITPlan"
AssemblyClass="ITPlanSolution.SolutionRightsManager"
UpdatePeriod="120"
></AlfaRightsManager>
    <AlfaMethod Type="OnGetObjectRights" Name = "OnGetObjectRights"
    />
    <MasterUser Name='CLIENTE' />
    <MasterUserGroup Name='Object Owners' />
    <RightsRule Name="Market Product Access for Architects" [...]/>
```

Configuring the Propagation/Inheritance of User Group Rights

The XML object ***RightsManager*** allows you to configure rules for propagation or inheritance for user group permissions. The concept of access permissions for user groups is described in the section Access Permissions for Authorized User Groups. The following is determined via the configuration of inheritance and propagation in the XML object ***RightsManager***:

- **Propagation**: All user groups in the user group hierarchy that are ascendant to a user group should have the same access permissions to that object.
- **Inheritance**: All user groups in the user group hierarchy that are descendant to a user group should have the same access permissions to that object.

This XML object should be defined as part of the initial configuration of the Alfabet solution. These definitions should not be changed once the Alfabet database is in use.

To edit the XML object ***RightsManager***:

1) Go to the **Presentation** tab, expand the **XML Objects** folder, and then expand the **Administration** folder by clicking the + symbol. You will see all XML objects that can be edited.

2) Right-click the XML object ***RightsManager*** and select **Edit XML**. The XML editor is displayed in the center pane.

For general information about editing XML objects in Alfabet Expand, see Working with XML Objects. All attributes that are described in the following are permissible attributes in the XML object. If they are not automatically displayed in the XML editor with a default value when opening the XML object, you can write them to the appropriate position in the editor. The example provided with this documentation shows the correct position of the attribute within the XML code.

3) In the XML attribute `Inheritance` of the XML element ***AlfaRightsManager***, enter "`true`" if a user group's access permissions to a specific object should be automatically inherited by all descendant groups in the user group hierarchy. Enter "`false`" if access permissions should not be inherited by all descendant groups in the user group hierarchy.

4) In the XML attribute `Propagation` of the XML element ***AlfaRightsManager*** enter "`true`" if a user group's permissions to a specific object should be automatically propagated to all ascendant groups in the user group hierarchy. Enter "`false`" if access permissions should not be propagated to the ascendant groups in the user group hierarchy.

The following example shows an XML object ***RightsManager*** in which the XML attribute `Inheritance` is set to "`false`" and the XML attribute `Propagation` is set to "`true`":

```
<?xml version="1.0" encoding="utf-8" ?>
<AlfaRightsManager
    Inheritance="false"
    Propagation="true"
    MandateMode="true"
    AssemblyName="ITPlan"
    AssemblyClass="ITPlanSolution.SolutionRightsManager" >
<AlfaMethod Type="OnGetObjectRights" Name =
"OnGetObjectRights" />
</AlfaRightsManager>
```

5) In the toolbar, click the **Save**  button to save the XML definition.

Chapter 4: Localization and Multi-Language Support for the Alfabet Interface

Software AG provides localization possibilities as well as multi-language support in order to adapt the Alfabet interface to suit the needs and expectations of the various locales in which your enterprise operates. All Alfabet components are delivered with the default interface in `English (United States)` as well as a translated interface for the supported secondary languages. A vocabulary, which constitutes the set of standard original and translated strings for a language, is available for each supported language.

The Alfabet interface is available in the following languages:

Language	Locale ID
Arabic (Saudi Arabia)	1025
German (Germany)	1031
English (United States)	1033
French (France)	1036
Portuguese (Brazil)	1046
Polish (Poland)	1045

Custom strings for configuration objects (such as custom properties, custom editors, configured reports, etc.) must be captured in English, regardless of the primary culture definition. All strings for captions for configuration objects and guide page/guide view content will be displayed in the `Original` column in the **Translation Editor** or XLSX files of the relevant vocabulary.

The following is possible in order to localize the Alfabet interface for your enterprise's needs:

- Specify cultures that capture the locale ID, the interface language, and the date, time, and number formats. Users can select the relevant culture to display on the user interface via the **Language** option in the < **Alfabet User** > menu.
- Modify the standard English strings and provide alternative terminology that is relevant for your enterprise. For example, if the English term `Software` is used instead of `Application` in your enterprise, you can customize all strings for the `Original` vocabulary in which the term `Application` is used so that the word `Software` is used instead. Likewise, if the German term `Anwendung` is used instead of `Applikation` in your enterprise, you can customize all translated strings for the `German` vocabulary in which the term `Applikation` is used so that the word `Anwendung` is used instead.

- Manually or automatically translate the custom strings that your enterprise has created when configuring the class model, custom editors, selectors, object views/object cockpits, workflows, wizards, configured reports, guide pages/guide views, etc. to the languages supported by Software AG.
- Specify whether manual or automated translation is allowed for the translation of object data by users in the Alfabet user interface. You can specify the object classes and properties for which data translation is allowed. The translation of object data typically includes the captions, descriptions, and other custom properties of the type `String` defined for objects that users create in the Alfabet user interface.
- Specify a statutory language to address regulatory requirements that mandate that objects are captured in a specific language. When a user creates an object based on the object class/object class stereotype, a statutory language selector will open in which the language can be selected. In the object editor that opens, the user can capture all attributes that support automated translation in the statutory language.
- Provide a custom online help.
- Translate the online help files `(HTML)` provided with Alfabet.

The following information is available:

Specifying the Cultures Relevant to Your Enterprise

A culture constitutes the base configuration of the default primary language displayed on the user interface when the user first logs in as well as the date, time, and number formats, etc. used to capture and render dates, times and numbers in the Alfabet user interface.

The Alfabet interface is available in the following languages:

Language	Locale ID
Arabic (Saudi Arabia)	1025
German (Germany)	1031
English (United States)	1033
French (France)	1036

Language	Locale ID
Portuguese (Brazil)	1046
Polish (Poland)	1045

A base culture must be specified for each culture you define. The base culture is associated with a culture code which is the equivalent of a locale code assigned by Microsoft® Windows®. The culture code determines which vocabulary (the set of original and translated strings) to display in the Alfabet user interface - in other words, the base culture determines the language displayed on the Alfabet user interface. In addition to the base culture, you can also specify date and time patterns used to capture dates and timestamps, whether a point or comma is used as the decimal symbol in numbers, or whether measurements for printing are in the metric or imperial system, etc. A base culture may be defined for only one culture.

You may define multiple cultures associated with the supported languages. If only one culture is defined for your enterprise, then the automated and manual data translation functionalities will be disabled in the Alfabet user interface.

If your enterprise has various corporate locations around the world that all use the same corporate language but require different date patterns, time patterns, measurement units, you can create two cultures, each that reference the same language but with different date and time patterns and measurement units. For example, if you have a subsidiary in the United States and one in Ireland, you could create an additional English language culture setting such as English (Ireland) and specify the date pattern, time pattern, and measurement units for the cultures English (United States) and English (Ireland) as needed.

To specify a new or existing culture:

1) In the **Meta-Model** tab, right-click the **Cultures** node and select **Add New Culture** . The new culture is displayed below the **Cultures** node and is labelled **Invariant Language**.
2) Click the new culture to display the attribute window and define the following attributes:

- **Base Culture**: Select the locale that you are defining. The primary language displayed in the user interface is determined by the **Base Culture** attribute. Therefore, if you specify `French (France)`, then the `French (France)` vocabulary will be called when the base culture `French (France)` is displayed in the user interface. If a base culture is selected for which no vocabulary has been created, the default `English (United States)` will be displayed in the Alfabet interface.
- **Language Culture Name**: Displays the name of the language associated with the base culture. This information may be relevant for the specification of REST API calls.
- **Current Machine Date Pattern**, **Current Machine Date Separator**, and **Current Machine Time Pattern**: Displays the local settings of the server machine. These default settings will be used if the **Date Pattern**, **Date Separator**, and **Time Pattern** attributes are not explicitly set. In order to ensure that the user has consistent date and number formating, the following attributes should be defined.

  Please note that the dates in editor fields are always displayed as a numeric value, regardless of the date format. For example, whereas the date format dddd, dd MMMM yyyy will be displayed as Tuesday, 22 August 2006 in the object profile, it will

be displayed as 22/08/2006 in the editor field. For more information about the syntax and possible output values defined for date/separator/time specification, see the section Permissible Date and Time Formats in Alfabet .

- **Date Format**: Select the locale of the date and time format to be displayed in the solution interface. An error message will be displayed in the user interface if the user attempts to enter a date that has a different date format. Please note the following:
- If no locale is selected in the **Date Format** field, the default date and time formats defined for the server machine will be displayed. These are displayed in the **Current Machine Date Pattern**, **Current Machine Date Separator**, and **Current Machine Time Pattern** fields.
- The date format determines the representation of the date pattern, date separator and time pattern. If a different date pattern, date separator, or time pattern is required than that specified by the locale selected in the **Date Format** field, you must explicitly define the **Date Pattern**, **Date Separator**, and /or **Time Pattern** attribute.
- **Date Pattern**: If the date pattern should deviate from the date pattern specified in the **Date Format** attribute or the **Current Machine Date Pattern** attribute, enter the date pattern to implement for the culture setting.

  Please note that users will typically define dates in editor and filter fields using the **Calendar** button available for the field. However, if you would like to enable users to define dates manually, you must ensure that the **Date Separator** attribute is also defined.

- **Date Separator**: If the date separator should deviate from the date separator associated with the **Date Format** attribute or the **Current Machine Date Separator** attribute, enter the date separator to implement for the culture setting. This definition supersedes the value in the **Current Machine Date Pattern** field. Please note that if both the **Date Separator** and the **Date Pattern** attribute are specified, the "/" character must be used as the separator when specifying the **Date Pattern** attribute.
- **Time Pattern**: If the time pattern should deviate from the time pattern associated with the **Date Format** attribute or the **Current Machine Time Pattern** attribute, enter the time pattern to implement for the culture setting.
- **Number Format**: If the number format should deviate from the locale associated with the **Base Culture** attribute, select the locale of the number format to be displayed in the solution interface. An error message will be displayed if a user attempts to enter a number with a different number format.
- **Decimal Symbol**: If the decimal symbol should deviate from the decimal symbol associated with the **Number Format** attribute or the **Base Culture** attribute, enter the decimal symbol that should be used for integers and real numbers. (For example, the English language decimal symbol typically uses a period (10.50) and the German language decimal symbol typically uses a comma (10,50).
- **Digit Grouping Symbol**: If the digit grouping symbol should deviate from the digit grouping symbol associated with the **Number Format** attribute or the **Base Culture** attribute, enter the digit grouping symbol that should be used for integers and real numbers defined for standard and custom attributes. (For example, the English language digit grouping symbol typically uses a comma (100,000,000) and the German language digit grouping symbol typically uses a period (100.000.000).
- **Icon**: Typically, an icon of the national flag associated with the base culture is displayed in the Alfabet user interface for the currently selected culture. If a different icon should be displayed, select the icon in the **Icon** field. For more information about adding custom icons, see Uploading Custom Icons to the Icon Gallery.

- **Is Primary Culture**: Select `True` if this culture is the base culture. The primary culture is the culture displayed in the Alfabet user interface upon login. It is also the language that is considered the primary language when users create or modify objects in editors and wizards in the Alfabet user interface. To change the **Is Primary Culture** definition, see the section Specifying the Primary Culture for Your Enterprise.
- **Help Culture**: Select the locale of the context-sensitive online help that shall be available for the culture. A standard online help is provided only for `English (United States)1033` and `German (Germany)1031`. If the **Base Culture** attribute is set to a language other than `English (United States)` or `German (Germany)`, an online help will not be available unless you explicitly provide one. For more information about translating the online help, see the section Creating a New Language Version of the Online Help
- **Measurement Units**: Select the measurement unit relevant for the printing functionality (margin settings, etc.).
- **Support Data Translation**: Specifies whether instance translation of object data is allowed for the selected culture. Data translation constitutes the translation of the values defined for properties of the type `String` or `Text`defined for objects. The data captured may be translated to any of the languages supported by the enterprise. If the **Support Data Translation** attribute is set to `False`, then the automated and manual data translation functionalities will be disabled and the language code will not be displayed in the language field in all object editors. If the **Support Data Translation** attribute is set to `True` for the culture, data translation will be possible per default for all object classes for predefined protected properties and custom properties of the type `String` or `Text`. However, you can restrict data translation to specified classes and specified custom properties. For more information about configuring the data translation capability, see the section Configuring the Translation of Object Data.

  To change the data translation definition, right-click the culture  and select:

- **Activate Support for Data Translation** to specify that data translation is allowed for the culture. The **Support Data Translation** attribute will be set to `True` of you select this option.
- **Deactivate Support for Data Translation** to specify that data translation is not allowed for the culture. The **Support Data Translation** attribute will be set to `False` of you select this option.
- **Automated Assistant Parameter**: If your enterprise provides translations of the content specified for automated help assistants, the translated content may be displayed when the user interface is rendered in a secondary language. The URL defined for the automated help assistant should include a {CURRENT_LANGUAGE} parameter whereby {CURRENT_LANGUAGE} is replaced with the language of the user interface at the time of the call. For example:
  `http://autohelp.alfabet.com/\{CURRENT_LANGUAGE}/showcase/user_profile/Full_Access.html`

In the **Automated Assistant Parameter** attribute, specify the replacement value for the {CURRENT_LANGUAGE} parameter. This could be, for example, DE or Deutsch or 1031 or DE-de. If no parameter is specified, the four-digit ISO code will be used (for example 1031). For more information about configuring automated help assistants, see the section Understanding the Automated Help Assistant.

3) In the toolbar, click the **Save**  button to save the culture definition.

Permissible Date and Time Formats in Alfabet

Please note that the dates in editor fields are always displayed as a numeric value, regardless of the date format. For example, whereas the date format dddd, dd MMMM yyyy will be displayed as Tuesday, 22 August 2006 in the object profile, it will be displayed as 22/08/2006 in the editor field.

Please consider the following syntax for the specification of date and time formats.

- To specify the 24-hour clock system, use "H" or "HH" for hours.
- To specify the 12-hour clock system, use "h" or "hh". Users will need to specify AM or PM.
- To specify a time format, use "M" for month and "m" for minute.

The following formats can be specified:

Date/Time/Separator Syntax	Date/Time/Separator Output
MM/dd/yyyy	08/22/2006
dddd, dd MMMM yyyy	Tuesday, 22 August 2006
dddd, dd MMMM yyyy HH:mm	Tuesday, 22 August 2006 06:30
dddd, dd MMMM yyyy hh:mm tt	Tuesday, 22 August 2006 06:30 AM
dddd, dd MMMM yyyy H:mm	Tuesday, 22 August 2006 6:30
dddd, dd MMMM yyyy h:mm tt	Tuesday, 22 August 2006 6:30 AM
dddd, dd MMMM yyyy HH:mm:ss	Tuesday, 22 August 2006 06:30:07
MM/dd/yyyy HH:mm	08/22/2006 06:30
MM/dd/yyyy hh:mm tt	08/22/2006 06:30 AM
MM/dd/yyyy H:mm	08/22/2006 6:30
MM/dd/yyyy h:mm tt	08/22/2006 6:30 AM

Date/Time/Separator Syntax	Date/Time/Separator Output
MM/dd/yyyy h:mm tt	08/22/2006 6:30 AM
MM/dd/yyyy h:mm tt	08/22/2006 6:30 AM
MM/dd/yyyy HH:mm:ss	08/22/2006 06:30:07
MMMM dd	August 22
MMMM dd	August 22
yyyy'-'MM'-'dd'T'HH':'mm':'ss.fffffffK	2006-08-22T06:30:07.7199222-04:00
yyyy'-'MM'-'dd'T'HH':'mm':'ss.fffffffK	2006-08-22T06:30:07.7199222-04:00
ddd, dd MMM yyyy HH':'mm':'ss 'GMT'	Tue, 22 Aug 2006 06:30:07 GMT
ddd, dd MMM yyyy HH':'mm':'ss 'GMT'	Tue, 22 Aug 2006 06:30:07 GMT
yyyy'-'MM'-'dd'T'HH':'mm':'ss	2006-08-22T06:30:07
HH:mm	06:30
hh:mm tt	06:30 AM
H:mm	6:30
h:mm tt	6:30 AM
HH:mm:ss	06:30:07
yyyy'-'MM'-'dd HH':'mm':'ss'Z'	2006-08-22 06:30:07Z
dddd, dd MMMM yyyy HH:mm:ss	Tuesday, 22 August 2006 06:30:07

Date/Time/Separator Syntax	Date/Time/Separator Output
`yyyy MMMM`	`2006 August`
`yyyy MMMM`	`2006 August`

Specifying the Primary Culture for Your Enterprise

As part of the configuration of the Alfabet solution, you must specify which culture is the base culture for the enterprise. The primary culture constitutes the culture code, primary language, and date and time formats specified for the culture. Only one culture can be defined as the primary culture.

Custom strings for configuration objects (such as custom properties, custom editors, configured reports, etc.) must be captured in English, regardless of the primary culture definition. All strings for captions for configuration objects and guide page/guide view content will be displayed in the `Original` column in the **Translation Editor** or XLSX files of the relevant vocabulary.

The primary culture will be displayed when users open the Alfabet user interface upon first login. After the user has accessed Alfabet for the first time, the user can then change the language displayed in the Alfabet user interface via the **Language** button in the upper-right corner of the Alfabet interface. The language used for the user's last session will be automatically displayed when the user opens a new user session.

To define the primary culture that is automatically displayed in the Alfabet user interface when a user first logs in to Alfabet, click the relevant culture below the **Cultures** node and in the attribute window, select `True` in the **Is Primary Culture** attribute. Click the **Save**  button to save the culture definition.

Please consider the following:

- The **Support Data Translation** attribute must be set to `False` for the culture that you want define as the primary culture.
- In order to set a different culture as the primary culture, you must first set the **Is Primary Culture** attribute of the original primary culture to `False` and then set the **Is Primary Culture** attribute for the new primary culture to `True`.

Modifying, Translating and Managing the Vocabularies

A vocabulary is the set of original and translated strings that are implemented in the Alfabet interface. Each vocabulary is associated with a base culture via the locale ID.

The Alfabet interface is available in the following languages:

Language	Locale ID
Arabic (Saudi Arabia)	1025
German (Germany)	1031
English (United States)	1033
French (France)	1036
Portuguese (Brazil)	1046
Polish (Poland)	1045

For technical reasons, the translatable strings are divided into four separate vocabularies which constitute terminology sets based on their source of origin in the software. These are `METAMODEL`, `ITPlan`, `Platform`, `Extensions`, and `GUIDEPAGES`. The vocabularies are displayed as columns in the **Translation Editor**. If you choose to export the vocabularies for translation, each terminology set will be exported as a separate vocabulary file (as either a VOC or XLSX file). The file names include the locale ID of the translated language as well as the name of the specific vocabulary set. For example, the following file syntax would be displayed for the meta-model file containing the English strings and any existing German translations: `AlfaVoc_127_METAMODEL_`<date>`_`<timestamp>.xlsx.

An automated translation that leverages a translation service is available to provide a translation of terms in the standard vocabularies as well as the object data defined by users. For more information, see the section Configuring the Translation of Object Data.

The following explains the content in each of the vocabularies:

- `METAMODEL`: Strings derived from the meta-model. This includes strings for the following:
- Standard configuration objects. These strings are protected strings and are typically already translated by Software AG. They include, for example, the caption of standard object classes, object class properties, page view names and descriptions, toolbar buttons, menu options, and the captions and tooltips available for editor and filter fields. The standard English terminology as well as the translated terms may be changed, if needed.
- Custom configuration objects. These are public strings and may be translated. They include, for example, captions of object class stereotypes and custom properties, the captions and tooltips defined for custom editor and filter fields, the captions of custom object views and object cockpits, the

captions of explorer names in custom selectors, and the text defined in the context of guide views and splash screens. These strings will be added to the vocabulary after the configuration has been saved. Translation to any language other than `English (United States)` must be provided by the customer.

Custom strings for configuration objects (such as custom properties, custom editors, configured reports, etc.) must be captured in English, regardless of the primary culture definition. All strings for captions for configuration objects and guide page/guide view content will be displayed in the `Original` column in the **Translation Editor** or XLSX files of the relevant vocabulary.

- The strings for the following configuration objects require additional configuration steps in order to make the relevant strings available in the `METAMODEL` vocabulary:
- Enumerations. For more information, see the section below Understanding the Configuration Required to Translate Enumerations.
- Object states. For more information, see the section below Understanding the Configuration Required to Translate Object States.
- Release statuses. For more information, see the section below Understanding the Configuration Required to Translate Release Statuses.
- Lifecycle phases. For more information, see the section below Understanding the Configuration Required to Translate Lifecycle Phases.
- Indicators (of type `Text`). For more information, see the section below Understanding the Configuration Required to Translate Semantic Indicators.
- User profiles: The translated user profile name will be displayed in the masthead of the Alfabet user interface as well as in the **Change User Profile** menu in the <Alfabet User Name> menu in the main toolbar. Please note however that the translation of the user profile name will not be displayed in the **User Profiles Administration** and **User Administration** functionalities in the Alfabet user interface as well as the **User Profiles** node in Alfabet Expand.
- Workflows and wizards: The captions, comments, and user messages configured for workflows, workflow steps, wizards, and wizard steps are public strings and may be translated. For more information about the additional step required to translate workflows, see the section Understanding the Configuration Required to Translate Workflows and Wizards.
- HTML in custom editors and object cockpits. For more information, see the section below Translating HTML Defined in Custom Editors and Object Cockpits.
- Configured reports: The caption and description configured for custom reports are public strings and may be translated. For more information about the additional configuration required to translate configured reports, see the section Understanding the Additional Configuration Required to Translate Configured Reports.
- Text templates. For more information, see the section below Translating Text Templates Configured for Email Notifications.
- `ITPlan`: Strings derived from the standard solution code. These strings are protected strings and may be modified, if necessary. Please note that only a subset of strings derived from code-triggered functions are available in the vocabulary and may be translated.
- `Platform`: Strings from the platform code. This includes underlaying functionality and error messages. These strings are protected strings and may be modified, if necessary. Please note that

only a subset of strings derived from code-triggered functions are available in the vocabulary and may be translated.

Some error messages such as those sourced by Microsoft® .NET Framework are not part of the standard translation and will typically be displayed in English regardless of the language selected to display the Alfabet user interface.

- `GUIDEPAGES`: Strings defined in the context of guide page configured by the solution designer. These strings are public strings and may be translated. Please note that the translation of guide pages requires additional configuration steps in order to make the relevant strings available in the vocabulary. For more information, see the section below Translating Guide Pages and Guide Views. For detailed information about the configuration of guide pages, see the reference manual *Designing Guide Pages for Alfabet*.

Please note that the strings specified for guide pages will be extracted to the `GUIDEPAGES` vocabulary and the strings specified for guide views will be extracted to the `METAMODEL` vocabulary.

Custom strings for guide pages and guide views must be captured in English, regardless of the primary culture definition. All strings for captions for configuration objects and guide page/guide view content will be displayed in the `Original` column in the **Translation Editor** or XLSX files of the relevant vocabulary.

The following information is available:

- Modifying the Original Strings Provided by Software AG
- Manually Translating Custom Strings from Your Solution Configuration
- Understanding the Configuration Required to Translate Enumerations
- Understanding the Configuration Required to Translate Object States
- Understanding the Configuration Required to Translate Release Statuses
- Understanding the Configuration Required to Translate Lifecycle Phases
- Understanding the Configuration Required to Translate Semantic Indicators
- Understanding the Configuration Required to Translate Workflows and Wizards
- Updating the Translation of Workflow Templates
- Translating HTML Used in Workflows and Wizards
- Translating HTML Defined in Custom Editors and Object Cockpits
- Understanding the Additional Configuration Required to Translate Configured Reports
- Translating the Caption and Description of a Configured Report
- Translating Enumerations, Object States, Release Statuses, and Indicator Values in a Configured Report
- Translating Column Headers or Captions of a Configured Report
- Translating Text Templates Configured for Email Notifications
- Translating Guide Pages and Guide Views

Modifying the Original Strings Provided by Software AG

You can modify the standard strings provided for English or a supported secondary language and provide alternative terminology that is relevant for your enterprise. A custom term may be provided for any public or protected string vocabulary, allowing you to redefine terms in the vocabulary to match the specific terminology used in your company. For example, if the English term `software` is used instead of `application` in your enterprise, you can customize all strings in the `Original` vocabulary and replace the term `application` with the word `software`. Likewise, you could modify messages or strings displayed in the user interface in order to customize the information.

The changes to the vocabularies must be saved to the database. If an entry is not made for a string in the custom terminology, the term in the standard `English (United States)` vocabulary will be used. In this way, you only need to modify those terms that deviate from the standard Alfabet terminology. If you change a string, your text will be displayed instead of the original Alfabet term on the Alfabet interface. If you do not modify a term, the original English terminology will be used.

The changes can be made directly in the **Translation Editor** of Alfabet Expand or in an exported XLSX or XML (*VOC) file. In this case, you would export the vocabulary, edit the terminology in the XLSX file, and import the altered file to the Alfabet database. For more information about using the **Translation Editor**, see the section XXXTranslationInCustomEditor. For more information about exporting the vocabulary and translating the strings in an XLSX or XML file, see the section Translating Strings via XLSX files, XML (*VOC), or Simple Text Files.

To define alternative terms in English, you would need to do one of the following:

- Create a new culture with a base culture that references the same language as the culture with the terms you are modifying. For example, to replace the English term `application` with the term `software`, you could create a new culture with a base culture that uses English such as `English (Australia)` or `English (Zimbabwe)`. Other than the **Base Culture** attribute, all other culture attributes should be identical to the standard `English (United States)` culture (with the exception of the **Icon** attribute, which will allow you to specify a custom icon to display for the custom terminology in the Alfabet interface).

- Search for all instances of the term `application` in the new vocabulary and modify the term as needed. Please note that if an entry is not made for a string in the custom terminology, the term in the standard `English (United States)` vocabulary will be used.
- Save the modified vocabulary to the database and review the translated strings in the Alfabet interface.

To create new terms in any supported secondary language, you would need to do one of the following:

- Modify the translated strings directly in the vocabulary for the secondary language (for example, for `German (Germany)`. Any modified terms will be displayed as custom translations in the **Translation Editor**. Save the modified vocabulary to the database and review the translated strings in the Alfabet interface.
- Create a new culture and modify strings with the new terminology. The following steps are required to do this:

- Create a new culture for the language you want to modify. For example, to alter the vocabulary for `German (Germany)`, you could create a new culture for `German (Schweiz)`.
- Export the vocubularies to an XLSX files for the supported secondary language. For example, export the `MetaModel` vocabulary for `German (Germany)`.
- In the Excel file, copy all strings from the `Translation` column to the `Custom Translation` column.
- In the name of the XLSX file, replace the locale ID with the locale ID of the new culture. For the culture `German (Schweiz)`. For example, you want to change the file name the exported `MetaModel` vocabulary for `German (Germany)`, which is `AlfaVoc_1031_METAMODEL` to `AlfaVoc_2055_METAMODEL`.
- Import the modified XLSX file to Alfabet.
- In the **Translation Editor**, check that the new strings are available in the secondary language. Modify the translated strings as needed to.
- Save the modified vocabulary to the database and review the translated strings in the Alfabet interface.

Translation can be performed during runtime of the Alfabet Web Application with an Alfabet Expand instance directly connecting to the Alfabet database with a server alias in parallel to the Alfabet Web Application.

Updates to the translation table require the Web server to be restarted or the meta-model to be reread if the server is running in Design mode (if in the `web.config` file the key attribute "`appmode`" is set to `Design`).

Manually Translating Custom Strings from Your Solution Configuration

The custom strings that your enterprise has created when configuring the class mode may be manually translated for any of the languages supported by Software AG. If only one culture is defined for your enterprise, then the manual data translation functionality will be disabled in the Alfabet user interface. Translations can be created for the custom strings defined for custom editors, custom selectors, custom object views/object cockpits, workflows, custom wizards, configured reports, guide pages/guide views, etc. Most

of the terminology displayed in the Alfabet interface is translated in the vocabularies available with your Alfabet product.

Custom strings for configuration objects (such as custom properties, custom editors, configured reports, etc.) must be captured in English, regardless of the primary culture definition. All strings for captions for configuration objects and guide page/guide view content will be displayed in the `Original` column in the **Translation Editor** or XLSX files of the relevant vocabulary.

The vocabularies can either be translated in the **Translation Editor** in Alfabet Expand or exported and translated XLSX files or XML (*VOC) files. A translated string will be displayed instead of the original Alfabet string in the Alfabet interface. If a translation does not exist for a string, the original English string will be used.

The strings for the following configuration objects are NOT added to the vocabularies and therefore require separate treatment in order to be made available in a secondary language:

- Text templates: Text templates used in the context of the email capability. For more information about providing translated text templates, see the section Translating Text Templates Configured for Email Notifications.
- HTML templates: HTML templates used in the context of workflow and wizard configuration. For more information about providing translated HTML templates, see the section Understanding the Configuration Required to Translate Workflows and Wizards.
- Publications: Publications of data in Microsoft® Word® format. For more information about provided translated publications, see the section Configuring Alfabet Functionalities Implemented in the Solution Environment in the chapter Publishing Data In Microsoft® Word Format.
- Object data: The translation of the **Name** and **Description** properties of Alfabet objects are created and maintained in Alfabet. The translation of the name and description of such objects like applications, components, etc. is made by users in the context of the editors where these objects are edited. For more information about the translation of names and descriptions of Alfabet objects, see the section Configuring the Translation of Object Data.

Updates to the translation table require the Web server to be restarted or the meta-model to be reread if the server is running in Design mode (if in the `web.config` file the key attribute "`appmode`" is set to `Design`).

The following information is available:

- Understanding the Configuration Required to Translate Enumerations
- Understanding the Configuration Required to Translate Object States
- Understanding the Configuration Required to Translate Release Statuses
- Understanding the Configuration Required to Translate Lifecycle Phases
- Understanding the Configuration Required to Translate Semantic Indicators
- Understanding the Configuration Required to Translate Workflows and Wizards
- Updating the Translation of Workflow Templates

Understanding the Configuration Required to Translate Enumerations

An enumerations is a set of preconfigured enumeration items available to users when defining an object class property in Alfabet. An enumeration is associated with a standard or custom property. Each enumeration will have multiple enumerations items which represent the values that can be selected by users to define the custom property. The enumeration items can be translated as needed. For more information about the definition of custom and protected enumerations, see the section Defining Protected and Custom Enumerations section in the chapter Configuring the Class Model

Translation support is available for the preconfigured values available for enumeration items, which can be modified, as well as for the enumeration items created in the context of custom properties. Please note that the **Translatable in Meta-Model Vocabulary** for the custom property that the enumeration is assigned to must be set to `True` so that the strings defined for the enumeration items will be available in the `METAMODEL` vocabulary for translation.

The translated enumeration values will only be displayed on the Alfabet interface if the following requirements are met.

- The **Extract for Translation** attribute for the enumeration must be set to `True` and the **Enable Data Translation** attribute for the custom or protected property that the enumeration is assigned to must also be set for to `True`. This mechanism allows the enumeration to be translated for one object class property of the type `String` and displayed in the original language for another object class property.
- For object profiles: For an enumeration based on a custom or protected property displayed in an object profile, the attribute **Enable Data Translation** must be set to `True` for the property that has been added to the **Attributes** section of the object profile.

- For object cockpits: For an enumeration based on a custom or protected property displayed in an object cockpit, the attribute **Enable Data Translation** must be set to `True` for the Value Control interface control associated with the object class property.
- For configured reports: Instructions must be specified in order to display the translated enumeration values. This is described in detail in the section below Translating Enumerations, Object States, Release Statuses, and Indicator Values in a Configured Report.

Please note that properties of the type `StringArray` cannot be translated.

Translations of strings for enumeration items are NOT used in the standard history tracking information: In the standard history tracking report provided by Software AG, enumerations are not translated. In order to view history tracking information with translated enumerations, you should create a configured report displaying the history tracking information. For more information about configuring history tracking-related configured reports, see Defining Audit Management Related Configured Reports in the chapter Configuring Reports.

Understanding the Configuration Required to Translate Object States

The captions of the standard values available for object states and their hints can be modified by the solution designer in the XML object ***ObjectStateManager***. For more information about the configuration of the values available for object states, see the section Configuring Object State Definitions for Object Classes in the chapter Configuring the Class Model.

Translation support is available for the values defined in the XML element `ObjectStateDef` in the XML object ***ObjectStateManager*** and the strings will be available in the `METAMODEL` vocabulary for translation. However, please note that the translated strings for object state values will only be displayed on the Alfabet interface if the following requirements are met.

- For object profiles: The attribute **Enable Data Translation** must be set to `True` for the `ObjectState` property that has been added to the **Attributes** section of the object profile.
- For object cockpits: The attribute **Enable Data Translation** must be set to `True` for the Value Control interface control associated with the `ObjectState` property.
- For configured reports: Instructions must be specified in order to display the translated enumeration values. This is described in detail in the section below Translating Enumerations, Object States, Release Statuses, and Indicator Values in a Configured Report.

For a small subset of standard selectors used in the context of the editor search fields as well as the **Simple Search** functionality, the translated strings for the values for object state values cannot be entered in the secondary language rendered in the Alfabet interface but must rather be entered in the English original. These search selectors include: `COM_SelectorDef`, `ICTO_SelectorDef`, `ICTO_ICTOV_SelectorDef`. Similarly, the search for object class stereotypes must be entered in English in the following selectors: `DOM_SelectorDef`, `DVC_SelectorDef`, `ICTO_SelectorDef`, `SRVPRD_SelectorDef`, `VMND_SelectorDef`.

Translations of strings for values defined in the XML element `ObjectStateDef` in the XML object ***ObjectStateManager*** are NOT used in the standard history tracking information: In the standard history tracking report provided by Software AG. In order to view history tracking information with translated values for `ObjectState` properties, you should create a configured report

displaying the history tracking information. For more information about configuring history tracking-related configured reports, see Defining Audit Management Related Configured Reports in the chapter Configuring Reports.

Understanding the Configuration Required to Translate Release Statuses

Translation support is available for the values configured for release statuses. The captions and the hints specified for release statuses can be configured by the solution designer in the XML object ***ReleaseStatusDefs***. For more information about the configuration of release statuses, see the section Configuring Release Status Definitions for Object Classes in the chapter Configuring the Class Model.

Translation support is available for the values defined in the XML element `ReleaseStatusDef` in the XML object ***ReleaseStatusDefs*** and the strings will be available in the `METAMODEL` vocabulary for translation. However, please note that the translated strings for release status values will only be displayed on the Alfabet interface if the following requirements are met.

- For object profiles: The attribute **Enable Data Translation** must be set to `True` for the `Status` property that has been added to the **Attributes** section of the object profile.
- For object cockpits: The attribute **Enable Data Translation** must be set to `True` for the Value Control interface control associated with the `Status` property.
- For configured reports: Instructions must be specified in order to display the translated enumeration values. This is described in detail in the section below Translating Enumerations, Object States, Release Statuses, and Indicator Values in a Configured Report.

For a small subset of standard selectors used in the context of the editor search fields as well as the **Simple Search** functionality, the translated strings for the values for release status values cannot be entered in the secondary language rendered in the Alfabet interface but must rather be entered in the English original. These search selectors include: `COM_SelectorDef`, `ICTO_SelectorDef`, `ICTO_ICTOV_SelectorDef`. Similarly, the search for object class stereotypes must be entered in English in the following selectors: `DOM_SelectorDef`, `DVC_SelectorDef`, `ICTO_SelectorDef`, `SRVPRD_SelectorDef`, `VMND_SelectorDef`.

Translations of strings for values defined in the XML element `ReleaseStatusDef` in the XML object ***ReleaseStatusDefs*** are NOT used in the standard history tracking information: In the standard history tracking report provided by Software AG. In order to view history tracking information with translated values for `Status` properties, you should create a configured report displaying the history tracking information. For more information about configuring history tracking-related configured reports, see Defining Audit Management Related Configured Reports in the chapter Configuring Reports.

Understanding the Configuration Required to Translate Lifecycle Phases

Translation support is available for the preconfigured values available for lifecycle definitions. The captions and the hints specified for lifecycle phases can be configured by the solution designer in the XML object ***ObjectLifeCycleManager***. For more information about the configuration of object lifecycles, see the section Configuring Lifecycle Definitions for Object Classes in the chapter Configuring the Class Model.

Translation support is available for the values defined in the XML element `ObjectLifecycle` in the XML object ***ObjectLifeCycleManager*** and the strings will be available in the `METAMODEL` vocabulary for translation. However, please note that the translated strings for lifecycle phases will only be displayed on the Alfabet interface if the following requirements are met.

- For object profiles: The attribute **Enable Data Translation** must be set to `True` for the relevant `TimeStatus` property that has been added to the **Attributes** section of the object profile.
- For object cockpits: The attribute **Enable Data Translation** must be set to `True` for the Value Control interface control associated with the relevant `TimeStatus` property.
- For configured reports: Instructions must be specified in order to display the translated enumeration values. This is described in detail in the section below Translating Enumerations, Object States, Release Statuses, and Indicator Values in a Configured Report.

Translations of strings for values defined in the XML element `ObjectLifecycle` in the XML object ***ObjectStateManager*** are NOT used in the standard history tracking information: In the standard history tracking report provided by Software AG. In order to view history tracking information with translated values for `TimeStatus` properties, you should create a configured report displaying the history tracking information. For more information about configuring history tracking-related configured reports, see Defining Audit Management Related Configured Reports in the chapter Configuring Reports.

Understanding the Configuration Required to Translate Semantic Indicators

An indicator is the value defined for an indicator type in the context of an object in the IT landscape. In some cases, your company may configure a range of permissible indicator values that may be defined for each indicator type. These are semantic indicators that users will manually enter values for. These values are configured in the **Range** property in the **Indicator Type** editor.

All valid values for manually-entered indicators for an indicator type must be written in accordance with the following convention: <NumericalValue>-<SemanticValue> (for example, `1-low, 2-medium, 3-high`). For more information about the configuration of a range of indicators for an indicator type, see the section *Configuring Evaluations, Prioritization Schemes, and Portfolios* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.

Translation support is available for the preconfigured values available for the semantic values configured for the **Range** property for indicator types and the strings will be available in the `METAMODEL` vocabulary for translation. The text specified for the semantic indicators and defined in the **Range** property for an indicator type will be available in the `METAMODEL` vocabulary for translation. Please note the following:

- The translation of the range values may not exceed 128 characters.
- The numerical value and the hyphen are not added to the vocabulary.
- If icons are specified to visualize the range values for an indicator type, the captions of the icons will be added to the vocabulary and can be translated.

However, please note that the translated strings for semantic indicators will only be displayed on the Alfabet interface if the following requirements are met.

- For object profiles: The attribute **Enable Data Translation** must be set to `True` for the `IndicatorType` property that has been added to the **Attributes** section of the object profile.

- For object cockpits: The attribute **Enable Data Translation** must be set to `True` for the Value Control interface control associated with the `IndicatorType` property.
- For configured reports: Instructions must be specified in order to display the translated enumeration values. This is described in detail in the section below Translating Enumerations, Object States, Release Statuses, and Indicator Values in a Configured Report.

Translations of strings for indicators are NOT used in the standard history tracking information: In the standard history tracking report provided by Software AG, enumerations are not translated. In order to view history tracking information with translated indicators, you should create a configured report displaying the history tracking information. For more information about configuring history tracking-related configured reports, see Defining Audit Management Related Configured Reports in the chapter Configuring Reports.

Understanding the Configuration Required to Translate Workflows and Wizards

The captions, descriptions, technical comments, and user messages configured for workflows, workflow steps, workflow step actions, and pre- and post-conditions can be made available in the vocabularies and translated to a secondary language for the user community. In the case of workflows, an explicit update of the strings associated with workflow templates must be carried out in order to update the relevant strings to the vocabulary.

If HTML templates are configured for either workflows or wizards, these well not be added to the vocabulary but must be explicitly translated in the context of the HTML template.

The following information is available:

- Updating the Translation of Workflow Templates
- Translating HTML Used in Workflows and Wizards

Updating the Translation of Workflow Templates

Captions, descriptions, technical comments, and user messages configured for workflows, workflow steps, workflow step actions, and pre- and post-conditions can be translated to a secondary language for the user community. In the case of workflows, an explicit update of the strings associated with workflow templates must be carried out regardless of the value of the **Workflow State** attribute for the workflow template.

To update the translated strings available in the vocabularies for all workflow templates, click **Globalization** > **Update Workflow Translation** in the menu bar of Alfabet Expand. A message will be displayed stating that the translated strings available for workflow templates, workflow steps, workflow step actions, and pre- and post-conditions have been updated for all workflow templates.

It is also possible to update individual workflow templates for which the **Workflow State** attribute of the workflow template is set to `Active`. In this case, right-click the workflow template and select **Update Translation**. A message will be displayed stating that the translated strings available for the selected workflow template and its workflow steps, workflow step actions, and pre- and post-conditions have been updated.

Please note that the workflow step states (`Confirmed`, `Pending`, `Running`, `Finished`, etc.) cannot be translated.

Translating HTML Used in Workflows and Wizards

Configurable HTML templates based on XHTML using HTML5 standards may be implemented in the context of workflows. The HTML templates contain basic elements to capture information about a workflow step and are used to render the specified data in the **Workflow Activities Explorer** functionality (`WFS_Explorer`) as well as in custom explorers configured for workflow activities. For more information about the configuration of the **Workflow Activities Explorer** functionality, see the section Specifying the Customized Workflow Activities Explorer (WFS_Explorer) or a Custom Explorer in the chapter Configuring Workflows.

In the case of wizards, HTML text based on XHTML using HTML5 standards may be used to display information in the header of a wizard step. The HTML may contain basic elements to capture information about the wizard step as well as the object targeted by the wizard step and specify the layout and visualization of the data. For more information about the configuration of the header for a wizard step, see the section Configuring the Header Text of a Wizard Step in the chapter Configuring Wizards.

If the text defined in the HTML implemented for workflows and wizards is to be translated, you must provide the translation directly in the HTML specification.

For example:

```
<!DOCTYPE HTML>
<xhtml>
    <culture_1031 >
        <html>
        ..
        </html>
    <culture_1031>
    <culture_1033>
        <html>
        ..
        </html>
    </culture_1033>
</xhtml>
```

Please note the following:

- The element `<culture_xxx>` must be specified as a child element of the root element `<xhtml>`, whereby the `_XXX` specifies the Alfabet locale ID.
- A element `<culture_xxx>` should be created for each language for which a translation should be available.

- Each <culture_xxx> contains the <html> element with the HTML specification in the relevant language.

Translating HTML Defined in Custom Editors and Object Cockpits

An HTML Control interface element can also be added to a custom editor to display text with HTML formatting (font and color). The code must start with an <html> tag and end with </html>. The definition of <head> and <body> specification is optional. The HTML code can include CSS style definitions and links to external URLs. The inclusion of images is not supported.

If multiple languages are used in the Alfabet interface, a language version must be defined directly in the HTML defined in HTML control of the custom editors with an element <Culture_xxx> whereby the_XXX specifies the Alfabet locale ID. No root element is required. Each <culture_xxx> contains the <html> element with the HTML specification in the relevant language. The translated text can be captured in the relevant locale ID element <culture_xxx>.

Understanding the Additional Configuration Required to Translate Configured Reports

If your enterprise requires the Alfabet interface to be displayed in any of the secondary supported languages, then you must explicitly ensure that the data displayed in configured reports is also translated to the relevant language. The following must be carried out in order to translate the content of a configured report:

- For each report configured in Alfabet Expand, the **Caption** and **Description** attributes will be displayed in the Alfabet interface. These attributes can be translated as described below in the section Translating the Caption and Description of a Configured Report.
- For relevant configured reports configured in Alfabet Expand, the column headers of tabular reports are created based on either the alias definitions in the Show properties of the Alfabet query, the SELECT clause of the native SQL query, or the instructions defined for the query. The column headers can be translated as described in the section Translating Column Headers or Captions of a Configured Report.
- String values in a graphic report that have been defined directly in an attribute of the report assistant are usually added automatically to the vocabulary files. This includes, for example, the captions for lanes in lane reports, the names of color rules and indicator rules that are used as legend headers, static text definitions in widget reports, or the graphic title, X-axis title, and Y-axis title of business chart reports or portfolio reports. For gauge and map chart reports, this also includes the captions defined for color range members. For general information about vocabulary translation, see Modifying, Translating and Managing the Vocabularies.
- Text defined for the cell tooltip in the LinkAssignment_Edit instruction as well as text defined as legend text in the Alfabet query language instructions ColorAssignment, PictureAssignment, RowColorAssignment, FontStyleAssignment, FontStyleColorAssignment are added automatically to the vocabulary files. For general information about vocabulary translation, see Modifying, Translating and Managing the Vocabularies.
- In the case of configured portfolio, lane, and grid reports, the graphic title, X-axis title, and Y-axis will not be added automatically to the vocabulary files. Texts provided for the calculation of a total value for pie charts will also not be added automatically to vocabulary files. These values can be translated as described in the section Translating Column Headers or Captions of a Configured Report.

Please note that the labels displayed in configured portfolio charts and gauge reports and the labels for enterprise milestones in configured Gantt chart reports cannot be translated.

If configured reports are translated to Arabic, you should be aware that the text is right-aligned instead of left-aligned in user interface elements. For example, for the display of a **Caption** defined for a **Lane** in a lane report, you will see the text above the lanes instead of above the object boxes because the space for the caption spans both object boxes and lanes. To display a caption above the object boxes, you must define the caption in the **Caption** attribute of the **Node** element below the **Lane** element.

If the Alfabet user interface is rendered in Arabic language, the order of columns and display of graphic elements in graphic reports is changed to right to left instead of left to right, as for all other supported languages. Only configured reports based on the template `MatrixMapReport` that have the subtype `Diagram` and Pivot Grids rendered with the embedded DevExpress® component are still displayed left to right.

- For Alfabet objects displayed in a configured report, the translation of relevant object properties is read from the database tables. The following object class properties are translatable and can be displayed in a secondary language:

- Predefined protected properties such as the object's **Name** and **Description** properties as well as custom properties of the type `String` and `Text`. These properties will be displayed in the target language if the **Support Data Translation** attribute of the relevant culture is set to `True`, and a translation has been made in the relevant object class editor in the Alfabet user interface. The translated values for object data will be displayed automatically in configured reports based on Alfabet query language. For native SQL queries, however, a special code triggering the translation must be added to native SQL queries in order to enable the display of translation values for object data. For more information, see Displaying Translated Object Data In the Current Language of the User Interface in the chapter Defining Queries. For more information about object data translation, see the section Configuring the Translation of Object Data.

- The caption of the object class and object class stereotype. These captions can be added to a configured report via instructions. If this is the case, the captions of the object class and object class stereotype can be displayed in a target language if they are translated in the vocabularies available in Alfabet Expand. For general information about vocabulary translation, see Configuring the Translation of Object Data.

- Values defined for object states, release statuses, indicator values, enterprise releases, project milestones, and enumerations. These values can be displayed in a target language if the following applies:

- The **Enable Data Translation** attribute of the relevant object class property (properties of the type `String` or `Text`) is set to `Manual` or `ManualAndAutomated`.

- Translations are provided in the vocabularies available in Alfabet Expand. For more information about vocabulary translation, see Configuring the Translation of Object Data.

- The values are not automatically displayed translated in configured reports. An instruction triggering the translation must be added to queries to enable the display of translation values. For more information, see Translating Enumerations, Object States, Release Statuses, and Indicator Values in a Configured Report.

Status values that are editable in the **Report** editor in the **Reports Administration** functionality available in the Alfabet interface is neither a release status, object state, nor based on an enumeration and therefore cannot be translated in the **Report** editor.

Translating the Caption and Description of a Configured Report

The **Caption** and **Description** attributes will be displayed in the Alfabet interface for each report configured in Alfabet Expand. These attributes can be translated to the languages supported by your enterprise.

Once the **Caption** and **Description** attributes have been specified for a configured report, they will be automatically added to the vocabularies available in Alfabet Expand. Once they are available in the vocabularies, they can be translated like any other string in the **Translation Editor** or exported to a Microsoft Excel file and translated there. For more information about this process, see the section Modifying, Translating and Managing the Vocabularies.

For technical reasons, the translation of report captions and descriptions is coupled with the mechanisms of object data translation. Therefore, the translation provided via the vocabularies is only displayed in the Alfabet interface if the following applies:

- The **Support Data Translation** attribute must be set to `True` for the respective culture.
- The translation must be updated in the data translation columns of the database table of the object class `Report`. To write the translated **Caption** and **Description** attributes to the database table of the object class `Report`:
  1) Set the **Report State** attribute for the configured report to `Plan`.
  2) Right-click the configured report and select **Update Translation**.

     To batch update the translation of multiple configured reports, right-click the **Reports** node and select **Update Translation**. The captions and descriptions for all configured reports for which the **Report State** attribute is set to **Plan** will be updated.

  3) Set the **Report State** attribute for the configured report to `Active`.

Translating Enumerations, Object States, Release Statuses, and Indicator Values in a Configured Report

Configured reports require the specification of instructions to display the translated values. Two new Alfabet instructions are available that trigger the display of translated values for indicator ranges, enumerations, object states, and status definition values:

- To display enumeration, object state, and release status values in the translated language, the following Alfabet instruction must be added to the query. The `ColumnName` is the name of the column containing the information that shall be translated:

  ```
  TranslateEnums("ColumnName,ColumnName,..");
  ```

- To display indicator range values in the translated language, the following Alfabet instruction must be added to the query. The `ColumnName` is the name of the column containing the information that shall be translated:


```
TranslateIndicators("ColumnName,ColumnName,..");
```

For detailed information about the definition of the Alfabet instructions see Displaying Translated Values of Enumerations, Object States, Statuses, Milestones and Indicators in Configured Reports in the section Using Instructions to Format the Results of an Alfabet or Native SQL Query.

Translating Column Headers or Captions of a Configured Report

The definition of column header or graphic report elements in a native SQL query or Alfabet query is typically a complex process. This is the case, for example, for a native SQL query containing a `WITH` statement or if instructions are used for renaming and restructuring the results of the query. In such cases, it is not possible to automatically scan the query definitions for strings that need to be translated.

Therefore, if you want to translate column headers, you must write the column header strings in the XML object ***VocXML***. The strings defined in the XML object ***VocXML*** are automatically added to the meta-model section of the vocabularies and can be translated to a defined target language as described in the section Modifying, Translating and Managing the Vocabularies.

Strings that are defined directly in report assistants such as the caption of lane reports or strings that are explicit in the query definition such as column headers defined in `SetColumnShowName` instructions are automatically added to the translation tables. You can check the vocabularies prior to adding strings to the XML object ***VocXML*** to evaluate which strings have already been added automatically to the vocabulary.

To add column headers to the translation table of the Alfabet vocabularies:

1) In the **Presentation** tab of Alfabet Expand, expand the **XML Objects** node and double click the XML object ***VocXML***. The XML object ***VocXML*** opens in the center pane. The editor automatically displays the root XML element `VocXml` of the XML object.
2) For each column header, enter a child element `Entry` to the `VocXml` root element. In the `Entry` element, write the string of the column header in the original language.

Alfabet vocabularies are string-based. Therefore, if two configured report have the same column header, you must only add the string once to the XML object t ***VocXML***. If the header column string is identical to a string that is used somewhere else in the configuration of the Alfabet meta-model (for example, if the header column is identical to the caption of an object class property), the string will already exist in the vocabulary and does not need to be entered in the XML object t ***VocXML***. If you enter a duplicate string in the XML object ***VocXML*** it will be ignored.

3) In the toolbar, click the **Save**  button to save your changes.

For example, a configured report is based on the following native SQL query:

```
SELECT proj.REFSTR, proj.NAME AS 'Project Name',proj.STEREOTYPE
FROM PROJECT proj
```

The query is combined with the following Alfabet instructions:

```
InsertColumn("1","StereotypeCaption");
SetColumnShowName("StereotypeCaption","Project Stereotype");
```



```
SetStereotypeCaption("REFSTR","STEREOTYPE", "StereotypeCaption");
RemoveColumns("STEREOTYPE");
```

The resulting configured report will have two column headers:

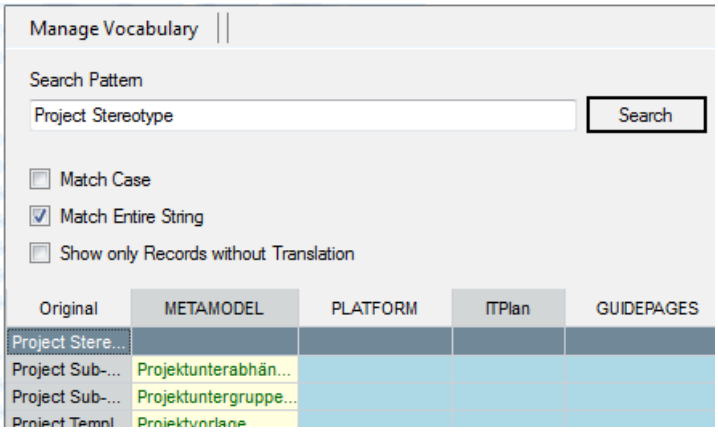
- "Project Name", derived from the alias definition in the `SELECT` clause of the native SQL query
- "Project Stereotype" defined via the `SetColumnShowName` instruction.

	Project Name	Project Stereotype
1	Consolidate Trading Applications	Program
2	Retire GL Applications	Project
3	Upgrade GenLManager	Project
4	Reshape Core Trading Applications	Project
5	Enhance TradeNet	Project
6	Implement Unified Trade Solution	Project
7	UTS phase 1	Project Step
8	UTS phase 2	Project Step
9	UTS phase 3	Project Step
10	Streamline CRM Applications	Program
11	Migrate CRM Opti Retail to CRM CSS	Project
12	Integrate CRM with SAP	Project
13	Retire GL Applications Solution	Project

To translate the headers of the configured report, the two definitions must be written in the XML object ***VocXML***:

```
<VocXml>
    <Entry>Project Name</Entry>
    <Entry>Project Stereotype</Entry>
</VocXml>
```

The terms are then translated to the target language (in this case, German) in the **Translation Editor** of Alfabet Expand:



The translations will be displayed if German is selected as the base culture for th---e culture displayed in the Alfabet interface.

	Projektname	Projekt-Stereotyp
1	Consolidate Trading Applications	Programm
2	Retire GL Applications	Projekt
3	Upgrade GenLManager	Projekt
4	Reshape Core Trading Applications	Projekt
5	Enhance TradeNet	Projekt
6	Implement Unified Trade Solution	Projekt
7	UTS phase 1	Projektschritt
8	UTS phase 2	Projektschritt

Please note: You can see from the example that the content of the configured report is also only partially translated. The project names are displayed in English because data translation has not yet been activated. The stereotype captions are automatically added to the vocabulary and have been translated. Therefore, the translated values are displayed in the configured report. For more information about data translation, see the section Configuring the Translation of Object Data.

Translating Text Templates Configured for Email Notifications

A text template is the predefined text in an email that is automatically generated as the result of an action triggered by a user. The text template defines the email text in English, the relevant references to objects and object class properties in Alfabet, and hyperlinks to the relevant objects in Alfabet.

The text defined in a text template will not be added to the vocabulary but must be explicitly translated in the context of the text template. For this purpose, locale text templates can be created in order to translate the content of the text template to the secondary languages implemented in your solution interface. It is possible to create one or more locale text templates for each text template implemented in the enterprise. In other words, each text template required by the user community can be translated to all non-English languages that you will use in the Alfabet interface. The translation of the text template occurs directly in the locale text template configuration object.

The language displayed for a user's email notifications is specified by a user administrator in the **Email Notification Language** attribute for the relevant user in the **User Administration** functionality accessible via the `Admin` user profile. For more information, see the section *Defining and Managing Users* in the reference manual *User and Solution Administration*.

To create and translate a locale text template:

1) Go to the **Presentation** tab and expand the **Text Templates** node. Expand the relevant text template folder.
2) Right-click the text template that you want to create a language version for and select **Create Locale Text**. You will see a copy of the text template appear in the explorer tree below the original English-language text template.

   The locale text template must be stored in the appropriate folder with its parent text template. For example, all text templates for workflows must be stored in the `WF` folder, all text templates for consistency monitors must be stored in the `M_CON` folder, etc.

3) Click the new locale text template to open the attribute window and define the following attributes:

- **Locale Name**: Select the base culture specified in the culture that this local text template is translated for.
- **Text**: Edit the text of the email message as you would for any protected text template. Please note the following:
- The text displayed in the templates can be translated, as needed.
- Predefined variables as well as configurable object variables can be moved via copy and paste within a text template.
- The expression in curly brackets `{XXX}` in predefined variables may not be edited.
- For detailed information about specifying a text template, see the section *Editing a Protected Text Template*.

4) In the toolbar, click the **Save**  button to save your changes.

Translating Guide Pages and Guide Views

Guide page projects allow an enterprise to configure start pages with links and information for the enterprise's users in order to efficiently guide them to specific functionalities in the Alfabet interface. Both guide pages and guide views are HTML files that are created in the Guide Pages Designer. Once they have been configured, the guide page project must be uploaded to the Alfabet database. Once the guide pages and guide views have been uploaded to the Alfabet database, their strings will be available in the English original vocabularies and can be translated to the secondary languages implemented in your company.

Please note that the strings specified for guide pages will be extracted to the `GUIDEPAGES` vocabulary and the strings specified for guide views will be extracted to the `METAMODEL` vocabulary.

Custom strings for guide pages and guide views must be captured in English, regardless of the primary culture definition. All strings for captions for configuration objects and guide page/guide view content will be displayed in the `Original` column in the **Translation Editor** or XLSX files of the relevant vocabulary.

The translation of the guide pages can be carried out in the **Translation Editor** or in an XLSX file as described in the section Translating Strings via XLSX files, XML (*VOC), or Simple Text Files. For more information about creating and uploading guide page projects, see the reference manual *Designing Guide Pages for Alfabet*.

To delete all translated strings for all guide page vocabularies, click **Globalization** > **Delete Guide Pages Translation** in the menu bar of Alfabet Expand and confirm the message box.

Translating Strings via the Translation Editor

A **Translation Editor** is available in Alfabet Expand in which you can translate your custom strings or modify the standard strings that are displayed in the Alfabet interface. Before you can begin to translate, the culture must already be created, and the base culture must be specified with the correct locale. For more information about creating a culture setting, see the section Specifying the Cultures Relevant to Your Enterprise.

Alternative to modifying the strings in the context of the **Translation Editor**, the vocabulary file (.voc) for a translated language can be exported to an XML file or to a Microsoft Excel format. In this way, you could edit the vocabularies in the translation tools used in your company.

Translation can be performed during runtime of the Alfabet Web Application with an Alfabet Expand instance directly connecting to the Alfabet database with a server alias in parallel to the Alfabet Web Application.

All terminology displayed in the **Translation Editor** can be translated for the various cultures implemented by your enterprise. Translation and modification of the strings is done on a string-by-string basis. You can modify the strings in the vocabularies available for any of the supported languages.

The translation of languages not provided with the Alfabet product is not supported. The initial translation for a specific language must be provided by Software AG. Custom translations can only be created for existing vocabularies.

Extreme caution should be used when translating vocabularies: Please note the following important issues regarding the Alfabet vocabularies:

- Strings may not be longer than 600 characters. If a text is longer than 600 characters, it will NOT be stored in the Alfabet database.
- Do NOT edit or translate the following when found in the vocabularies:
- Variables enclosed in curly brackets: {ExampleVariable}. The text in curly brackets will be automatically generated by the system. Variables MUST remain untranslated and in curly brackets {..} so that they can be read by the software. Changing curly brackets or the variable inside the curly brackets may cause an error in the system.
- Compound words that have no empty spaces. For example: `CentraSiteManager`, `SearchManager`, `SolutionOptions`.
- Words written in all capital letters. For example: `WHERE` or `SELECT`.
- Terms enclosed in single quotes (‘arg’, property of the type ‘String’)
- Special characters used in the source language such as square brackets [ ], angle brackets < >, ampersand &, and backslash \ . For example, "`Application|Component`" can be customized to "`Program|Software`" but not to "`Program` and `Software`" and "save as *.html" can be customized to "format *.html" but not to "save as html file".
- If configuration objects created by your enterprise are deleted, their translations will NOT be automatically deleted from the vocabularies. However, during the meta-model update process, any translation entries that cannot be matched to an original will be deleted from the Alfabet database. An XLSX file will be generated during the meta-model update process that lists all orphaned translations that have been removed from the Alfabet database. Solution administrators should analyze this list to determine whether any of the orphaned translations require the creation of new (custom) translations.

To translate strings in the **Translation Editor**:

1) In the menu bar, click **Globalization** > **Edit Translation**. The **Translation Editor** opens.
2) A message is displayed asking you whether a scan for new originals shall be executed. If you click **Yes**, new original strings that are available due to changes to the solution configuration will be

scanned, updated in the `ALFA_SYS_VOCABULARY` database table, and available for translation. The **Translation Editor** will open once the scan has been completed. If you click **No**, the **Translation Editor** will open immediately but the original strings will not be updated.

3) In the **Translation Editor** toolbar, click **Manage Vocabulary** > **Open** and select the culture for which you want to translate the vocabularies. The target language for translation is displayed in the header of the **Translation Editor**. You will see the following in the **Translation Editor**:

- The `Original` column displays the English terms.
- For technical reasons, the translatable strings are divided into four separate vocabularies which constitute terminology sets based on their source of origin in the software. These are `METAMODEL`, `ITPlan`, `Platform`, `Extensions`, and `GUIDEPAGES`. The following columns are displayed:
- `METAMODEL`: Strings derived from the meta-model. This includes the following:
    - Protected strings from the standard configuration objects.
    - Custom strings resulting from the customization of the meta-model via Alfabet Expand.
- `ITPlan`: Strings derived from the standard solution code.
- `Platform`: Strings from the platform code.
- `Extensions`: Strings derived from the extensions code.
- `GUIDEPAGES`: Strings from the guide pages designed by the solution designer with the tool Guide Pages Designer provided by Software AG.

For a more detailed description of the content of the terminology sets, see the section Manually Translating Custom Strings from Your Solution Configuration.

- A legend is displayed at the bottom of the **Translation Editor** explaining the color-coding for the standard translation and the custom translation. Please note the following:
- White cells indicate that the `Original` string can be translated for the corresponding vocabulary. In some cases, the same `Original` string may be available in multiple vocabularies (`METAMODEL`, `ITPlan`, `Platform`, `Extensions`, and `GUIDEPAGES`.). Each entry may have a different translated string. Typically, translations will be available with the standard Alfabet product. The standard translation can be modified but not deleted, but it should not be overwritten unless there is an explicit reason. If you attempt to delete these terms, they will remain in the Alfabet database and will continue to be available in the **Translation Editor**. The standard translation will be reinstantiated if a custom string that replaced it is deleted.
- Yellow cells indicate that the string is a custom string. The cell will only be colored yellow if the string has been saved to the database. The custom translations defined for configuration objects that have been created for your solution (custom properties, custom editors, etc) can be modified or deleted from the vocabularies.
- Blue cells indicate that no translation is required for the `Original` string for the corresponding vocabulary.
- To sort the terminology alphabetically in each column, click the column caption. The sort can be either a descending or ascending alphabetical order. Undefined terminology is displayed before terms starting with "A". To change the sort order, re-click the column caption. The translatable term will be displayed in a white cell in the column displaying the terminology set from which it originates.

- To find a specific term, enter the complete term in the **Search Pattern** field. You can use the star symbol * as a wildcard. Define the following filters as needed and click the **Search** button:
- **Match Case**: Select the checkbox if only strings with matching capitalization should be displayed
- **Match Entire String**: Select the checkbox if only strings containing the entire string should be displayed.
- **Show Only Strings with No Translation**: Select the checkbox to display all strings that have no translation defined. All strings with empty cells in the vocabulary columns will be displayed.
- **Show Only Strings with Automated Translation**: Select the checkbox to display all strings that have a translation that has been generated via the automated translation capability. These are the strings that are imported when the **Get Low-Confidence Translations** functionality is executed. For more information, see the section Translating Custom Strings via the Automated Translation Capability.
- The first entry matching the search criteria will be highlighted. Click the **Search** button to highlight the next match.

4) Translate the `Original` strings to the targeted culture, as needed. To translate a term, click the relevant cell displaying either a standard translation or custom translation and enter your translation of the term. Please note that if you edit a translation in a field in the **Translation Editor**, it will not be automatically saved. You must move the focus from the last edited field in the **Translation Editor** to any other field and then save the vocabulary.
5) After you have translated all necessary terms, click **Manage Vocabulary** > **Save to Database** to save the translation.
6) Click the **Close** button to close the **Translation Editor**.

Updates to the translation table require the Web server to be restarted or the meta-model to be reread if the server is running in Design mode (if in the `web.config` file the key attribute "`appmode`" is set to `Design`).

Translating Strings via XLSX files, XML (*VOC), or Simple Text Files

You might want to define your terminology in XLSX or XML (*VOC) files rather than directly in the **Translation Editor**. (*VOC files are Alfabet proprietary files in XML language used for the explicit purpose of importing and exporting vocabulary strings.) This would allow you, for example, to translate the terminology in the translation tools used by your company. In this case, you can export the vocabularies for a selected target language as either XLSX or XML (*VOC) files, translate the strings to the target language, and import the edited XLSX or XML (*VOC) files to Alfabet. A separate file will be generated for each terminology set (`METAMODEL`, `ITPlan`, and `Platform`, and `GUIDEPAGES`).

As an alternative to manually translating the empty strings in the vocabularies, you can configure a translation service such as Google Translate®, Amazon Translate®, or DeepL® Translator to automatically translate the empty strings. This is described in more detail in the section Translating Custom Strings via the Automated Translation Capability.

All terminology displayed in the **Translation Editor** can be translated for the various culture settings implemented by your enterprise. Translation and modification of terminology is done on a term-by-term basis. You can modify the terms in the vocabularies available for any of the supported languages.

Extreme caution should be used when translating vocabularies: Please note the following important issues regarding the Alfabet vocabularies:

- Strings may not be longer than 600 characters. If a text is longer than 600 characters, it will NOT be stored in the Alfabet database.
- Do NOT edit or translate the following when found in the vocabularies:
  - Variables enclosed in curly brackets: {ExampleVariable}. The text in curly brackets will be automatically generated by the system. Variables MUST remain untranslated and in curly brackets {..} so that they can be read by the software. Changing curly brackets or the variable inside the curly brackets may cause an error in the system.
  - Compound words that have no empty spaces. For example: `CentraSiteManager`, `SearchManager`, `SolutionOptions`.
  - Words written in all capital letters. For example: `WHERE` or `SELECT`.
  - Terms enclosed in single quotes ('arg', property of the type 'String')
  - Special characters used in the source language such as square brackets [ ], angle brackets < >, ampersand &, and backslash \ . For example, "`Application|Component`" can be customized to "`Program|Software`" but not to "`Program` and `Software`" and "save as *.html" can be customized to "format *.html" but not to "save as html file".
- If configuration objects created by your enterprise are deleted, their translations will NOT be automatically deleted from the vocabularies. However, during the meta-model update process, any translation entries that cannot be matched to an original will be deleted from the Alfabet database. An XLSX file will be generated during the meta-model update process that lists all orphaned translations that have been removed from the Alfabet database. Solution administrators should analyze this list to determine whether any of the orphaned translations require the creation of new (custom) translations.

The following information is available:

- Exporting the Vocabulary Files
- Translating the Vocabulary Files
- Importing the Vocabulary Files

Exporting the Vocabulary Files

To export Alfabet vocabularies to XLSX or XML (*VOC) files:

1) In the menu bar, click **Globalization** > **Open Vocabulary Manager**.

   To import and export simple text files displaying the unique identifier and the string (ID=Value), click **Globalization** > **Export Vocabularies with Unique Identifiers**. Once the strings have been modified, you can import them via **Globalization** > **Import Vocabularies with Unique Identifiers**.

2) A message is displayed asking you whether a scan for new originals shall be executed. If you click **Yes**, new original strings that are available due to changes to the solution configuration will be

scanned, updated in the `ALFA_SYS_VOCABULARY` database table, and available for translation. The **Vocabulary Manager** will open once the scan has been completed. If you click **No**, the **Vocabulary Manager** will open immediately but the original strings will not be updated.

3) In the **Vocabulary Manager**, select the source language and terminology set that you want to customize. You can select multiple terminology sets by holding down the SHIFT key while selecting.

All existing vocabularies are listed in the **Vocabulary Manager**. The **Source** column displays where the vocabulary string is stored. A `METAMODEL`, `ITPlan`, `Platform`, `Extensions`, and `GUIDEPAGES`. file is typically available for each culture supported by Software AG. The entry `Original` in the **Vocabulary** column contains the `English (United States)` vocabularies with no translated strings.

4) In the toolbar, click the **Export Vocabularies**  button. The **Translation Export** dialog box opens.

For more information about the **Get Low-Confidence Translations** menu, see the section Translating Custom Strings via the Automated Translation Capability.

5) Click the **Browse** button next to the **Target Folder** field and select the target directory for the export.

6) In the **Export Type** drop-down list, select `Excel` to export the selected terminology sets to XLSX files or select `Voc` to export the selected terminology sets to XML files.

The XML files will have the extension VOC.

7) Click **OK** to start the export of terminology. A message will be displayed indicating how many files were created. Click **OK** to confirm the message.

Translating the Vocabulary Files

You can translate the vocabulary in XLSX or XML (*VOC) files which are Alfabet proprietary files in XML language used for the explicit purpose of importing and exporting vocabulary strings. A separate file will be generated for each terminology set (`METAMODEL`, `ITPlan`, and `Platform`, and `GUIDEPAGES`).

Extreme caution should be used when translating vocabularies: Please note the following important issues regarding the Alfabet vocabularies:

- Strings may not be longer than 600 characters. If a text is longer than 600 characters, it will NOT be stored in the Alfabet database.
- Do NOT edit or translate the following when found in the vocabularies:
- Variables enclosed in curly brackets: {ExampleVariable}. The text in curly brackets will be automatically generated by the system. Variables MUST remain untranslated and in curly brackets {..} so that they can be read by the software. Changing curly brackets or the variable inside the curly brackets may cause an error in the system.

- Compound words that have no empty spaces. For example: `CentraSiteManager`, `SearchManager`, `SolutionOptions`.
- Words written in all capital letters. For example: `WHERE` or `SELECT`.
- Terms enclosed in single quotes (‘arg’, property of the type ‘String’)
- Special characters used in the source language such as square brackets [ ], angle brackets < >, ampersand &, and backslash \ . For example, "`Application|Component`" can be customized to "`Program|Software`" but not to "`Program` and `Software`" and "save as *.html" can be customized to "format *.html" but not to "save as html file".

- If configuration objects created by your enterprise are deleted, their translations will NOT be automatically deleted from the vocabularies. However, during the meta-model update process, any translation entries that cannot be matched to an original will be deleted from the Alfabet database. An XLSX file will be generated during the meta-model update process that lists all orphaned translations that have been removed from the Alfabet database. Solution administrators should analyze this list to determine whether any of the orphaned translations require the creation of new (custom) translations.

Please note the following regarding the different file formats:

- XLSX files: The following columns are relevant for translation:

- ***Original***: The original string from the `English (UnitedStates)` vocabulary provided by Software AG. This column must not be edited.
- ***Translation***: The standard translation provided by Software AG. This column is empty for all exported vocabulary files except for files exported for the `German (Germany)` vocabulary. This column must not be edited. Translations added to a vocabulary in the `Translation` column are ignored during update of vocabulary by the vocabulary file import mechanisms of the **Vocabulary Manager** editor.
- ***CustomTranslation***: The customized terminology added by the customer. Enter the translated or modified terms in the column `CustomTranslation`.

- XML (*VOC) files: The XML file contains one XML element `Entry` for each term. The following XML attributes are relevant for translation:

- ***Original***: The original string from the `English (UnitedStates)`) vocabulary provided by Software AG. This column must not be edited.
- ***Translation***: The standard translation provided by Software AG. This column is empty for all exported vocabulary files except for files exported for the `German (Germany)` vocabulary. This column must not be edited. Translations added to a vocabulary in the `Translation` column are ignored during update of vocabulary by the vocabulary file import mechanisms of the **Vocabulary Manager** editor.
- ***CustomTranslation***: The translation of the original string. Enter a translation for the string in the XML attribute `Original`.

Importing the Vocabulary Files

To import the customized terminology from Microsoft Excel files or XML files:

1) In the menu bar, click **Globalization** > **Open Vocabulary Manager.**

If you have exported simple text files via the **Export Vocabularies with Unique Identifiers** functionality, then you must import the files via **Globalization** > **Import Vocabularies with Unique Identifiers**.

2) In the **Vocabulary** toolbar, click the **Import Vocabularies** button. The **Translation Import** editor opens.

3) In the **Translation Import** editor, click the **Add Vocabulary** button and select the relevant files to import.

The XML files for the export of terminology have the extension VOC.

4) The selected files are displayed in the **Translation Import** dialog box. The column **File** displays the file that you have selected to import.

- If the column **Translation Source** is empty, select the type of terminology set that you are importing in the drop-down list.
- If the column **Culture** is empty, select the locale that you are importing in the drop-down list.

5) Click **OK** to import the terminology.

Updates to the translation table require the Web server to be restarted or the meta-model to be reread if the server is running in Design mode (if in the `web.config` file the key attribute "`appmode`" is set to `Design`).

Optionally you can select the checkbox **Clear All Translations** to remove all translations from the terminology set and vocabulary that were imported from the import file.

Make sure that the import file contains all required translations prior to import with a selected **Clear All Translations** checkbox. For example, a terminology set already includes a translation of both standard and configuration terms. You have exported **Standard** terms only without exporting the **Configuration** part of the terminology. After changing the terminology in the exported file, you import the translations and select the **Clear All Translations** checkbox. All custom translations will be deleted from the terminology set and then the translations from the import file are added to the custom translation in the terminology set. The import file only contains standard terminology, which results in all configuration related translations to be irrevocably deleted from the terminology set during import.

Translating Custom Strings via the Automated Translation Capability

An automated translation capability is available that supports interoperability with a translation service. This capability allows the enterprise to provide a translated user interface for the supported secondary languages quickly, with a minimal amount of effort and cost.

The custom strings that your enterprise has created when configuring the class model may be automatically translated via the automated translation capability for any of the languages supported by Software AG. If only one culture is defined for your enterprise, then the automated data translation functionality will be disabled in the Alfabet user interface. Translations can be fetched via a configured translation service such as Google Translate® for the custom strings defined for custom editors, custom selectors, custom

object views/object cockpits, workflows, custom wizards, configured reports, guide pages/guide views, etc. Any English string that does not have a translated string available will be translated to the target language that you select. Via the **Get Low-Confidence Translations** functionality, you can automatically translate the custom strings to any of the languages supported by Software AG.

The following translation services are supported for the automated translation capability:

- Google Translate®
- AWS Translate®
- DeepL® Translator

For more information about using the automated translation capability to translate object data to the languages supported by your enterprise, see the section Configuring the Translation of Object Data.

Please note the following:

- The automated translations will be saved to the `Custom Translation` column in the vocabulary for the relevant language.
- If a string is used in multiple vocabularies (for example, in both `MetaModel` and `GuidePages`), the same translation string will be used for all occurrences of the string.
- The translation service will not overwrite existing custom translations that have been either manually or automatically translated.
- The translation service will not overwrite standard strings provided by Software AG.
- The enterprise must have a valid license to one of the following translation services:
- Google Translate®
- AWS Translate®
- DeepL® Translator
- Microsoft® Azure® Translate Text
- The XML object ***AlfaTranslationServicesConfig*** must be configured and the connection to the translation service must be activated. For more information about the configuration of the XML object ***AlfaTranslationServicesConfig***, see the section *Configuring Interoperability with a Translation Service* in the reference manual *API Integration with Third-Party Components*.
- The pre-conditions for activating the Rest API must be fulfilled. For more information, see the reference manual *Alfabet RESTful API*.
- One user in the enterprise must be specified to execute self-reflective events and must start the Alfabet Server. For more information, see the chapter *Setting a User as a Self-Reflective User to Execute Events* in the reference manual *Configuring Alfabet with Alfabet Expand*.
- The Alfabet Server must be running and able to connect to the Internet.

To provide automated translations for empty strings in the vocabularies:

1) In the menu bar, click **Globalization** > **Edit Translation**. You will see the **Translation Editor**

2) Click **Manage Vocabulary** > **Open** and select the culture for which you want to automatically translate the vocabularies. The target language for translation is displayed in the header of the **Translation Editor**.

3) In the header, click **Get Low-Confidence Translations** > **Meta-Model Configuration**. An message dialog is displayed asking confirmation to start the automated translation process. Click **OK** to confirm the message and start the automated translation process.

The **Get Low-Confidence Translations** functionality may not be displayed in Alfabet Expand if the XML object ***AlfaTransltionServicesConfig*** is not correctly configured.

4) When the automated translation process has been completed, a message will be displayed with the number of automated translations that have been uploaded. Confirm the information message by clicking **OK**. The strings added to the vocabularies via the automated translation capability will be displayed as custom translations in the **Translation Editor**.

5) Click the **Close** button to close the **Vocabularies** editor.

Managing Imported Vocabularies

It is possible to track the vocabularies that have been updated via imported Microsoft Excel or XML files. To do so, click **Globalization** > **Show Vocabulary Update History** in the menu bar. The following information is displayed:

- **Culture**:The locale ID of the language.
- **Source**: The name of the vocabulary that has been updated.
- **Update File**: The name of the file imported for the translation update.
- **Update Time**: The timestamp when the file was imported for the translation update.
- **Standard Translation Count**: The number of strings updated in the standard translation.
- **Custom Translation Count**: The number of strings updated in the custom translation.

Deleting Vocabularies

You can delete a vocabulary because, for example, the culture that the vocabulary is associated with is no longer supported in the enterprise. Please note the following about deleting a vocabulary:

- After deleting a vocabulary, the vocabulary file no longer exists, and the strings cannot be imported and exported using the **Vocabulary Manager**.
- Nevertheless, the vocabulary of a language culture can be opened and edited via the translation editor as long as the vocabulary is defined as **Base Culture** in a culture setting defined in Alfabet Expand. If you open the **Translation Editor** and add a string to a terminology set of a vocabulary that was formerly deleted, the terminology set is re-created and can then be exported via the vocabulary manager.

If you delete the vocabularies for a culture, the custom strings will be irrevocably deleted from the Alfabet database. In order to restore standard strings provided by Software AG, see the section Restoring Deleted Vocabularies.

To delete a vocabulary:

1) In the menu bar, click **Globalization** > **Open Vocabulary Manager**. You will see the **Vocabularies** editor.
2) In the table, select one or multiple vocabularies that you want to delete.
3) In the toolbar of the **Vocabularies** editor, click the **Delete** button. The vocabularies are removed for the selected vocabulary. No translation will be available for the strings in the **Translation Editor**:

You can alternatively remove vocabularies by means of the **Translation Editor**:

1) In the menu bar, click **Globalization** > **Edit Translation.** The **Translation Editor** opens.
2) In the **Translation Editor** toolbar, click **Manage Vocabulary** > **Open** and select the culture for which you want to delete the vocabularies.
3) In the **Translation Editor** toolbar, click **Manage Vocabulary** > **Remove Vocabulary**. Confirm the warning message by clicking **OK**. No translation will be available for the strings in the **Translation Editor**:

Restoring Deleted Vocabularies

If the standard Alfabet vocabularies have been accidentally deleted, it is possible to restore them by means of the option **Globalization** > **Restore All Vocabularies**. Please contact Software AG Support to attain the necessary binary file required to restore the standard Alfabet vocabularies.

Transferring the Custom Translation to the Alfabet Administrator

If you have changed some standard translation strings that are displayed in the Alfabet Administrator, you must explicitly transfer the custom translation to the Alfabet Administrator if you want the custom translation displayed in the interface of the Alfabet Administrator. This must be done after the custom translation has been imported to the Alfabet database. The `LocTr.bin` file that is created will be stored in the Alfabet `Programs` directory. Please note that the `Programs` directory must have Write permissions at run time.

To implement the custom translation in the interface of the Alfabet Administrator, click **Globalization** > **Transfer Translation to Alfabet Administrator**. Click **OK** to confirm the message.

Configuring the Translation of Object Data

Depending on the configuration of your Alfabet solution, object data such as names, descriptions, and other relevant custom properties may be captured in the enterprise's primary language, any of its secondary languages, or in a statutory language that is mandated for relevant object classes. If the enterprise supports multiple languages, the data captured for protected and custom properties of the type `String` and `Text` may be translated to the other languages implemented in your Alfabet solution. When the user interface is rendered in a one of the languages for which translations are available, the translated names of objects will be displayed in explorers, object profiles, page views, previews, etc. Objects with names defined in a secondary language will be ordered accordingly in explorer trees and page views.

If data translation is supported for the class **Business Process**, a user could capture the **Name** and **Description** of their business processes in the primary language. The same user or other users with access to the business processes could then either manually provide a translation for the names and descriptions of the business processes in the editor/wiazárd or, if the automated translation capability is configured, the data can be translated via a translation service to the languages supported by the enterprise.

As solution designer, you can configure on a per-class basis whether the translation of object data is possible and which properties may be manually translated or translated via the automated translation capability. The following scenarios to capture and translate object data are supported:

- Data can be captured in the primary language only. If only one culture is defined for the enterprise, then the **Manual Translation** and **Automated Translation** icons will be disabled in the editor/wizard.
- Data can be captured in the primary language and may be translated either manually and/or automatically to one or more secondary languages. It is possible to configure on a per-class basis the object classes where only manual translation is possible, where only automated translation is possible, or where an automated translation is provided if no manual translation has been entered.
- The manual translation can be captured by changing the language selector in the editor/wizard and entering the translation in the relevant field.
- The automated translation will be triggered when the **OK** button in the editor or **Next** button in the wizard is clicked.

In some cases, a new object will inherit the translated name, description, etc. of its parent object. This is the case, for example, for information flows, local components, or business services. If translations are available for the name of the parent object, the new

information flow, local component, or business service will inherit the translated name of its parent object. For example, a business service will inherit the translated name of the business function it is based on and an information flow will inherit the name of its source and target applications.

- Data can be captured in a secondary language. The Alfabet user interface must be rendered in the secondary language to capture data in that language. In this case, it is recommended that the automated translation capability is configured in order to provide a translation of the name for the object in the primary language. For more information about capturing data in a secondary language, see the section *Capturing Data in a Secondary Language*.
- Data can be captured in a statutory language, which can be the primary language or a secondary language. The Alfabet user interface should be rendered in the statutory language to capture data in that language. If the statutory language is a secondary language, it is recommended that the automated translation capability is configured in order to provide a translation of the name for the object in the primary language. For more information about capturing data in a statutory language, see the section *Capturing Data in a Statutory Language*.
- Data can be captured via XLSX files based on data capture templates configured in the **Extended Data Capture Templates** functionality. In this case, data can also be captured in a primary, secondary, or statutory language. For more information about configuring and using data capture templates to import object data, see the chapter *Capturing Data with Data Capture Templates* in the reference manual *User and Solution Administration*.

The following configuration is required to specify data translation:

- Specify which cultures support object data translation.
- Specify which object classes allow object data translation. For each object class, you must consider the following:
  - Which protected and custom properties of the type `String` and `Text` shall be translated. The protected properties that may be translated are predefined by Software AG.
  - Which properties shall be translated manually by users when they create a new object in the editor/wizard.
  - Which properties shall be translated via the automated translation capability in the editor/wizard.
  - You can specify that the data for an object class must be captured in a statutory language.
  - Specify whether users may initially capture object data in a secondary language.
  - Specify whether users should capture object data in a statutory language.

  The configuration of the object classes and their properties is applicable to all cultures for which object data translation is permissible.

- In addition, if the automated translation capability shall be implemented, the following must be configured:
  - The enterprise must have a valid licence to one of the following translation services:
  - Google Translate®
  - AWS Translate®

- DeepL® Translator
- Microsoft® Azure® Translate Text
- The XML object ***AlfaTranslationServicesConfig*** must be configured and the connection to the translation service must be activated. For more information about the configuration of the XML object ***AlfaTranslationServicesConfig***, see the section *Configuring Interoperability with a Translation Service* in the reference manual *API Integration with Third-Party Components*.
- The pre-conditions for activating the Rest API must be fulfilled. For more information, see the reference manual *Alfabet RESTful API*.
- One user in the enterprise must be specified to execute self-reflective events and must start the Alfabet Server. For more information, see the chapter *Setting a User as a Self-Reflective User to Execute Events* in the reference manual *Configuring Alfabet with Alfabet Expand*.
- The Alfabet Server must be running and able to connect to the Internet.

Translated values for object data will be displayed automatically in configured reports that are based on the Alfabet query language. For configured reports based on native SQL queries, however, a special syntax triggering the translation must be added to the native SQL query in order to enable the display of translation values for object data. For more information, see Displaying Translated Object Data In the Current Language of the User Interface in the section Defining Native SQL Queries in the chapter Defining Queries.

The following information is available:

- Enabling Object Data Translation for a Culture
- Enabling the Manual Translation of Object Data
- Configuring Automated Translation of Object Data
- Specifying Object Classes and Properties for Automated Translation
- Configuring the Connection to the Translation Service
- Allowing Data To Be Captured in a Non-Primary Language
- Specifying a Statutory Language for the Enterprise

Enabling Object Data Translation for a Culture

The **Support Data Translation** attribute of a culture determines whether object data translation is possible for the culture. In order for users to be able to either manually or automatically translate object data to a language available in the Alfabet solution, the **Support Data Translation** attribute must be set to `True` for the respective culture. If the **Support Data Translation** attribute is set to `True` for the culture, manual and automatic data translation is permissible for all classes per default. You can refine this and configure the permissibility of data translation for specified object classes as well as their protected and custom properties. This is described in the sections Enabling the Manual Translation of Object Data and Configuring Automated Translation of Object Data.

The configuration of the object classes and their properties is applicable to all cultures for which object data translation is permissible. For more information about configuring cultures, see the section Specifying the Cultures Relevant to Your Enterprise.

If the **Support Data Translation** attribute is set to `True` for a culture, the database tables will include the necessary columns needed for translation purposes for the permissible object classes. Please note that the translations are stored in separate columns in the database tables with the column name having the following syntax: < **Tech Name** attribute of object class property>_<locale ID>. For example, the `Name` property of the object class `Application` can be translated to German. The number 1031 is the locale ID for German. Therefore, the column ***NAME_1031*** will be available in the `Application` database table.

To specify the permissibility of manual or automatic data translation on object classes:

- If the **Support Data Translation** attribute is set to `True` in the attribute grid of the relevant the culture , data translation will be possible per default for predefined protected properties and custom properties of the type `String` or `Text` for all object classes. The language associated with the culture will be available for selection in the language selector in the relevant editors/wizards in the Alfabet user interface.
- If the **Support Data Translation** attribute is set to `False`, data translation will not be possible at all for the culture and the language will not be displayed in the language selector in object editors/wizards.

To change the data translation definition for a culture, right-click the culture node in the **Class Model** explorer and select:

- **Activate Support for Data Translation** to specify that data translation is allowed for the culture. The **Support Data Translation** attribute will be set to `True` if you select this option. In the toolbar, click the **Save** button to save the culture definition.
- **Deactivate Support for Data Translation** to specify that data translation is not allowed for the culture. The **Support Data Translation** attribute will be set to `False` if you select this option. In the toolbar, click the **Save** button to save the culture definition.

Enabling the Manual Translation of Object Data

In order to allow users to manually translate object data, you must ensure that the relevant object classes and properties are enabled for data translation. The translation of object data typically includes the captions, descriptions, and other custom properties of the type `String` and `Text` defined for objects that users create in the Alfabet user interface. For each object class for which object data may be translated, you should review which standard and custom properties of the type `String` and `Text` shall be manually translated.

For an overview of how users can translate object data in editors and wizards, see the section *Multi-Language Support in Editors and Wizards* in the reference manual *Getting Started with Alfabet*.

The following configuration must be carried out in order to ensure that the values defined for standard and custom properties may be manually translated:

- For all cultures for which object data translation should be possible: Set the **Support Data Translation** attribute to `True` for the culture. For more information about specifying a culture, see the section Specifying the Cultures Relevant to Your Enterprise.

  It is important to note that the configuration regarding the permissibility of manual translation and automated translation for object classes and/or properties is relevant for all cultures for which the **Support Data Translation** attribute is set to `True`.

- For all object classes for which object data translation should be possible:
  - Set the **Enable Data Translation** attribute to `True` for the object class to ensure that the object class may be translated. For more information about configuring object classes, see the section Editing a Protected Object Class in the chapter Configuring the Class Model. Please consider the following:
  - The **Enable Data Translation** attribute is set to `True` per default for all object classes that are visible in Alfabet Expand. Therefore, if specific object classes shall not support object data translation, you must set the **Enable Data Translation** attribute to `False` for these object classes.
  - If you set the **Enable Data Translation** attribute to `False`, then the translation of object data for that object class will not be permissible for all cultures.
  - If you change the **Enable Data Translation** attribute setting for an object class from `True` to `False`, any existing automated translations will be deleted from the Alfabet database for that class.
  - Set the **Enable Automated Data Translation** attribute to `False` for the object class to ensure that the automated translation capability is disabled per default for the object class.
- For custom properties of the type `String` and `Text`: Set the **Enable Data Translation** attribute to either `Manual` if the custom property may be only manually translated or `ManualAndAutomated` if the custom property may be manually translated or translated via the automated translation capability. For more information about specifying a custom property, see the section Configuring Custom Properties for Protected or Public Object Classes in the chapter Configuring the Class Model. Please consider the following:
  - If the **Enable Data Translation** attribute is set to `ManualAndAutomated`, the **Enable Automated Data Translation** attribute for the object class must be set to `True` to ensure that the automated translation capability is enabled.
  - If you change the **Enable Data Translation** attribute setting for an object class property from `ManualAndAutomated` to `Manual`, any existing automated translations will be deleted from the Alfabet database for that property.

Configuring Automated Translation of Object Data

In order to allow users to translate object data via the automated translation capability, you must ensure that the relevant object classes and properties are enabled for data translation. The translation of object data typically includes the captions, descriptions, and other custom properties of the type `String` and `Text` defined for objects that users create in the Alfabet user interface. For each object class for which object data may be translated, you should review which standard and custom properties of the type `String` and `Text` shall be translated via the automated translation capability.

If users shall capture data in a secondary language or a statutory language, then the automated translation capability should be implemented in order to ensure that the data captured in a secondary language will also have values in the primary language. For more information about capturing data in a secondary language, see the sections Allowing Data To Be Captured in a Non-Primary Language and Specifying a Statutory Language for the Enterprise.

In addition to configuring the permissibility of object classes and properties for automated translation, the following must be configured in order to implement the automated translation capability:

- The enterprise must have a valid license to one of the following translation services:
- Google Translate®
- AWS Translate®
- DeepL® Translator
- Microsoft® Azure® Translate Text
- The XML object ***AlfaTranslationServicesConfig*** must be configured and the connection to the translation service must be activated. For more information about the configuration of the XML object ***AlfaTranslationServicesConfig***, see the section *Configuring Interoperability with a Translation Service* in the reference manual *API Integration with Third-Party Components*.
- The pre-conditions for activating the Rest API must be fulfilled. For more information, see the reference manual *Alfabet RESTful API*.
- One user in the enterprise must be specified to execute self-reflective events and must start the Alfabet Server. For more information, see the chapter *Setting a User as a Self-Reflective User to Execute Events* in the reference manual *Configuring Alfabet with Alfabet Expand*.
- The Alfabet Server must be running and able to connect to the Internet.

In the Alfabet user interface, the user will define a value for a relevant property in an editor field displaying the **Automated Translation**  icon. Upon clicking the **OK** button in the editor/the **Next** button in the wizard, a private event template `GetAutomatedTranslation_For_Instance` will be triggered that sends a Rest API call. The Rest API call then triggers the execution of an ADIF import scheme that fetches the translation strings from the configured translation service for all relevant languages. Please note the following:

- The translated string will not be immediately displayed. A few minutes may be needed to fetch the translations.
- If an object is created as a copy of an object, then the automated translation will be copied to the new object. For example, if an application variant is created based on an application or if a local component is created for an application, the translation of the application's description will be

copied to the respective application variant or local component. The translation string can be changed as needed.

- If the original string is used for multiple object data, the same translation string will be used for all occurrences of the original string in the Alfabet database.
- If the original string is changed in the editor/wizard after translation strings have been fetched, the automated translation will be triggered again when the **OK** button in the editor/the **Next** button in the wizard is clicked and new translation strings will be fetched from the translation service.
- If an existing translation string has already been translated manually or via the automated translation capability, it will not be retranslated if the automated translation capability is triggered for other fields in the editor/wizard.
- If an existing translation string has been translated via the automatic translation capability, it may be manually revised in the editor/wizard or in the *Automated Data Translation Functionality*. If the original string is used for multiple object data, then only the explicit translation string that is being revised will be changed in the Alfabet database. Other instances of the translation string will remain unchanged and will not be impacted by the revised translation string. In this way, you can revise the translation string for a specific context.
- A user with an administrative profile may review the translation strings fetched via the automated translation capability and modify them as needed in the *Automated Data Translation Functionality*. For more information, see the chapter *Managing Automated Translation Strings* in the reference manual *User and Solution Administration*.
- A user with an administrative profile may review whether the event has been successfully triggered in the *Events Administration Functionality*. If the event is not displayed or has an `Error` status, then the prerequisites should be reviewed. For more information about events administration, see the chapter *Managing Events* in the reference manual *User and Solution Administration*.
- A user with an administrative profile may view the status of the ADIF import schemes (`Get Automated Instance Translations from Service Provider` or `Get Automated Translations for an Instance from Service Provider`) that are executed for the automated translations in the *ADIF Jobs Administration Functionality*.
- If automated translations have not been fetched due to Internet connection outages, a solution designer can trigger the ADIF import scheme `Get_Instance_Automated_Translations_For_Empty_Texts` or similar ADIF jobs to retrieve the automated data translations that have been missed as a result of the Internet connection outages.

Furthermore, several private ADIF import scheme are also available in Alfabet Expand to support the automated translation capability. For more information about the predefined ADIF schemes for the automated translation capability, see the chapter *Predefined ADIF Schemes* and for more information about executing the ADIF job based on an ADIF scheme, see the chapter *Configuring ADIF Schemes* in the reference manual *Alfabet Data Integration Framework*.

For an overview of how users can translate object data in editors and wizards, see the section *Multi-Language Support in Editors and Wizards* in the reference manual *Getting Started with Alfabet*.

A **Get Low-Confidence Translations** functionality is available to automatically translate the custom strings defined for custom editors, custom selectors, custom object views/object cockpits, workflows, custom wizards, configured reports, guide pages/guide views, etc. to any of the

languages supported by Software AG. For more information, see the section Translating Custom Strings via the Automated Translation Capability.

- Specifying Object Classes and Properties for Automated Translation
- Configuring the Connection to the Translation Service

Specifying Object Classes and Properties for Automated Translation

In order to allow users to translate object data with the automated translation capability, you must ensure that the relevant object class is enabled for data translation and for automated data translation. For each object class for which object data may be translated, you must specify which protected and custom properties of the type `String` and `Text` may be translated via the automated translation capability. The protected properties that are permissible for data translation are predefined by Software AG.

You should carefully review whether it is meaningful for a particular property to be translated. For example, it may not be appropriate to translate the name of applications or components as these are typically product names whereas it would be meaningful to translate business process names as these tend to be descriptive texts.

If you specify that a particular property may be translated via the automated data translation capability, users can disable the automated data translation capability for the property for in the context of the object's editor. For example, in the case of the object class **Location**, you may specify that the automated translation capability shall fetch the translations for the names of locations. Whereas it may be meaningful for a city name to be translated (for example, `Berlin` would be translated to `Berlino` in Italian, `Berlim` in Portuguese, and `Berlina` in Polish), it may make no sense for the names of small towns to be translated because no actual translation of the town's name exists. The user can click the **Automated Translation** icon next to a field in the editor to disable it. The **Disable Automated Translation** icon will then be displayed in the editor and will be ignored by the automated translation capability.

The following configuration must be carried out in order to ensure that the values defined for standard and custom properties may be translated via the automated translation capability:

- For all cultures for which object data translation should be possible: Set the **Support Data Translation** attribute to `True` for the culture. For more information about specifying a culture, see the section Specifying the Cultures Relevant to Your Enterprise.

  It is important to note that the configuration regarding the permissibility of manual translation and automated translation for object classes and/or properties is relevant for all cultures for which the **Support Data Translation** attribute is set to `True`. In other words, it is not possible to

- For all object classes for which object data translation should be possible:

- Set the **Enable Data Translation** attribute to `True` for the object class. For more information about configuring object classes, see the section Editing a Protected Object Class in the chapter Configuring the Class Model.
-
-

-

Please consider the following:

- The **Enable Data Translation** attribute is set to `True` per default for all object classes that are visible in Alfabet Expand. Therefore, if specific object classes shall not support object data translation, you must set the **Enable Data Translation** attribute to `False` for these object classes.
- If you set the **Enable Data Translation** attribute to `False`, then the translation of object data for that object class will not be permissible for all cultures.
- If you change the **Enable Data Translation** attribute setting for an object class from `True` to `False`, any existing automated translations will be deleted from the Alfabet database for that class.

- Set the **Enable Automated Data Translation** attribute to `True` for the object class to ensure that the automated translation capability is enabled for the object class. Please consider the following:
- The specification of the **Enable Data Translation** attribute applies to all cultures defined for your enterprise. Therefore, if you specify that a data translation is not permissible for an object class property, then object data cannot be specified for that object class for all defined cultures. Please note that regardless of the specification of the **Enable Data Translation** attribute for an object class/object class property, if the **Support Data Translation** attribute is set to `False` for a culture, data translation will not be possible at all for the culture and the language code will not be displayed in the language field in all object editors.
- The specification of the **Enable Data Translation** attribute for an object class property will have precedence over the specification made in the **Automated Translation Rules for Class Properties** attribute for the object class. In other words, if the **Enable Data Translation** attribute for the property is set to `None` or `Manual` and the property is set to `True` in the **Automated Translation Rules for Class Properties** attribute for the object class, the property will not be automatically translated.
- Specify the **Automated Translation Rules for Class Properties** attribute for the object class. Click the **Browse** button to open the editor to specify which protected and custom properties may be translated automatically. For each property listed in the **Automated Translation Rules for Class Properties** editor, select `True` in the **Automated Translation Enabled** column to permit the property to be automatically translated. Please consider the following:
- The `Name` property for the following object classes has been set to `False` per default for the automated translation capability. This too can be changed as needed via the **Automated Translation Rules for Class Properties** attribute in the attribute grid of the object class:
  - Application
  - Component
  - Component Catalog Element
  - Connection Data Format
  - Platform Catalog Element
  - Platform Element
  - Platform Information Flow
  - Platform Layer
  - Platform Tier
  - Solution Platform Element
  - Solution Platform Information Flow
  - Solution Standard Platform
  - Standard Platform

- Connection Frequency
- Connection Method
- Connection Type
- Device
- Information Flow
- Interface System
- Local Component
- Peripheral
- Platform Template
- Solution Application
- Solution Component
- Solution Device
- Solution Information Flow
- Solution Local Component
- Solution Peripheral
- Stack
- Stack Element
- Stack Item
- Technology
- Technology Group
- Threat
- Threat Group
- Vendor
- Vendor Product

- For many classes, the `Name` property may be automatically translated per default. In some cases, a new object such as an information flow, local component, or business service will inherit the translated name of its parent object. For example, a business service will inherit the translated name of the business function it is based on and an information flow will inherit the name of its source and target applications. If translations are available for the name of the parent object, the new information flow, local component, or business service will inherit the translated name of its parent object.
- The specification of the **Enable Data Translation** attribute for an object class property will have precedence over the specification made in the **Automated Translation Rules for Class Properties** field for the object class. If the **Enable Data Translation** attribute is changed to `Manual` or `None` for a property, then the respective property will not be displayed in the **Automated Translation Rules for Class Properties** editor for the object class.
- If the **Enable Data Translation** attribute is set to `None` or `Manual` for a custom property, the value in the **Automated Translation Enabled** column will be set to `False` per default and cannot be automatically translated.
- For custom properties  of the type `String` and `Text`: Set the **Enable Data Translation** attribute to `ManualAndAutomated` to enable the automated translation capability for the custom property. All custom properties for which the **Enable Data Translation** attribute is set to `ManualAndAutomated` will be added to the **Automated Translation Rules for Class Properties** editor for the object class. Users will be able to either provide a manual translation or translate the custom property's value via the automated translation capability. For more information about specifying a custom property, see the section Configuring Custom Properties for Protected or Public Object Classes in the chapter Configuring the Class Model. Please consider the following:
- If you change the **Enable Data Translation** attribute setting for an object class property from `ManualAndAutomated` to `Manual`, any existing automated translations will be deleted from the Alfabet database for that property.

Configuring the Connection to the Translation Service

The XML object ***AlfaTranslationServicesConfig*** allows you to specify interoperability with a translation service for the purposes of providing an automated translation for custom strings in the vocabularies that have been defined in the context of the solution configuration as well as for object data translation may be

users in the context of editors and wizards in the Alfabet user interface. For more information, see the section *Configuring Interoperability with a Translation Service* in the reference manual *API Integration with Third-Party Components*.

The following translation services are supported for the automated translation capability:

- Google Translate®
- AWS Translate®
- DeepL® Translator

The XML object usually includes an example definition. In addition, a template is available via the **XML Template** in the attribute grid of the XML object ***AlfaTranslationServicesConfig***. The template can be copied to the XML object to avoid manually writing the configuration. In this case, you would edit the XML elements described below. The following information describes a configuration from scratch.

To edit the XML object ***AlfaTranslationServicesConfig***:

1) Go to the **Presentation** tab, expand the **XML Objects** folder, and expand by the **Integration Solutions** folder.
2) Right-click ***AlfaTranslationServicesConfig*** and select **Edit XML.** . The XML object ***AlfaTranslationServicesConfig*** opens.
3) Optionally, you can configure the integration interface to send requests to a translation service instance via a proxy server. To define a proxy server, add a child XML element `Proxy` to the XML element ***AlfaTranslationServicesConfig*** and define the following XML attributes for the XML element `Proxy`:

- `url`: Define the URL of the proxy server.
- `user`: If required, enter the user name for access to the proxy server. The domain name for authentication is defined separately with the XML attribute domain and must not be specified as part of the user name.
- `password`: If required, enter the password for access to the proxy server.
- `domain`: If required, define the domain name that shall be used as part of the user name for authentication at the proxy server

Please note that server variables read the value of the XML attribute at runtime from the server alias configuration of the Alfabet Web Application when a connection to the integration solution is established. For information about configuration the server variables in the server alias, see *Configuring Server Variables for Integration and Interoperability Solutions*.

4) For each translation service that you want to define in the XML object, create a child XML element `TranslationServiceInfo`.
5) Define the following XML attributes for each XML element `TranslationServiceInfo`:

- `Name`: Enter a name for the translation service connection.

- `IsActive`: Enter "`true`" to activate interoperability with the translation service. One translation service may be active at any given time. If multiple connections are set to `IsActive` = `true`, then the first active connection will be used.
- `Type`: Enter one of the following, depending on the translation service that you will connect to and for which you have a valid license:
- `Google`
- `AWS`
- `DeepL`
- `Azure`
- `ServiceType`: You must specify `AzureCognitive` if the XML element `Type` is set to `Azure`. Failure to specify this XML attribute will result in an error if interoperability is specified with Microsoft® Azure® Translate Text.
- `Timeout`: Define the HTTP request timeout for the data connection.
- `AccessKey`: The access key for connection to the translation service.
- `SecretKey`: The secret key for connection to the translation service.

6) Automated translation is available for HTML texts including embedded tables if the **Enable Data Translation** attribute is set to `True` for the custom property. Because most translation engines impose a limitation on the maximum number of translated characters, any HTML that exceeds 5000 characters will not be translated. In order to specify how the translation mechanism in Alfabet shall deal with HTML that exceeds 5000 characters, the XML attribute `TranslateContentExceedingHTMLLengthLimit` should be specified. If set to `False`, automated translation will not be performed if the text in HTML or ASCII format exceeds 5000 characters. If set to `True`, the text in HTML format will be translated and displayed as ASCII format in the relevant translated languages.

Columns will be added to the relevant class table in the database for properties for which the **Can Have HTML Content** attribute has been set to `True`. This ensures that texts will be stored in the database in ASCII format as well as HTML format. The column name for the text with HTML format will consist of Class.Property.TechName, the ISO-code of the translated language, and the suffix _RT (rich text). Due to the implementation of _RT columns in the database tables, the number of characters that may be used for the technical name of a property that supports HTML and shall be translated is restricted to 23 characters.

7) In the toolbar, click the **Save**  button to save the XML definition.

Allowing Data To Be Captured in a Non-Primary Language

The possibility to capture data in a secondary language provides end users with the means to capture and maintain IT portfolio information in their preferred language and ensures high data quality in multi-language organizations.

The Alfabet user interface must be rendered in the secondary language that the user wants to capture the original data for. For details regarding how to capture data in a secondary language in the user interface, see the section *Capturing Data in a Secondary Language* in the reference manual *Getting Started with Alfabet*.

The following configuration is required in order to allow users to capture data in a secondary language:

- The automated translation capability as described in the section Configuring Automated Translation of Object Data must be configured and all requirements to successfully fetch translations from the configured translation service must be fulfilled.

  Every new object that is created requires a value in the **Name** attribute for the primary language. If the data is originally captured in a secondary language, then the **Name** attribute must be specified in the primary language either via a manual translation or via the translation capability.

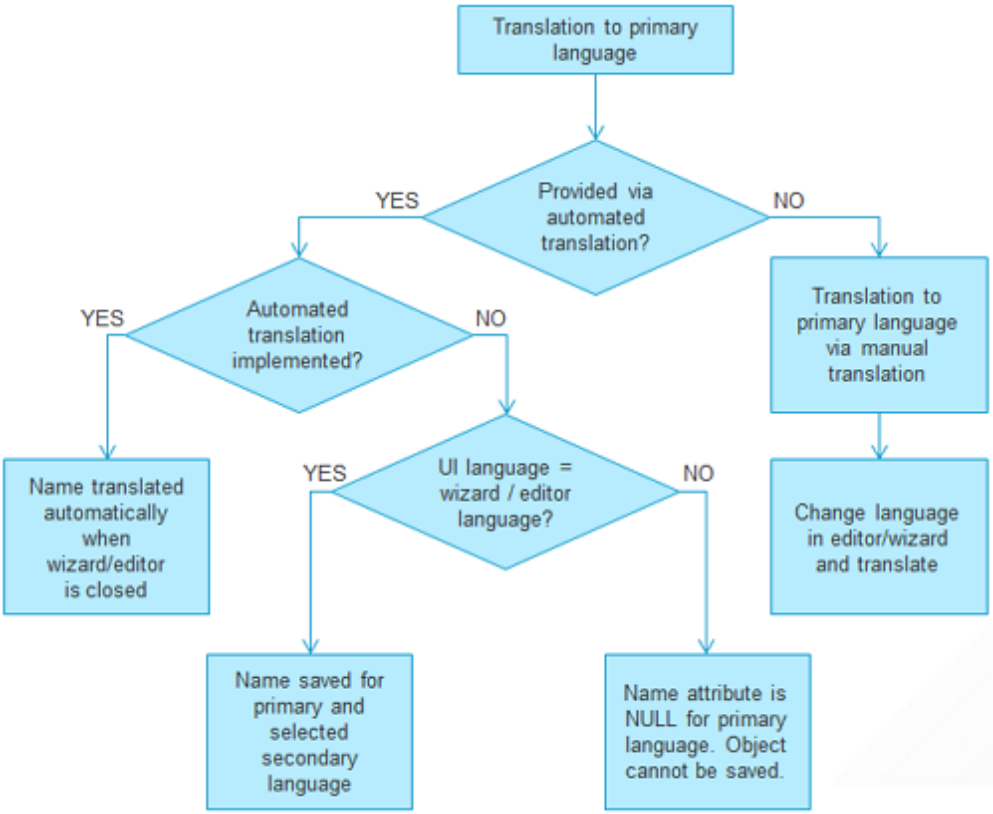


FIGURE: Translation to primary language

Please note the following:

- If the translation to the primary language is provided via the automated translation capability:
  - If the automated translation capability is implemented, the name will be automatically translated to the primary language when the editor/wizard is closed.
  - If the automated translation capability is **not** implemented, please consider the following:
  - If the language of the user interface is the same as the language selected in the editor/wizard, the value entered in the **Name** field will be saved for both the selected secondary language as well as the primary language. The language can be manually translated to the primary language in the editor after the new object has been saved.
  - If the language of the user interface is not the same as the language selected in the editor/wizard, the object cannot be saved because the **Name** attribute is NULL for the primary language. In this case, you must change the language of the user interface to the language that you want to capture the data in.
  - If a translated string is changed, the automated translation will be triggered again for all languages. The original string captured in the secondary or statutory language will not be changed via the automated translation.
- If the translation to the primary language shall be defined via a manual translation:
  - Enter the value in the editor field, change the language selector at the bottom of the editor to the primary language, and enter the translated value in the relevant editor. Click the **OK** button in the editor/the **Next** button in the wizard to save the definitions.
  - If no manual translation is defined for the primary language, the original string defined in the secondary language will be displayed for the **Name** when the user interface is rendered in the primary language. This may be revised at any time in the editor/wizard.
  - If no manual translation is defined for any of the other secondary languages, the value displayed for the **Name** when the user interface is rendered in the primary language will also be displayed when the user interface is rendered in the secondary language.

- The XML attribute `EnableTranslationToPrimaryLanguage` in the XML object ***SolutionOptions*** must be set to `True`. For more information, see the section Configuring Automated Translation of Object Data.
- In order for users to be able to capture language in a secondary language, they must select the checkbox for the **Capture Translations in Language of User Interface** field in their **User Settings** editor. For more information, see the section *Defining Your User Settings in Alfabet*. For more information about the configuration requirements to capture data in a secondary language, see the section Allowing Data To Be Captured in a Non-Primary Language in the reference manual *Configuring Alfabet with Alfabet Expand*.
- A user with an administrative profile may review the translation strings fetched via the automated translation capability and modify them as needed in the *Automated Data Translation Functionality*.

For more information, see the chapter *Managing Automated Translation Strings* in the reference manual *User and Solution Administration*.

Specifying a Statutory Language for the Enterprise

A statutory language capability is available in order to address regulatory requirements that mandate that objects are captured in a specific language. For example, if an enterprise has English as its primary language, but an organization in the enterprise is located in Germany and is required to capture data in German, than users can capture the data in the statutory language of German.

When a user creates an object based on a relevant object class/object class stereotype, a statutory language selector will open in which the statutory language can be selected. All languages supported by the enterprise will be available for selection. In the object editor that opens, the user can capture all attributes that support data translation in the statutory language. The automated translation capability will then translate the data to the primary language as well as other relevant secondary languages.

Please note the following:

- If the **Enable Statutory Language** attribute for the relevant object class is set to `True`, all languages will be available in the statutory language selector.
- There is no enforcement mechanism that requires users to capture data in a statutory language.
- The Alfabet user interface must be rendered in the statutory language that the user wants to capture the original data for.
- The value defined for the selected statutory language will never be replaced by an automated translation string.

Please note that if the user interface is rendered in a language that is different than the selected statutory language, a warning will be displayed in the editor. If the user nevertheless proceeds to capture data in the non-statutory user interface, an automated translation string will not be fetched for the statutory language and the value for the statutory language will be empty.

- For details regarding how to capture data in a statutory language in the Alfabet user interface, see the section *Capturing Data in a Statutory Language* in the reference manual *Getting Started with Alfabet*.

The following configuration is required to implement the statutory language capability:

- For all object classes  for which object data translation should be possible: Set the **Enable Statutory Language** attribute for the relevant object class to `True`. For more information about specifying a statutory language for an object class, see the section Configuring Custom Properties for Protected or Public Object Classes in the chapter Configuring the Class Model.
- For all object class stereotypes for which object data translation should be possible: Set the XML attribute `StatutoryLanguage` in the **Stereotypes** attribute of the object class to "`true`". If the XML attribute `EnableStatutoryLanguage` is not set for an object class stereotype in the Stereotypes attribute, the object class stereotype will inherit the definition of the **Enable Statutory Language** attribute specified for the object class. For more information about specifying a statutory language for an object class stereotype, see the section Configuring Object Class Stereotypes for Object Classes in the chapter Configuring the Class Model.

- The automated translation capability as described in the section Configuring Automated Translation of Object Data must be configured and all requirements to successfully fetch translations from the configured translation service must be fulfilled.

  Every new object that is created requires a value in the **Name** attribute for the primary language. If the data is originally captured in a secondary language, then the **Name** attribute must be specified in the primary language either via a manual translation or via the translation capability.

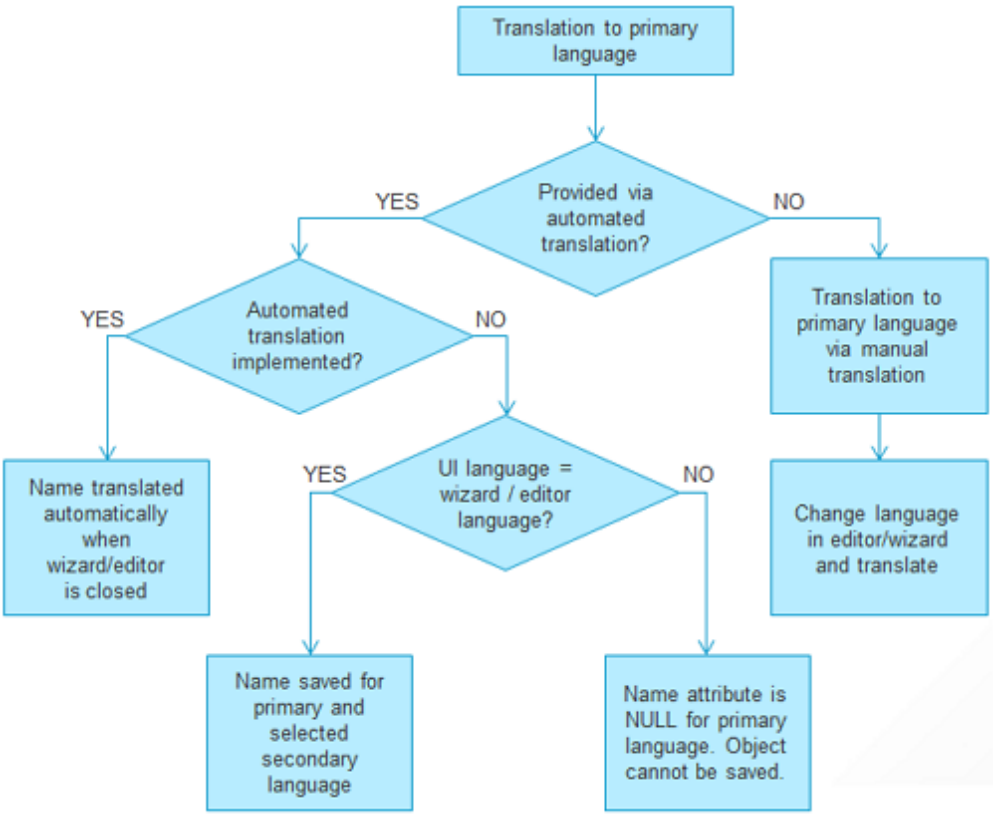


FIGURE: Translation to primary language

Please note the following:

- If the translation to the primary language is provided via the automated translation capability:
  - If the automated translation capability is implemented, the name will be automatically translated to the primary language when the editor/wizard is closed.
  - If the automated translation capability is **not** implemented, please consider the following:
    - If the language of the user interface is the same as the language selected in the editor/wizard, the value entered in the **Name** field will be saved for both the selected secondary language as well as the primary language. The language can be manually translated to the primary language in the editor after the new object has been saved.
    - If the language of the user interface is not the same as the language selected in the editor/wizard, the object cannot be saved because the **Name** attribute is NULL for

the primary language. In this case, you must change the language of the user interface to the language that you want to capture the data in.

- If a translated string is changed, the automated translation will be triggered again for all languages. The original string captured in the secondary or statutory language will not be changed via the automated translation.

- If the translation to the primary language shall be defined via a manual translation:

- Enter the value in the editor field, change the language selector at the bottom of the editor to the primary language, and enter the translated value in the relevant editor. Click the **OK** button in the editor/the **Next** button in the wizard to save the definitions.

- If no manual translation is defined for the primary language, the original string defined in the secondary language will be displayed for the **Name** when the user interface is rendered in the primary language. This may be revised at any time in the editor/wizard.

- If no manual translation is defined for any of the other secondary languages, the value displayed for the **Name** when the user interface is rendered in the primary language will also be displayed when the user interface is rendered in the secondary language.

- The XML attribute `EnableTranslationToPrimaryLanguage` in the XML object ***SolutionOptions*** must be set to `True`. For more information, see the section Configuring Automated Translation of Object Data. Please note that the XML attribute `EnableTranslationToPrimaryLanguage` must be set to `True` and the **Capture Translations in Language of User Interface** field in the user's **User Settings** editor must be selected (=True) so that the statutory language selected by the user in the statutory language selector will also be selected per default in the editor/wizard.

- In order for users to be able to capture language in a secondary language, they must select the checkbox for the **Capture Translations in Language of User Interface** field in their **User Settings** editor. For more information, see the section *Defining Your User Settings in Alfabet*. For more information about the configuration requirements to capture data in a secondary language, see the section Allowing Data To Be Captured in a Non-Primary Language in the reference manual *Configuring Alfabet with Alfabet Expand*.

- A user with an administrative profile may review the values defined for the **Name** attribute of the objects with a statutory language definition and change them as needed in the *Statutory Language Review for Data Translations* functionality. Furthermore, the administrative user can change the statutory language or remove the statutory language definition if needed For more information, see the chapter *Managing the Translations for the Enterprise's Statutory Language* in the reference manual *User and Solution Administration*.

- A user with an administrative profile may review the translation strings fetched via the automated translation capability and modify them as needed in the *Automated Data Translation Functionality*. For more information, see the chapter *Managing Automated Translation Strings* in the reference manual *User and Solution Administration*.

Translating the Standard Online Help Provided with Your Solution

After translating the Alfabet interface to a new language or after adapting the Alfabet terminology to you company's terminology, you may want to translate or modify the context-sensitive online help of the Alfabet interface. Software AG provides a context-sensitive online help for the Alfabet software. The online help delivered with your Alfabet software contains documentation for all functionalities in the software, including those that may not be part of your company's Alfabet configuration.

A standard online help is provided for `English (United States)1033` and `German (Germany)1031`. If the **Base Culture** attribute is set to a language other than `English (United States)` or `German (Germany)`, an online help will not be available unless you explicitly specify one. You can either copy the existing online help for `English (United States)1033` or `German (Germany)1031` or provide a translated version of the copied help files.

Context-sensitive custom online help can be made available for a standard business function/custom explorer, custom object profile and object cockpit, standard page view, or configured report included in the Alfabet user interface. Additionally, a custom help link may be made available for a user profile, thus allowing information to be available to relevant users regardless of the view that they are currently looking at. This type of custom help is independent of the help files specified in the Help folder as well as the culture setting definition. The custom online help is configured on a per-view basis and is based on the URL configured for that view. The custom help is viewed in a browser. For more information, see the chapter Providing Custom Online Help to the User Community.

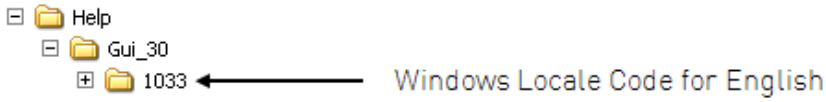
For more information about using the Alfabet online Help, see the section *Using the Context-Sensitive Help* in the reference manual *Getting Started with Alfabet*.

The following information is available:

- Finding the Online Help Files for Alfabet
- Adapting the Online Help Files to Include Custom Terminology
- Creating a New Language Version of the Online Help

Finding the Online Help Files for Alfabet

The Alfabet online Help files are located in the sub-directory `Help` of the Alfabet Web Application. The directory `Help` contains the sub-directory `Gui_30`, which contains a sub-folder for each language version of the online help. By default, Software AG provides an `English (United States)` and `German (Germany)` version of the online help.



The name of the folder containing the help files for a language version is identical to the name of the Microsoft® locale ID (LCID) of the language that the online help is used for

The Alfabet interface is available in the following languages:

Language	Locale ID
Arabic (Saudi Arabia)	1025
German (Germany)	1031
English (United States)	1033
French (France)	1036
Portuguese (Brazil)	1046
Polish (Poland)	1045

Therefore, if a user opens the Alfabet interface with a culture setting where the **Base Culture** attribute is set to `German (Germany)`, for example, the online help will be called from the folder with the locale identifier 1031. Within each language folder, four sub-folders are available:

- `images`: This folder contains images for all parts of the help in GIF or PNG format.
- `Alfabet`: This folder contains the context-sensitive HTML files for the Alfabet online Help
- `eXpand`: This folder contains the HTML files for the reference manual *Configuring Alfabet with Alfabet Expand*. The files are accessible via hyperlinks from both the Alfabet online Help and the `eXpandTech` help files. Please note that the German language folder contains English HTML files because the documentation of the configuration tool Alfabet Expand is not translated.
- `eXpandTech`: This folder contains the context-sensitive HTML files for the configuration tool Alfabet Expand. Please note that the German language folder contains English HTML files because the documentation of the configuration tool Alfabet Expand is not translated.

Adapting the Online Help Files to Include Custom Terminology

An HTML file provided by Software AG for the online help can be opened and changed with any HTML compliant editor. If, for example, your company uses a different term for the phrase "application variant", you could change the terms in the online Help file describing the Alfabet functionality for creating and maintaining application variants.

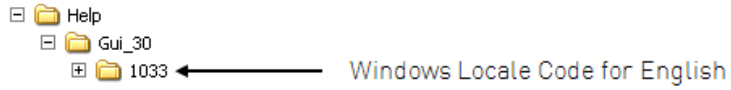
The name of each HTML file reflects the technical name of the view. If you do not know the technical name of the relevant view, you could call up the context-sensitive Help for that view in Alfabet. The technical name of the view is displayed in the path information when opening the context-sensitive help for the view. For example, the address bar displays the following path information for the **Application Variants** page view: `http://pc-customer/ALFABET_HELP/Gui_30/1033/OpenHelp.html?Path=Alfabet,APP_Variants.html`

The name of the HTML file for this view is `APP_Variants.html`. You could either access this HTML file in the `Alfabet` folder or edit the HTML file directly via an HTML compliant editor.

Changes applied to the HTML files located in the `Alfabet` folder will be automatically displayed to the users opening the help via the Alfabet interface.

Creating a New Language Version of the Online Help

You can create new language versions of the online help. The new online help requires the same folder structure as the standard online help and the names of the HTML files may not be changed.



The translated online help files must be placed in a sub-folder of the `Gui_30` folder. The sub-folder containing the translated HTML files must be named with the relevant Microsoft® locale ID (LCID) relevant for the culture specified in the **Base Culture** attribute of the relevant culture setting.

The full-text search available for the standard online help will not work for the new version because no help index is available for the online help. A help index can only be created by Software AG Support.

To create a new language version of the online Help:

1) In the sub-folder `Gui_30` of the online help, create a new folder with the locale identifier (LCID) associated with the language that online help will be translated to.
2) Copy the folder labelled `1033` containing the English help, paste it to a new folder below the `Gui_30` node labeled with the standard locale ID for the language you want to provide. For example, for a `Portuguese (Brazil)` culture setting, you would label the folder `1046`. The folder containing the custom help should correspond to the locale ID for the locale specified in the **Base Culture** attribute.
3) Translate the content of the HTML files. The online help files are written in standard compliant HTML. The online help can be translated with any standard translation tool or opened and modified with any standard HTML editor. Please note that the online help is context-sensitive and the file names of the HTML files may not be changed.

   Custom help can be made available for a standard business function/custom explorer, custom object profile and object cockpit, standard page view, or configured report included in the Alfabet user interface. Additionally, a custom help link may be made available for a user profile, thus allowing information to be available to relevant users regardless of the view that they are currently looking at. This type of custom help must be available via a URL that can be viewed in a browser. For more information, see the chapter Providing Custom Online Help to the User Community.
4) Replace the original language files with the translated files.
5) Ensure that the culture associated with the LCID is specified in the **Base Culture** attribute of the relevant culture.

Chapter 5: Configuring the Class Model

The standard Alfabet meta-model includes a class model that is made up of object classes used in the Alfabet solution, the properties of the object classes, and the relationships between object classes. The following chapter describes information important to understanding the standard meta-model as well how to customize it to meet the needs of your enterprise.

The **Meta-Model** tab allows you to view and edit the class model including the creation and definition of new custom object class properties required to meet the specific data input needs of your organization as well as the creation of object class stereotypes for a specified set of object classes. In the context of class model configuration, information is also provided about the creation of enumerations, which are necessary for the configuration of some types of custom object class properties as well as the configuration of class keys which determine uniqueness constraints for specified properties when users create a new object.

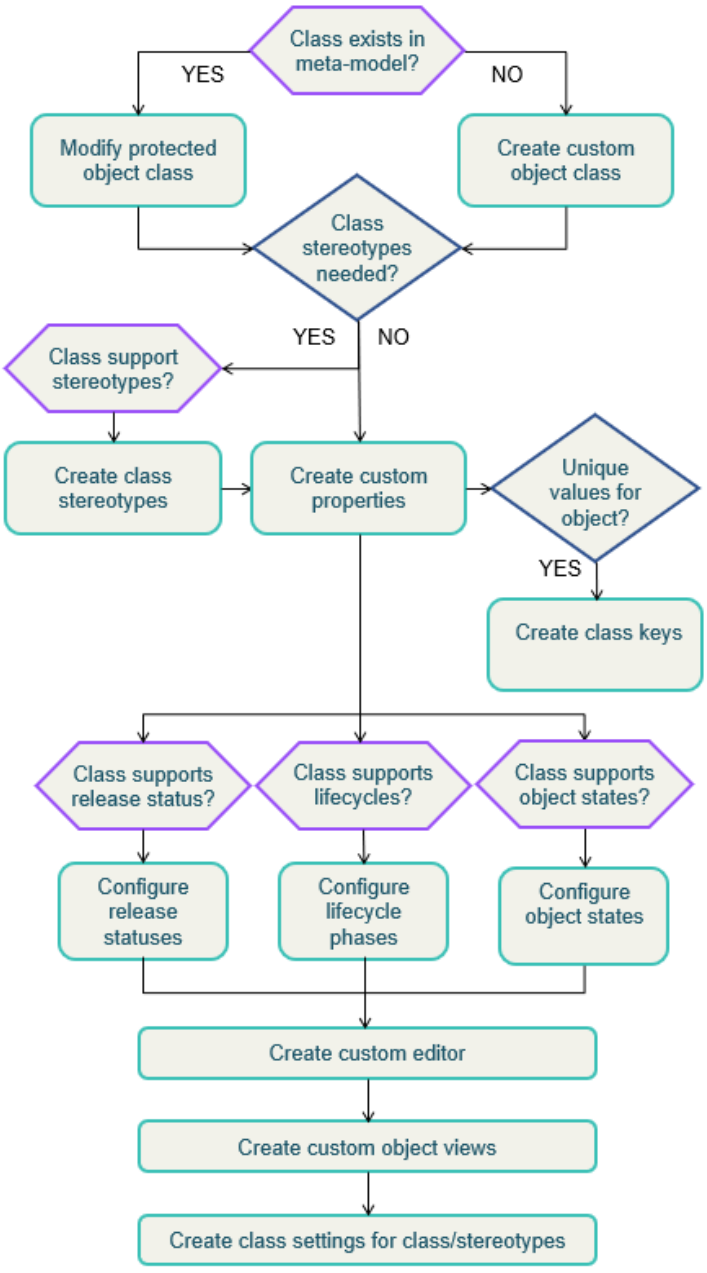


FIGURE: Steps to configure a standard or custom object class

Please note that the reference manual *Alfabet Meta-Model* provides detailed information about all standard object classes and their standard object class properties.

Furthermore, this chapter includes the specification of the release status, object state, and lifecycle definition for those object classes that support these concepts. For an overview of the object classes that implement the release status, object state, and lifecycle concepts, see the chapter *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*

Any changes made to the meta-model will impact both the relevant class table and audit table in the Alfabet database. For example, if a custom object class property is deleted from the class model, it will also be deleted from the audit table and thus will no longer be displayed in the *Object Audit Page View* available in your Alfabet solution.

It is highly recommended that you do not change the meta-model once the auditing capability is implemented, otherwise errors may occur in the audit information.

A custom object class can be created for special circumstances. For example, some companies may require a custom object class that represents an interface to support data migration. A custom object class is a public class and can be deleted at any time. Custom object classes should only be created with the assistance of Software AG Support.

The following information is available:

- Configuring the XML Object ReleaseStatusDefs
- Configuring Object State Definitions for Object Classes
- Configuring Lifecycle Definitions for Object Classes
- Configuring Class Keys for Object Classes
- Specifying History Tracking for an Object Class

Understanding the Class Model

The Alfabet class model is based on object classes, some of which are visible in the configuration tool Alfabet Expand. Each object class has a set of preconfigured object class properties and attributes.

The class model can be grouped into three different categories of object classes:

- The extendable meta-model that represents artifact object classes. Artifact object classes represent any object class for which object classes can be created in the Alfabet solution. These classes are typically protected object classes  for which protected object class properties can be modified and custom object class properties can be created. These classes are used to persistently store data about the customer's IT landscape in the Alfabet database. Most of these classes are customizable and customers can add custom object class properties to the classes to store company specific data that is not reflected in the standard properties of the object class. These object classes are visible in the **Class Model** explorer in Alfabet Expand.
- Object classes that represent objects designed to support functionality. These classes are used to persistently store data about object classes supporting functionality. This includes, for example, the object class `TimeStatus` that is used to store the lifecycle of objects or the object class `RoleTypeConfig` that is used to store information about which roles a user can have with regard to an object. Only the most important object classes for the storage of functionality-related data are visible in the explorer in Alfabet Expand.
- Auxiliary constructs required for Alfabet functionality. Some of the auxiliary object classes are persistently stored in Alfabet database tables, whereas others are not persistently stored. These object classes are not visible in Alfabet Expand and are not explicitly relevant to tasks of the solution designer.

An Alfabet database table exists for each object class displayed in the Alfabet Expand interface (as well as those that are not visible). Each class property displayed below its object class node corresponds to a column in the Alfabet database table.

The object classes that are displayed are relevant to the configuration of the Alfabet solution. Please note that the **Class Model** explorer displays the object classes as a flat list. Each object class in the meta-model has a preconfigured set of object class properties that serve to semantically describe the object class. These are nested below the object classes.

The following information is available:

- Overview of the Classes Node in the Class Model Explorer
- About Object Class Names and IDs
- About the Stereotypes Attribute for an Object Class

- About the Inheritance of Access Permissions
- About the Dependencies of an Object Class
- About the Tracking of Object Changes for an Object Class
- About XML-Based Configurations Relevant to Object Classes
- About the Configuration of Reference and Evaluation Data for an Object Class
- About Mandatory Object Class Properties

Overview of the Classes Node in the Class Model Explorer

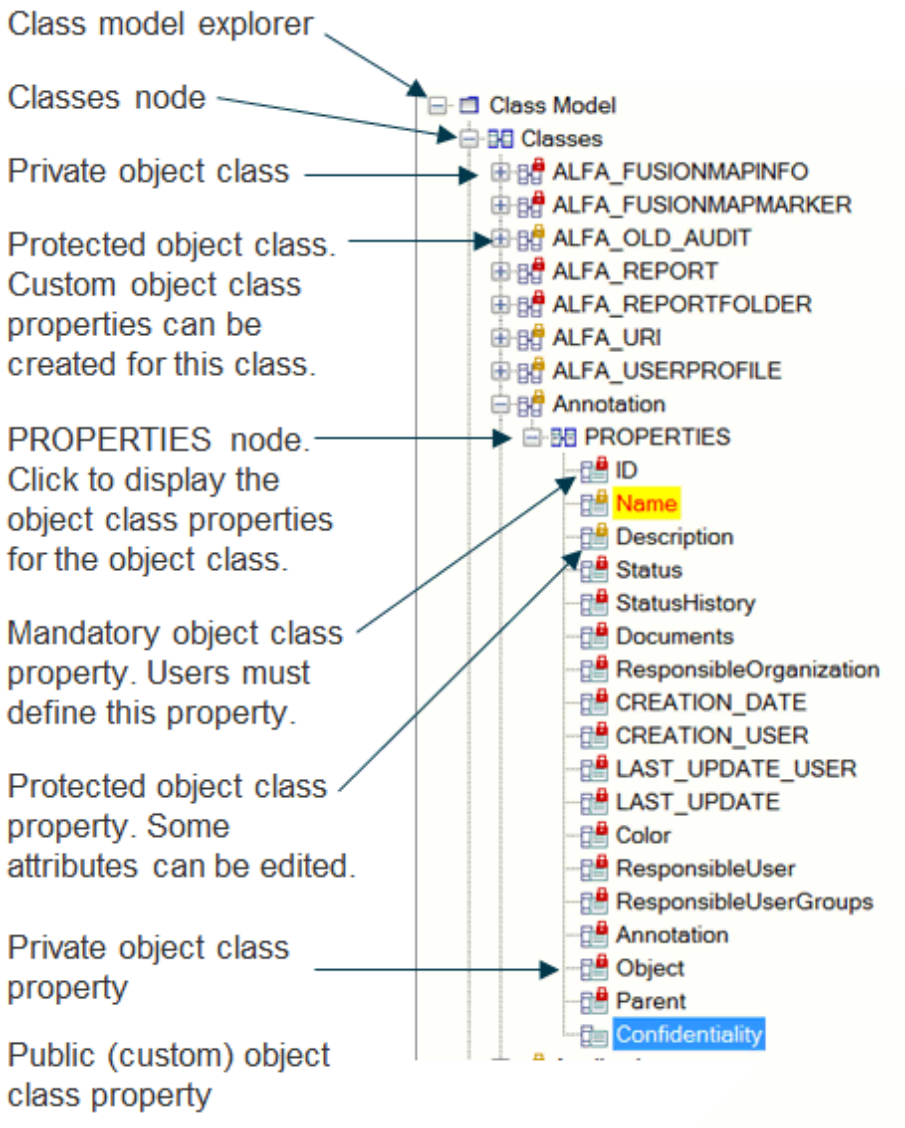


FIGURE: Class Model explorer in the Meta-Model tab

When you expand the **Classes** node, you may see any of the following:

- A private object class cannot be modified. Custom object class properties cannot be configured for private object classes. Private object classes are displayed for informational purposes and can be configured in Alfabet queries.
- A protected object class can be edited in a limited way. A protected object class typically has private object class properties that cannot be edited, protected object class properties that may be edited (for example, `ResponsibleUser`, `Documents`, `CREATION_DATE`, etc.) but allows for custom object class properties to be created and configured for a protected object class.
- A public object class is a custom class that has typically been created by your enterprise with the help of Software AG Support. All relevant object class properties may be edited and the custom class may be deleted. The attribute **Tech Name** must be explicitly defined for call custom object classes. Custom object class properties can be created and configured for a public object class.
- Some private and protected object class properties are highlighted yellow in the **Class Model** explorer. These are standard object class properties that are recommended by Software AG to be mandatory properties. No enforcement mechanism is implemented if the object class property is undefined. However, if the object class property is included in a class key definition, an error will occur if the object class property is not defined. Therefore, if you include a mandatory object class property in a class key, the object class property may not be hidden in the editor or wizard used to create new objects of the relevant property class.

In Alfabet Expand, object classes are displayed with their names, which means the value of the **Name** attribute of the object class. In some cases, this may differ from the caption used in the standard Alfabet user interface.

When you expand the **Properties** node below an object class, you may see any of the following:

- A private object class property cannot be modified. Private object class properties are displayed to provide guidance in defining new objects.

  Please note that the **Hint** attribute cannot be defined for private object class properties . If you need to provide custom tooltips for private object class properties, you can either:

  - Display the object class property in an object cockpit and configure a hint for the interface control. In this way, you can define a customized tooltip for the specific context that the object class property is displayed in each relevant object cockpit. For more information, see the section Configuring Object Cockpits for a Custom Object View.
  - Define a custom translation to replace the standard tooltip via the translation capability available in Alfabet. The translation of the standard tooltip will be displayed in all instances in Alfabet in which it appears. For inherited object class properties such as `Name` and `ID` which are used in most protected object classes in Alfabet, the custom translations will be displayed for all `Name` and `ID` properties for all relevant object classes. For more information, see the chapter Localization and Multi-Language Support for the Alfabet Interface.

- A protected object class property  is a standard property that allows some attributes for the object class property to be edited including, for example, the **Caption** attribute, which is displayed for the object class property in the Alfabet interface, and the **Hint** attribute, which is the text for the tooltip that is displayed in the **Attributes** section of an object view and the editor. Both the **Caption** attribute and **Hint** attribute can be translated for the Alfabet interface. A protected object class property may not be deleted.
- A public object class property  is a custom object class property that has typically been created by your enterprise. All relevant attributes may be edited and the custom object class property may be deleted.

Click any object class node in the **Class Model** explorer to display its attributes in the attribute window. These attributes specify the technical data about the object class property. The attributes displayed in the attribute window will depend on the object class property type definition in the **Type** attribute. For more information about configuring object classes and object class properties, see the chapter Configuring the Class Model.

Some object classes have a **Class Key Group** node below the object class which will display any existing class keys. A class key  allows for the specification of a unique combination of values for a specified set of standard or custom object class properties. This may be implemented to specify uniqueness constraints for the definition of an object in the object class as well as to create indexes to improve performance. For more information about the configuration of class keys, see the section Configuring Class Keys for Object Classes.

Below the **Class Model** node, you will also see a node labelled **Enums**, which may contain private, protected, and public enumerations. Enumerations allow you to define values that can be selected in combo-box or checked list box interface controls. Once an enumeration is defined, you may assign it to any custom object class property of the type `String`, `Real`, or `Integer` associated with a protected object class. An enumeration can be reused for numerous object class properties for multiple object classes. For more information about the configuration of enumerations, see the section Defining Protected and Custom Enumerations.

You may see any of the following under the **Enums** node:

- Private enumerations  are for reference purposes only and cannot be edited.
- Protected enumerations  are created by Software AG. The enumeration name cannot be changed but you can edit the values specified in the **Sort Enumeration Items** attribute in order to determine which values should be displayed to users in the Alfabet interface.
- Public enumerations  are custom enumerations. All attributes can be edited.

About Object Class Names and IDs

Select an object class in the **Class Model** explore to display its attribute window in the right pane. The attribute window displays the attributes that can be defined for the object class. The following attributes are relevant to the identification of the object class:

- **Caption**: Specifies the name of the object class or object class property displayed in the Alfabet user interface.

- **Name**: Used to identify the object class in Alfabet Expand and to specify the object class in Alfabet queries and when defining configuration objects such as XML objects. This attribute is unique for each object class in the meta-model as well as for each property in the object class. In some cases, the **Name** may differ from the caption used in the standard Alfabet interface.

  A validation mechanism checks for correct syntax when defining a technical name. The technical name of a configuration object may not begin with an empty space nor include any of the following special characters: **\ / * ? " > < | ':**

  Furthermore, please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.

  Furthermore, if the technical name of a configuration object is changed, the name will be correctly updated in other configuration objects referencing the changed object during design time. Please note that the name of a changed configuration object will not be updated in guide pages nor if the guide pages reference a configuration object listed in the **Show Usage** functionality. If you plan to change the name of a configuration object, the reference in the guide page should be changed prior to changing the configuration object name.

- **Tech Name**: Specifies the name of the Alfabet database table column for the object class property. This attribute is unique for each object class. The **Tech Name** attribute is used in native SQL queries to identify the object class.
- **ID Prefix**: The automatically-generated abbreviation of typically 2 or 3 letters to identify the object class (for example, `APP` for `Application` or `BD` for `BusinessData`). The prefix is used to generate the ID for a new object. The object identification number will include the prefix of the object class, followed by an automatically-generated identification number (for example: `APP-378`, whereby `APP` is the value for the **ID Prefix** attribute and `378` is the value for the **ID** attribute). Users can search and identify objects in Alfabet based on their object ID.
- **ID**: The automatically-generated identification number for the object class (for example: `APP-378`, whereby `APP` is the value for the **ID Prefix** attribute and `378` is the value for the **ID** attribute). This attribute cannot be edited and is not visible in the Alfabet interface.
- **Validator**: This attribute is only available for some object class properties of the type String, such as the object class property `Name`, `ShortName`, or `Version`. This attribute allows regular expressions according to Microsoft® syntax conventions to be specified that enforce that the values defined for the attribute have a specific structure. For example, on the object class property `Version`, the **Validator** attribute could specify that version numbers should always be defined as two digits made up of a period and another digit. The validator is enforced on all editors where the object class property can be edited for the relevant object class. Users will be prompted with an error message if the value provided does not match the specified validator. The error message will state that the input value has an invalid format and will display the informational text about the validator that your enterprise has specified in the **Comments** attribute for the custom object property. It is your responsibility to inform users about custom validators configured for object class properties in your enterprise's solution configuration. Please note that specific enforcement mechanisms must be implemented for existing records or records that are imported to Alfabet from external systems. For more information about the syntax conventions for regular expressions, see http://msdn.microsoft.com/en-us/library/hs600312(VS.71).aspx.

About the Stereotypes Attribute for an Object Class

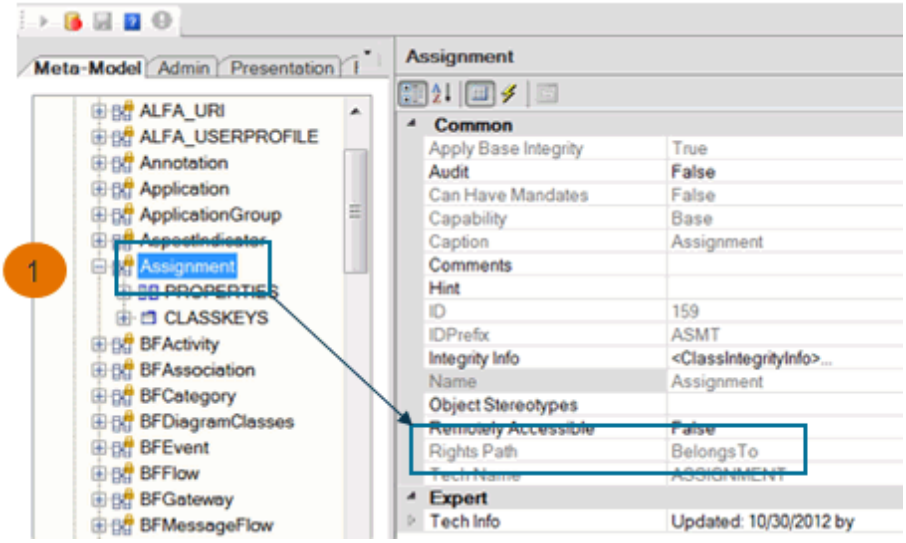
An object class stereotype is a sub-classification of an object class. A permissible object class can have multiple object class stereotypes, each of which captures a specified set of attributes, reference data, and class configurations. For example, the object class Application that may have the object class stereotypes Business Applications and Technical Applications.

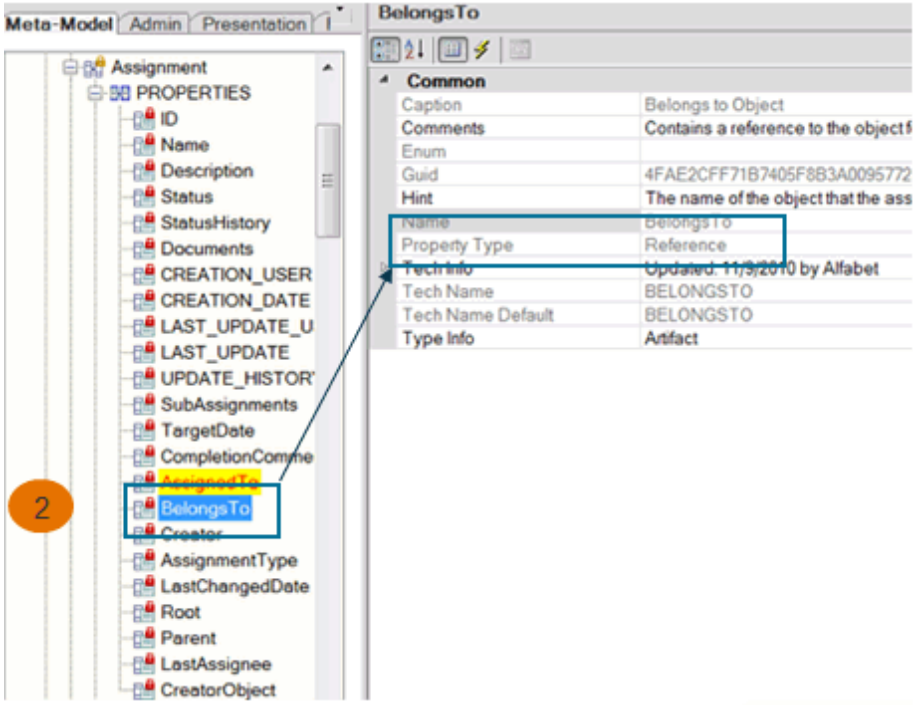
Only a limited number of object classes support the configuration of object class stereotypes.

For an overview of the object classes that support the object class stereotype concept, see the chapter *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*. For detailed information about how to configure object class stereotypes, see the section Configuring Object Class Stereotypes for Object Classes.

About the Inheritance of Access Permissions

In the Alfabet meta-model, objects in an object class may inherit the access permissions from another object class. The **Rights Path** attribute for an object class indicates if access permissions are inherited and which object class determines the access permissions for the object of that object class. If the **Rights Path** attribute is empty, access permissions are not inherited from another object class. If access permissions are inherited, the **Rights Path** attribute will indicate which object class property determines the access permission.





For example, for the object class `Assignment`, the **Rights Path** attribute is set to `BelongsTo`. The object class property `BelongsTo` is of the type `Reference`. Thus, the object that the assignment belongs to also determines the access permissions for the assignment. In other words, only the owner of an application, for example, has access permissions to its assignment.

Or for example, for the object class `BusinessProcessVariant`, the **Rights Path** attribute is set to `Object`. The object class property `Object` is set to `BusinessProcess`. Thus, the business process variant inherits its access permissions from the business process that it is derived from.

About the Dependencies of an Object Class

In the Alfabet meta-model, objects in an object class may have a dependent relationship to objects in other object classes. The **Integrity Info** attribute defined for an object class (considered the base object class) specifies the object classes that are dependent on that base object class. If an object in the base object class is deleted, all dependent objects will also be deleted. For example, if an application is deleted, all information flows for which the base application is either the owner application (if **Integrity Info** attribute = `Owner`) or the referenced incoming or outgoing application (if **Integrity Info** attribute = `To` or `From`) will also be deleted.

The dependencies between private and protected object classes are part of the standard configuration and should not be customized. An error message will be displayed if syntax errors are written in the XML definition of the **Integrity Info** attribute. For an overview of the objects that are impacted by the deletion of an object in Alfabet, see the description of the relevant object class in the reference manual *Alfabet Meta-Model*.

A service is offered by Software AG that allows the integrity information to be defined on the slave object classes in a master/slave class relationship if a custom class has been introduced to the meta-model. This service ensures that if an object in a custom object class is deleted, the

referenced Alfabet objects will also be deleted. For more information about the definition of the **Integrity Info** attribute for slave object classes, please contact Software AG Support.

About the Tracking of Object Changes for an Object Class

Software AG provides a history tracking functionality that documents the changes made to an object in the context of the Alfabet solution. All changes made to an object's standard and custom object class properties and relations are documented.

The history tracking functionality is implemented on a class-by-class basis. History tracking is available for all protected classes  and public classes  . Please keep the following in mind:

- The **Audit** attribute is set to `True` per default for relevant IT object classes so that the history tracking functionality activated. Please be aware that tracking the history of a significant number of object classes may lead to a decrease in performance. It is recommended that your enterprise consider whether some object classes do not require the history tracking functionality to be activated.
- The history tracking functionality tracks only the change history of objects in the Alfabet solution. A deleted object cannot be restored via the history tracking functionality. If an object has been erroneously deleted, the object must be recreated and redefined in the Alfabet solution.

For more information about configuring history tracking, see the section Specifying History Tracking for an Object Class.

About XML-Based Configurations Relevant to Object Classes

Many functionalities relevant to the implementation of object classes in the Alfabet solution are controlled via an XML object definition. For example, multiple object classes have an object class property **Status** that allows the release status of the object in the approval process to be defined; the possible values for release status definitions must be configured in the XML object ***ReleaseStatusDefs***.

For information about some of the typical configurations required for an object class, see the following sections:

- Configuring Release Status Definitions for Object Classes
- Configuring Object State Definitions for Object Classes
- Configuring Lifecycle Definitions for Object Classes

About the Configuration of Reference and Evaluation Data for an Object Class

The data that can be captured for some object class properties may be determined by the configurations made for the associated object class. Evaluation types and their indicator types, cost types and income types, role types, and diagram views are examples of some of the configured objects that may be assigned to a set of object classes used in your enterprise.

The configuration of evaluation types, indicator types, cost types, income types, role types and diagram types is carried out in the **Configuration** module of Alfabet. The configuration objects are created independent of an object class. For example, an evaluation type may be created for the enterprise and then assigned to many different object classes in the context of the **Class Configuration** functionality in the **Configuration** module. The configuration objects must explicitly be assigned to each object class that they are relevant for. This allows you to create reference or evaluation data that is relevant to a specific object class context. For example, indicators required to evaluate applications typically differ from the indicators required to evaluate devices.

The `TechnicalName` property for the object classes `EvaluationType`, `IndicatorType`, and `RoleType` has been set to protected. It is recommended that you specify the **Validator** attribute of the **Name** property for these classes to indicate that only letters, numbers, blank spaces, underscores, and slashes are allowed as characters for the technical name. To do so, you should enter the following for the **Validator** attribute of the **Name** property for these classes: ^[A-Za-z_0-9\s\/]+$

The validator is enforced on all editors where the object class property is editable. Users will be prompted with an error message if the value provided does not match the validator specified. The error message will state that the input value has an invalid format and will display the text defined in the **Comments** attribute for the custom object class property. It is your responsibility to inform users about custom validators configured for object class properties in your enterprise's solution configuration. Please note that specific enforcement mechanisms must be implemented for existing records or records that are imported into Alfabet from external systems. For more information about the syntax conventions for regular expressions, see http://msdn.microsoft.com/en-us/library/hs600312(VS.71).aspx.

For detailed information about the configuration of reference and evaluation data, see the reference manual *Configuring Evaluation and Reference Data in Alfabet*.

About Mandatory Object Class Properties

Private and protected object class properties that are preconfigured to be mandatory are highlighted yellow in the **Class Model** explorer. These are standard object class properties that are recommended by Software AG to be mandatory properties. No enforcement mechanism is implemented if the object class property is undefined. Mandatory properties may be hidden from editors and wizards as long as they are not included in a class key definition. If the mandatory property is hidden in an editor or wizard, the user is not required to enter a value for the object class property

However, if the object class property is included in a class key definition, an error will occur if the object class property is not defined and users will be prompted to enter a value for the mandatory property. Therefore, if you include a mandatory object class property in a class key, the object class property may not be hidden in the editor or wizard used to create new objects of the relevant property class. For information about the definition of class keys, see the section Specifying History Tracking for an Object Class.

For more information about hiding object class properties in editors or wizards, see the section Refining Visibility Issues in the View Scheme in the chapter Configuring User Profiles for the User Community.

Custom properties may be configured to be mandatory in the context of the custom editor that they are implemented in. In this case, the custom object class property itself is not mandatory but rather the interface control associated with the object class property. For more information, see the section Specifying the Interface Control To Be Mandatory in the chapter Configuring Custom Editors.

Editing a Protected Object Class

A protected object class can be edited in a limited way. A protected object class typically has private object class properties that cannot be edited, protected object class properties that may be edited (for example, `ResponsibleUser`, `Documents`, `CREATION_DATE`, etc.). New object class properties (referred to as custom or public object class properties) can also be created for a protected object class. If custom properties are created for a protected object class, you will typically require a custom appropriate editor in order to allow data to be captured for the custom properties. The creation of custom properties is described in detail in the section Configuring Custom Properties for Protected or Public Object Classes. In addition to creating custom properties for the protected object class or modifying its protected properties, some attributes of the protected object class can be changed.

The **Anonymize** and **Property Anonymization** attributes are relevant for the data anonymization capability, which ensures data transparency and accountability across the enterprise. This capability requires multiple configuration steps and therefore the **Anonymize** and **Property Anonymization** attributes are not described here. For more information about configuring data anonymization, see the section Anonymizing Data in the chapter Executing Administrative Tasks in Alfabet Expand.

To edit the attributes of a projected object class:

1) Expand the **Classes** node and click the protected object class that you want to edit. The attribute grid is displayed.
2) Edit the following attributes, as needed:

- **Can Have Mandates**: Select `True` if the mandate capability should be available for the selected class. Select `False` if the mandate capability should not be available. For detailed information about the implementation and configuration of a federated architecture and mandates, see the section Implementing the Mandate Capability for a Federated Architecture in the chapter Configuring Access Permissions for Alfabet .
- **Audit**: Select `True` if an audit table should be generated in the Alfabet database. Select `False` if an audit table should not be generated in the Alfabet database. Please be aware that tracking the history of a significant number of object classes may lead to a decrease in performance. Furthermore, changes to some properties for some classes may also produce an excessive number of entries in the Alfabet database table. For more information about configuring the history capability, see the section Specifying History Tracking for an Object Class.
- **Comments**: Specify comments relevant to the configuration of the object class. The comments will not be displayed in the Alfabet user interface.
- **Enable for Data Capture Templates**: Select `True` if the object class may be captured via data capture templates. Select `False` if the object class may not be captured via data capture templates. For each object class for which the **Enable for Data Capture Templates** attribute is set to `True`, you must enable the individual properties that shall be captured for the object class by setting the **Enable for Data Capture Templates** attribute for each object class property to `True`. For information about specifying the **Enable for Data Capture Templates** attribute for an object class property, see the section Editing a Protected Property.

  Data capture is limited to `Artifact` object classes only and the **Enable for Data Capture Templates** attribute may only be specified for those classes for which data

capture is permissible. If data capture is not permissible for a class, the attribute will be greyed out.

For more information about data collection via data capture templates, see the section *Capturing Data with Data Capture Templates* in the reference manual *User and Solution Administration*.

- **Integrity Info**: In the Alfabet meta-model, objects in an object class may have a dependent relationship to objects in other object classes. The **Integrity Info** attribute defined for an object class (considered the base object class) specifies the object classes that are dependent on that base object class. If an object in the base object class is deleted, all dependent objects will also be deleted. For example, if an application is deleted, all information flows for which the base application is either the owner application (if **Integrity Info** attribute = `Owner`) or the referenced incoming or outgoing application (if **Integrity Info** attribute = `To` or `From`) will also be deleted. For an overview of the consequences of deleting an object in Alfabet, see the class descriptions in the reference manual *Alfabet Meta-Model*.
- **Stereotypes**: If necessary, specify object class stereotypes for the public object class. For more information about specifying object class stereotypes, see the section Configuring Object Class Stereotypes for Object Classes.
- **Enable Data Translation**: If the **Support Data Translation** attribute is set to `True` for a culture, data translation will be permissible for all classes per default. The **Enable Data Translation** attribute allows you to refine the culture definition and specify the permissibility of data translation for some but not all object classes as well as selected object class properties of the permissible object classes. You can refine permissibility of data translation as follows:

  Please consider the following:

  - Data translation is possible for some protected properties and all public properties of the type `Text` and `String`. You can further refine the permissibility to translate data for selected protected object class properties as well as public properties configured for the object classes for which the **Enable Data Translation** attribute is set to `True`. The specification of data translation for a protected property is defined in defined via the **Automated Translation Rules for Class Properties** attribute described below. For more information about the definition of the data translation for a custom property, see the section Configuring Custom Properties for Protected or Public Object Classes.
  - The specification of the **Enable Data Translation** attribute applies to all cultures defined for your enterprise. Therefore, if you specify that a data translation is not permissible for an object class, then object data cannot be specified for that object class for all defined cultures. Please note that regardless of the specification of the **Enable Data Translation** attribute for an object class, if the **Support Data Translation** attribute is set to `False` for a culture, data translation will not be possible at all for that culture and the language code will not be displayed in the language field in all object editors.If the **Support Data Translation** attribute is set to `False`, then the automated and manual data translation functionalities will be disabled and the language code will not be displayed in the language field in all object editors available for the class.

- The **Enable Data Translation** attribute must be set to `True` in order to enable the object class for either manual data translation or automatic data translation.
- If the **Enable Data Translation** attribute is set to `False`, then the automated and manual data translation functionalities will be disabled, and the language code will not be displayed in the language field in all object editors available for the class.
- For more information about the configuration of object data translation, see the section Configuring the Translation of Object Data in the chapter Localization and Multi-Language Support for the Alfabet Interface.

- To enable an object class for data translation, navigate to the relevant object class and set the **Enable Data Translation** attribute to `True`.
- To disable an object class for data translation, navigate to the relevant object class and set the **Enable Data Translation** attribute to `False`.
- **Enable Automated Data Translation**: Set to `True` if the object class may be translated automatically by the translation service specified in the XML object `AlfaTranslationServicesConfig`. For more information about the configuration of object data translation, see the section Configuring the Translation of Object Data in the chapter Localization and Multi-Language Support for the Alfabet Interface.
- **Automated Translation Rules for Class Properties**: If the **Enable Automated Data Translation** attribute is set to `True`: Click the **Browse** button to open the editor to specify which protected and custom properties may be translated automatically. For each property listed in the **Automated Translation Rules for Class Properties** editor, select True in the **Automated Translation Enabled** column to permit the property to be automatically translated.

  Please note that if the **Enable Data Translation** attribute is set to `None` or `Manual` for a custom property, the value in the **Automated Translation Enabled** column will be set to `False` per default and cannot be automatically translated.

- **Enable Statutory Language**: Set to `True` if the statutory language capability shall be available for the object class. The statutory language capability is available in order to address regulatory requirements that mandate that objects are captured in a specific language. When a user creates an object based on a relevant object class/object class stereotype, a statutory language selector will open in which the language can be selected. In the object editor that opens, the user can capture all attributes that support automated translation in the statutory language. If the XML attribute `EnableStatutoryLanguage` is not set for an object class stereotype in the Stereotypes attribute, the object class stereotype will inherit the definition of the **Enable Statutory Language** attribute specified for the object class. For more information about specifying a statutory language, see the section Specifying a Statutory Language for the Enterprise in the chapter Localization and Multi-Language Support for the Alfabet Interface.
- Attributes in the section **AlfaBot Support**: The attribute in this section are only editable for custom object class. They define if and how the AlfaBot can be used with respect to this object class. Each object class in the meta-model has an **AlfaBot Support** attribute section with four attributes. For standard private and protected object classes, the attributes are visible but not editable. The attributes can only be set for public, customer-defined object classes. The attributes **Creation Mode**, **Editing Mode**, and **Navigation Mode** specify whether and how creation of object, editing of objects or navigation to objects is supported for the object class. The attribute **Analysis Intent Mode** specifies

whether the object class is included into the search in configured reports that can be performed via the `Analyze` intent of the AlfaBot. The attributes can be set to one of the following:

- `None`: The functionality is not supported for the AlfaBot.
- `ContextDependent`: The functionality is supported by the AlfaBot and will take a required context into account. For example information flows can only meaningfully be created in the context of a source or target object. If the user wants to create an information flow via the AlfaBot, the relevant view for creation of the information flow will open, like for example the **Information Flows** page view of an application.
- `Full`: The functionality is supported by the AlfaBot context free. For the creation of objects with the **Creation Mode** attribute set to `Full`, the relevant editor or wizard will open.

> For an overview of the AlfaBot capability, see the section *Using the AlfaBot Capability* in the reference manual *Getting Started with Alfabet*. For information about how to implement the AlfaBot capability, see the section Implementing the AlfaBot for Navigation via a Full-Text Command.

- **Alias**: If the **Analysis Intent Mode** attribute of th object class is set to `Full`, a list of synonyms for the object class can be defined in this attribute. If a user searches for one of the synonyms in the `Analyze` intent of the AlfaBot, the AlfaBot will provide search results for the object class that the synonym is defined for.

3) In the toolbar, click the **Save** button to save the new custom object class.

Creating a Public Object Class

A public object class can be created for special circumstances. For example, some companies may require a custom object class to support setup interoperability to external repositories. In this case, a new object class must be created to capture the reference information to objects in the external repository. The new object class may have multiple properties, one of which typically references an object class/object class stereotype in Alfabet and another property that typically captures the ID of object in the linked repository. For example, the custom object class Application Incident would have a custom property referencing the Alfabet object class Application and a custom property to capture the ID of the linked object in the external incident management repository. Other custom properties can be specified that allow some of the data about the object from the external repository to be captured in Alfabet while other properties may be used to further describe the reference context in Alfabet. The latter would require an appropriate editor to be defined for the reference class (for example, Application Incidents).

Please note the following:

- A custom object class can be deleted at any time. Custom object classes should only be created with the assistance of Software AG Support.
- It is highly recommended that you do not change the technical name of an object class. if you do changed the technical name of an existing public class, the name of the associated database table must also be changed. Otherwise, an error will occur if an attempt is made to update the meta-model and the update of the meta-model will be aborted.

The **Anonymize** and **Property Anonymization** attributes are relevant for the data anonymization capability, which ensures data transparency and accountability across the enterprise. This capability requires multiple configuration steps and therefore the **Anonymize** and **Property Anonymization** attributes are not described here. For more information about configuring data anonymization, see the section Anonymizing Data in the chapter Executing Administrative Tasks in Alfabet Expand.

To create a public object class:

1) Right-click the **Classes** node and select **Add New Class**. The **Create New Class** editor opens. Define the following fields:

- **Class Name**: Enter a unique name for the custom object class. This value is displayed in the solution interface if the **Caption** attribute is not defined.
- **Tech Name**: Enter a unique technical name for the custom object class that will be used in the Alfabet database table. Click **OK** to save the definition.

Please note that the `Tech Name` is the name of the database table that will be created in the Alfabet database for storing object property data of objects of the custom object class. The following rules apply:

- The technical name may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.
- The technical name should only contain standard `ASCII` characters.
- The technical name should consist of upper-case letters only.
- The maximum length of a technical name may not exceed 30 characters.
- The technical name may not coincide with any of the reserved key words of the relational database management system.

A validation mechanism checks for correct syntax when defining a technical name.

2) The public object class is added to the explorer. Click the public object class and define the following attributes in the attribute grid:

- **Name**: This is the technical name of the object class. It is highly recommended that you do not change the technical name of an object class. if you do changed the technical name of an existing public class, the name of the associated database table must also be changed. Otherwise, an error will occur if an attempt is made to update the meta-model and the update of the meta-model will be aborted.
- **Caption**: Specify the caption of the object class that displayed in the Alfabet user interface. If no caption is defined, then no caption will be displayed in the Alfabet user interface.
- **Caption Plural**: Specify the plural form of the caption of the object class.
- **ID Prefix**: Enter the automatically-generated abbreviation of typically 2 or 3 letters to identify the object class (for example, `APP` is used for `Application` or `BD` is used for `BusinessData`). The prefix is used to generate the ID for a new object. The object identification number will include the prefix of the object class, followed by an automatically-generated identification number (for example: `APP-378`,

whereby `APP` is the value for the **ID Prefix** attribute and `378` is the value for the **ID** attribute). Users can search and identify objects in Alfabet based on their object ID.

- **Hint**: Enter informational text about the custom object class. Please note that the **Hint** attribute defined for the custom object class will not be displayed in the Alfabet user interface.
- **Rights Path**: In the Alfabet meta-model, objects in an object class may inherit the access permissions from another object class. The **Rights Path** attribute for an object class indicates if access permissions are inherited and which object class determines the access permissions for the object of that object class. Before you can specify the **Rights Path** attribute you must create the custom property that determines the access permissions for the object class. Once this has been done, enter the name of the custom property in the **Rights Path** attribute. If the **Rights Path** attribute is empty, access permissions will not be inherited from another object class. For more information about creating a custom property, see the section Configuring Custom Properties for Protected or Public Object Classes.

  For example, for the object class `Assignment`, the **Rights Path** attribute is set to `BelongsTo`. The object class property `BelongsTo` is defined for the object class `Assignment` and is of the type `Reference`. Thus, the object that the assignment has been created for (`BelongsTo`) also determines the access permissions for the assignment. In other words, only the owner of an application, for example, has access permissions to its assignment. Or for example, for the object class `BusinessProcessVariant`, the **Rights Path** attribute is set to `Object`. The object class property `Object` is set to `BusinessProcess`. Thus, the business process variant inherits its access permissions from the business process that it is derived from.

- **Apply Base Integrity**: Select `True` if all references to the deleted object should also be deleted if an object of the object class is deleted. The references are specified in the **Integrity Info** attribute. Select `False` if no integrity management should be performed for the object class. In order to improve performance, the value `False` may be set for some custom object classes if no object classes references the custom object class. However, it is highly recommended that you contact Software AG Support if you want to set the value to `False` for a custom object class. Setting the **Apply Base Integrity** attribute to `False` may result in serious inconsistency problems in the database.
- **Audit**: Select `True` if an audit table should be generated in the Alfabet database. Select `False` if an audit table should not be generated in the Alfabet database. Please be aware that tracking the history of a significant number of object classes may lead to a decrease in performance. Furthermore, changes to some properties for some classes may also produce an excessive number of entries in the Alfabet database table. For more information about configuring the history capability, see the section Specifying History Tracking for an Object Class.
- **Comments**: Specify comments relevant to the configuration of the object class. The comments will not be displayed in the Alfabet user interface.
- **Enable for Data Capture Templates**: Select `True` if the object class may be captured via data capture templates. Select `False` if the object class may not be captured via data capture templates. For each object class for which the **Enable for Data Capture Templates** attribute is set to `True`, you must enable the individual properties that shall be captured for the object class by setting the **Enable for Data Capture Templates** attribute for each object class property to `True`. For information about specifying the **Enable for Data Capture Templates** attribute for an object class property, see the section Editing a Protected Property.

  Data capture is limited to `Artifact` object classes only and the **Enable for Data Capture Templates** attribute may only be specified for those classes for which data

capture is permissible. If data capture is not permissible for a class, the attribute will be greyed out.

For more information about data collection via data capture templates, see the section *Capturing Data with Data Capture Templates* in the reference manual *User and Solution Administration*.

- **Integrity Info**: In the Alfabet meta-model, objects in an object class may have a dependent relationship to objects in other object classes. The **Integrity Info** attribute defined for an object class (considered the base object class) specifies the object classes that are dependent on that base object class. If an object in the base object class is deleted, all dependent objects will also be deleted. For example, if an application is deleted, all information flows for which the base application is either the owner application (if **Integrity Info** attribute = `Owner`) or the referenced incoming or outgoing application (if **Integrity Info** attribute = `To` or `From`) will also be deleted. For an overview of the consequences of deleting an object in Alfabet, see the class descriptions in the reference manual *Alfabet Meta-Model*.
- **Stereotypes**: If necessary, specify object class stereotypes for the public object class. For more information about specifying object class stereotypes, see the section Configuring Object Class Stereotypes for Object Classes.
- **Enable Data Translation**: If the **Support Data Translation** attribute is set to `True` for a culture, data translation will be permissible for all classes per default. The **Enable Data Translation** attribute allows you to refine the culture definition and specify the permissibility of data translation for some but not all object classes as well as selected object class properties of the permissible object classes. You can refine permissibility of data translation as follows:

  Please consider the following:

  - Data translation is possible for some protected properties and all public properties of the type `Text` and `String`. You can further refine the permissibility to translate data for selected protected object class properties as well as public properties configured for the object classes for which the **Enable Data Translation** attribute is set to `True`. The specification of data translation for a protected property is defined in defined via the **Automated Translation Rules for Class Properties** attribute described below. For more information about the definition of the data translation for a custom property, see the section Configuring Custom Properties for Protected or Public Object Classes.
  - The specification of the **Enable Data Translation** attribute applies to all cultures defined for your enterprise. Therefore, if you specify that a data translation is not permissible for an object class, then object data cannot be specified for that object class for all defined cultures. Please note that regardless of the specification of the **Enable Data Translation** attribute for an object class, if the **Support Data Translation** attribute is set to `False` for a culture, data translation will not be possible at all for that culture and the language code will not be displayed in the language field in all object editors. If the **Support Data Translation** attribute is set to `False`, then the automated and manual data translation functionalities will be disabled and the language code will not be displayed in the language field in all object editors available for the class.

- The **Enable Data Translation** attribute must be set to `True` in order to enable the object class for either manual data translation or automatic data translation.
- If the **Enable Data Translation** attribute is set to `False`, then the automated and manual data translation functionalities will be disabled and the language code will not be displayed in the language field in all object editors available for the class.
- For more information about the configuration of object data translation, see the section Configuring the Translation of Object Data in the chapter Localization and Multi-Language Support for the Alfabet Interface.

- To enable an object class for data translation, navigate to the relevant object class and set the **Enable Data Translation** attribute to `True`.
- To disable an object class for data translation, navigate to the relevant object class and set the **Enable Data Translation** attribute to `False`.
- **Enable Automated Data Translation**: Set to `True` if the object class may be translated automatically by the translation service specified in the XML object `AlfaTranslationServicesConfig`. For more information about the configuration of object data translation, see the section Configuring the Translation of Object Data in the chapter Localization and Multi-Language Support for the Alfabet Interface.
- **Automated Translation Rules for Class Properties**: If the **Enable Automated Data Translation** attribute is set to `True`: Click the **Browse** button to open the editor to specify which protected and custom properties may be translated automatically. For each property listed in the **Automated Translation Rules for Class Properties** editor, select True in the **Automated Translation Enabled** column to permit the property to be automatically translated.

  Please note that if the **Enable Data Translation** attribute is set to `None` or `Manual` for a custom property, the value in the **Automated Translation Enabled** column will be set to `False` per default and cannot be automatically translated.

- **Enable Statutory Language**: Set to `True` if the statutory language capability shall be available for the object class. The statutory language capability is available in order to address regulatory requirements that mandate that objects are captured in a specific language. When a user creates an object based on a relevant object class/object class stereotype, a statutory language selector will open in which the language can be selected. In the object editor that opens, the user can capture all attributes that support automated translation in the statutory language. If the XML attribute `EnableStatutoryLanguage` is not set for an object class stereotype in the Stereotypes attribute, the object class stereotype will inherit the definition of the **Enable Statutory Language** attribute specified for the object class. For more information about specifying a statutory language, see the section Specifying a Statutory Language for the Enterprise in the chapter Localization and Multi-Language Support for the Alfabet Interface.
- Attributes in the section **AlfaBot Support**: The attribute in this section are only editable for custom object class. They define if and how the AlfaBot can be used with respect to this object class. Each object class in the meta-model has an **AlfaBot Support** attribute section with four attributes. For standard private and protected object classes, the attributes are visible but not editable. The attributes can only be set for public, customer-defined object classes. The attributes **Creation Mode**, **Editing Mode**, and **Navigation Mode** specify whether and how creation of object, editing of objects or navigation to objects is supported for the object class. The attribute **Analysis Intent Mode** specifies

whether the object class is included into the search in configured reports that can be performed via the `Analyze` intent of the AlfaBot. The attributes can be set to one of the following:

- `None`: The functionality is not supported for the AlfaBot.
- `ContextDependent`: The functionality is supported by the AlfaBot and will take a required context into account. For example information flows can only meaningfully be created in the context of a source or target object. If the user wants to create an information flow via the AlfaBot, the relevant view for creation of the information flow will open, like for example the **Information Flows** page view of an application.
- `Full`: The functionality is supported by the AlfaBot context free. For the creation of objects with the **Creation Mode** attribute set to `Full`, the relevant editor or wizard will open.

> For an overview of the AlfaBot capability, see the section *Using the AlfaBot Capability* in the reference manual *Getting Started with Alfabet*. For information about how to implement the AlfaBot capability, see the section Implementing the AlfaBot for Navigation via a Full-Text Command.

- **Alias**: If the **Analysis Intent Mode** attribute of th object class is set to `Full`, a list of synonyms for the object class can be defined in this attribute. If a user searches for one of the synonyms in the `Analyze` intent of the AlfaBot, the AlfaBot will provide search results for the object class that the synonym is defined for.

3) In the toolbar, click the **Save** button to save the new custom object class.

Configuring Object Class Stereotypes for Object Classes

An object class stereotype is a sub-classification of an object class. A permissible object class can have multiple object class stereotypes, each of which captures a specified set of attributes, reference data, and class configurations. For example, the object class Application that may have the object class stereotypes Business Applications and Technical Applications.

Only a limited number of object classes support the configuration of object class stereotypes.

> For an overview of the object classes that support the object class stereotype concept, see the chapter *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

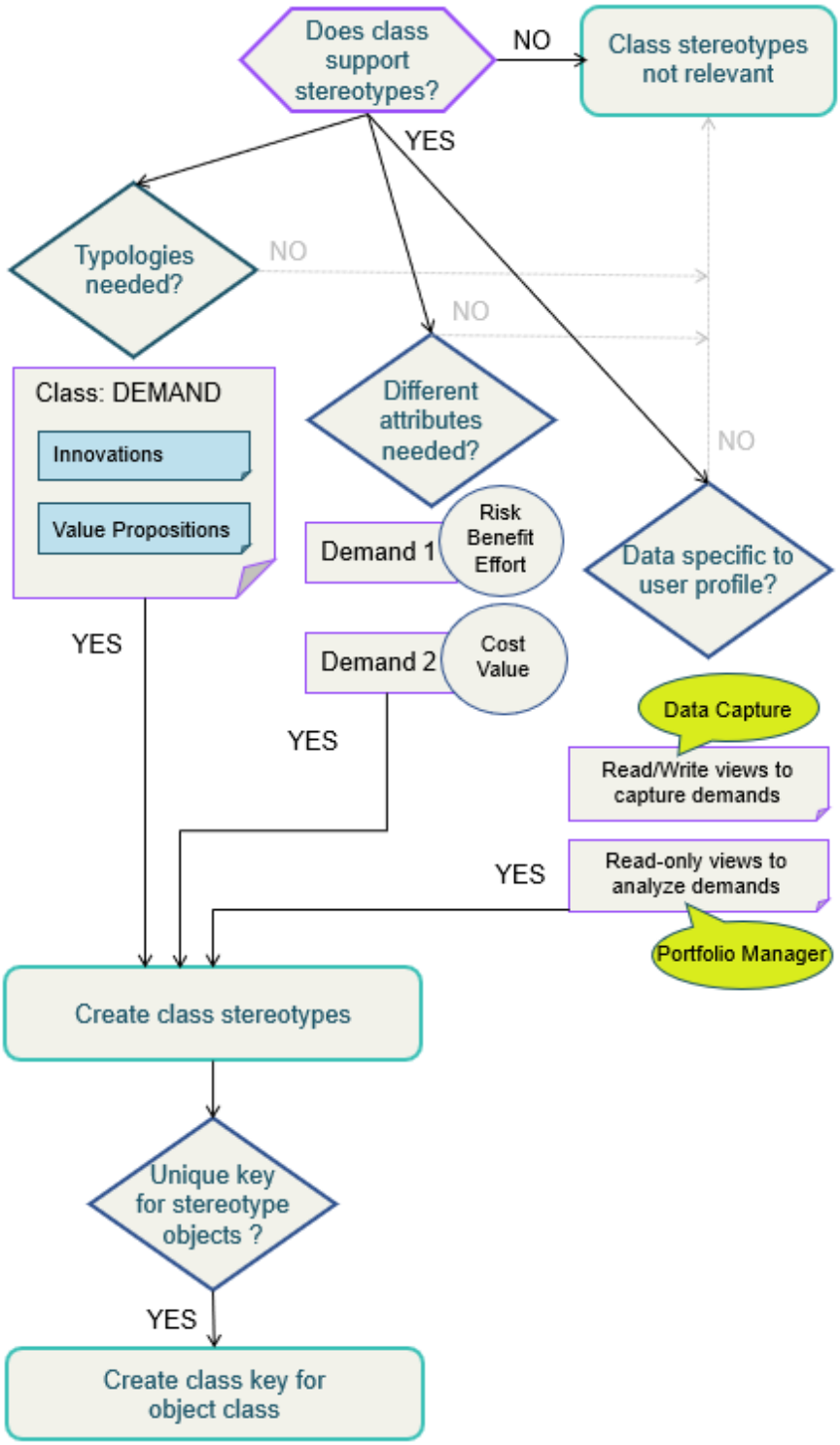


FIGURE: Determining whether to configure class stereotypes

The following can be configured for each object class stereotype created for an object class:

Configuration Option	Explanation
Statuses and Release Statuses	Release statuses can be configured for object class stereotypes in the XML object ***ReleaseStatusDef***. For more information, see the section Configuring Release Status Definitions for Object Classes.
Object States	Object states cannot be configured for individual object class stereotypes in the XML object ***ObjectStateManager***. If the concept of object states is available for the object class that the object class stereotype is based on, then it will also be available to the object class stereotype. For more information, see the section Configuring Object State Definitions for Object Classes.
Lifecycles	Lifecycle definitions cannot be configured for individual object class stereotypes in the XML object ***ObjectLifeCycleManager***. The object class stereotype will inherit the lifecycle concept of the object class that it is based on. For more information, see the section Configuring Lifecycle Definitions for Object Classes.
Searchability	If object class stereotypes have been defined for an object class, the solution designer can specify searchability on the level of the object class and/or object class stereotype. The searchability of object classes/object class stereotypes is defined by means of the **Searchable** attribute for class settings: • If the entire class is to be searchable the **Searchable** attribute for the relevant class settings defined for the object class must be set to `True`. The name of the object class will be displayed in the **Search for** filter in the search functionality. This setting will override the specification of the **Searchable** attribute for the relevant class settings defined for the object class stereotype(s). • If one or more specific object class stereotypes are to be searchable, the **Searchable** attribute for the relevant class settings defined for the object class stereotype(s) must be set to `True`. For more information, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.
Mandates	Defined via the XML attribute `HasMandates` in the XML definition of the **Stereotypes** attribute for the relevant object class. If the XML attribute `HasMandates` is set to "`true`", the objects based on the object class stereotype can be defined to have mandates in a federated architecture. For detailed information about the implementation and configuration of a federated architecture and mandates, see the section Implementing the Mandate Capability for a Federated Architecture in the chapter Configuring Access Permissions for Alfabet.
Custom online Help	A custom help can be specified for the object class stereotype via the object view configured for the object class stereotype. For information about the

Configuration Option	Explanation
	syntax of the path, see the chapter Providing Custom Online Help to the User Community.
Custom properties	Custom properties are defined for the base object class that the object class stereotypes are configured for and then assigned to the relevant custom editor and custom object view that they are relevant for. Configure as for a conventional object class. For more information, see the section Configuring Custom Properties for Protected or Public Object Classes.
Custom editors	One or more custom editors must be created to capture data for an object class stereotype. Configure as for a conventional object class. For more information, see the chapter Configuring Custom Editors.
Wizard	Multiple wizards may be created to capture data for an object class stereotype. Configure as for a conventional object class. For more information, see the chapter Configuring Wizards.
Custom object view	One or more custom object views must be created for an object class stereotype to make the information available in the Alfabet user interface. Configure as for a conventional object class. For more information, see the chapter Configuring Object Views.
Class settings	One or more class settings must be created per object class stereotype. Class settings for stereotypes are configured as for a conventional object class. For more information, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.
Workflows	Multiple workflows may be created per object class stereotype. Configure as for a conventional object class. For more information, see the chapter Configuring Workflows.
Reference data including, for example, cost types, evaluation types, portfolios, and role types.	Configure as for a conventional object class in the **Configuration** module. The object class stereotypes will be displayed in the explorer of the **Class Configuration** functionality. For more information, see the reference manual *Configuring Evaluation and Reference Data in Alfabet*. **NOTE**: If reference data, evaluation types, etc. are NOT assigned to an object class stereotype in the **Class Configuration** functionality, the reference data, evaluation types, etc. assigned to the base object class of the object class stereotype will be implemented. For example, if no evaluation types are assigned to `Application:<ApplicationStereotype>`, then the evaluations types assigned to the class `Application` will be available for objects based on `<ApplicationStereotype>`

Configuration Option	Explanation
	For some types of configurations, the specifications for the object class stereotype will be ignored. In this case, the configuration specified for the base object class of the object class stereotype will apply. This is the case for: • Prioritization Schemes • Portfolios (but not configured report portfolio views) • Diagram View Items • Aspect Evaluation Types • Aspect Prioritization Schemes • Aspect Portfolios

If object class stereotypes are defined for an object class, then a user creating a new object will be required to first define the object class stereotype that the new object is to be based on in the **Stereotype Selector** that will automatically open when the **Create New <ObjectClass>** action is selected. The **Stereotype Selector** lists the possible object class stereotypes for the selected object class. The **Stereotypes** attribute is thus specified for the object via the selection in the **Stereotype Selector**. Once the selector is closed, the relevant editor will open and the user can continue to define the new object. The defined object can then be viewed in the relevant custom object view configured for the object class stereotype. The **Stereotypes** attribute must be specified whenever an object is created that is based on an object class for which object class stereotypes are configured.

A **Change Stereotype** button is available in the toolbar of object profiles for which object class stereotypes have been configured thus allowing the user to easily change the stereotype that an object is based on. The menu of the **Change Stereotype** button will display all stereotypes configured for the object class. If the **Change Stereotype** button should not be used by users with a specific user profile, the button can be hidden for that user profile. For more information, see the section Hiding Functionalities in an Object View in the chapter Configuring User Profiles for the User Community.

Please note that the **Change Stereotype** button will only be available for object classes that are not relevant in the configuration of an XML object available in the **SolutionManagers** folder in Alfabet Expand. The following classes do not support the **Change Stereotype** button: **Application**, **Component**, **Demand**, **Demand Group**, **Domain**, **Feature**, **ICT Object**, **Organization**, **Policy**, **Policy Group**, **Project**, **Resource**, **Service Product**, **Service Product Item**, **Standard Platform**, **Technical Service**, **Technical Service Operation**, **Technical Service Operation Method**, and **Value Node**, and **Vendor Product**.

If object class stereotypes have been configured for an object class, the **Stereotype** attribute must be specified for every object of that object class. This is particularly important to take into consideration if objects are to be created by means of the **Extended Data Capture** functionality. Once the object class stereotype has been defined for a new object, it can be changed via the **Change Stereotype** button in the toolbar of object profile (unless it is based one of the object classes listed above that do not support the **Change Stereotype** button. For more information about the **Extended Data Capture** functionality, see the chapter *Capturing Data with Data Capture Templates* in the reference manual *User and Solution Administration*.

The following information is available:

- Creating Object Class Stereotypes for an Object Class
- Configuring Uniqueness Constraints for an Object Class Stereotype
- Providing Custom Online Help for an Object Class Stereotype

Creating Object Class Stereotypes for an Object Class

Object class stereotypes must be created for the relevant object class via the **Stereotypes** attribute. By means of the **Stereotypes** attribute, the solution designer creates the object class stereotypes by defining a stereotype name, the singular and plural form of the object class stereotype's caption, and whether the object class stereotype may have mandates assigned. Once the object class stereotype has been created, class settings can be configured to specify whether the object class stereotype is searchable in the search functionalities and which icon and preview properties to display for the object class stereotype. For more information, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.

The configuration of object class stereotypes should not be changed once the object class stereotype is implemented. It is important that all relevant object class stereotypes that you plan to implement in your enterprise are well-considered at the stage of configuration.

The following information describes the general procedure to configure object class stereotypes for all object classes that allow stereotype configuration. The specification object class stereotypes is made in an XML definition in the **Stereotypes** attribute for the relevant object class. For an overview of the object classes that support the object class stereotype concept, see the chapter *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

Example of the XML definition in the **Stereotypes** attribute for the object class `Domain`:

```
<ClassStereotypes>
   <Stereotype Name="Area" Caption="Area" CaptionPlural="Areas"
   Comments="" HasMandates="false" IDPrefix="AR" />
   <Stereotype Name="SubArea" Caption="Sub-Area" CaptionPlural="Sub-
   Areas" Comments="" HasMandates="false" IDPrefix="SAR"/>
   <Stereotype Name="BusinessDomain" Caption="Business Domain"
   CaptionPlural="Business Domains" Comments="" HasMandates="false"
   IDPrefix="BDOM"/>
   <Stereotype Name="TechnologyDomain" Caption="Technology Domain"
   CaptionPlural="Technology Domains" Comments=""
   HasMandates="false" IDPrefix="TDOM"/>
</ClassStereotypes>
```

For some object classes, a hierarchy is typically configured for the object class stereotypes of these object classes and thus additional configuration is required. This is described in detail in the following sections of the chapter Configuring Alfabet Functionalities Implemented in the Solution Environment.

- For the object class `Domain`, see Configuring Domain Models and Domain Planning.

Please note that the mandate capability may be differently configured for the individual object class stereotypes defined for an object class, with the exception of the object class `Domain`. The object class `Domain` requires that the configuration of the `HasMandates` attribute is the same for all domain stereotypes. For more information about the configuration of the mandate capability for the object class `Domain`, see the section Implementing Mandates for the Domain Model in the chapter Configuring Domain Models and Domain Planning.

- For the object class `Project`, see Configuring the Project Management Capability.
- For the object class `ValueNode`; see Configuring the Strategy Deduction Capability.
- For the object class `OrgaUnit`, see Configuring the Organizational Hierarchy.
- For the object class `ICTObject`, see Configuring the ICT Object Hierarchy.
- For the object class `Demand`, see Configuring the Demand Hierarchy.
- For the object class `Feature`, see Configuring the Feature Capability.
- For the object class `ITPolicy`, see Configuring the Policy Hierachy.
- If object class stereotypes are configured for an object class that can be defined in the *To-Be Architecture Page View* (such as `Application`, `Component`, `Device`, or `StandardPlatform`) then you must create an identical configuration for the corresponding solution object class (for example, for `SolutionApplication` or `SolutionComponent`, `SolutionDevice`, or `SolutionStandardPlatform`). For example, the configuration defined in the **Stereotypes** attribute for the object class `Application` should be copied and pasted to the **Stereotypes** attribute for the object class `SolutionApplication`. If the configuration is not the same, errors may occur when solution objects are checked in to the Alfabet inventory in the context of project and solution planning. For more information about configuration requirements for solution Alfabet, see the section Making the As-Is Architecture and To-Be Architecture Capabilities Available.

To configure object class stereotypes for a relevant object class:

1) Expand the **Classes** node and click the relevant protected object class for which it is permissible to configure object class stereotypes.
2) In the attribute window, define the **Stereotypes** attribute in the text editor available for the XML definition. For each object class stereotype, define the following XML attributes:

An error message will be displayed upon closing the text editor if syntax errors are written in the XML definition of the **Stereotypes** attribute.

- `Stereotype Name`: Enter the technical name for each object class stereotype. You may define a name of up to 64 characters. This name should not be changed once the object class stereotypes are implemented.

Changing the technical name after objects have been created based on the object class stereotype requires such objects to be updated by means of an ADIF scheme.

For more information, see the reference manual *Alfabet Data Integration Framework*.

- `Caption`: Enter the caption for the object class stereotype. This is the name that users will see in the Alfabet user interface.

  Please note that the following:

  - The caption of the object class stereotype should be different than the caption of its object class. This is particularly important for the implementation of the search functionality for the object class stereotype. For more information, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.
  - The values defined for the XML attribute `Caption` can be translated whereas the values defined for the XML attribute `Stereotype Name` cannot be translated. However, in the table of the **Simple Search** functionality as well as other search selectors, the values defined for the XML attribute `Stereotype Name` of the object class stereotype are displayed in the **Stereotypes** column. If the values defined for the XML attribute `Caption` should be displayed, a custom selector should be defined. For more information, see the section Configuring a Custom Selector for Search Functionalities.
  - If you want to enter a caption (`Caption`) or plural form for the caption (`CaptionPlural`) that contains special characters (for example, a greater then (>) or lesser then (<) symbol), you must replace the special characters with respective XML compliant code, for example:
    - > for >
    - < for <
    - " for "
    - [for [
    -] for]
  - The caption will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.

- `Description`: Provide a brief description about the object class stereotype. The description will be displayed in the **Stereotype Selector** that opens when the user must select which stereotype a new object is to be based on. The description should help the user to understand which stereotype the new object should be based on. The description will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.
- `CaptionPlural`: Enter a plural form for the caption. This is the name in plural that users will see in the Alfabet user interface.
- `IDPrefix`: Enter no more than 5 characters for the ID prefix used to generate the unique identification number for new objects based on the object stereotype. If the XML attribute `IDPrefix` is not specified, the ID prefix of the object class that the object class stereotype is based on will be used to generate the unique ID. Please note that the integer generated for the unique ID is based on the number of instances

created for the object class that the object class stereotype is based on, and includes all instances of all object class stereotypes created for the object class.

The prefix ID of existing objects based on an object class stereotype can be updated via the ADIF import scheme `SetStereotypeID` available in the `Alfabet Standard Jobs` folder. For more information, see the section *Predefined ADIF Schemes* in the reference manual *Configuring Alfabet with Alfabet Expand*.

- `Comments`: Enter text in order to provide information about the object class stereotype. This information is only for the purposes of configuration. It is not displayed in the Alfabet user interface.
- `HasMandates`: Enter "`true`" if the mandate capability should be available for the object class stereotype. Enter "`false`" if the mandate capability should not be available for the object class stereotype. If the XML attribute `HasMandates` is defined as "`false`" for an object class stereotype, objects based on the object class stereotype will be visible to users with any mandate implemented in your enterprise. For information about additional configuration requirements necessary to implement the mandate capability, see the section *Implementing the Mandate Capability for a Federated Architecture* in the chapter Configuring Access Permissions for Alfabet .
- `EnableStatutoryLanguage`: Set to `True` if the statutory language capability shall be available for the object class stereotype. The statutory language capability is available in order to address regulatory requirements that mandate that objects are captured in a specific language. When a user creates an object based on a relevant object class/object class stereotype, a statutory language selector will open in which the language can be selected. In the object editor that opens, the user can capture all attributes that support automated translation in the statutory language. If the XML attribute `EnableStatutoryLanguage` is not set, the object class stereotype will inherit the definition of the **Enable Statutory Language** attribute specified for the object class. Specifying a Statutory Language for the Enterprise

3) In the toolbar, click the **Save** button to save your changes.

Configuring Uniqueness Constraints for an Object Class Stereotype

Uniqueness constraints cannot be explicitly defined for a specific object class stereotype. Uniqueness constraints are configured in the class key on the level of the base object class. The class key configuration is thus inherited by all object class stereotypes configured for the object class. For example, a class key defined for the object class `Application` will inherently be applied to all object class stereotypes configured for the object class `Application`. For more information about configuring class keys, see the section Configuring Class Keys for Object Classes.

Providing Custom Online Help for an Object Class Stereotype

You can provide a custom online Help for the object class stereotypes that you configure in order to help your user community understand the content of the object class stereotype. The path to the custom help must be defined in the **Custom Context-Sensitive Help URL** attribute for the object view configured for the object class stereotype. Please note that the hyperlinked caption displayed in the Help window will display the name of the object class that the object class stereotype is defined for and not the caption of the object class stereotype. For example, the help link will display **Help on Applications** (and not, for example, **Help on Technical Applications**).

In the **Custom Context-Sensitive Help URL** attribute for the object view configured for the object class stereotype, enter the URL or server variable that targets the custom context-sensitive help.

For more information about configuring an object view for an object class stereotype, see the chapter Configuring Object Views. For detailed information about the various options for configuring custom help, see the chapter Providing Custom Online Help to the User Community.

Configuring Custom Properties for Protected or Public Object Classes

You can create multiple custom object class properties for an object class. For reasons of performance, it is recommended that you do not create more than 30 custom object class properties per object class. All standard and custom object class properties are listed in the **Class Model** explorer below the relevant object class node. The properties are listed in the order of their creation. To organize the object class properties in alphabetical order, right-click the **Classes** node in the **Meta-Model** tab and select **Sort Properties by Name**. All properties will be sorted for all classes for the current user session.

The number of object class properties that may be created for an object class is limited to 255. If a user attempts to create more than 255 object class properties, a warning will be displayed and the property exceeding the limit will be discarded. If more than 255 properties exist and a meta-model update is executed, a warning will be added to the log file that the class properties for the specific class exceeds the limit. The limit of 255 applies only to the properties displayed in Alfabet Expand and does not include database properties such as RefStr, INSTGUID, etc.

Any changes made to the meta-model will impact both the relevant class table and audit table in the Alfabet database. For example, if a custom object class property is deleted from the class model, it will also be deleted from the audit table and thus will no longer be displayed in the *Object Audit Page View* available in your Alfabet solution.

It is highly recommended that you do not change the meta-model once the auditing capability is implemented, otherwise errors may occur in the audit information.

The following information is available:

- What to Consider When Creating Custom Properties
- Implementing the Generic Attribute Concept Instead of Custom Properties
- Overview of Data Types Available for Custom Properties
- Creating a New Custom Property
- Configuring Custom Properties of the Type String and StringArray
- Configuring Custom Properties of the Type Boolean
- Configuring Custom Properties of the Type Date and DateTime
- Configuring Custom Properties of the Type Text
- Configuring Custom Properties of the Type Real
- Configuring Custom Properties of the Type Integer

- Configuring Custom Properties of the Type URL
- Configuring Custom Properties of the Type Reference
- Configuring Custom Properties of the Type ReferenceArray
- Configuring Custom Properties of the Type Email
- Changing an Existing Custom Property

What to Consider When Creating Custom Properties

You can define custom object class properties  for any protected object class  that you see displayed in the **Classes** node in the **Meta-Model** tab of Alfabet Expand.

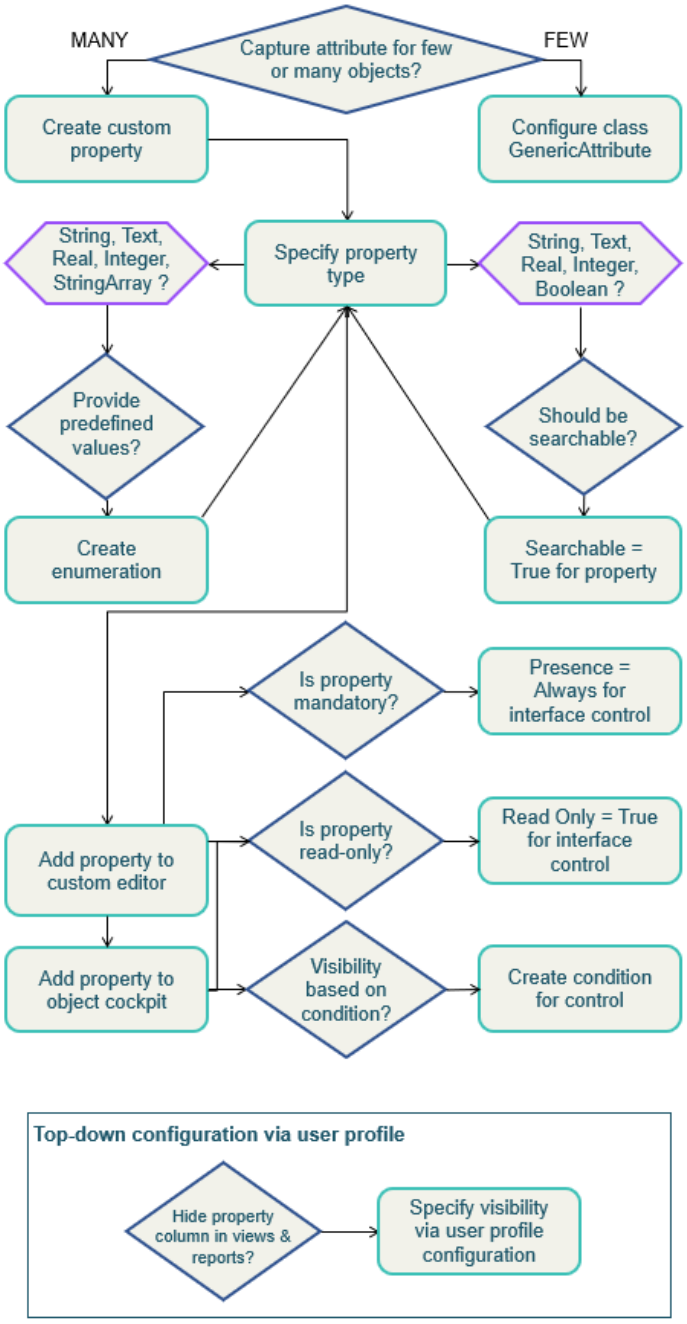


FIGURE: Steps to configure object class properties

The following issues should be taken into consideration when you configure custom object class properties for your Alfabet solution:

- What kind of data must the user enter for the custom object class property? This will determine the property type that you define for the custom object class property. For example, will the user enter a `True/False` value (`Boolean`), a number (`Real` or `Integer`), or text (`String` or `Description`)? For an overview of the available property types, see the section Overview of Data Types Available for Custom Properties.
- Will the data generally be captured for most objects or only in rare cases. A generic attribute can be used if one or more attributes are required for informational purposes only and each attribute is only used for a small subset of objects rather than for all objects in an object class. A generic attribute is

an alternative to configuring a custom property for an object class. For more information about working with generic attributes, see the section Implementing the Generic Attribute Concept Instead of Custom Properties.

- Can the user enter a value of his/her choice or must he/she select a value from a drop-down list displaying predefined values? If a drop-down list with pre-defined values is needed, you will first need to create an enumeration, which is the set of values that are to be displayed in the drop-down list. An enumeration can only be assigned to custom object class properties of the type `String`, `Text`, `Real`, `Integer`, or `StringArray`. For more information about configuring enumerations for a custom object class property, see the section Defining Protected and Custom Enumerations

- How many custom object class properties will be required for an object class? How many custom tabs will be required for the custom object class properties in the custom editor that you will later create? For more information about the types of interface controls available for custom editors, see the section Overview of the Interface Controls Available for Custom Editors in the chapter Configuring Custom Editors.

- Are rules (class keys) required so that users must enter a unique set of values for a combination of standard and/or custom object class properties? If this is the case, you must ensure that the custom object class properties that are to be included in the class key definition are of the object class property types `Date`, `Integer`, `Real`, or `String`. For more information about configuring class key for a custom object class property, see the section Configuring Class Keys for Object Classes.

- Do the captions for custom object class properties need to be translated for other interface languages used in your Alfabet solution? For general information about translating captions, see the section Modifying, Translating and Managing the Vocabularies in the chapter Localization and Multi-Language Support for the Alfabet Interface.

- Should the custom object class property be displayed in relevant page views or configured reports. If this is the case, you should specify it to be visible as a column in the relevant page view/configured report. For more information, see the section Refining Visibility Issues in the View Scheme in the chapter Configuring User Profiles for the User Community.

Keep the following in mind when defining custom object class properties:

- When you define a custom object class property, you can choose from a limited number of property types. For an overview of all data types and the possible visualization of these data types in a custom editor, see the section Overview of Data Types Available for Custom Properties.

- A custom editor can display multiple custom object class properties distributed across multiple tabs. For the sake of usability and performance, however, it is recommended that you do not create more than a total of 30 custom object class properties per object class nor design more than three tabs per custom editor.

- Each property type is automatically associated per default with a specific interface control. Once the custom object class properties are defined, you can automatically generate the custom editor. The custom object class properties are assigned to the default interface controls and automatically placed in the custom editor. If more than ten custom object class properties are defined for the object class, additional tabbed pages will be generated. You can later modify the visualization of the custom editor, add additional tabbed pages, and opt for a different interface control for a specific custom object class property.

- The searchability of private properties is predefined by Software AG, but the searchability of protected properties as well as public properties may be specified by the solution designer via the **Searchable** attribute available for the protected/public property. Therefore, private properties that are preconfigured by Alfabet as well as all public and protected properties of the type `String`,

`Integer`, `Real`, `Date`, and `Boolean` for which the **Searchable** attribute has been set to `True` will be available in the **Search Attributes** field. Keep the following in mind:

- Any protected or custom object class properties that you configured to be searchable can be defined by users as search criteria in the **Search Attributes** field in the **Simple Search** tab of the object selector or **Search** functionality when searching for objects in the relevant class.
- Any protected or custom object class properties that you configured to be searchable and that are either 1) of the type `String` and have an enumeration assigned or 2) of the type `Boolean` can be selected in multiple selection combo box filters (such as the **Object Filter** fields typically available in diagrams. In this way, users can filter the results shown in a diagram based on the object class properties. The multiple selection combo box displays searchable standard and custom object class properties using the following nomenclature: `<ClassName>.<CustomPropertyName> = <CustomPropertyValue>`.For more information about the use of object filters in diagrams and other views, see the section *Defining Multiple Selection Combo Box Fields* in the reference manual *Getting Started with Alfabet*

Implementing the Generic Attribute Concept Instead of Custom Properties

The object class **Generic Attribute** (`GenericAttribute`) has the following object class properties.

- `Name`: The name of the generic attribute. This attribute corresponds to the caption of a custom object class property.
- `Type`: The data type defined for the generic attribute. Generic attributes can have the type `String`, `Boolean`, `Integer`, `Date`, `Real`, `Color` or `Icon`. Alternatively, an existing enumeration can be specified as the type and an enumeration item can be selected as the value for the generic attribute. Please note that the attribute **Enabled for Generic Attribute** must be set to `True` for the enumeration if it is to be available in the context of a generic attribute. Only enumerations set to **True** will be displayed in the relevant **Type** field in the **Generic Attribute** editor.
- `Owner`: The object that the generic attribute is defined for. Generic attributes can be created for objects of the object classes `Application`, `Component`, `Deployment`, `Deployment Element`, `Standard Platform`, `Standard Platform Element`, `Stack`, `Stack Element`, and `Stack Item` (**Stack Configuration Item**).
- `Value`: The value for the generic attribute. The possible value that can be defined will depend on the data type specified for the **Type** attribute.
- `Group`: Generic attributes can be structured in generic attribute groups. Generic attributes with identical string values for the property `Group` belong to the same group.

Please note the following regarding the implementation of generic attributes:

- Generic attributes can be configured for an object class or object class stereotype in the *Generic Attributes Page View* (`SET_GenericAttributes`) available in the **Class Configuration** functionality. Generic attributes can be created here with a predefined value and reused in the *Generic Attributes Page View* (`ObjectGenericAttributes`) available for objects of the relevant object classes/object class stereotypes. The value can be changed for an object as needed.
- The *Generic Attributes Page View* (`ObjectGenericAttributes`) is available per default in the **Structure** workspace of the standard object views for the object classes `Application`, `Component`, `Deployment`, `Deployment Element`, `Standard Platform`, `Standard Platform Element`, `Stack`, `Stack Element`, and `Stack Item` (**Stack Configuration Item**). The *Generic*

Attributes Page View (`ObjectGenericAttributes`) allows new generic attributes to be created or existing generic attributes created for an object of the same object class or in the context of the **Class Configuration** functionality to be created for a selected object. The value of the generic attribute can be changed, as needed.

- To display generic attributes in object profiles or object cockpits, see the sections Adding a Generic Attribute to the Attributes Section and Adding a Generic Attribute to the Object Cockpit in the chapter Configuring Object Views.
- Properties that are stored as objects of the class `GenericAttribute` can be imported or exported via the Alfabet RESTful API v2. For more information, see the reference manual *Alfabet RESTful API*

Overview of Data Types Available for Custom Properties

When you define a custom object class property, you can choose from a limited number of property types. The attributes that are available for an object class property type will vary. The table also indicates whether it is possible to specify a default value for the custom object class property, the searchability of the custom object class property, or an enumeration providing preconfigured values for user to define for the custom object class property.

Please note that a `NULL` value will be stored in the Alfabet database for a data type if no value is defined for the custom object class property. In the case of custom object class properties of the type `Boolean`, it is highly recommended that a default value is configured to ensure that the value `False` (rather than `NULL`) is stored for a custom object class property if the associated checkbox that is not selected.

The following data types can be selected in the **Property Type** attribute for a custom object class property.

Custom Property Data Types	Description	Default Value	Is Searchable	Enumeration	Interface Control
`String`	An object class property of the type `String` may have between 1-1999 characters of text.	yes	yes	yes	`Edit`, `ComboBox`, `RadioGroupButton`
`Boolean`	An object class property of the type `Boolean` has the permissible values `True` or `False`.	yes	yes	no	`CheckBox`
`Date`	Users must specify a date.	no	yes	no	`Edit` (The **Calendar** button will be automatically added to the `Edit` field.)

Custom Property Data Types	Description	Default Value	Is Searchable	Enumeration	Interface Control
`DateTime`	Users must specify a date and time.	no	yes	no	`Edit` (Users must explicitly enter a date and time in the editor field.)
`Integer`	An object class property of the type Integer requires a positive or negative integer number as value.	yes	yes	yes	`Edit`
`Real`	An object class property of the type `Real` requires a positive or negative decimal number as value.	yes	yes	yes	`Edit`
`Reference`	An object class property of the type Reference is used to define relationship information between two objects that have a 1:n relationship. **NOTE**: A custom object class property of the type `Reference` should be defined with the help of Software AG Support	no	no	no	`EditSearch`, `ComboBox`, `ListBox`, `RadioButtonGroup`
`ReferenceArray`	An object class property of the type `ReferenceArray` is used to define relationship information between two objects that have an n:n relationship. **NOTE**: A custom object class property of the type `ReferenceArray` should be defined with the help of Software AG Support	no	no	no	`CheckedListBox`, `EditSearch`
`StringArray`	For properties of the type `StringArray`, multiple values of the enumeration can be selected.	no	no	yes	`CheckedListBox`
`Text`	An object class property of the type `Text` allows a large amount of text to be captured. An HTML editor may be implemented in custom editors, object profiles,	no	no	no	`Memo`, `CheckedListBox`, `ListBox`, `ComboBox`

Custom Property Data Types	Description	Default Value	Is Searchable	Enumeration	Interface Control
	and object cockpits for properties of type `Text`.				
`URL`	An object class property of the type `URL` allows users to define a URL. The URL can be displayed as a link in the **Attributes** section of the object's profile or an object cockpit. Clicking the link in the object profile/object cockpit or preview will open the URL in a separate browser window.	no	no	no	`Edit`

Creating a New Custom Property

The procedure described below is a general approach to configuring custom object class properties and therefore not all attributes available for all property types are described below. The attributes available in the attribute window will depend on the object class property type that you select for the custom object class property.

If you plan to use native SQL instead of the Alfabet query language for queries, you must ensure that a technical name exists for each custom object class property.

To create a new custom object class property for a selected object class:

1) Expand the **Classes** node and right-click the relevant object class  and select **Add New Property**. The **Create New Property** editor opens. Define the following fields:

- **Property Name**: Enter a unique name for the custom object class property. This value is displayed in the solution interface if the **Caption** attribute is not defined.
- **Caption**: Specify a caption to be displayed for the property in the Alfabet user interface.
- **Tech Name**: Enter a unique technical name for the custom object class property that will be used in the Alfabet database table. Click **OK** to save the definition.

Please note that the `Tech Name` is the name of the column that will be created in the database table of the object class for storing values for the custom property. The following rules apply:

- The technical name may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.

- The technical name should only contain standard `ASCII` characters.
- The technical name should consist of upper-case letters only.
- The maximum length of a technical name may not exceed 30 characters for properties that are not translatable and may not exceed 25 characters for properties that have the **Enable Data Translation** attribute set to `True`.
- The technical name may not coincide with any of the reserved key words of the relational database management system.

A validation mechanism checks for correct syntax when defining a technical name.

2) In the toolbar, click the **Save**  button to save the new custom object class property.

For detailed information about the configuration of the various types of custom object class property:

- Configuring Custom Properties of the Type String and StringArray
- Configuring Custom Properties of the Type Boolean
- Configuring Custom Properties of the Type Date and DateTime
- Configuring Custom Properties of the Type Text
- Configuring Custom Properties of the Type Real
- Configuring Custom Properties of the Type Integer
- Configuring Custom Properties of the Type URL
- Configuring Custom Properties of the Type Reference
- Configuring Custom Properties of the Type ReferenceArray
- Configuring Custom Properties of the Type Email

Configuring Custom Properties of the Type String and StringArray

A property of the type `String` allows a value to be manually entered in a custom editor by the end user. It is also possible to provide a set of preconfigured values for the user to select from. In this case, the custom object class property must be associated with an enumeration. An enumeration is made up of enumeration items that can be selected as the value for the custom object class property.

An enumeration can be associated with an object class property of the type `String`, whereby a single enumeration item to be selected as a value. An enumeration can also be associated with an object class property of the type `StringArray`, whereby multiple enumeration items can be selected.

Please note that an enumeration cannot be translated if it is assigned to an object class property of the type `StringArray`. If you need to configure an object class property in which multiple selection of values is possible, then the enumeration must be assigned to an object class property of the type `Text`. For more information, see the section Configuring Custom Properties of the Type Text. The translation capability is described in detail in the section Modifying, Translating

and Managing the Vocabularies in the chapter Localization and Multi-Language Support for the Alfabet Interface.

If the custom object class property is associated with an enumeration, the enumeration items can also be selected in a multiple selection combo box available in diagrams associated with the relevant object class.

If the custom object class property is to be associated with an enumeration, the enumeration must first be defined before it can be assigned to the custom object class property. For information about how to configure an enumeration for a `String` or `StringArray` property, see the section Defining Protected and Custom Enumerations.

To add a custom object class property of the type `String` to a custom editor, add an `Edit`, `ComboBox`, or `RadioGroupButton` interface control to the custom editor. To add a custom object class property of the type `StringArray` to a custom editor, add a `CheckedListBox` interface control to the custom editor. Please note that a `CheckedComboBox` should not be used. For more information, see the section Configuring the Interface Controls in the Custom Editor.

A color selector can be added to any custom editor created for an object class that has a `Color` property of the type `String`. When the user creates or edits an object in the custom editor, he/she can change the color in the custom editor. The color will then be used for the display of that object in business graphics for all Alfabet views and configured reports for all users. For more information, see the section Adding a Color Selector to the Custom Editor.

To define a custom object class property of the type `String` or `StringArray`:

1) In the attribute window, define the following attributes, as needed:

- **Property Type**: Select `String` or `StringArray`.
- **Name**: Enter a name of the object class property. The name must be unique in the scope of the selected object class.
- **Caption**: Specify a caption to be displayed for the property in the Alfabet user interface. If no caption is defined, then no caption will be displayed in the Alfabet user interface.
- **Enable for Data Capture Templates**: Select `True` if the object class property may be captured via data capture templates. Select `False` if the object class property may not be captured via data capture templates. The **Enable for Data Capture Templates** attribute must be set to `True` for the object class that the object class property is assigned to. For more information about data collection via data capture templates, see the section *Capturing Data with Data Capture Templates* in the reference manual *User and Solution Administration*.

  If the protected property is inherited from the parent object class `Artifact` or `ArtifactAuthorized`, then the **Enable for Data Capture Templates** attribute will be greyed out. In this case, you must specify a local setting for the property that will apply to the property only in the context of this object class. To do so, expand the **Local Settings** section and set the **Enable for Data Capture Templates** attribute to `True` to override the inherited setting from the parent object class. Please note that in order to prevent the ID of objects being erroneously changed via data capture templates, the `ID` property cannot be enabled as a class property.

  Properties of the type `BinData`, `Picture`, `DateArray`, `POSIX`, `ByteArray`, `DateArray`, `DateTimeArray`, `BindataArray`, `IntegerArray`, `TimeArray`, `PropertySet`, `StringArray`, and `URLArray` are not suitable for data capture templates.

- **Size** attribute: Specify the maximum number of characters that the string may consist of. An object class property of the type `String` may have between 1-1999 characters of text.

  Please note however, strings with more than 600 characters might exceed the maximum size restrictions of your enterprise's database server that hosts the Alfabet database.

- **Searchable**: Please consider the following:

- For custom object class properties of the type `String` only: Set to `True`. if the object class property is to be searchable in the search functionalities. The custom object class property will be displayed in the searchable properties field in the standard search functionalities as well as relevant standard or custom selectors implemented for search filters. If you select `False`, the custom object class property will not be searchable in the standard search functionalities nor in the standard or custom selectors implemented for search filters.

- . For custom properties of type `String` that have an enumeration specified for the **Enum** attribute. Set to `True` to include the property in the **Object Filter** field in diagrams. The multiple selection combo box displays searchable standard and custom object class properties using the following nomenclature: `<ClassName>.<CustomPropertyName> = <CustomPropertyValue>` For more information about the use of object filters in diagrams and other views, see the section *Defining Multiple Selection Combo Box Fields* in the reference manual *Getting Started with Alfabet*.

  You should carefully consider which custom object class properties should be searchable. Unwarranted searchable custom object class properties make drop-down lists excessively long. Additionally, some properties (such as telephone numbers) should not be searchable for reasons of data security. For more information about configuring the search functionalities in Alfabet, see the chapter Configuring Custom Selectors and Search Functionalities. For more information about the use of the search functionalities available, see the section *Searching for Your Objects* in the reference manual *Getting Started with Alfabet*.

  .

- **Enable Data Translation**: Optional for a `String` property only: If the **Enable Data Translation** attribute is set to `True` for the object class, you can further refine the permissibility to translate data for the custom property.

  Please consider the following:

  - The specification of the **Enable Data Translation** attribute applies to all cultures defined for your enterprise. Therefore, if you specify that a data translation is not permissible for an object class property, then object data cannot be specified for that object class for all defined cultures. Please note that regardless of the specification of the **Enable Data Translation** attribute for an object class/object class property, if the **Support Data Translation** attribute is set to `False` for a culture, data translation will not be possible at all for the culture and the language code will not be displayed in the language field in all object editors.

  - The specification of the **Enable Data Translation** attribute for an object class property will have precedence over the specification made in the **Automated Translation Rules for Class Properties** field for the object class. In other words, if the **Enable Data Translation** attribute for the

property is set to `None` or `Manual` and the property is set to `True` in the **Automated Translation Rules for Class Properties** field for the object class, the property will not be automatically translated.

- For more information about the translation capability, see the chapter Localization and Multi-Language Support for the Alfabet Interface.

- Set the **Enable Data Translation** attribute to `Manual` to allow data translation for the custom property to be translated manually by the user in the object editor.
- Set the **Enable Data Translation** attribute to `ManualAndAutomated` to allow data translation for the custom property to be translated manually by the user in the object editor or translated via the automated translation capability.
- Set the **Enable Data Translation** attribute to `None` to disable data translation for the custom property.
- **Translatable in Meta-Model Vocabulary**: Optional for a `String` property that has an enumeration defined: Set to `True` if the values defined for the associated enumeration should be available in the vocabularies for translation. For more information about the translation capability, see the section Modifying, Translating and Managing the Vocabularies in the chapter Localization and Multi-Language Support for the Alfabet Interface.
- **Enumeration**: Optional for a `String` property only: If the possible values available are to be provided based on a configured enumeration, select the custom enumeration. If the enumeration items configured for the custom enumeration should be available in the vocabularies for translation, set the **Translatable in Meta-Model Vocabulary** attribute to `True`. For information about how to configure an enumeration for a `String` or `StringArray` property, see the section Defining Protected and Custom Enumerations.

  Please note that an enumeration cannot be translated if it is assigned to an object class property of the type `StringArray`. If you need to configure an object class property in which multiple selection of values is possible, then the enumeration must be assigned to an object class property of the type `Text`. For more information, see the section Configuring Custom Properties of the Type Text. For more information about the translation capability, see the section Modifying, Translating and Managing the Vocabularies in the chapter Localization and Multi-Language Support for the Alfabet Interface.

- **Default Value**: Optional for a `String` property only: To specify a default value for the object class property, enter a default value for the object class property. The default value is automatically entered for the object class property when a new object is created but can be edited by the user as needed. The value entered must correspond to the object class property type defined. If an enumeration is specified for the **Enum** attribute, enter a value that has been configured for the enumeration.

  In some cases, you may want to display a predefined default value that cannot be edited in the custom editor. In this case, you could define the interface control to be non-editable (ReadOnly). It will be displayed in grey and thus disabled and therefore cannot be edited. For more information about configuring an editor field to be non-editable, see the section Specifying an Editor Field To Be Non-Editable.

- **Validator**: Optional for a `String` property only: In order to enforce values with a specific structure (for example, that version numbers should always be defined as two digits made up of a period and another digit), define a regular expression. In the **Comments** attribute, provide information to help users define the object class property according to the requirements defined in the **Validator** attribute. The comments will be shown in the error message displayed in the input value has an invalid format.

The validator is enforced on all editors where the object class property is editable. Users will be prompted with an error message if the value provided does not match the validator specified. The error message will state that the input value has an invalid format and will display the text defined in the **Comments** attribute for the custom object class property. Please note that specific enforcement mechanisms must be implemented for existing records or records that are imported into Alfabet from external systems. For more information about the syntax conventions for regular expressions, see http://msdn.microsoft.com/en-us/library/hs600312(VS.71).aspx.

- **Hint**: Enter informational text about the custom object class property to be displayed in the Alfabet interface. Users can view the tooltip for the custom object class property in the **Attributes** section of the respective object view. Please note that the **Hint** attribute defined for the custom object class property will not be displayed in a custom editor. The **Hint** attribute must be explicitly configured for the interface control associated with the custom object class property in the custom editor. For more information about the specification of custom editors, see the section Adding Help Text to Interface Controls in the Custom Editor in the chapter Configuring Custom Editors.

  Please note the following:

  - The text entered in the **Hint** attribute may not be longer than 600 characters. Texts exceeding 600 characters cannot be saved to the Alfabet database. Please note that if the hint is to be translated, the translated text may also not exceed 600 characters.
  - Hints are displayed in the tooltip functionality available via Microsoft Internet Explorer. Therefore, a tooltip will only remain open for a few seconds. As a consequence, the content of the tooltip should not be excessively long. If extensive information is required to guide users in defining the object class property, you should consider configuring static text that is displayed on the editor. For more information, see the section Adding Static Text to the Custom Editor in the chapter Configuring Custom Editors.

- **Alias**: If the **Analysis Intent Mode** attribute of the object class is set to `Full`, a list of synonyms for the object class property caption can be defined in this attribute. If a user searches for one of the synonyms in the `Analyze` intent of the AlfaBot, the AlfaBot will provide search results for the object class property that the synonym is defined for.

2) In the toolbar, click the **Save** button to save your changes.

Configuring Custom Properties of the Type Boolean

An object class property of the type `Boolean` has the permissible values `True` or `False`.

Please note that a `NULL` value will be stored in the Alfabet database for a data type if no value is defined for the custom object class property. In the case of custom object class properties of the type `Boolean`, it is highly recommended that a default value is configured to ensure that the value `False` (rather than `NULL`) is stored for a custom object class property if the associated checkbox that is not selected.

To add a custom object class property of the type `Boolean` to a custom editor, add a `CheckBox` interface control to the custom editor. For more information, see the section Adding a Checkbox to the Custom Editor in the chapter Configuring Custom Editors.

To define a custom object class property of the type `Boolean`:

1) In the attribute window, define the following attributes, as needed:

- **Property Type**: Select `Boolean`.
- **Name**: Enter a name of the object class property. The name must be unique in the scope of the selected object class.
- **Caption**: Specify a caption to be displayed for the property in the Alfabet user interface. If no caption is defined, then no caption will be displayed in the Alfabet user interface.
- **Comments**: Provide information relevant to solution designers regarding the maintenance of the custom object class property.
- **Enable for Data Capture Templates**: Select `True` if the object class property may be captured via data capture templates. Select `False` if the object class property may not be captured via data capture templates. The **Enable for Data Capture Templates** attribute must be set to `True` for the object class that the object class property is assigned to. For more information about data collection via data capture templates, see the section *Capturing Data with Data Capture Templates* in the reference manual *User and Solution Administration*.

  If the protected property is inherited from the parent object class `Artifact` or `ArtifactAuthorized`, then the **Enable for Data Capture Templates** attribute will be greyed out. In this case, you must specify a local setting for the property that will apply to the property only in the context of this object class. To do so, expand the **Local Settings** section and set the **Enable for Data Capture Templates** attribute to `True` to override the inherited setting from the parent object class. Please note that in order to prevent the ID of objects being erroneously changed via data capture templates, the `ID` property cannot be enabled as a class property.

  Properties of the type `BinData`, `Picture`, `DateArray`, `POSIX`, `ByteArray`, `DateArray`, `DateTimeArray`, `BindataArray`, `IntegerArray`, `TimeArray`, `PropertySet`, `StringArray`, and `URLArray` are not suitable for data capture templates.

- **Searchable**: Please consider the following:
- Set to `True`. if the object class property is to be searchable in the search functionalities. The custom object class property will be displayed in the searchable properties field in the standard search functionalities as well as relevant standard or custom selectors implemented for search filters. If you select `False`, the custom object class property will not be searchable in the standard search functionalities nor in the standard or custom selectors implemented for search filters.
- Set to `True` to include the property in the **Object Filter** field in diagrams. The multiple selection combo box displays searchable standard and custom object class properties using the following nomenclature: `<ClassName>.<CustomPropertyName> = <CustomPropertyValue>` For more information about the use of object filters in diagrams and other views, see the section *Defining Multiple Selection Combo Box Fields* in the reference manual *Getting Started with Alfabet*.

You should carefully consider which custom object class properties should be searchable. Unwarranted searchable custom object class properties make drop-down lists excessively long. Additionally, some properties (such as telephone numbers) should not be searchable for reasons of data security. For more information about configuring the search functionalities in Alfabet, see the chapter Configuring Custom Selectors and Search Functionalities. For more information about the use of the search functionalities available, see the section *Searching for Your Objects* in the reference manual *Getting Started with Alfabet*.

.

- **Default Value**: Optional for a `String` property only: Enter a default value for the object class property. The default value is automatically entered for the object class property when a new object is created, but can be edited by the user as needed. The value entered must correspond to the object class property type defined. If an enumeration is specified for the **Enum** attribute, enter a value that has been configured for the enumeration.

  In some cases, you may want to display a predefined default value that cannot be edited in the custom editor. In this case, you could define the interface control to be non-editable (ReadOnly). It will be displayed in grey and thus disabled and therefore cannot be edited. For more information about configuring an editor field to be non-editable, see the section Specifying an Editor Field To Be Non-Editable in the chapter Configuring Custom Editors.

- **Hint**: Enter informational text about the custom object class property to be displayed in the Alfabet interface. Users can view the tooltip for the custom object class property in the **Attributes** section of the respective object view. Please note that the **Hint** attribute defined for the custom object class property will not be displayed in a custom editor. The **Hint** attribute must be explicitly configured for the interface control associated with the custom object class property in the custom editor. For more information about the specification of custom editors, see the section Adding Help Text to Interface Controls in the Custom Editor in the chapter Configuring Custom Editors.

  Please note the following:

  - The text entered in the **Hint** attribute may not be longer than 600 characters. Texts exceeding 600 characters cannot be saved to the Alfabet database. Please note that if the hint is to be translated, the translated text may also not exceed 600 characters.
  - Hints are displayed in the tooltip functionality available via Microsoft Internet Explorer. Therefore, a tooltip will only remain open for a few seconds. As a consequence, the content of the tooltip should not be excessively long. If extensive information is required to guide users in defining the object class property, you should consider configuring static text that is displayed on the editor. For more information, see the section Adding Static Text to the Custom Editor in the chapter Configuring Custom Editors.

- **Alias**: If the **Analysis Intent Mode** attribute of the object class is set to `Full`, a list of synonyms for the object class property caption can be defined in this attribute. If a user searches for one of the synonyms in the `Analyze` intent of the AlfaBot, the AlfaBot will provide search results for the object class property that the synonym is defined for.

2) In the toolbar, click the **Save** button to save your changes.

Configuring Custom Properties of the Type Date and DateTime

A property of the type `Date` allows a date (day, month, year) to be captured for the custom object class property. User will be able to enter a date or select a date in a calendar. A property of the type `DateTime` allows a date (day, month, year, timestamp) to be captured for the custom object class property. In this case, users must manually enter the date and time information in the edit field.

The format used to display and capture the date/time information on the user interface is determined by the cultures configured by your enterprise for the Alfabet solution. For more information about configuring the cultures, see the section Specifying the Cultures Relevant to Your Enterprise in the chapter Localization and Multi-Language Support for the Alfabet Interface.

The format used to store the date/time information in the Alfabet database is determined by the setup of the Alfabet database.

To add a custom object class property of the type `Date`or `DateTime` to a custom editor, add an `Edit` interface control to the custom editor. For more information, see the section Adding a Checkbox to the Custom Editor in the chapter Configuring Custom Editors.

To define a custom object class property of the type `Date` or `DateTime`:

1) In the attribute window, define the following attributes, as needed:

- **Property Type**: Select `Date` or `DateTime`.
- **Name**: Enter a name of the object class property. The name must be unique in the scope of the selected object class.
- **Caption**: Specify a caption to be displayed for the property in the Alfabet user interface. If no caption is defined, then no caption will be displayed in the Alfabet user interface.
- **Comments**: Provide information relevant to solution designers regarding the maintenance of the custom object class property.
- **Enable for Data Capture Templates**: Select `True` if the object class property may be captured via data capture templates. Select `False` if the object class property may not be captured via data capture templates. The **Enable for Data Capture Templates** attribute must be set to `True` for the object class that the object class property is assigned to. For more information about data collection via data capture templates, see the section *Capturing Data with Data Capture Templates* in the reference manual *User and Solution Administration*.

If the protected property is inherited from the parent object class `Artifact` or `ArtifactAuthorized`, then the **Enable for Data Capture Templates** attribute will be greyed out. In this case, you must specify a local setting for the property that will apply to the property only in the context of this object class. To do so, expand the **Local Settings** section and set the **Enable for Data Capture Templates** attribute to `True` to override the inherited setting from the parent object class. Please note that in order to prevent the ID of objects being erroneously changed via data capture templates, the `ID` property cannot be enabled as a class property.

Properties of the type `BinData`, `Picture`, `DateArray`, `POSIX`, `ByteArray`, `DateArray`, `DateTimeArray`, `BindataArray`, `IntegerArray`, `TimeArray`, `PropertySet`, `StringArray`, and `URLArray` are not suitable for data capture templates.

- **Searchable**: If the custom object class property is to be searchable in the search functionalities, select `True`. The custom object class property will be displayed in the searchable properties field in the standard search functionalities as well as relevant standard or custom selectors implemented for search filters. If you select `False`, the custom object class property will not be searchable in the standard search functionalities nor in the standard or custom selectors implemented for search filters.

  You should carefully consider which custom object class properties should be searchable. Unwarranted searchable custom object class properties make drop-down lists excessively long. Additionally, some properties (such as telephone numbers) should not be searchable for reasons of data security. For more information about the use of the search functionalities available, see the section *Searching for Your Objects* in the reference manual *Getting Started with Alfabet*.

- **Hint**: Enter informational text about the custom object class property to be displayed in the Alfabet interface. Users can view the tooltip for the custom object class property in the **Attributes** section of the respective object view. Please note that the **Hint** attribute defined for the custom object class property will not be displayed in a custom editor. The **Hint** attribute must be explicitly configured for the interface control associated with the custom object class property in the custom editor. For more information about the specification of custom editors, see the section Adding Help Text to Interface Controls in the Custom Editor in the chapter Configuring Custom Editors.

  Please note the following:

  - The text entered in the **Hint** attribute may not be longer than 600 characters. Texts exceeding 600 characters cannot be saved to the Alfabet database. Please note that if the hint is to be translated, the translated text may also not exceed 600 characters.
  - Hints are displayed in the tooltip functionality available via Microsoft Internet Explorer. Therefore, a tooltip will only remain open for a few seconds. As a consequence, the content of the tooltip should not be excessively long. If extensive information is required to guide users in defining the object class property, you should consider configuring static text that is displayed on the editor. For more information, see the section Adding Static Text to the Custom Editor in the chapter Configuring Custom Editors.

- **Alias**: If the **Analysis Intent Mode** attribute of the object class is set to `Full`, a list of synonyms for the object class property caption can be defined in this attribute. If a user searches for one of the synonyms in the `Analyze` intent of the AlfaBot, the AlfaBot will provide search results for the object class property that the synonym is defined for.

2) In the toolbar, click the **Save**  button to save your changes.

Configuring Custom Properties of the Type Text

An object class property of the type `Text` allows a large amount of text to be captured. In this case, you would add a `Memo` interface control to the custom editor. An HTML editor may also be implemented in custom editors, object profiles, and object cockpits to capture custom properties of the type `Text`. For more information, see the section Adding a Text Box to Capture ASCII and HTML in the Custom Editor in the chapter Configuring Custom Editors.

You can also use an object class property of the type `Text`in conjunction with an enumeration in order to allow the users to select a single value or to select multiple values for an object class property. If you want users to select a single value, you could add a `ListBox`, `ComboBox`, and `RadioButtonGroup` interface control to the custom editor. If you want users to have the open of selecting multiple values for an object class property, you would add a `CheckedListBox` interface control to the custom editor. In this case, the enumeration items can be translated via the translation capability.

Please note that translation is not possible for an object class property of the type `StringArray` and that is why an object class property of the type `Text` is the preferred method for selection of multiple properties.

Please refer to the following documentation in the chapter Configuring Custom Editors to further configure the object class property of the type `Text` in a custom editor.

- To add an `Memo` interface control to the custom editor. For more information, see the section Adding a Text Box to Capture ASCII and HTML in the Custom Editor.
- To add an `ListBox` interface control to the custom editor. For more information, see the section Adding a List Box to Display Enumerations in the Custom Editor.
- To add a `ComboBox` interface control to the custom editor. For more information, see the section Adding a Combo Box to the Custom Editor.
- To add a `RadioButtonGroup` interface control to the custom editor. For more information, see the sectionAdding a Radio Button Group to the Custom Editor .
- To add a `CheckedListBox` interface control to the custom editor. For more information, see the section Adding a Checked List Box to the Custom Editor.

.To define a custom object class property of the type `Text`:

1) In the attribute window, define the following attributes, as needed:

- **Property Type**: Select `Text`.
- **Name**: Enter a name of the object class property. The name must be unique in the scope of the selected object class.
- **Caption**: Specify a caption to be displayed for the property in the Alfabet user interface. If no caption is defined, then no caption will be displayed in the Alfabet user interface.
- **Comments**: Provide information relevant to solution designers regarding the maintenance of the custom object class property.
- **Can Have HTML Content**: Select `True` to allow the implementation of an HTML editor to capture values for the property in custom editors, object profiles, and object cockpits.

If the **Can Have HTML Content** is set to `True` for a property, the **Enable HTML Content** attribute will automatically be set to `True` for the associated Memo interface control in the standard and custom editor. The **HTML Content** attribute can be set to `False` for the Memo interface control if needed. For more information about configuring HTML editors in custom editors, see the section Adding a Text Box to Capture ASCII and HTML in the Custom Editor.

If the HTML editor is enabled and the **Enable Data Translation** attribute is set to `True`for the custom property, the XML attribute `TranslateContentExceedingHTMLLengthLimit` must be configured in the XML object ***AlfaTranslationServicesConfig***. For more information about configuring the XML object

AlfaTranslationServicesConfig, see the section Configuring the Connection to the Translation Service.

- **Enable Data Translation**: Optional for a `Text` property only: If the **Enable Data Translation** attribute is set to `True` for the object class, you can further refine the permissibility to translate data for the custom property.

  Please consider the following:

  - The specification of the **Enable Data Translation** attribute applies to all cultures defined for your enterprise. Therefore, if you specify that a data translation is not permissible for an object class property, then object data cannot be specified for that object class for all defined cultures. Please note that regardless of the specification of the **Enable Data Translation** attribute for an object class/object class property, if the **Support Data Translation** attribute is set to `False` for a culture, data translation will not be possible at all for the culture and the language code will not be displayed in the language field in all object editors.
  - The specification of the **Enable Data Translation** attribute for an object class property will have precedence over the specification made in the **Automated Translation Rules for Class Properties** field for the object class. In other words, if the **Enable Data Translation** attribute for the property is set to `None` or `Manual` and the property is set to `True` in the **Automated Translation Rules for Class Properties** field for the object class, the property will not be automatically translated.
  - For more information about the translation capability, see the section Configuring the Translation of Object Data in the chapter Localization and Multi-Language Support for the Alfabet Interface.

- Set the **Enable Data Translation** attribute to `Manual` to allow data translation for the custom property to be translated manually by the user in the object editor.
- Set the **Enable Data Translation** attribute to `ManualAndAutomated` to allow data translation for the custom property to be translated manually by the user in the object editor or translated via the automated translation capability.
- Set the **Enable Data Translation** attribute to `None` to disable data translation for the custom property.
- **Searchable**: If the custom object class property is to be searchable in the search functionalities, select `True`. The custom object class property will be displayed in the searchable properties field in the standard search functionalities as well as relevant standard or custom selectors implemented for search filters. If you select `False`, the custom object class property will not be searchable in the standard search functionalities nor in the standard or custom selectors implemented for search filters.

  You should carefully consider which custom object class properties should be searchable. Unwarranted searchable custom object class properties make drop-down lists excessively long. Additionally, some properties (such as telephone numbers) should not be searchable for reasons of data security. For more information about the use of the search functionalities available, see the section *Searching for Your Objects* in the reference manual *Getting Started with Alfabet*.

- **Hint**: Enter informational text about the custom object class property to be displayed in the Alfabet interface. Users can view the tooltip for the custom object class property in the **Attributes** section of the respective object view. Please note that the **Hint** attribute defined for the custom object class property will not be displayed in a custom editor. The **Hint** attribute must be explicitly configured for the interface control associated with the custom object class property in the custom editor. For more information about the specification of custom editors, see the section Adding Help Text to Interface Controls in the Custom Editor in the chapter Configuring Custom Editors.

  Please note the following:

  - The text entered in the **Hint** attribute may not be longer than 600 characters. Texts exceeding 600 characters cannot be saved to the Alfabet database. Please note that if the hint is to be translated, the translated text may also not exceed 600 characters.
  - Hints are displayed in the tooltip functionality available via Microsoft Internet Explorer. Therefore, a tooltip will only remain open for a few seconds. As a consequence, the content of the tooltip should not be excessively long. If extensive information is required to guide users in defining the object class property, you should consider configuring static text that is displayed on the editor. For more information, see the section Adding Static Text to the Custom Editor in the chapter Configuring Custom Editors.

- **Alias**: If the **Analysis Intent Mode** attribute of the object class is set to `Full`, a list of synonyms for the object class property caption can be defined in this attribute. If a user searches for one of the synonyms in the `Analyze` intent of the AlfaBot, the AlfaBot will provide search results for the object class property that the synonym is defined for.

2) In the toolbar, click the **Save** button to save your changes.

Configuring Custom Properties of the Type Real

An object class property of the type `Real` requires a positive or negative decimal number as value. The values that may be defined for an object class property of the type Integer may be restricted by associating an enumeration with the custom object class property. The digit symbol used to display numbers on the user interface is determined by the cultures configured by your enterprise for the Alfabet solution. For more information about configuring the cultures, see the section Specifying the Cultures Relevant to Your Enterprise in the chapter Localization and Multi-Language Support for the Alfabet Interface.

To add a custom object class property of the type `Real` to a custom editor, add an `Edit` interface control to the custom editor. For more information, see the section Adding a Checkbox to the Custom Editor in the chapter Configuring Custom Editors.

To define a custom object class property of the type `Real`:

1) In the attribute window, define the following attributes, as needed:

- **Property Type**: Select `Real`.
- **Name**: Enter a name of the object class property. The name must be unique in the scope of the selected object class.

- **Caption**: Specify a caption to be displayed for the property in the Alfabet user interface. If no caption is defined, then no caption will be displayed in the Alfabet user interface.
- **Comments**: Provide information relevant to solution designers regarding the maintenance of the custom object class property.
- **Enable for Data Capture Templates**: Select `True` if the object class property may be captured via data capture templates. Select `False` if the object class property may not be captured via data capture templates. The **Enable for Data Capture Templates** attribute must be set to `True` for the object class that the object class property is assigned to. For more information about data collection via data capture templates, see the section *Capturing Data with Data Capture Templates* in the reference manual *User and Solution Administration*.

  If the protected property is inherited from the parent object class `Artifact` or `ArtifactAuthorized`, then the **Enable for Data Capture Templates** attribute will be greyed out. In this case, you must specify a local setting for the property that will apply to the property only in the context of this object class. To do so, expand the **Local Settings** section and set the **Enable for Data Capture Templates** attribute to `True` to override the inherited setting from the parent object class. Please note that in order to prevent the ID of objects being erroneously changed via data capture templates, the `ID` property cannot be enabled as a class property.

  Properties of the type `BinData`, `Picture`, `DateArray`, `POSIX`, `ByteArray`, `DateArray`, `DateTimeArray`, `BindataArray`, `IntegerArray`, `TimeArray`, `PropertySet`, `StringArray`, and `URLArray` are not suitable for data capture templates.

- **Precision**: To specify the number of decimal places allowed, enter an integer for the object class property.
- **Enumeration**: If the possible values available are to be provided based on a configured enumeration, select the custom enumeration. For information about how to configure an enumeration for a `Integer` property, see the section Defining Protected and Custom Enumerations.
- **Searchable**: If the custom object class property is to be searchable in the search functionalities, select `True`. The custom object class property will be displayed in the searchable properties field in the standard search functionalities as well as relevant standard or custom selectors implemented for search filters. If you select `False`, the custom object class property will not be searchable in the standard search functionalities nor in the standard or custom selectors implemented for search filters.

  You should carefully consider which custom object class properties should be searchable. Unwarranted searchable custom object class properties make drop-down lists excessively long. Additionally, some properties (such as telephone numbers) should not be searchable for reasons of data security. For more information about the use of the search functionalities available, see the section *Searching for Your Objects* in the reference manual *Getting Started with Alfabet*.

- **Hint**: Enter informational text about the custom object class property to be displayed in the Alfabet interface. Users can view the tooltip for the custom object class property in the **Attributes** section of the respective object view. Please note that the **Hint** attribute defined for the custom object class property will not be displayed in a custom editor. The **Hint** attribute must be explicitly configured for the interface control associated with the custom object class property in the custom editor. For more information about the specification of custom editors, see the section Adding Help Text to Interface Controls in the Custom Editor in the chapter Configuring Custom Editors.

Please note the following:

- The text entered in the **Hint** attribute may not be longer than 600 characters. Texts exceeding 600 characters cannot be saved to the Alfabet database. Please note that if the hint is to be translated, the translated text may also not exceed 600 characters.
- Hints are displayed in the tooltip functionality available via Microsoft Internet Explorer. Therefore, a tooltip will only remain open for a few seconds. As a consequence, the content of the tooltip should not be excessively long. If extensive information is required to guide users in defining the object class property, you should consider configuring static text that is displayed on the editor. For more information, see the section Adding Static Text to the Custom Editor in the chapter Configuring Custom Editors.

- **Alias**: If the **Analysis Intent Mode** attribute of the object class is set to `Full`, a list of synonyms for the object class property caption can be defined in this attribute. If a user searches for one of the synonyms in the `Analyze` intent of the AlfaBot, the AlfaBot will provide search results for the object class property that the synonym is defined for.

2) In the toolbar, click the **Save** button to save your changes.

Configuring Custom Properties of the Type Integer

An object class property of the type `Integer` requires a positive or negative integer number as value. The values that may be defined for an object class property of the type Integer may be restricted by associating an enumeration with the custom object class property. The digit symbol used to display numbers on the user interface is determined by the cultures configured by your enterprise for the Alfabet solution. For more information about configuring the cultures, see the section Specifying the Cultures Relevant to Your Enterprise in the chapter Localization and Multi-Language Support for the Alfabet Interface.

To add a custom object class property of the type `Integer` to a custom editor, add an `Edit` interface control to the custom editor. For more information, see the section Adding an Edit Field to the Custom Editor in the chapter Configuring Custom Editors.

To define a custom object class property of the type `Integer`:

1) In the attribute window, define the following attributes, as needed:

- **Property Type**: Select `Integer`.
- **Name**: Enter a name of the object class property. The name must be unique in the scope of the selected object class.
- **Caption**: Specify a caption to be displayed for the property in the Alfabet user interface. If no caption is defined, then no caption will be displayed in the Alfabet user interface.
- **Comments**: Provide information relevant to solution designers regarding the maintenance of the custom object class property.
- **Enable for Data Capture Templates**: Select `True` if the object class property may be captured via data capture templates. Select `False` if the object class property may not be captured via data capture

templates. The **Enable for Data Capture Templates** attribute must be set to `True` for the object class that the object class property is assigned to. For more information about data collection via data capture templates, see the section *Capturing Data with Data Capture Templates* in the reference manual *User and Solution Administration*.

If the protected property is inherited from the parent object class `Artifact` or `ArtifactAuthorized`, then the **Enable for Data Capture Templates** attribute will be greyed out. In this case, you must specify a local setting for the property that will apply to the property only in the context of this object class. To do so, expand the **Local Settings** section and set the **Enable for Data Capture Templates** attribute to `True` to override the inherited setting from the parent object class. Please note that in order to prevent the ID of objects being erroneously changed via data capture templates, the `ID` property cannot be enabled as a class property.

Properties of the type `BinData`, `Picture`, `DateArray`, `POSIX`, `ByteArray`, `DateArray`, `DateTimeArray`, `BindataArray`, `IntegerArray`, `TimeArray`, `PropertySet`, `StringArray`, and `URLArray` are not suitable for data capture templates.

- **Counter Start**: Enter a start value to be entered in order to set the value larger than any of the current IDs of objects imported from a legacy record system. The value in this property is populated only on creation of an object as the sum of the value specified in the **Counter Start** attribute and the numerical part of the object ID (i.e. `REFSTR`).
- **Enumeration**: If the possible values available are to be provided based on a configured enumeration, select the custom enumeration. For information about how to configure an enumeration for a `Integer` property, see the section Defining Protected and Custom Enumerations.
- **Searchable**: If the custom object class property is to be searchable in the search functionalities, select `True`. The custom object class property will be displayed in the searchable properties field in the standard search functionalities as well as relevant standard or custom selectors implemented for search filters. If you select `False`, the custom object class property will not be searchable in the standard search functionalities nor in the standard or custom selectors implemented for search filters.

You should carefully consider which custom object class properties should be searchable. Unwarranted searchable custom object class properties make drop-down lists excessively long. Additionally, some properties (such as telephone numbers) should not be searchable for reasons of data security. For more information about the use of the search functionalities available, see the section *Searching for Your Objects* in the reference manual *Getting Started with Alfabet*.

- **Hint**: Enter informational text about the custom object class property to be displayed in the Alfabet interface. Users can view the tooltip for the custom object class property in the **Attributes** section of the respective object view. Please note that the **Hint** attribute defined for the custom object class property will not be displayed in a custom editor. The **Hint** attribute must be explicitly configured for the interface control associated with the custom object class property in the custom editor. For more information about the specification of custom editors, see the section Adding Help Text to Interface Controls in the Custom Editor in the chapter Configuring Custom Editors.

Please note the following:

  - The text entered in the **Hint** attribute may not be longer than 600 characters. Texts exceeding 600 characters cannot be saved to the Alfabet database. Please note that if the hint is to be translated, the translated text may also not exceed 600 characters.

- Hints are displayed in the tooltip functionality available via Microsoft Internet Explorer. Therefore, a tooltip will only remain open for a few seconds. As a consequence, the content of the tooltip should not be excessively long. If extensive information is required to guide users in defining the object class property, you should consider configuring static text that is displayed on the editor. For more information, see the section Adding Static Text to the Custom Editor in the chapter Configuring Custom Editors.

- **Alias**: If the **Analysis Intent Mode** attribute of the object class is set to `Full`, a list of synonyms for the object class property caption can be defined in this attribute. If a user searches for one of the synonyms in the `Analyze` intent of the AlfaBot, the AlfaBot will provide search results for the object class property that the synonym is defined for.

2) In the toolbar, click the **Save** button to save your changes.

Configuring Custom Properties of the Type URL

A custom object class property of the type URL allows the user to enter a URL or a link that opens a document located in the **Internal Document Selector**. The URL can be displayed as a hyperlink in the **Attributes** section of the object's profile, object cockpit, or configured report. Properties of the type URL that are displayed in previews may be clicked in order to open the target link. The URL must start with `http://` or `https://`. The target link will open in a new browser window.

Please note that custom object class properties that display URLs (**Property Type** = `URL`) cannot be displayed via columns in a page view/configured report.

To add a custom object class property of the type `URL` to a custom editor, add an `Edit` interface control to the custom editor. For more information, see the section Adding an Edit Field to the Custom Editor in the chapter Configuring Custom Editors. To add the custom object class property of the type `URL` to a custom object view, see the section Adding Standard or Custom Properties to the Attributes Section. To add the custom object class property of the type `URL` to an object cockpit, see the section Adding a URL to the Object Cockpit.

Alternatively, the interface control `HTML Content` can be added to a custom editor in order to make a URL, document link, or HTML text available in the custom editor. In this case, the HTML is available only in the custom editor. For more information, see the section Adding an HTML Document Stored in the Internal Document Selector to the Custom Editor in the chapter Configuring Custom Editors.

To define a custom object class property of the type `URL`:

1) In the attribute window, define the following attributes, as needed:

- **Property Type**: Select `URL`.
- **Name**: Enter a name of the object class property. The name must be unique in the scope of the selected object class.
- **Caption**: Specify a caption to be displayed for the property in the Alfabet user interface. If no caption is defined, then no caption will be displayed in the Alfabet user interface.

- **Comments**: Provide information relevant to solution designers regarding the maintenance of the custom object class property.
- **Enable for Data Capture Templates**: Select `True` if the object class property may be captured via data capture templates. Select `False` if the object class property may not be captured via data capture templates. The **Enable for Data Capture Templates** attribute must be set to `True` for the object class that the object class property is assigned to. For more information about data collection via data capture templates, see the section *Capturing Data with Data Capture Templates* in the reference manual *User and Solution Administration*.

  If the protected property is inherited from the parent object class `Artifact` or `ArtifactAuthorized`, then the **Enable for Data Capture Templates** attribute will be greyed out. In this case, you must specify a local setting for the property that will apply to the property only in the context of this object class. To do so, expand the **Local Settings** section and set the **Enable for Data Capture Templates** attribute to `True` to override the inherited setting from the parent object class. Please note that in order to prevent the ID of objects being erroneously changed via data capture templates, the `ID` property cannot be enabled as a class property.

  Properties of the type `BinData`, `Picture`, `DateArray`, `POSIX`, `ByteArray`, `DateArray`, `DateTimeArray`, `BindataArray`, `IntegerArray`, `TimeArray`, `PropertySet`, `StringArray`, and `URLArray` are not suitable for data capture templates.

- **Hint**: Enter informational text about the custom object class property to be displayed in the Alfabet interface. Users can view the tooltip for the custom object class property in the **Attributes** section of the respective object view. Please note that the **Hint** attribute defined for the custom object class property will not be displayed in a custom editor. The **Hint** attribute must be explicitly configured for the interface control associated with the custom object class property in the custom editor. For more information about the specification of custom editors, see the section Adding Help Text to Interface Controls in the Custom Editor in the chapter Configuring Custom Editors.

  Please note the following:

  - The text entered in the **Hint** attribute may not be longer than 600 characters. Texts exceeding 600 characters cannot be saved to the Alfabet database. Please note that if the hint is to be translated, the translated text may also not exceed 600 characters.
  - Hints are displayed in the tooltip functionality available via Microsoft Internet Explorer. Therefore, a tooltip will only remain open for a few seconds. As a consequence, the content of the tooltip should not be excessively long. If extensive information is required to guide users in defining the object class property, you should consider configuring static text that is displayed on the editor. For more information, see the section Adding Static Text to the Custom Editor in the chapter Configuring Custom Editors.

- **Alias**: If the **Analysis Intent Mode** attribute of the object class is set to `Full`, a list of synonyms for the object class property caption can be defined in this attribute. If a user searches for one of the synonyms in the `Analyze` intent of the AlfaBot, the AlfaBot will provide search results for the object class property that the synonym is defined for.

2) In the toolbar, click the **Save**  button to save your changes.

Configuring Custom Properties of the Type Reference

A custom object class property of the type `Reference` is used to define relationship information between two objects that have a one-to-one (1:n) relationship.

A custom object class property of the type `Reference` should only be defined with the help of Software AG Support.

To add a custom object class property of the type `Reference` to a custom editor, you must add either an `EditSearch`, `ComboBox`, or `RadioButtonGroup` interface control to the custom editor. For more information, see the chapter Configuring Custom Editors.

To define a custom object class property of the type `Reference`:

1) In the attribute window, define the following attributes, as needed:

- **Property Type**: Select `Reference`.
- **Name**: Enter a name of the object class property. The name must be unique in the scope of the selected object class.
- **Caption**: Specify a caption to be displayed for the property in the Alfabet user interface. If no caption is defined, then no caption will be displayed in the Alfabet user interface.
- **Comments**: Provide information relevant to solution designers regarding the maintenance of the custom object class property.
- **Enable for Data Capture Templates**: Select `True` if the object class property may be captured via data capture templates. Select `False` if the object class property may not be captured via data capture templates. The **Enable for Data Capture Templates** attribute must be set to `True` for the object class that the object class property is assigned to. For more information about data collection via data capture templates, see the section *Capturing Data with Data Capture Templates* in the reference manual *User and Solution Administration*.

  If the protected property is inherited from the parent object class `Artifact` or `ArtifactAuthorized`, then the **Enable for Data Capture Templates** attribute will be greyed out. In this case, you must specify a local setting for the property that will apply to the property only in the context of this object class. To do so, expand the **Local Settings** section and set the **Enable for Data Capture Templates** attribute to `True` to override the inherited setting from the parent object class. Please note that in order to prevent the ID of objects being erroneously changed via data capture templates, the `ID` property cannot be enabled as a class property.

  Properties of the type `BinData`, `Picture`, `DateArray`, `POSIX`, `ByteArray`, `DateArray`, `DateTimeArray`, `BindataArray`, `IntegerArray`, `TimeArray`, `PropertySet`, `StringArray`, and `URLArray` are not suitable for data capture templates.

- **Type Info**: This attribute allows you to restrict the values that may be defined for an object class property of the type `Reference`. Select the object classes that the reference may target. This attribute should only be defined with the help of Software AG Support.
- **Hint**: Enter informational text about the custom object class property to be displayed in the Alfabet interface. Users can view the tooltip for the custom object class property in the **Attributes** section of the respective object view. Please note that the **Hint** attribute defined for the custom object class

property will not be displayed in a custom editor. The **Hint** attribute must be explicitly configured for the interface control associated with the custom object class property in the custom editor. For more information about the specification of custom editors, see the section Adding Help Text to Interface Controls in the Custom Editor in the chapter Configuring Custom Editors.

Please note the following:

- The text entered in the **Hint** attribute may not be longer than 600 characters. Texts exceeding 600 characters cannot be saved to the Alfabet database. Please note that if the hint is to be translated, the translated text may also not exceed 600 characters.
- Hints are displayed in the tooltip functionality available via Microsoft Internet Explorer. Therefore, a tooltip will only remain open for a few seconds. As a consequence, the content of the tooltip should not be excessively long. If extensive information is required to guide users in defining the object class property, you should consider configuring static text that is displayed on the editor. For more information, see the section Adding Static Text to the Custom Editor in the chapter Configuring Custom Editors.

- **Alias**: If the **Analysis Intent Mode** attribute of the object class is set to `Full`, a list of synonyms for the object class property caption can be defined in this attribute. If a user searches for one of the synonyms in the `Analyze` intent of the AlfaBot, the AlfaBot will provide search results for the object class property that the synonym is defined for.

2) In the toolbar, click the **Save** button to save your changes.

Configuring Custom Properties of the Type ReferenceArray

An object class property of the type `ReferenceArray` is used to define relationship information between two objects that have a one-to-many (n:n) relationship. For details about the Alfabet database storage of a custom object class property of the type `ReferenceArray` based on the configuration of the custom object class property, see the entry for `ReferenceArray` in the *Glossary* in the reference manual *Alfabet Meta-Model*.

A custom object class property of the type `ReferenceArray` should only be defined with the help of Software AG Support.

To add a custom object class property of the type `ReferenceArray` to a custom editor, you must add an `EditSearch` or `CheckedListBox` interface control to the custom editor. For more information, see the sections Adding an Edit Search Field to the Custom Editor and Adding a Checked List Box to the Custom Editor in the chapter Configuring Custom Editors.

To define a custom object class property of the type `ReferenceArray`:

1) In the attribute window, define the following attributes, as needed:

- **Property Type**: Select `ReferenceArray`.
- **Name**: Enter a name of the object class property. The name must be unique in the scope of the selected object class.

- **Caption**: Specify a caption to be displayed for the property in the Alfabet user interface. If no caption is defined, then no caption will be displayed in the Alfabet user interface.
- **Comments**: Provide information relevant to solution designers regarding the maintenance of the custom object class property.
- **Enable for Data Capture Templates**: Select `True` if the object class property may be captured via data capture templates. Select `False` if the object class property may not be captured via data capture templates. The **Enable for Data Capture Templates** attribute must be set to `True` for the object class that the object class property is assigned to. For more information about data collection via data capture templates, see the section *Capturing Data with Data Capture Templates* in the reference manual *User and Solution Administration*.

  If the protected property is inherited from the parent object class `Artifact` or `ArtifactAuthorized`, then the **Enable for Data Capture Templates** attribute will be greyed out. In this case, you must specify a local setting for the property that will apply to the property only in the context of this object class. To do so, expand the **Local Settings** section and set the **Enable for Data Capture Templates** attribute to `True` to override the inherited setting from the parent object class. Please note that in order to prevent the ID of objects being erroneously changed via data capture templates, the `ID` property cannot be enabled as a class property.

  Properties of the type `BinData`, `Picture`, `DateArray`, `POSIX`, `ByteArray`, `DateArray`, `DateTimeArray`, `BindataArray`, `IntegerArray`, `TimeArray`, `PropertySet`, `StringArray`, and `URLArray` are not suitable for data capture templates.

- **Reference Support**: The values that may be defined for an object class property of the type `ReferenceArray` may be restricted by the **Reference Support** attribute and the **Type Info** attribute defined for the object class property:
- If the **Reference Support** attribute is set to `False`, the references can be created to objects of the object classes specified in the **Type Info** attribute. The data is stored in the Alfabet database table of the object class <`TECHNAME` of Object Class> in the column <`TECHNAME` of Object Class Property>.These attributes should only be defined with the help of Software AG Support.
- If the **Reference Support** attribute is set to `True`, the references specified by the object class property are stored in the `RELATIONS` table in the Alfabet database. No database column is available for the object class property in the table of the object itself. This attribute should only be defined with the help of Software AG Support.
- **Hint**: Enter informational text about the custom object class property to be displayed in the Alfabet interface. Users can view the tooltip for the custom object class property in the **Attributes** section of the respective object view. Please note that the **Hint** attribute defined for the custom object class property will not be displayed in a custom editor. The **Hint** attribute must be explicitly configured for the interface control associated with the custom object class property in the custom editor. For more information about the specification of custom editors, see the section Adding Help Text to Interface Controls in the Custom Editor in the chapter Configuring Custom Editors.

  Please note the following:

  - The text entered in the **Hint** attribute may not be longer than 600 characters. Texts exceeding 600 characters cannot be saved to the Alfabet database. Please note that if the hint is to be translated, the translated text may also not exceed 600 characters.

- Hints are displayed in the tooltip functionality available via Microsoft Internet Explorer. Therefore, a tooltip will only remain open for a few seconds. As a consequence, the content of the tooltip should not be excessively long. If extensive information is required to guide users in defining the object class property, you should consider configuring static text that is displayed on the editor. For more information, see the section Adding Static Text to the Custom Editor in the chapter Configuring Custom Editors.

- **Alias**: If the **Analysis Intent Mode** attribute of the object class is set to `Full`, a list of synonyms for the object class property caption can be defined in this attribute. If a user searches for one of the synonyms in the `Analyze` intent of the AlfaBot, the AlfaBot will provide search results for the object class property that the synonym is defined for.

2) In the toolbar, click the **Save** button to save your changes.

Configuring Custom Properties of the Type Email

An object class property of the type `Email` is used to define email addresses for persons or artifacts in Alfabet.

To define a custom object class property of the type `Email`:

1) In the attribute window, define the following attributes, as needed:

- **Property Type**: Select `Email`.
- **Name**: Enter a name of the object class property. The name must be unique in the scope of the selected object class.
- **Caption**: Specify a caption to be displayed for the property in the Alfabet user interface. If no caption is defined, then no caption will be displayed in the Alfabet user interface.
- **Comments**: Provide information relevant to solution designers regarding the maintenance of the custom object class property.
- **Enable for Data Capture Templates**: Select `True` if the object class property may be captured via data capture templates. Select `False` if the object class property may not be captured via data capture templates. The **Enable for Data Capture Templates** attribute must be set to `True` for the object class that the object class property is assigned to. For more information about data collection via data capture templates, see the section *Capturing Data with Data Capture Templates* in the reference manual *User and Solution Administration*.

If the protected property is inherited from the parent object class `Artifact` or `ArtifactAuthorized`, then the **Enable for Data Capture Templates** attribute will be greyed out. In this case, you must specify a local setting for the property that will apply to the property only in the context of this object class. To do so, expand the **Local Settings** section and set the **Enable for Data Capture Templates** attribute to `True` to override the inherited setting from the parent object class. Please note that in order to prevent the ID of objects being erroneously changed via data capture templates, the `ID` property cannot be enabled as a class property.

Properties of the type `BinData, Picture, DateArray, POSIX, ByteArray, DateArray, DateTimeArray, BindataArray, IntegerArray, TimeArray,`

`PropertySet`, `StringArray`, and `URLArray` are not suitable for data capture templates.

- **Enumeration**: If the possible values available are to be provided based on a configured enumeration, select the custom enumeration. For information about how to configure an enumeration for an `Email` property, see the section Defining Protected and Custom Enumerations.
- **Hint**: Enter informational text about the custom object class property to be displayed in the Alfabet interface. Users can view the tooltip for the custom object class property in the **Attributes** section of the respective object view. Please note that the **Hint** attribute defined for the custom object class property will not be displayed in a custom editor. The **Hint** attribute must be explicitly configured for the interface control associated with the custom object class property in the custom editor. For more information about the specification of custom editors, see the section Adding Help Text to Interface Controls in the Custom Editor in the chapter Configuring Custom Editors.

  Please note the following:

  - The text entered in the **Hint** attribute may not be longer than 600 characters. Texts exceeding 600 characters cannot be saved to the Alfabet database. Please note that if the hint is to be translated, the translated text may also not exceed 600 characters.
  - Hints are displayed in the tooltip functionality available via Microsoft Internet Explorer. Therefore, a tooltip will only remain open for a few seconds. As a consequence, the content of the tooltip should not be excessively long. If extensive information is required to guide users in defining the object class property, you should consider configuring static text that is displayed on the editor. For more information, see the section Adding Static Text to the Custom Editor in the chapter Configuring Custom Editors.
- **Alias**: If the **Analysis Intent Mode** attribute of the object class is set to `Full`, a list of synonyms for the object class property caption can be defined in this attribute. If a user searches for one of the synonyms in the `Analyze` intent of the AlfaBot, the AlfaBot will provide search results for the object class property that the synonym is defined for.

2) In the toolbar, click the **Save** button to save your changes.

Changing an Existing Custom Property

With the exception of the **Name** attribute, all editable attributes for a custom object class property must be edited in the test environment and merged to the production environment.

The **Name** attribute can only be changed directly in the production environment. If the **Name** attribute of an existing custom object class property is changed in the test environment and the configuration is merged/read to an existing configuration (production environment), the custom object class property cannot be correctly identified during the merging process and data loss can occur. For more information about changing the technical name of a custom property, see the section Changing the Technical Name for a Custom Object Class Property or Custom Object Class.

It is highly recommended that the **Property Type** attribute of a custom object class property is not changed once the object class property has been implemented in the production environment and users have already defined and saved property values to the Alfabet database. Please note that if the **Property Type** attribute is changed for a custom object class property for an object class for which the history tracking capability is enabled, history data may be lost. The data in an object class property being converted to an object class property type `String` or `StringArray` cannot be more than 255 characters. Otherwise, information will be truncated.

The following table lists the changes that can be made to property types without losing history data.

FROM	TO
`String`	`Text, StringArray`
`Text`	`String, StringArray`
`StringArray`	`Text, String`
`Date, DateTime, Real, Integer`	`String, Text, StringArray`

Editing a Protected Property

A protected object class property  has an orange lock displayed on the icon. This is a standard property that allows some property values to be edited. A protected object class property may not be deleted.

To edit a protected object class property, click the protected object class property  in the explorer to open its attribute window. Define the following, as needed, and click the **Save** button to save the configuration:

- If the protected property is inherited from the parent object class `Artifact` or `ArtifactAuthorized`, then some attributes including the **Enable for Data Capture Templates** attribute will be greyed out. In this case, you must specify a local setting for the property that will apply to the property only in the context of this object class. To do so, expand the **Local Settings** section and define the relevant attributes to override the inherited setting from the parent object class. Expand the **Local Settings** section of the table and define the relevant attributes for the protected property:
- **Caption**:
- **Default Caption**:
- **Hint**: Specify the tooltip that shall be displayed for the property in the **Attributes** section of an object profile and, if relevant, in extended data capture templates. Please note that the **Hint** attribute will not be displayed for properties of the type `ReferenceArray`.

- **Default Hint**:
- **Can Have HTML Content**: Select True to allow the implementation of an HTML editor to capture values for the property in custom editors, object profiles, and object cockpits. the HTML editor will be available for the property for all object class stereotypes specified for the parent object class.
- **Enable for Data Capture Templates**: Select `True` if the object class property may be captured via data capture templates. Select `False` if the object class property may not be captured via data capture templates. The **Enable for Data Capture Templates** attribute must be set to `True` for the object class that the object class property is assigned to. For more information about data collection via data capture templates, see the section *Capturing Data with Data Capture Templates* in the reference manual *User and Solution Administration*.

Please note that in order to prevent the ID of objects being erroneously changed via data capture templates, the `ID` property cannot be enabled as a class property.

Properties of the type `BinData`, `Picture`, `DateArray`, `POSIX`, `ByteArray`, `DateArray`, `DateTimeArray`, `BindataArray`, `IntegerArray`, `TimeArray`, `PropertySet`, `StringArray`, and `URLArray` are not suitable for data capture templates.

- **Caption**: Edit the caption of the object class property that is displayed in the Alfabet interface. The **Caption** attribute will be added to the vocabulary and can be translated.
- **Default Value**: Optional for a `String` property only: To specify a default value for the object class property, enter a default value for the object class property. The default value is automatically entered for the object class property when a new object is created but can be edited by the user as needed. The value entered must correspond to the object class property type defined. If an enumeration is specified for the **Enum** attribute, enter a value that has been configured for the enumeration.
- **Searchable**: Please consider the following:
- Set to `True`. if the object class property is to be searchable in the search functionalities. The custom object class property will be displayed in the searchable properties field in the standard search functionalities as well as relevant standard or custom selectors implemented for search filters. If you select `False`, the custom object class property will not be searchable in the standard search functionalities nor in the standard or custom selectors implemented for search filters.
- . For custom properties of type `Boolean` and `String` that have an enumeration specified for the **Enum** attribute. Set to `True` to include the property in the **Object Filter** field in diagrams. The multiple selection combo box displays searchable standard and custom object class properties using the following nomenclature: `<ClassName>.<CustomPropertyName>` = `<CustomPropertyValue>` For more information about the use of object filters in diagrams and other views, see the section *Defining Multiple Selection Combo Box Fields* in the reference manual *Getting Started with Alfabet*.

You should carefully consider which custom object class properties should be searchable. Unwarranted searchable custom object class properties make drop-down lists excessively long. Additionally, some properties (such as telephone numbers) should not be searchable for reasons of data security. For more information about configuring the search functionalities in Alfabet, see the chapter Configuring Custom Selectors and Search Functionalities. For more information about the use of the search functionalities available, see the section *Searching for Your Objects* in the reference manual *Getting Started with Alfabet*.

.

Defining Protected and Custom Enumerations

An enumeration is a set of preconfigured enumeration items available to users when defining an object class property in Alfabet. An enumeration is associated with a standard or custom object class property. Enumerations are reusable and the same enumeration may be assigned to multiple custom object class properties in different object classes. An enumeration has two or more enumeration items.

The enumeration items represent the values that users can select in a combo-box, checked combo box, list box or checked list box in a standard or custom editor or in standard object filters available in relevant views. For example, an enumeration could consist of the four enumeration items `Mainframe`, `Client Server`, `eBusiness`, and `Other` that are assigned to the custom object class property `Application Type`.

There are two types of enumerations displayed in the **Class Model** explorer in Alfabet Expand:

- A protected enumeration  is an enumeration that has been preconfigured by Alfabet Expand. A protected enumeration cannot be deleted but the enumeration items can be modified according to the needs of your enterprise.
- A public enumeration  is a custom enumeration created by your enterprise. A custom enumeration can be created and configured for custom object class properties of the type `String`, `Text`, `Real`, `Integer`, or `StringArray`. A custom enumeration can be edited and deleted, if necessary.

Please be aware that any modifications that you make to protected and public enumerations will only impact objects created after the modification to the enumeration! All existing objects in the Alfabet database will have the values defined before the modification was made to the enumeration. As a result, there may be inconsistencies in the Alfabet database that can only be rectified by editing the relevant objects in the Alfabet database. For more information about the available protected enumerations, see the section *Overview of Protected Enumerations* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

Please note that a service is offered by Software AG that allows erroneous enumeration item names to be corrected for objects in the Alfabet database. For more information about renaming such enumeration items, please contact Software AG Support.

If a custom object class property that has enumerations assigned to it is associated with an object class for which diagrams are available in Alfabet, then the enumerations will be displayed in the **Object Filter** field for all relevant diagrams. For more information about filter fields, see the chapter *Navigating the Alfabet Interface* in the reference manual *Getting Started with Alfabet*.

The object class `GenericReferenceData` allows objects representing classifications to be captured and provides an alternative to enumerations to capture custom data. Typical classifications that already exist in Alfabet are the reference data types **Connection Types**, **Connection Methods**, **Connection Frequencies**, and **Connection Data Formats**.

For example, the object class `GenericReferenceData` could allow accountability levels for strategic decisions (Direct), management checks (Control), and business actions (Execute) such as those in the IBM® Business Capability Model to be captured for domains/business capabilities in Alfabet. Object class stereotypes must be configured for the object class `GenericReferenceData` to capture each relevant classification system. A custom property should then be configured for the relevant object class to allow the data for the object class stereotype to be captured. In the example object, for example, the object class stereotypes Direct, Control, and Execute would be configured for the class `GenericReferenceData` and a custom property

would be defined for each object class stereotype on the object class Domain (which is used to capture data for business capabilities).

The full range of configuration options such as custom properties (including properties of the type `Reference` and `ReferenceArray`), custom editors, wizards, object views, selectors, and configured reports to analyze and understand the information are available. For more information about configuring object class stereotypes and custom properties for the class `GenericReferenceData`, see the sections Configuring Object Class Stereotypes for Object Classes and Configuring Custom Properties for Protected or Public Object Classes.

The objects for the object class stereotypes can be created in the **Generic Reference Data** page view in the **Reference Data** functionality. For more information, see the section *Configuring Objects for the Object Class Generic Reference Data* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.

The following information is available:

- Creating a Custom Enumeration
- Modifying Protected and Custom Enumerations
- Assigning an Enumeration to a Custom Property
- Providing a Translation for the Custom Enumeration
- Deleting a Custom Enumeration

Creating a Custom Enumeration

A public enumeration  is a custom enumeration created by your enterprise. A custom enumeration can be created and configured for custom object class properties of the type `String`, `Real`, or `Integer`. A custom enumeration can be edited and deleted, if necessary.

The following steps must be carried out in order to implement the enumeration in Alfabet:

- Create the enumeration.
- Create all relevant enumeration items for the custom enumeration. An enumeration item with no value can be created if definition of the custom object class property is optional.
- Assign the enumeration to the relevant custom object class property via the **Enum** attribute of the custom object class property. The **Property Type** attribute of the custom object class property must be either `String`, `Text`, `Real`, `Integer`, or `StringArray`.

Please note that an enumeration cannot be translated if it is assigned to an object class property of the type `StringArray`. If you need to configure an object class property in which multiple selection of values is possible, then the enumeration must be assigned to an object class property of the type `Text`.

- It is recommended that you define a default value for the custom object class property in the **Default Value** attribute.

To create a custom enumeration:

1) Go to the **Meta-Model** tab and expand the **Enums** node. You will see all existing protected enumerations and any existing custom enumerations .

2) Right-click the **Enums** node and select **Add New Enum** . You will see a new custom enumeration labelled `Noname` in the **Enums** node.

3) In the attribute window, edit the **Name** attribute. The enumeration name is a technical name and must be unique.

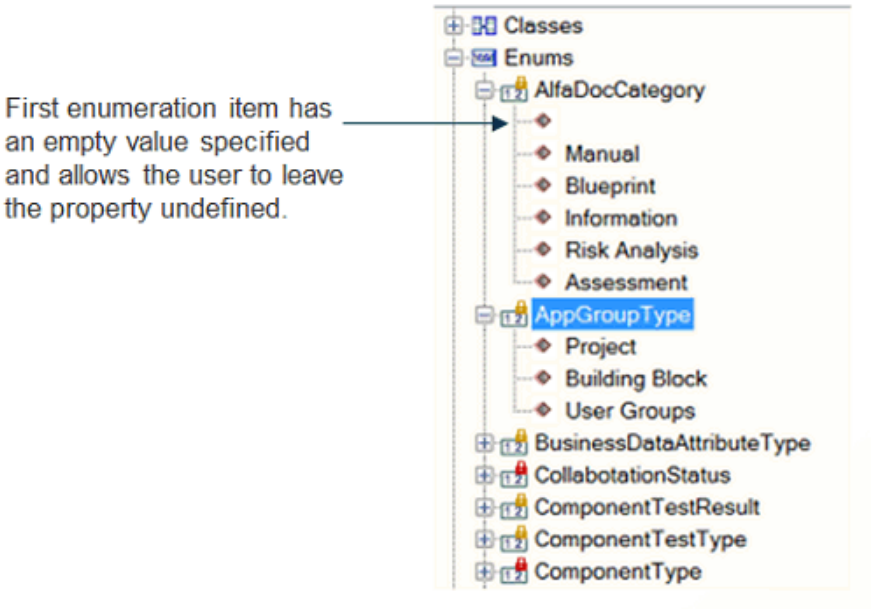
   A validation mechanism checks for correct syntax when defining a technical name. The technical name of a configuration object may not begin with an empty space nor include any of the following special characters: **\ / * ? " > < | ':**

   Furthermore, please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.

   Furthermore, if the technical name of a configuration object is changed, the name will be correctly updated in other configuration objects referencing the changed object during design time. Please note that the name of a changed configuration object will not be updated in guide pages nor if the guide pages reference a configuration object listed in the **Show Usage** functionality. If you plan to change the name of a configuration object, the reference in the guide page should be changed prior to changing the configuration object name.

4) In the **Caption** field, enter a caption for the enumeration. The caption will be displayed in the Alfabet user interface.

5) Next, create the enumeration items for the enumeration. To do so, right-click the custom enumeration and select **Add New Enumeration Items**. The text editor opens.



6) In the text editor, type in the items that should be displayed for the enumeration. These are the values that will appear in the relevant editor fields or object filters. An empty line should be included in the list of items if the users will not be required to select a specific value. A value

should only be entered once in the text editor, otherwise it will be repeated in the interface control. Click **OK** to close the editor.

Only ASCII characters made be used in the name of an enumeration item. Please note that NO error will be displayed if you use characters that are not conform to ASCII.

7) Next, order the sequence of the enumeration items as they should appear in the editor field. Click the **Browse** button for the **Sort Enumeration Items** attribute and in the editor that opens, click each enumeration item as needed and click the **Up/Down** buttons to sequence the items. Click **OK** to save the defined sequence of enumeration items.

8) In order to provide users with information about the meaning of the enumeration values when capturing data in editors, enter information in the **Hint** attribute for each enumeration item. Please note that the **Hint** attribute must be defined for the custom object class property in order to display the hints defined for the enumeration. The text defined in the **Hint** attributes for the enumeration items will be line-separated and appended below the Help text for the object class property.

   The text defined in the **Hint** attribute for the enumeration will be line-separated and appended below the texts for the custom object class property, which is defined via the **Hint** attribute on the custom object class property. The enumeration item Help text will be listed in the sequence defined in the **Sort Enumeration Items** attribute of the enumeration. To define Help text providing information about the enumeration and enumeration items:

- Click the custom enumeration to open its attribute window in the right pane. In the **Hint** field, click the **Browse** button and enter the Help text in the editor. Click **OK** to save the Help text.
- Click the custom enumeration item to open its attribute window in the right pane. In the **Hint** field, click the **Browse** button and enter the Help text in the editor. Click **OK** to save the hint text. Repeat for all enumeration items that require Help text.

9) If the enumeration is to be available in the vocabulary for translation, set the **Extract for Translation** field to `True`. Only enumerations set to **True** will be added to the vocabularies for translation. Note that the enumeration values will only be displayed in the Alfabet interface for supported languages if the **Translate in Meta-Model Vocabulary** attribute of the object class property associated with the enumeration is also set to `True`. This mechanism allows the enumeration to be translated for one object class property of the type `String` and displayed in the original language for another object class property. For more information about translating enumerations, see the section Understanding the Configuration Required to Translate Enumerations in the chapter Localization and Multi-Language Support for the Alfabet Interface.

10) If the enumeration is to be available in the context of a *generic attribute*, set the **Enabled for Generic Attribute** field to `True`. Only enumerations set to **True** will be displayed in the relevant **Type** field in the **Generic Attribute** editor. For more information about generic attributes, see the section Implementing the Generic Attribute Concept Instead of Custom Properties.

11) In the toolbar, click the **Save** button.

You can now assign the custom enumeration to a custom object class property. For more information, see the section Assigning an Enumeration to a Custom Property.

Modifying Protected and Custom Enumerations

For an overview of all protected enumerations that are available below the **Enums** node, see the section *Overview of Protected Enumerations* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

To edit a protected enumeration:

1) Go to the **Meta-Model** tab and expand the **Enums** node. You will see all existing protected enumerations and any existing custom enumerations.
2) To understand which standard and custom object class properties the protected enumerations are assigned to, right-click the protected enumeration  that you want to edit and select **Show Usage in Meta-Model**. An information box will show the all standard and custom object classes and properties <ObjectClass.Property> that use the enumeration. Click **OK** to close the information box.
3) Expand the protected enumeration  to view all preconfigured enumeration items.
4) If the enumeration is to be available in the context of a generic attribute, set the **Enabled for Generic Attribute** field to `True`. For more information about generic attributes, see the section Implementing the Generic Attribute Concept Instead of Custom Properties.
5) In the **Caption** field, enter a caption for the enumeration. The caption will be displayed in the Alfabet user interface.
6) To change the caption of a enumeration item, click the enumeration item  and edit the caption in the **Value** attribute. Repeat for all enumeration items, as needed.

   If the option to specify an undefined value shall be available, you can create an enumeration item with an empty caption in the **Value** attribute. The enumeration item with the empty value should be defined as the first enumeration item in the **Sort Enumeration Items** attribute of the enumeration.

7) To order the sequence of the enumeration items as they should appear in the editor field, click the protected enumeration  to open its attribute window. Click the **Browse**  button in the **Sort Enumeration Items** field. In the editor that opens, click an enumeration item and click the **Up/Down**  buttons to sequence the items. Click **OK** to save the defined sequence of enumeration items.
8) In order to provide the user community with information about the enumeration and the enumeration values, you can optionally provide additional Help text for the associated property. The text defined in the **Hint** attribute for the enumeration will be line-separated and appended below the standard Help text available for the object class property in the relevant editor. The text defined in the **Hint** attributes for the enumeration items will be line-separated and appended below the text for the enumeration Help text. The enumeration item Help texts will be listed in the sequence defined in the **Sort Enumeration Items** attribute of the enumeration. To define additional Help text providing information about the enumeration and enumeration items:

- Click the protected enumeration  to open its attribute window in the right pane. In the **Hint** field, click the **Browse**  button and enter the Help text in the editor. Click **OK** to save the hint text.

- Click the enumeration item  to open its attribute window in the right pane. In the **Hint** field, click the **Browse**  button and enter the Help text in the editor. Click **OK** to save the hint text. Repeat for all enumeration items that require Help text.

9) In the toolbar, click the **Save**  button.

Assigning an Enumeration to a Custom Property

You may assign a protected or public enumeration to one or more custom object class properties of the type `String`, `Text`, `Real`, `Integer`, or `StringArray`. For an overview of all protected enumerations that are available below the **Enums** node, see the section *Overview of Protected Enumerations* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

To assign an enumeration to a custom object class property:

1) Go to the **Meta-Model** tab, navigate to the relevant protected class  or custom class  and click the custom object class property  that you want to assign the enumeration to. The attribute window is displayed in the right pane.
2) In the **Enum** field, select the enumeration that you want to assign to the custom object class property in the drop-down list.
3) Ensure that on of the values `String`, `Text`, `Real`, `Integer`, or `StringArray` is defined for the **Property Type** attribute of the custom object class property.
4) If necessary, select an enumeration item as the default value for the custom object class property in the **Default Value** attribute.
5) In the toolbar, click the **Save**  button. The enumeration is now assigned to the custom object class property.

You can review the enumeration in the custom editor. For more information, see the section Creating a Custom Editor in the chapter Configuring Custom Editors.

Providing a Translation for the Custom Enumeration

The enumeration items configured for protected and custom enumeration items can be translated to a language other than English (United States). For more information about the translation of terms in the Alfabet interface, see the section Modifying, Translating and Managing the Vocabularies in the chapter Localization and Multi-Language Support for the Alfabet Interface.

The following requirements must be met to display translation enumeration items:

- Enumeration values will only be displayed in the Alfabet interface for supported languages if the attribute **Translatable in Meta-Model Vocabulary** of the object class property associated with the enumeration is set to `True`. This mechanism allows the enumeration to be translated for one object class property and displayed in the original language for another object class property.

- The translations specified for enumerations will only be displayed in custom object views and object cockpits if the associated object class property is specified as translatable in the object view or object cockpit:
- For object views, the **Enable Data Translation** attribute must be set to `True` for the object class property displayed in the **Attributes** section.
- For object cockpits, the **Enable Data Translation** attribute must be set to `True` for the Value Control interface control associated with the object class property.
- Configured reports require the specification of instructions to display the translated enumeration values. This is described in detail in the section below Understanding the Additional Configuration Required to Translate Configured Reports in the chapter Localization and Multi-Language Support for the Alfabet Interface.

Deleting a Custom Enumeration

You can delete a custom enumeration that is no longer needed. If you delete an enumeration, you must adjust all properties that reference the enumeration.

To delete a custom enumeration:

1) Go to the **Meta-Model** tab and expand the **Enums** node. You will see all enumerations including any existing custom enumerations.
2) Right-click the custom enumeration that you want to delete and select **Delete**.
3) Confirm the warning by clicking **Yes**.
4) In the toolbar, click the **Save** button. The custom enumeration is removed from the custom editor.

Changing the Technical Name for a Custom Object Class Property or Custom Object Class

In order to allow the Alfabet database to be accessed from external applications or interfaces, the technical name of a custom object class property or, if relevant, custom object class can be changed.

You can assign a new technical name to any custom object class property/custom object class that you create or that already exists in the Alfabet database. Before the technical name of an object class or object class property can be changed, all other changes that have been made must first be saved to the Alfabet database. The changed technical name is written directly to the Alfabet database.

Please keep the following in mind:

- The technical name may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.

- The technical name should only contain standard `ASCII` characters.
- The technical name should consist of upper-case letters only.
- The maximum length of a technical name may not exceed 30 characters.
- The technical name may not coincide with any of the reserved key words of the relational database management system.

A validation mechanism checks for correct syntax when defining a technical name. Please note that if the technical name of a configuration object is changed, the name will be correctly updated in other configuration objects referencing the changed object during design time. However, the name of a changed configuration object will not be updated in guide pages nor if the guide pages reference a configuration object listed in the **Show Usage** functionality. If you plan to change the name of a configuration object, the reference in the guide page should be changed prior to changing the configuration object name.

To change the technical name of a custom object class property or custom object class:

1) Right-click the custom object class property/object class and select **Database** < **Change Class/Property** > **Technical Name**.
2) In the dialog box that opens, enter the new technical name and click **OK**. You will see the name in the **Tech Name** attribute in the attribute window of the relevant object.
3) In the toolbar, click the **Save**  button.

Configuring Release Status Definitions for Object Classes

A release status describes the state of approval for or agreement about an object in the enterprise. Typical status values could be, for example, `Draft`, `Described`, `Reviewed`, `Approved`,`Rejected`, and `Retired`. However, the release statuses will largely depend on the object class that they describe. For example, the release statuses for an architecture element such as an application, component, or standard platform is typically used to express agreement to the state of the documented information whereas, the release statuses for a planning artifact like a demand or project reflects an approval status. The release status for an assignment describes the state of progress or completion of the task that the assignment is about.

An object's release status can be changed by a user with Read/Write access permissions. For all release statuses that are configured as non-editable release statuses, users will not be able to edit the attributes in the object's editor and page views. Users will see the  icon at the top of the object profile of an object that may not be edited due to its current release status definition.

Release status definitions can be customized per object class.

For each relevant object class, you should consider if a release status concept should be implemented for that object class or object class stereotype and, if so, which release statuses are necessary.

For an overview of the object classes that support the concept of release statuses, see the chapter *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*. For more information about configuring release statuses

for object class stereotypes, see the section Configuring the Release Status Definition of Object Class Stereotypes.

When configuring the release statuses for an object class, you should also consider the sequence of release statuses. In other words, which release statuses may transition to a specified release status (for example, which release statuses may transition to a `Retired` status). You should also specify which release statuses allow an object to be edited and which release statuses should be non-editable (for example, typically an object with a `Retired` status would not be editable. Custom tooltips can be defined for each release status in order to help the Alfabet user community to understand the meaning of the release status for the object they are working with.

The XML object ***ReleaseStatusDefs*** in the **Presentation** tab in Alfabet Expand allows you to configure the release statuses available for various objects.

This XML object ***ReleaseStatusDefs*** should be defined as part of the initial configuration of the Alfabet solution. You should NOT change these definitions once the Alfabet database is in use. Please note that a service is offered by Software AG that allows erroneous release status names to be corrected for objects in the Alfabet database. For more information about renaming such release statuses, please contact Software AG Support.

A user logged on with an administrative user profile has edit permissions to all objects at all times, regardless of the user authorization definition or the release status assigned to an object.

Release status definitions may be differently configured for different object classes depending on their context in Alfabet. The number of statuses defined for an object class is determined by the needs of your enterprise. For example:

- Assignments might have the status definitions Created, Accepted, In Progress, Work Completed, Returned, Re-Assigned, Closed. Additional attributes must be defined for the object class Assignment that are necessary for the assignment process in Alfabet. The release status definition for Assignment is described in the table below.
- Architecture-related objects like applications or components might have the status definitions Draft, Under Review, Approved, Archived, Closed.
- Architecture-related objects like business supports might have the status definitions Draft, Under Review, Approved, Retired.
- Test-related objects like component tests might have the status definitions Proposed, Planned, In Execution, Completed, Signed Off, Discarded.
- Planning-related objects like demands might have the status definitions New, Discarded, Described, Reviewed, Assessed, Designed, Redefined, Approved, Rejected, Complete.
- Planning-related objects like projects might have the status definitions New, Created, Supported, Endorsed, Pre-Approved, Approved, Rejected.
- Planning-related objects like solution domain projects or solution business process models might have the status definitions Draft, In Review, Completed.

Please note that an Approved Status (XML element `ApprovedStatus` in the XML object ***ReleaseStatusDefs***) definition is required for the following object classes

- `SolutionProject`

- `SolutionDomainProject`
- `SolutionBusinessProcessModel` (Please note that in the case of the object class `SolutionBusinessProcessModel`, the value entered in the XML element `ApprovedStatus` may not be entered in the XML element `EditableStatusSet`.)
- `ComplianceControlSet`
- `ComplianceProject`

For detailed information about the definition of the attributes in the XML object `ReleaseStatusDefs`, see the following:

- Configuring the Release Status Definition of Object Class Stereotypes
- Defining the Release Statuses Used for the Assignment Functionality
- Configuring the Release Status for Objects Requiring Approval
- Configuring the Release Status Definition for Enterprise Releases
- Configuring the Release Status Definition for Demands
- Configuring the Release Status Definition for Project Stereotypes
- Configuring a Default Release Status for Copied or Version Objects
- Configuring the XML Object ReleaseStatusDefs

Configuring the Release Status Definition of Object Class Stereotypes

If object class stereotypes are configured for an object class, a release status definition must be created for the object class as a whole as well as each object class stereotype. Please note the following:

- The release status definition for the object class must include the complete set of release statuses configured for its object class stereotypes. This is defined in the XML attribute `StatusSet` for the object class. This ensures that the release statuses are available in the **Status** filters available in relevant functionalities. Other than the XML attributes `ClassNames` and `StatusSet`, the other attributes are irrelevant for the release status definition of the object class because the object class stereotype definition will have precedence.
- The release status definition for the object class stereotype should contain only the release statuses relevant for the object class stereotype as well as the sequences of release statuses that are available in order to reach a specific target release status.

For detailed information about the configuration of the XML object ***ReleaseStatusDefs***, see the section Configuring the XML Object ReleaseStatusDefs

Defining the Release Statuses Used for the Assignment Functionality

Assignments allow users to collaborate with one another about objects in the enterprise architecture. For more information about the implementation and configuration required for the assignment capability, see

the section Configuring the Assignment Capability in the chapter Configuring Alfabet Functionalities Implemented in the Solution Environment.

The XML object ***ReleaseStatusDefs*** allows you to configure the statuses available for the assignment capability as well as the transition from one status to the next. A default configuration is available in the XML object. However, the status definitions can be edited and adapted to the needs of your enterprise.

This XML object should be defined as part of the initial configuration of the Alfabet solution. These definitions should not be changed once the Alfabet database is in use.

To edit the XML object ***ReleaseStatusDefs*** for assignments

1) Go to the **Presentation** tab and expand the **XML Objects** node.
2) Right-click the XML object ***ReleaseStatusDefs*** and select **Edit XML**. The XML editor is displayed in the center pane.

```
<ReleaseStatusDef
        ClassNames = "Assignment"

        StatusSet = "Created,Accepted,In Progress,Work Completed,Returned,
                Re-Assigned,Closed"
        RetiredStatusSet = "Created,Closed"
        EditableStatusSet = "Created,Accepted,In Progress,Returned,Re-Assigned,
                Work Completed"
        PrivateStatus = "Returned"
        RedefinedStatus = "Re-Assigned"
        ApprovedStatus = "Work Completed"
        ClosedStatus = "Closed"
        DefaultStatus = "Created">

        <StatusTransition ToStatus="Created" FromStatuses=""/>
        <StatusTransition ToStatus="Accepted" FromStatuses="Created,Re-Assigned"/>
        <StatusTransition ToStatus="In Progress"  FromStatuses="Created,Accepted,
                Work Completed,Re-Assigned"/>
        <StatusTransition ToStatus="Work Completed"  FromStatuses="Created,Accepted,
                In Progress,Re-Assigned"/>
        <StatusTransition ToStatus="Returned" FromStatuses=""/>
        <StatusTransition ToStatus="Re-Assigned" FromStatuses="Created,Accepted,
                In Progress,Returned"/>
        <StatusTransition ToStatus="Closed" FromStatuses=""/>

</ReleaseStatusDef>
```

3) Edit the XML attributes in the status definition for the object class `Assignment`. The table below explains the XML attributes that may be defined.

XML Element (bold) / XML Attribute	Description
ReleaseStatusDef	
`ClassNames`	Enter `Assignment`.

XML Element (bold) / XML Attribute	Description
`StatusSet`	Enter a comma-separated list of all statuses that are to be implemented in the assignment workflow.  `"Created,Accepted,In Progress,Work Completed,Returned,Re-Assigned,Closed"`  Only ASCII characters made be used in the name of an enumeration item. Please note that NO error will be displayed if you use characters that are not conform to ASCII.
`RetiredStatusSet`	Enter a comma-separated list of statuses defined in the XML attribute `StatusSet` in order to indicate that an assignment is retired and may be deleted. If an object with the release status specified in the XML attribute `RetiredStatusSet` may not be edited by users, it should not be listed in the XML attribute `EditableStatusSet`. Users with relevant access permissions may delete an assignment with a status defined in the XML attribute `RetiredStatusSet`. If no status is defined in the XML attribute `RetiredStatusSet`, the deletion of assignments will be forbidden for all users other than a user with administrative permissions.
`EditableStatusSet`	Enter a comma-separated list of statuses defined in the XML attribute `StatusSet` that may be edited by users with relevant access permissions. If an object with the release status specified in the XML attribute `RetiredStatusSet` may not be edited by users, it should not be listed in the XML attribute `EditableStatusSet`.
`PrivateStatus`	Enter one status defined in the XML attribute `StatusSet` that should be assigned to expired mandatory assignments. If the XML attribute `PrivateStatus` is not included in the XML object ***ReleaseStatusDefs*** of it is left unspecified, the value will be defined as `Private` per default.  Expired mandatory assignments are automatically reassigned to the assignment's originator: Only the originator will have edit permissions for the reassigned assignment. In contrast, expired optional assignment will be automatically assigned the XML attribute `ClosedStatus`.
`RedefinedStatus`	Enter one status defined in the XML attribute `StatusSet` to indicate that an assignment has been reassigned to a different user. The status value of the assignment will automatically be changed to the valued specified in the

XML Element (bold) / XML Attribute	Description
	XML attribute `RedefinedStatus` if the assignment is reassigned to another user. If the XML attribute `RedefinedStatus` is not included in the XML object ***ReleaseStatusDefs*** of it is left unspecified, the value will be defined as `Redefined` per default.
`ApprovedStatus`	Enter one status defined in the XML attribute `StatusSet` that indicates that the assignment is complete. Once a user assigns the status specified in the XML attribute `ApprovedStatus` to the assignment, the assignment will no longer be automatically reassigned to the originator when the target date is reached. If the XML attribute `ApprovedStatus` is not included in the XML object ***ReleaseStatusDefs*** of it is left unspecified, the value will be defined as `Approved` per default.
`ClosedStatus`	Enter one status defined in the XML attribute `StatusSet` that should be assigned to an assignment that has been completed by the assignee and has been confirmed as completed by the originator. If the XML attribute `ClosedStatus` is not included in the XML object ***ReleaseStatusDefs*** of it is left unspecified, the value will be defined as `Closed` per default. Expired optional assignments will be automatically assigned the status specified in the XML attribute `ClosedStatus`. In contrast, expired mandatory assignments will automatically be assigned the status specified in the XML attribute `PrivateStatus`.
`DefaultStatus`	Enter one status that is the default status for assignments. The default status is automatically assigned to a new assignment upon its creation.

4) The XML element ***StatusTransition*** allows the sequence of statuses to be defined. You should create an XML element ***StatusTransition*** for each status listed in the XML attribute `StatusSet`. The definition for status transitions is determined by the configuration of the XML attributes `FromStatuses` and `ToStatus`. The status defined for the XML attribute `ToStatus` can be selected by the user subsequent to the status(es) defined in the XML attribute `FromStatuses`. The table below explains the XML attributes that may be defined.

For example, an object must have the statuses `Created` or `Re-Assigned` in order to allow a transition to the status `Accepted`.

```
<StatusTransition ToStatus="Accepted"
FromStatuses="Created,Re-Assigned"/>
```

If you do not enter a value in the quotation marks, no statuses can be selected in order to transition for a status by the user.

XML Element (bold) / XML Attribute	Description
StatusTransition	
`ToStatus`	Enter one status that may follow the statuses described in the XML attribute `FromStatuses`. An XML attribute `ToStatus` definition should be made for each status in the status set.
`FromStatuses`	Enter one or more statuses in a comma-separated list that may precede the status described in the XML attribute `ToStatus`. It is recommended that a XML attribute `FromStatuses` definition should be made for each XML attribute `ToStatus` definition in the status set. Enter the word "any" in the quotation marks if all statuses may be defined for either the XML attribute `ToStatus` or the XML attribute `FromStatuses`. For example: `<StatusTransition ToStatus="Planned" FromStatuses="any"/>`

5) The XML element `Status` definitions allow tooltip texts to be defined for the statuses available for assignments.

XML Element (bold) / XML Attribute	Description
Status	Allows you to provide custom tooltips for each status. Users will see the tooltips for the statuses in the relevant editor. The tooltip text you define in the context of the status configuration will be added to the tooltip hover box for the object class property `Status`. The text defined in the XML attribute `Hint` for the statuses will be line-separated and appended below the text for the standard tooltip.
`Name`	Enter the name of the status.
`Hint`	Enter a Help text that will help users to understand the meaning of the status. It is recommended that the text is not excessively long. The value defined in the XML attribute `Hint` will be displayed as custom Help in editors, filters, and legends. The texts for the individual statuses will be line-separated. The texts defined for custom Help can be translated in the **Translation**

XML Element (bold) / XML Attribute	Description
	Editor. For more information about the translation of custom terminology, see the chapter Localization and Multi-Language Support for the Alfabet Interface.

6) In the toolbar, click the **Save**  button to save the XML definition.

Configuring the Release Status for Objects Requiring Approval

Several functionalities in Alfabet target the approval process of objects. For example, several functionalities include a check-in process where objects are checked into the inventory and thus overwrite existing objects in the Alfabet database. Such processes require an explicit approval before check-in can occur. Therefore, for such Alfabet functionalities to work properly, the XML attribute `ApprovedStatus` is required as part of the objects release status definition in the XML object ***ReleaseStatusDefs***.

To define the `ApprovedStatus` attribute, enter one status that is the approved status for the object's release status. The XML attribute `ApprovedStatus` must be defined for the following object classes:

- `SolutionDomainProject`
- `SolutionBusinessProcessModel` (Please note that in the case of the object class `SolutionBusinessProcessModel`, the value entered in the XML element `ApprovedStatus` may not be entered in the XML element `EditableStatusSet`.)
- `SolutionProject` (For more detailed information about configuring release statuses for projects, see the section Configuring Release Status Definitions for Projects in the chapter Configuring Alfabet Functionalities Implemented in the Solution Environment.).
- `ComplianceProject`and `ComplianceControlSet` (For more detailed information about configuring release statuses for the compliance management capabilities, see the section Configuring Release Status Definitions for Compliance Projects in the chapter Configuring Alfabet Functionalities Implemented in the Solution Environment.).

For detailed information about the configuration of the XML object ***ReleaseStatusDefs***, see the section Configuring the XML Object ReleaseStatusDefs.

Configuring the Release Status Definition for Enterprise Releases

A release status definition must be created for the object classes `EnterpriseRelease` and `ReleaseItem` in the XML object ***ReleaseStatusDefs*** in the configuration tool Alfabet Expand. The release status definition includes the complete set of release statuses implemented in the context of the `ENTRLS_CaptureReleases` functionality as well as the possible sequences permissible for the release statuses to reach a specific target status.

Please note that the `RetiredStatusSet` property for the object class `EnterpriseRelease` should be defined. Otherwise all release statuses will be considered a retired status for an enterprise release and can be deleted.

For detailed information about the configuration of enterprise releases, see the section Configuring Release Status Definitions for the Capture Enterprise Releases Capability. in the chapter Configuring Alfabet Functionalities Implemented in the Solution Environment. For detailed information about the configuration of the XML object ***ReleaseStatusDefs***, see the section Configuring the XML Object ReleaseStatusDefs.

Configuring the Release Status Definition for Demands

In the *Redefining Demands Page View*, users can redefine a demand that is perhaps too vaguely or too broadly defined. Depending on the configuration of release statuses for the object class Demand, the user may be able to create one or multiple new demands. The release status of the original demand will automatically be set to the status `Redefined` and the release status of the new demand(s) will automatically be set to `New`.

Whether a user can create only one new demand to replace the original demand or whether he/she can create multiple demands will depend on the configuration of the release status `Redefined`. If the status `Redefined` is included in the set of release statuses specified in the XML attribute `EditableStatusSet`, then it is possible to create multiple demands for any demand with the `Redefined` status (as well as demands with any other status specified in the XML attribute `EditableStatusSet`.) If the status `Redefined` is not specified in the XML attribute `EditableStatusSet`, then a demand can no longer be redefined once its status has changed to `Redefined`.

For detailed information about the configuration of the XML object ***ReleaseStatusDefs***, see the section Configuring the XML Object ReleaseStatusDefs

Configuring the Release Status Definition for Project Stereotypes

A release status definition must be created for the object class `Project` and for each object class stereotype created for the object class `Project`. The release status definition for the object class `Project` must include the complete set of release statuses implemented in the project management functionalities. These must be specified in the XML attribute `StatusSet` in the release status definition for the object class `Project`. Furthermore, an element `Project:<ProjectStereotype>` should be defined in the XML object ***ReleaseStatusDefs*** for each project stereotype in your enterprise's project framework.

Please note the following regarding the release status definition for projects:

- **Multiple release status set definitions should not be defined for the project management functionalities.** You should conceptualize one release status set that includes all possible release statuses for all project stereotypes. The release statuses in the XML attribute `StatusSet` can be configured as needed for each project stereotype specified in the XML attribute `ClassNames`. Release statuses that are irrelevant for a given project stereotype may be excluded in the XML attribute `StatusSet` of that stereotype.

For example, the XML attribute `StatusSet`of the overall project is conceptualized to include the release statuses: `New, Discarded, Described, InReview, ReviewFailed, Reviewed, Planned, Closed, Discarded`.

- For the object class Project, the following is specified: `StatusSet = "New,Discarded,Described,InReview,ReviewFailed,Reviewed,Planned,Closed,Discarded"`
- For the object class Project:Program, the following is specified: `StatusSet = "New,Discarded,Described,Reviewed,Planned,Closed"`
- For the object class Project:Project, the following is specified: `StatusSet = "New,Described,InReview,ReviewFailed,Reviewed,Planned,Closed,Discarded"`
- For the object class Project:ProjectStep, the following is specified: `StatusSet = "New,Described,Planned,Closed,Discarded"`

- It is recommended that the release status values `Draft` and `Closed` are not included in the XML attribute `RetiredStatusSet`. The status `Draft` typically indicates a new request for a project and the status `Closed` typically indicates that a project is completed. When configuring the retired release status definition for projects, it is recommended that the statuses `Discarded` or `Archived` are implemented in the `RetiredStatusSet`.
- Each project stereotype requires a release status to indicate that the project is approved. This is configured in the XML attribute `ApprovedStatus`. For more information about defining an approval workflow for projects, see the section Configuring the Release Status for Objects Requiring Approval.
- For general information about how to configure release statuses in the XML object ***ReleaseStatusDefs***, see the section Configuring the XML Object ReleaseStatusDefs.

Configuring a Default Release Status for Copied or Version Objects

Typically, when a new object is created as a copy, version, or variant of another object, the value of the **Status** attribute of the base object is copied to the new object. However, it is possible to specify that the XML attribute `DefaultStatus` specified for the object class in the XML object `ReleaseStatusDefs` is used instead.

This is relevant for the following classes: `Application, Component, ComponentTest, Contract, Device, ICTObject, InformationFlow, ITPolicy, ServiceProduct, Sack, StandardPlatform, VendorProduct` as well as the various classes available for integration solution database connections.

To specify that the default status configured for an object class shall be used if a new object is created as a copy, version, or variant of a base object:

1) Go to the **Presentation** tab and expand the **XML Objects** node. You will see all XML objects that can be edited.
2) Right-click the XML object ***SolutionObjects*** and select **Edit XML**. The XML editor is displayed in the center pane.

3) For the XML attribute `SetDefaultStatusOnCreateAsCopy`, specify `"true"` if the value of the XML attribute `DefaultStatus` specified in the XML object `ReleaseStatusDefs` for all relevant object classes shall be used when a new copy, version, or variant is created for an object of that object class. Specify `"false"` if the status of the base objects shall be copied to new objects.

4) In the toolbar, click the **Save** button to save the XML definition.

Configuring the XML Object ReleaseStatusDefs

The XML object ***ReleaseStatusDefs*** allows you to configure the release statuses available for the object classes used in the enterprise as well as the transition from one status to the next. A default configuration is available in the XML object. However, the release status definitions can be edited and adapted to the needs of your enterprise.

If object class stereotypes have been defined for an object class, a release status definition must be created for each object class stereotype. For each object class stereotype, a separate XML element ***ReleaseStatusDefs*** should be created, whereby each specifies `ClassNames = <Object Class:ObectClassStereotype>` should be defined.

This XML object should be defined as part of the initial configuration of the Alfabet solution. These definitions should not be changed once the Alfabet database is in use.

To edit the XML object ***ReleaseStatusDefs*** for object classes in the enterprise

1) Go to the **Presentation** tab and expand the **XML Objects** node. You will see all XML objects that can be edited.
2) Right-click the XML object ***ReleaseStatusDefs*** and select **Edit XML**. The XML editor is displayed in the center pane.
3) Edit the attributes in the release status definition for the relevant object classes. The image above shows how different release status definitions can be configured for various sets of object classes. The table below explains the XML attributes that may be defined.

XML Element (bold) / XML Attribute	Description
ReleaseStatusDefs	
`ClassNames`	Enter the name of the object class(es) that the release statuses specified in the XML attribute `StatusSet` apply to. Specify an XML element ***ReleaseStatusDefs*** for each relevant set of object classes that require a release status definition.  The value of the **Name** attribute of the object class must be used to specify the object class.

XML Element (bold) / XML Attribute	Description
StatusSet	Enter a comma-separated list of all release statuses that are to be implemented in order to describe the progress of the objects in the object classes specified in the XML attribute StatusSet. For example: enter "Draft, Under Review, Approved, Retired". The release statuses you specify will depend on the methodology implemented for the relevant object class. Please note the following: • If the XML attribute PrivateStatus is not included in the XML object ***ReleaseStatusDefs*** of it is left unspecified, the value will be defined as Private per default. • If the XML attribute RedefinedStatus is not included in the XML object ***ReleaseStatusDefs*** of it is left unspecified, the value will be defined as Redefined per default. • If the XML attribute ApprovedStatus is not included in the XML object ***ReleaseStatusDefs*** of it is left unspecified, the value will be defined as Approved per default. • If the XML attribute ClosedStatus is not included in the XML object ***ReleaseStatusDefs*** of it is left unspecified, the value will be defined as Closed per default. ⚠ Only ASCII characters made be used in the name of an enumeration item. Please note that NO error will be displayed if you use characters that are not conform to ASCII.
RetiredStatusSet	Enter a comma-separated list of statuses defined in the XML attribute StatusSet in order to indicate that an object is retired and may be deleted. If an object with the release status specified in the XML attribute RetiredStatusSet may not be edited by users, it should not be listed in the XML attribute EditableStatusSet. Users with relevant access permissions may delete an object with a status defined in the XML attribute RetiredStatusSet. If no status is defined in the XML attribute RetiredStatusSet, the deletion of objects will be forbidden for all users other than a user with administrative permissions.
EditableStatusSet	Enter a comma-separated list of statuses defined in the XML attribute StatusSet that may be edited by users with relevant access permissions.
DefaultStatus	Enter one status that is the default status for the object's release status. The default status is automatically assigned to a new object upon its creation. For example, enter "Draft".

4) The StatusTransition definitions allow the sequence of statuses to be defined. You should create an XML attribute ToStatus for each status listed in the XML attribute StatusSet. The definition for release status transitions is determined by the configuration of the XML attribute

FromStatuses and the XML attribute ToStatus. The release status defined for the XML attribute ToStatus can be selected by the user subsequent to the status(es) defined in the XML attribute FromStatuses. The table below explains the XML attributes that may be defined.

If you do not enter a value in the quotation marks, no release statuses can be selected in order to transition for a release status by the user.

For example, an object must have the release statuses Created or Re-Assigned in order to allow a transition to the release status Accepted.

```
<StatusTransition ToStatus="Accepted"
FromStatuses="Created,Re-Assigned"/>
```

XML Element (bold) / XML Attribute	Description
StatusTransition	
ToStatus	Enter one release status that may follow the statuses described in the XML attribute FromStatuses. An XML attribute ToStatus definition should be made for each release status in the status set.
FromStatuses	: Enter one or more release statuses in a comma-separated list that may precede the release status described in the XML attribute ToStatus. It is recommended that a n XML attribute FromStatuses should be specified for each XML attribute ToStatus in the status set.  Enter the word "any" in the quotation marks if all release statuses may be defined for either the XML attribute ToStatus or the XML attribute FromStatuses. For example: `<StatusTransition ToStatus="Planned" FromStatuses="any"/>`

5) The Status definitions allow tooltip texts to be defined for the release statuses.

XML Element (bold) / XML Attribute	Description
Status	Allows you to provide custom Help text for each release status. The Help text will be available for the release statuses in the relevant editor. The Help text you define in the context of the release status configuration will be in the editor Help for

XML Element (bold) / XML Attribute	Description
	the **Status** field in the relevant editor. The text defined in the ***Hint*** attribute for the release statuses will be line-separated and appended below the text for the standard Help text.
`Name`	Enter the name of the release status.
`Hint`	It is recommended that the text is not excessively long. Enter a Help text that will help users to understand the meaning of the release status. The value defined in the XML attribute `Hint` allows you to provide custom Help text that will be displayed for editors, filters, and legends. The texts for the individual release statuses will be line-separated. The texts defined for custom Help text can be translated in the **Translation Editor**. For more information about the translation of custom terminology, see the chapter Localization and Multi-Language Support for the Alfabet Interface.

6) In the toolbar, click the **Save** button to save the XML definition.

Configuring Object State Definitions for Object Classes

An object state describes the operational status of an object in the enterprise. The object state indicates whether an object is actively used, planned to be used, or has been used in the past. Because an object's start and end dates specify the planned period of activity for the object, the object state should be changed from **Plan** to **Active** once the object's start date is reached. Equally, the object state should be changed from **Active** to **Retired** when the object's end date is reached. If the lifecycle concept is available for an object class the object's start and end dates will initially be aligned with the active period of the object.

An object's object state can be changed by a user with Read/Write access permissions. For objects like applications or components, relevant object dependencies must be taken into consideration when the object state is changed. For applications, for example, the user changing the object state should consider whether the object state should be propagated to the application's associated information flows or business supports.

The number of available states Retired, Active, and Plan cannot be changed, however the names of the preconfigured object states can be customized per object class.

The XML object ***ObjectStateManager*** in the **Presentation** tab in Alfabet Expand allows you to configure the object states available for a preconfigured set of object classes in Alfabet. The object states configured in the XML object ***ObjectStateManager*** are relevant for all object classes that support the object state concept. If object class stereotypes are configured for a relevant object class, then the object state concept will also be available for the object class stereotype.

For an overview of the object classes that support the concept of object states, see the chapter *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

Only the three standard object states are permissible for the object state definition.

- `Retired`: This object state indicates that an object is no longer in use and that there is currently no plan to make it active again.
- `Active`: This object state indicates that an object is presently in active use.
- `Plan`: This object state indicates that an object is currently planned to be used and will be made active at some time in the future.

The number of object states (three) cannot be changed. However, you can edit the name of the object state (for example, you could change `Retired` to `Shut Down`) and change the color used to visualize the object state in page views. You can provide custom tooltips for each object state in order to help the Alfabet user community in specifying the object state definition for an object. The object states must be specified in the XML object ***ObjectStateManager*** in the order `Retired`, `Active`, `Plan`.

This XML object should be defined as part of the initial configuration of the Alfabet solution. These definitions should not be changed once the Alfabet database is in use.

Please note that a service is offered by Software AG that allows erroneous object state names to be corrected for objects in the Alfabet database. For more information about renaming such object states, please contact Software AG Support.

To edit the XML object ***ObjectStateManager***:

1) Go to the **Presentation** tab and expand the **XML Objects** node. You will see all XML objects that can be edited.

2) Right-click the XML object ***ObjectStateManager*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.

3) Define the attributes, as needed. The table below displays the attributes that can be edited for the XML object ***ObjectStateManager***:

XML Element (bold) / XML Attribute	Explanation
ObjectStateDef	 They ***ObjectStateDef*** elements must be specified in the order `Retired`, `Active`, `Plan`, otherwise errors may occur when users attempt to create information flows.
`Name`	Enter a name to represent the respective object state. For example, the object state `Retired` could be renamed `Shut-Down`.  Only ASCII characters made be used in the name of an enumeration item. Please note that NO error will be displayed if you use characters that are not conform to ASCII.
`ShortName`	Enter an abbreviation for the object state. This is useful for object state information displayed in business support matrices.
`Color`	Enter the color that should be used to represent this object state in the relevant page views. The value entered should be a Windows, Web, hexadecimal, or RGB color value.
`Hint`	It is recommended that the text is not excessively long. Enter a Help text that will help users to understand the meaning of the object state. The value defined in the XML attribute `Hint` allows you to provide custom Help text that will be displayed for editors, filters, and legends. The texts for the individual object states will be line-separated. The texts defined for custom Help text can be translated in the **Translation Editor**. For more information about the translation of custom terminology, see the chapter Localization and Multi-Language Support for the Alfabet Interface.

4) In the toolbar, click the **Save**  button to save the XML definition.

Configuring Lifecycle Definitions for Object Classes

When an object class is configured, you should consider if the lifecycle concept should be implemented and, if so, which lifecycle phases are necessary for the specific object class. The XML object ***ObjectLifeCycleManager*** allows you to configure the lifecycle definition and lifecycle phases for the object classes that support lifecycle definition.

For an overview of the object classes for which lifecycle definitions can be configured, see the chapter *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*. Lifecycle definitions cannot be configured for individual object class stereotypes. The object class stereotype will inherit the lifecycle concept of the object class that it is based on.

An object's lifecycle describes the succession of stages that an architecture element goes through. Many objects in Alfabet have a lifecycle, although a lifecycle does not have to be defined for all objects.

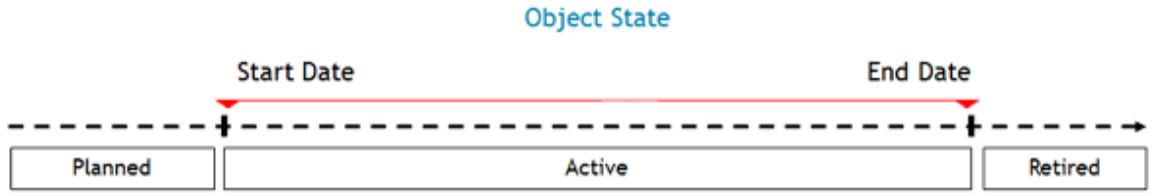


FIGURE: Application object state

The lifecycle definition includes the specification of the object's active period, which corresponds to the object's start and end dates. When the object is first created, the object state will be set to `Active` in alignment with the active period. Even if the object state is later changed, the active period of the lifecycle with correspond to the defined start and end dates of the object.

For reasons of performance, the maximum period of time displayed for lifecycles in the *Lifecycle Page View* is limited to 40 years. The time displayed will be based on 20 years prior to the current date and 20 years after the current date.

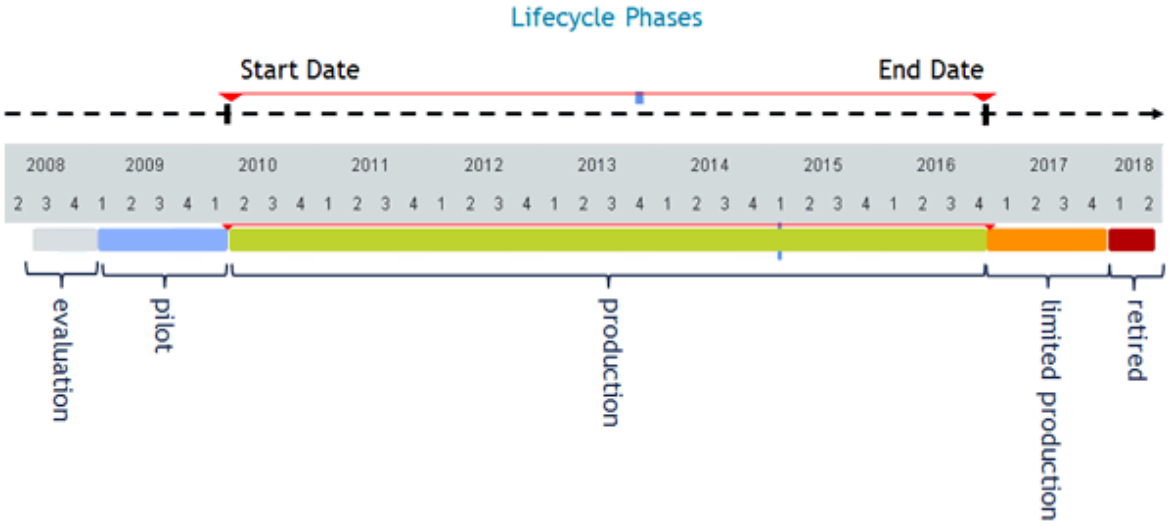


FIGURE: Application lifecycle

The lifecycle is comprised of lifecycle phases that describe the object's status of activity or production. Each lifecycle phase is aligned with its proceeding and succeeding lifecycle phase. A user may choose to skip a lifecycle phase if it is not relevant to the object. One or more lifecycle phases may constitute the active period.

The XML object ***ObjectLifeCycleManager*** allows you to configure the lifecycle definition and lifecycle phases for the object classes that support lifecycle definition. You may configure the number of lifecycle phases that make up the lifecycle definition of a specific object class as well as specify the name and default duration of each lifecycle phase. The actual duration of a lifecycle phase defined for an object may be manually edited in the **Lifecycle** editor available in the *Lifecycle Page View* by a user with relevant access permissions. You can also configure the color used to visualize the lifecycle phase in various relevant time schedule views as well as hint texts for each lifecycle phase in order to provide information about the meaning of the lifecycle phase for the Alfabet user community.

Additionally, you can configure which lifecycle phases are mapped to the active period, the period of time starting with the object's start date and ending with the object's end date. The active period may be configured to represent one or more specified lifecycle phases. For example, the solution designer may configure the active period to constitute the lifecycles phases `Pilot`, `Production`, and `LimitedProduction` but not the lifecycle phases `Evaluation` and `Shut Down`. When new object is created, the active period will correspond to the object's start and end dates. The lifecycle phases must be explicitly defined in the *Lifecycle Page View* available for the relevant object.

It is recommended that you do NOT change the configuration of an object class's lifecycle definition once object lifecycles have been defined in the production environment. If you modify the configuration of an object class's lifecycle definition by adding or removing configured lifecycle phases and objects with the original lifecycle configuration already exist in the Alfabet database, it will be necessary to delete the existing lifecycle and redefine a new lifecycle for all objects for which the new lifecycle phase definition should apply! If an object displayed in a Gantt chart has a set of lifecycle phases that are different than the actual lifecycle phases of the selected object, the misaligned lifecycle will be displayed as a light grey bar in the report.

Please note that a service is offered by Software AG that allows erroneous names of lifecycle phases to be corrected for objects in the Alfabet database. For more information about renaming such lifecycle phases, please contact Software AG Support.

For more information about the methodology of application lifecycle management, see the chapter *Lifecycle Management* in the reference manual *IT Planning Basic*.

To edit the XML object ***ObjectLifeCycleManager***:

1) Go to the **Presentation** tab and expand the **XML Objects** node. You will see all XML objects that can be edited.
2) Right-click the XML object ***ObjectLifeCycleManager*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) Define the attributes, as needed. The table below displays the attributes that can be edited for the XML object ***ObjectLifeCycleManager***:

XML Element (bold) / XML Attribute	Explanation
ObjectLifeCycle	
`ClassNames`	Enter a comma-separated list of object class names that the lifecycle definition defined in this XML element ***ObjectLifecycle*** applies to. **NOTE**: For more information about which object classes support the lifecycle concept, see the chapter *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.
ObjectStatusDef	Add an XML element ***ObjectStatusDef*** for each lifecycle phase that shall be available in the lifecycle of the object classes listed in the XML attribute `ClassNames`.
`Name`	Enter text to define the name of the lifecycle phase. **NOTE**: Only ASCII characters made be used in the name of an lifecycle phase. Please note that NO error will be displayed if you use characters that are not conform to ASCII.
`Color`	Enter text to define the color to be used when representing this lifecycle phase in the visualization of the lifecycle. The value entered should be a Windows, Web, hexadecimal, or RGB color value.
`Length`	Enter an integer that represents the default duration of the lifecycle phase in months in the lifecycle definition. Users can later change the length of the lifecycle phase, as needed. **NOTE**: Depending on the start and end dates defined for an object, the active period of an object may be longer than the sum of the lengths of all lifecycle phases.
`ActivePeriodStart`	Enter "`true`" if the lifecycle phase is the first lifecycle phase of the active period. Only one XML element ***ObjectStatusDef*** should have an XML attribute `ActivePeriodStart`. If only one lifecycle phase shall constitute the active period, then the lifecycle phase should have both the XML attribute `ActivePeriodStart` and XML attribute `ActivePeriodEnd` specified.
`ActivePeriodEnd`	Enter "`true`" if the lifecycle phase is the last lifecycle phase of the active period. Only one XML element ***ObjectStatusDef*** should have an XML attribute `ActivePeriodEnd`.

XML Element (bold) / XML Attribute	Explanation
`Hint`	The value defined in the XML attribute `Hint` allows you to provide custom Help text that may be displayed in editors, filters, and legends. The texts for the individual lifecycle phases will be line-separated. Enter a Help text that will help users to understand the meaning of the lifecycle phase. It is recommended that the text is not excessively long. The texts defined for custom Help text can be translated in the **Translation Editor**. For more information about the translation of custom terminology, see the chapter Localization and Multi-Language Support for the Alfabet Interface.

4) In the toolbar, click the **Save** button to save the XML definition.

Configuring Class Keys for Object Classes

A class key is the specification of one or a combination of standard and custom object class property for an object class. Typically, class keys are defined to implement uniqueness constraints. A standard or configured class key provides an enforcement mechanism for the definition of standard properties and custom object class properties when an object is created. In this case, the combination of properties must be unique for all objects in the object class. If the uniqueness constraint is violated by a user when defining data, a message will be displayed prompting the user to provide a unique combination of values. For example, class keys for the object class Application could require a unique combination of values for the object class properties Name, Version, and Object State.

Index creation is a side effect of a class key definition. For each class key that is configured for the object classes in Alfabet Expand, an index is created. The definition of class keys can therefore be used to speed up query-based searches and thus increase performance.

Please note the following:

- Multiple class keys can be defined as needed for an object class. All custom class keys are public class keys.
- Private class keys cannot be edited or deleted. Private class keys are typically preconfigured for indexing purposes.
- Protected class keys may be edited but not deleted.
- Public class keys may be edited and deleted.

Please keep the following in mind:

- Preconfigured keys override customer-defined class keys. You cannot create a key that duplicates or neutralizes an existing preconfigured class key.

- You cannot create a key that duplicates existing public or preconfigured class keys of the same object class. Two class keys are identical if the values for their attributes **Properties**, **Unique** and **Strict Null Handling** are identical.
- When you create a new class key, any existing data in the Alfabet database will be validated to see if it complies with the new class key constraint. If existing data violates the class key constraints you are currently creating, the class key will be refused. In this case, you will first need to edit the objects in the Alfabet database that violate the class key.
- Class keys are configured on the level of the base object class. Uniqueness constraints cannot be explicitly defined for a specific object class stereotype. The class key configuration is thus inherited by all object class stereotypes configured for the object class. For more information about configuring object class stereotypes, see the section Configuring Object Class Stereotypes for Object Classes.
- If a class key violation occurs when a user attempts to create a new object, a warning message will open listing the object class properties that caused the violation to the defined uniqueness constraints as well as a link to the existing object that is the source of the violation. The user can click the link to open the configured object view specified for the associated class setting in order to understand more about the conflicting object.
- For an overview of all preconfigured private, protected, and public class keys specified for protected object classes, see *Standard Class Keys of Protected Alfabet Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.
- An index is created for all data included in a class key. Class keys can also be defined to exclusively trigger index creation to increase the speed of query-based searches. For more information about index creation based on class keys, see Configuring Class Keys for Object Classes.

To create a class key for a selected object class:

1) Go to the **Meta-Model** tab and right-click the protected object class  that you want to create the class key for and select **Add New Class Key**. You will see the **Class Key Group** node and the new public class key  subordinate to the selected object class.
2) Click the new public class key  to open the attribute window.
3) In the attribute widow, specify the following attributes:

- **Name**: Edit the name for the class key, if necessary.

A validation mechanism checks for correct syntax when defining a technical name. The technical name of a configuration object may not begin with an empty space nor include any of the following special characters: **\ / * ? " > < | ':**

Furthermore, please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.

Furthermore, if the technical name of a configuration object is changed, the name will be correctly updated in other configuration objects referencing the changed object during design time. Please note that the name of a changed configuration object will not be updated in guide pages nor if the guide pages reference a

configuration object listed in the **Show Usage** functionality. If you plan to change the name of a configuration object, the reference in the guide page should be changed prior to changing the configuration object name.

- **Properties**: Click the **Browse** button to open the editor in which you can define the object class properties relevant for the class key constraint. Click each object class property that you want to include in the class key definition. After all necessary properties have been selected, click **OK**.
- **Unique**: Select `True` to implement the uniqueness definition for the object class. Select `False` to allow creation of an index for the key without implementing the uniqueness constraint for the object class.
- **Strict Null Handling**: This attribute is available in order to support various integration scenarios. Set to `False` if all entries with NULL values will be excluded from unique indexes. This allows duplicate entries to be inserted if one of the index attributes is set to NULL. Set to `True` if all entries with NULL values will be checked. The **Strict Null Handling** attribute is set to `True` per default.

4) In the toolbar, click the **Save** button to save your changes.

To delete all protected and custom class keys that have been configured for a protected or custom object class, click the **Class Key Group** node below the relevant object class and select **Delete All Public/Protected Class Keys**. The protected and public class keys will be deleted and no longer displayed below the **Class Key Group** node. Private class keys that have been configured by Software AG will not be deleted.

Specifying History Tracking for an Object Class

Alfabet provides a history tracking functionality that documents the changes made to an object. All changes made to an object's standard and custom object class properties and relations are documented. The *Object Audit Page View* (`Object_Audit`) displays the type of change to the object, the user making the change, the time of the change, and the object class property or relation that has been changed. The data can be exported to an MS Excel file or HTML file. For more information about viewing the audit history of an object, see the section *Viewing the Change History of an Object* in the reference manual *Getting Started with Alfabet*.

The history tracking functionality is implemented on a class-by-class basis. For an overview of the object classes for which an audit table is generated per default as well as an overview of the object classes for which the history tracking functionality is available, see the chapter *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

Please note that the `LAST_UPDATE_USER` and `LASTUPDATE` properties are only updated in the Alfabet database when changes are made via the object's editor, or the editor embedded in a wizard. The `LAST_UPDATE_USER` and `LASTUPDATE` properties are NOT updated when changes are made in page views.

Please note that the history tracking functionality tracks only the change history of objects in the Alfabet solution. An object deleted from the Alfabet database cannot be restored via the history tracking functionality. If an object has been erroneously deleted, the object must be recreated and redefined in the Alfabet solution.

An **Archive Manager** functionality is also available in the **Solution Administration** functionality that allows Alfabet objects to be deleted and an archive object to be created. For more

information about the **Archive Manager** functionality, see the section *Deleting and Archiving Alfabet Objects* in the reference manual *User and Solution Administration*.

Please note the following:

- All object classes in the meta-model have an **Audit** attribute. If the **Audit** attribute is set to `True`, an audit table will be generated in the Alfabet database. Per default, the **Audit** attribute has been set to `True` for most protected classes  . Please be aware that tracking the history of a significant number of object classes may lead to a decrease in performance. Furthermore, changes to some properties for some classes may also produce an excessive number of entries in the Alfabet database table. For example, if the class `Person` is specified to be audited, then every time the user logs in, a new audit record will be created due to changes to the `LASTLOGIN` property.

  The **Audit Key Group** node is only displayed if the **Audit** attribute of the object class is set to `True`. Below the **Audit Key Group** node, all private audit keys  and all custom audit keys  are displayed. An audit key can be configured for a protected object class  in order to trigger index creation to increase the speed of query-based searches in audit tables. For information about creating an index, see the section *Creating an Index to Improve Performance for Query Searches in Database Tables* in the chapter Defining Queries.

- Alfabet provides a standard *Object Audit Page View* (`Object_Audit` can be opened via the **Audit Trail**  displayed in the toolbar of the relevant object profile. This view is available for all artifact classes. Users with ReadOnly and ReadWrite access permissions may view the report. For other object classes requiring history tracking, a report can be configured to display history tracking information.

- The *Object Audit Page View* (`Object_Audit`) can be included in the custom object views configured for a public class  .

Any changes made to the meta-model will impact both the relevant class table and audit table in the Alfabet database. For example, if a custom object class property is deleted from the class model, it will also be deleted from the audit table and thus will no longer be displayed in the *Object Audit Page View* available in your Alfabet solution.

It is highly recommended that you do not change the meta-model once the auditing capability is implemented, otherwise errors may occur in the audit information.

It is highly recommended that the **Property Type** attribute of a custom object class property is not changed once the object class property has been implemented in the production environment and users have already defined and saved property values to the Alfabet database. Please note that if the **Property Type** attribute is changed for a custom object class property for an object class for which the history tracking capability is enabled, history data may be lost. The data in an object class property being converted to an object class property type `String` or `StringArray` cannot be more than 255 characters. Otherwise, information will be truncated.

The following table lists the changes that can be made to property types without losing history data.

FROM	TO
String	Text, StringArray
Text	String, StringArray
StringArray	Text, String
Date, DateTime, Real, Integer	String, Text, StringArray

Users with ReadOnly and ReadWrite access permissions may view the *Object Audit Page View* (Object_Audit). Because the *Object Audit Page View* (Object_Audit) displayed for a selected object displays all information about the object independent of the access permission available to the user viewing the audit information, it is recommended that you customize the display of audit information for users. Please note the following configuration requirements

- Create a native SQL-based configured reports to extract specific information from the audit history such as the changes made to the object by a specific user or specific types of changes made to the object (for example, new information flows added to an application). For information about the structure of the Alfabet database table for auditing and how to configure such a report, see the section Defining Audit Management Related Configured Reports in the chapter Configuring Reports.
- Once the report is configured, you must make it available in the relevant custom object view by configuring a custom button for the relevant custom object view. For more information about how to configure a custom button, see the section Configuring Custom Buttons for the Toolbar of a Custom Object View in the chapter Configuring Object Views.
- The standard **Audit Trail** button can be hidden from the relevant custom object view in the context of the user profile configuration. For more information, see the section Hiding Functionalities in an Object View in the chapter Configuring User Profiles for the User Community.

Please note the following configuration information:

- The parameter **Enable Audit** must be set to True in the server settings configuration in order for the history tracking functionality to be enabled. If the **Enable Audit** parameter is not set to True, history tracking will not occur regardless of the configuration of the **Audit** attribute on object classes. For more information about configuring server settings, see *Activating the History Tracking Functionality* in the reference manual *System Administration*. Please note that if the **Audit** attribute is changed to False, the audit table in the Alfabet database will be deleted.
- By default, the user name of the user performing a change to an object is stored in the audit table in the Alfabet database and displayed to other users in the history of the object when clicking the **Audit Trail** button. The Alfabet Server can be configured to use the **Technical Name** attribute of the user instead of the user name for the history

tracking functionality. This specification is carried out in the Update History User Name attribute in the server alias. For more information, see the section *Configuring User Information Displayed in History Tracking* in the reference manual *System Administration*.

- A change in an object's relationship to another object (captured via an object class property of the type `Reference` or `ReferenceArray`) will only be documented in the **Object Audit** view if BOTH object classes in the relationship have the **Audit** attribute set to `True`. For example, to capture the information that an assignment assigned to an application is reassigned to an ICT object, both of the object classes `Application` and `ICT Object` must have the history tracking capability enabled.
- Any properties that have been specified in the **Excluded Properties** attribute for a class setting will be hidden in he **Object Audit** view. Audit entries associated with hidden reference array properties will be completely hidden from the audit history. Audit entries for **Create Object**, **Update Object**, and **Delete Object** will not be hidden,but they will be displayed without the detail information about which property has been changed.
- If the Alfabet database already contains objects of the object classes for which audit is enabled, audit tables must be initiated by creating audit entries for the already existing objects. Software AG provides the executable `AuditInitTables.exe` to initialize audit tables. Initialization is only performed for empty tables. It is therefore important to execute `AuditInitTables.exe` immediately after activation of audit. For information about how to perform the initialization, see *Activating the History Tracking Functionality* in the reference manual *System Administration*.
- An index can be created for audit tables of individual object classes via the **Audit Key Group** node below the respective object class. The **Audit Key Group** node is only displayed if the **Audit** attribute of the object class is set to `True`. The availability of an index considerably increases the speed of query-based searches in audit tables. To trigger index creation, an audit key must be configured for the relevant object class. For more information about creating an index, see the section *Creating an Index to Improve Performance for Query Searches in Database Tables* in the chapter Defining Queries.
- A class that has the history tracking enabled can also be monitored for activity and inactivity. Therefore, if you plan to implement the activity monitors or inactivity monitors for an object class, you must also set the **Audit** attribute to `True`. For more information about configuring monitors, see the section Configuring Monitors in the chapter Configuring Alfabet Functionalities Implemented in the Solution Environment.
- If the **Audit** attribute is set to `True` for the object class `Indicator Type` and multiple indicator types are edited via the multi-edit functionality, a history entry will be added to the **Object Audit** view for each indicator. This could lead to a significantly large number of audit entries.

The **Audit** attribute can be set to `True` for all object classes by clicking the **Classes** node and selecting **Turn Audit On for All Protected/Public Classes**. If selected The **Audit** attribute will be set to `True` for all protected and public object classes. The **Audit** attribute can be set to `False` for individual classes or set to `False` for all protected and public classes by selecting **Turn Audit Off for All Protected/Public Classes**.

To implement the history tracking capability for a selected object class:

1) Go to the **Meta-Model** tab, expand the **Classes** node, and click the protected class  or public class  that you want to implement the history tracking for.

2) In the attribute window, set the **Audit** attribute to `True`. If the **Audit** attribute is set to `False`, the object class will not be audited. Please note that if the **Audit** attribute is changed to `False`, the audit table in the Alfabet database will be deleted.

3) In the toolbar, click the **Save**  button to save your changes.

Chapter 6: Configuring Custom Editors

A custom editor is a customized data entry view that allows users to capture data for the custom object class properties that have been created for a protected or public (custom) object class. A custom editor is typically configured for the needs of users working with a specific user profile. A custom editor can only be created for an object class that has custom object class properties defined. A custom editor cannot be created for a protected class that has no custom object class properties defined.

A custom editor typically consists of one or more tabbed pages that are automatically appended to the standard editor available for the object class. The solution designer designs the interface controls such as combo boxes, checkboxes, etc. that users need to capture the required data. Additional text fields or URLs may be included in the custom editor in order to provide users with additional information to help them define data. All captions created for the interface controls can be translated for the languages implemented in the Alfabet interface.

Custom editors are displayed in the context of standard data capture editors but can also be implemented in wizards and workflows configured for the enterprise. Multiple custom editors can be created for an object class, but only one custom editor can be assigned to a class setting. Therefore, only one custom editor can be implemented per object class per user profile.

The following describes the general procedure for configuring and implementing a custom editor:

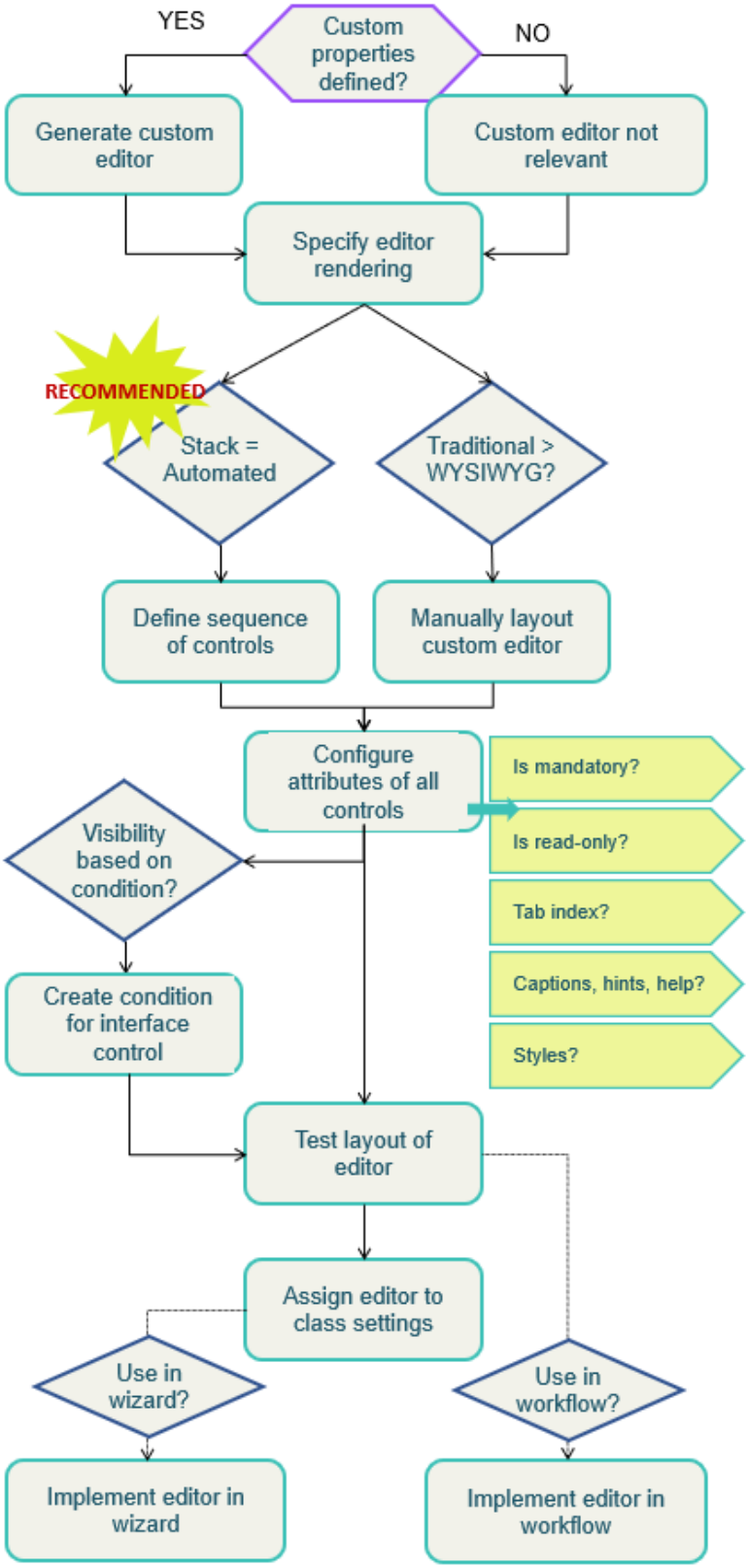


FIGURE: Steps to configure a custom editor

- Ideally, you have already configured all custom object class properties that you want to display in the custom editor.

- You can manually create a custom editor or automatically generate a default custom editor as a starting point for the editor configuration. Please note the following advantages if you automatically generate a default custom editor:
- Each property type is automatically converted to one predefined interface control and placed on a custom editor tab. This default interface control can be deleted and replaced with a different interface control, as needed.
- If more than ten custom object class properties have been defined, additional tabs will be automatically generated in the custom editor. The order of the custom object class properties is determined by the sequence in which they were created. The distribution of custom object class properties on tabbed pages can be modified, if needed.
- After you automatically generate the custom editor, you can implement it and display the default layout in Alfabet. If you are not satisfied with the layout of the custom editor or you want to further design it, you can manually revise the layout of the interface controls, edit the attributes specifying the interface control, and add or delete interface controls. Additionally, you can change the captions and size of the interface controls, redistribute the interface controls across tabbed pages, add instructional text to the editor, add URL or document links to the editor, and customize the custom editor by adding color and images to the custom editor.
- Add and refine the interface controls as needed. You can add and delete interface controls and add additional information to the custom editor such as instructional text, URL and document links, help text, icons, and color and formatting.
- Test the layout of the custom editor via the **Review Custom Editor** functionality.
- Assign the custom editor to a class setting, thus making it available for a specific user profile. For more information about assigning a custom editor to a class setting, see Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.
- If relevant, assign the custom editor to a wizard or workflow in which it is to be implemented. For more information, see the sections Configuring a Wizard Step to Display a Standard or Custom Editor in the chapter Configuring Wizards and Defining the View Implemented for a Workflow Stepin the chapter Configuring Workflows.

The following information is available:

- Conceptualizing a Custom Editor
- Creating a Custom Editor
- Creating a New Custom Editor from Scratch
- Generating a Default Custom Editor
- Copying an Existing Custom Editor
- Specifying the Size of the Custom Editor
- Specifying the Rendering Definition of the Custom Editor
- Configuring the Interface Controls in the Custom Editor
- Overview of the Interface Controls Available for Custom Editors
- Adding Tabbed Pages to the Custom Editor

- Adding an Edit Field to the Custom Editor
- Adding a Text Box to Capture ASCII and HTML in the Custom Editor
- Adding a Text Box to Capture ASCII Format
- Adding a Text Box with an Embedded HTML Editor
- Adding a Field to Capture Dates in the Custom Editor
- Adding a Checked List Box to the Custom Editor
- Adding a Checkbox to the Custom Editor
- Adding a Combo Box to the Custom Editor
- Adding a Radio Button Group to the Custom Editor
- Adding an Edit Search Field to the Custom Editor
- Adding a Role Edit Search Field to the Custom Editor
- Adding Static Text to the Custom Editor
- Adding a List Box to Display Enumerations in the Custom Editor
- Adding Icons to the Custom Editor
- Adding an HTML Document Stored in the Internal Document Selector to the Custom Editor
- Adding a URL to the Custom Editor
- Adding HTML Text to the Custom Editor
- Adding a Color Selector to the Custom Editor
- Specifying the Interface Control To Be Mandatory
- Specifying an Editor Field To Be Non-Editable
- Specifying Conditional Constraints in the Custom Editor
- Creating Conditions to Implement in a Custom Editor
- Assigning Conditions to Interface Controls in the Custom Editor
- Specifying the Tab Order of the Interface Controls
- Adding Help Text to Interface Controls in the Custom Editor
- Hiding Fields in the Custom Editor
- Reviewing the Visualization of an Existing Custom Editor
- Configuring Styles and GUI Scheme Settings for Standard and Custom Editors
- Deleting a Custom Editor from the Custom Editors Folder
- Configuring Editors for the Mass Update of Data Capture Objects and Information Flows
- Modifying Preconfigured Editors Provided by Software AG

Conceptualizing a Custom Editor

A custom editor allows you to make some or all of the custom object class properties defined for an object class available to Alfabet user community. A custom editor is appended to the standard editor as one or more tabbed pages. Once the custom editor is designed, you can associate the custom editor to the relevant standard editor by assigning it to one or more class settings. You can create multiple custom editors per object class and assign each to a different class setting, thus providing a custom editor for the various user profiles implemented to capture data for the object class.

In the context of configuring visibility issues for a user profile, you may exclude standard properties as well as entire standard tabs available in the standard editor. However, you should take careful consideration to not hide tabs with mandatory properties if users will be required to create new objects or edit mandatory properties when they access the editor via the configured user profile. For more information about configuring the visibility of a standard and/or custom editor, see the section Hiding Object Class Properties in Editors and Wizards in the chapter Configuring User Profiles for the User Community.

A custom editor can be reused in multiple contexts. Not only can custom editors be implemented in the context of standard data capture editors, they can also be used in standard and custom wizards as well as workflows. For more information about the configuration of wizards, see the chapter Configuring Wizards. For more information about the configuration of workflows, see the chapter Configuring Workflows.

Please note that some standard page views in Alfabet implement a predefined editor when creating and editing objects. For an overview of the page views with a predefined editor, see the section *Alfabet Page Views with Preconfigured Editing and Navigation Behavior* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

Please note the following regarding the inline editing functionality:

- Only scalar and reference properties that are editable in the editor/wizard available in the object profile or object cockpit can be edited via inline editing.
- All unique constraints defined for the class as well as any post-conditions configured for the wizard associated with the object profile/object cockpit will be applied to the data entered.

  Please note that data is saved to the Alfabet database prior to the post-condition being validated. Therefore, data entered via inline editing will be saved even if the post-condition is not fulfilled. This is because the data must be available in the Alfabet database in order to be evaluated for the post-condition. If the data entered does not fulfill the post-condition, the configured warning message will be displayed explaining that the input must be corrected in order to fulfill the post-condition.

- Inline editing may be limited or prevented if the syntax of the post-condition is not correct. In this case, an error will be displayed if the user tries to open the wizard.

The following issues must be considered when you configure custom editors:

- How will the custom editor be used? Is the custom editor being configured in order to capture enterprise-specific data in the standard editor? Will the custom editor be used in a wizard configuration? Will the custom editor be used in a workflow configuration?
- Which custom object class properties need to be included in the custom editor for the context for which it is being configured?
- How should the custom object class properties be grouped? Should all custom object class properties be on one tabbed page or split up among several tabbed pages.
- What kind of interface controls should be used to capture values for the custom object class properties? The type of interface control is usually dependent on the property type of the custom object class property it is associated with. For an overview of the interface controls available for a specific property type, see the section Configuring Custom Editors.
- Do the captions for custom object class properties need to be translated for other interface languages used in your Alfabet solution?

Creating a Custom Editor

Multiple custom editors can be created per object class. There are a number of different ways to create a custom editor. The method you choose will depend on the number of custom object class properties defined for an object class and the kind of interface controls that should visualize these custom object class properties:

- Create a new custom editor from scratch. This method requires that you add each individual custom object class property to the editor and define and specify the associated interface controls.
- Automatically generate a custom editor with a default layout. This method is easier than creating a custom editor from scratch because the default layout serves as a starting point in editor configuration. All custom object class properties including their captions are automatically added to the editor. The standard interface control type available for the property type of the custom object class property will be automatically generated. If more than ten custom object class properties have been defined for the object class, additional tabbed pages will be generated, and the custom object class properties will be distributed among the tabbed pages.

If you choose to modify the default design of the custom editor, you can change the type of interface controls generated for the custom object class properties, redistribute the custom object class properties among different tabbed pages in the custom editor, reposition and change the captions of the interface controls in a tabbed page, add or delete interface controls, and add text, color, links, and icons to the custom editor.

- Create a custom editor based on a copy of an existing custom editor for the object class. Once the custom editor is copied, you can delete and add interface controls, as needed.

Multiple class settings can be created for an object class and a different custom editor can be assigned to each class setting. For more information about class settings, see Configuring Class Settings for Object Classes and Object Class Stereotypes.

For an overview of the options to consider when working with custom editors, see the section Conceptualizing a Custom Editor.

The following information is available:

- Creating a New Custom Editor from Scratch
- Generating a Default Custom Editor
- Copying an Existing Custom Editor
- Specifying the Size of the Custom Editor

Creating a New Custom Editor from Scratch

You can create a new custom editor and add individual interface controls as needed.

1) In the **Presentations** tab, click the + symbol next to the **Custom Editors** folder. You will see all existing custom editors listed in the tree.
2) Right-click the **Custom Editors** folder and select **New Custom Editor**.
3) In the dialog that opens, select the object class defined for the custom editor that you want to copy and click **OK**. A new custom editor is displayed in the tree.
4) Right-click the existing custom editor that you want to copy and select **Copy**.
5) Right-click the new custom editor and select **Paste**.
6) Double-click the editor to view the definition of the new custom editor.
7) Enter a technical name in the **Name** field and the caption to display on the editor in the **Caption** field.

   A validation mechanism checks for correct syntax when defining a technical name. The technical name of a configuration object may not begin with an empty space nor include any of the following special characters: **\ / * ? " > < | ':**

   Furthermore, please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.

   Furthermore, if the technical name of a configuration object is changed, the name will be correctly updated in other configuration objects referencing the changed object during design time. Please note that the name of a changed configuration object will not be updated in guide pages nor if the guide pages reference a configuration object listed in the **Show Usage** functionality. If you plan to change the name of a configuration object, the reference in the guide page should be changed prior to changing the configuration object name.

8) If the editor should check the access permissions for the object managed by the editor when the user attempts to open the editor, select `True` in the **Check Rights** field. Select `False` if the editor should not check the access permissions for the object managed by the editor.

   It may be useful to set the **Check Rights** attribute to `False` in the context of a custom wizard which already verifies the access permissions and only shows objects the current user has Read/Write permissions for. Setting this attribute to `False` can improve performance. For more information about access permission concepts in Alfabet, see the

section Overview of Access Permissions for Objects in the chapter Configuring Access Permissions for Alfabet .

9) If custom help content should be available for the custom editor via the automated assistant capability, enter the URL or server variable that targets the content to display in the Assistant in the **Automated Assistant URL** attribute. For more information about configuring the automated assistant capability, see the chapter Providing Custom Online Help to the User Community.

10) You can modify the visualization of the new custom editor by adding and deleting interface controls, as needed. The options available to visualize the custom object class properties that are to be defined in the custom editor are described in detail in the section Configuring the Interface Controls in the Custom Editor.

11) In the toolbar, click the **Save** button.

Generating a Default Custom Editor

To generate a custom editor.

1) Go to the **Presentations** tab, right-click the **Custom Editors** folder, and select **Generate Custom Editor**.

2) In the dialog box that opens, select the object class for which you want to generate a custom editor and click **OK**. The new custom editor is added to the **Custom Editors** folder and is displayed in the editor in the center pane. All custom object class properties defined for the object class will be automatically distributed in the custom editor. If more than 10 custom object class properties have been defined for the object class, additional tabbed pages will be created and the custom object class properties distributed among the tabbed pages. See the section Specifying the Size of the Custom Editor. If the size of the custom editor must be altered.

3) Continue to define the basic attributes of the custom editor as described in the section Creating a New Custom Editor from Scratch.

After the custom editor is generated, you can review its visualization by right-clicking the custom editor node in the **Custom Editors** folder and selecting **Review Custom Editor.** If you are satisfied with the default layout that has been generated, you can implement the custom editor as is. Most likely, however, you will want to further refine and design the layout of the custom editor. The options available to visualize the custom object class properties that are to be defined in the custom editor are described in detail in the section Configuring the Interface Controls in the Custom Editor.

Copying an Existing Custom Editor

An existing custom editor can be copied and modified in order to create another custom editor. The custom editor may only be copied to the same class as the base object class defined for the original custom editor.

You should NOT copy an editor created for one object class and paste it to an editor created for a different object class.

1) Go to the **Presentations** tab, right-click the **Custom Editors** folder, and select **New Custom Editor** .
2) In the editor that opens, select the object class defined for the custom editor that you want to copy and click **OK**. A new custom editor is displayed in the tree.
3) Right-click the existing custom editor that you want to copy and select **Copy**.
4) Right-click the new custom editor and select **Paste**. Confirm the warning that you are replacing the content of the new custom editor.
5) Double-click the editor to view the definition of the new custom editor. You can modify the visualization of the new custom editor by adding and deleting interface controls, as needed.
6) Continue to define the basic attributes of the custom editor as described in the section Creating a New Custom Editor from Scratch.

Specifying the Size of the Custom Editor

The custom editor you create will be appended to the relevant standard editor for the object class that you specify in a class setting. The resulting size of the editor will be based on the maximum width and height defined for the standard editor and the custom editor. If the number of controls placed on the custom editor is high, you will need to add additional tabs to the custom editor. The size of static controls in custom editors will be adjusted automatically to the content to be displayed. Please note that the font and styles displayed in the editors are preconfigured by Software AG and cannot be customized.

Please note that all standard editors are designed to ensure display on a 1024 x 768 monitor screen. Because a standard editor may be implemented in a custom wizard, the height of standard editors is limited to 630 pixel. Therefore, it is highly recommended that the custom editor height does not exceed 630 pixel. The width of a custom editor should not exceed 850 pixel.

Once all of the interface controls have been added to the custom editor, you can refine the layout and design of the editor.

1) Go to the **Presentations** tab and expand the **Custom Editors** folder.
2) Right-click the custom editor that you want to define and select **Design Editor**. The custom editor is displayed in the editor in the center pane.
3) In the **Height** attribute, enter the height in pixel of the custom editor. The default height of the custom editor is 300 pixel. The value you enter should not exceed 630.
4) In the **Width** attribute, enter the width in pixel of the custom editor. The default width of the custom editor is 400 pixel. The value you enter should not exceed 850.
5) Click in the editor in the design editor to activate the **Edit** menu and **Format** menu.

- To cut, copy, paste, or delete an interface control, select the interface control, click **Edit** in the toolbar, and select the relevant option. For example, you can copy an interface control from one custom editor to another. To do so, select the interface control in the custom editor and select **Edit** > **Copy** in the main toolbar. Select **Edit** > **Paste** in the main toolbar, go to the other custom editor that is displayed in the center design pane and click in that custom editor to add the interface control to it.
- To define the alignment and spacing of the interface controls, click the relevant interface controls and select the relevant alignment or spacing options. **Format** menu with alignment and spacing options are added to the toolbar of Alfabet Expand.

- To sequence the interface controls in the editor in order to guide the user through the filter fields when using the keyboard shortcuts as well as to specify the order of hints in the custom online help, select **Format** > **Define Tab Order**. Click the blue boxes in the order that they should be sequenced.

6) In the toolbar, click the **Save** button to save your changes.

If you are dissatisfied with the layout of the custom editor you have created, you can modify the size, placement, and captions of the individual property types and interface controls. The options available to visualize the custom object class properties that are to be defined in the custom editor are described in detail in the section Configuring the Interface Controls in the Custom Editor.

Specifying the Rendering Definition of the Custom Editor

Before you begin adding interface controls to the custom editor, you should consider how you want to render the layout of the interface controls in the custom editor. Two options are possible and the method you choose will determine how you go about designing the custom editor. You can choose the traditional rendering style, which reflects the explicit design you make as you add and position the interface controls in the custom editor, or the stack rendering style which allows the interface controls to be automatically positioned at runtime in the editor based on a one- or two-column linear list. Please consider the following consequences of the different rendering styles:

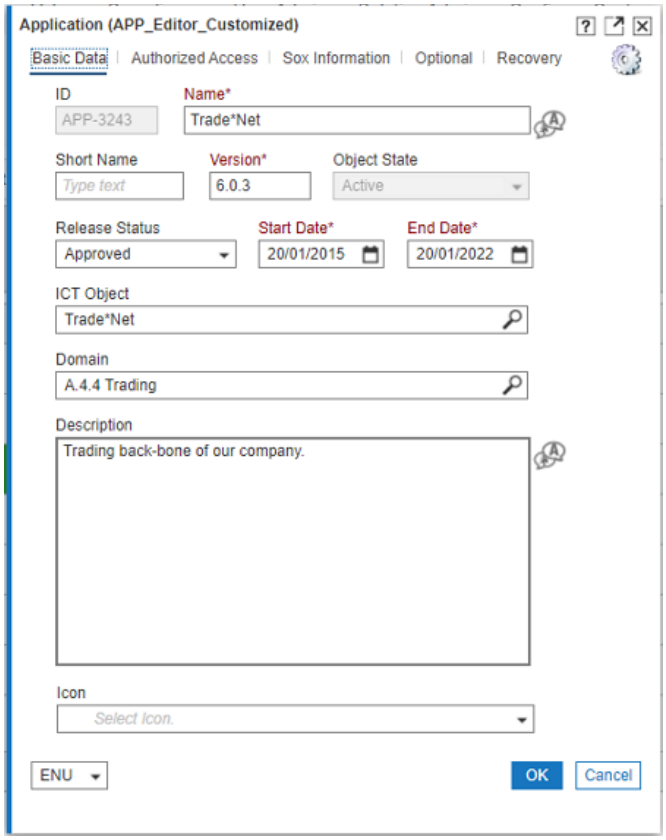


FIGURE: Editor above shows the editor with the traditional rendering style

Traditional Rendering Style: The traditional rendering of interface controls is determined by the explicit placement of the interface controls by the solution designer in the editor (WYSIWYG). In the traditional rendering style, the position, size, and alignment of the interface controls is defined by the solution designer. If

you configure conditional constraints for the interface controls and an interface control is not displayed as the result of a condition, the positioning of the visible interface controls will not be adjusted. Interface controls that are not visible due to the outcome of a conditional constraint will simply be displayed as a blank space in the editor. The traditional rendering can be reviewed when you right-click the custom editor and select **Review Custom Editor**.

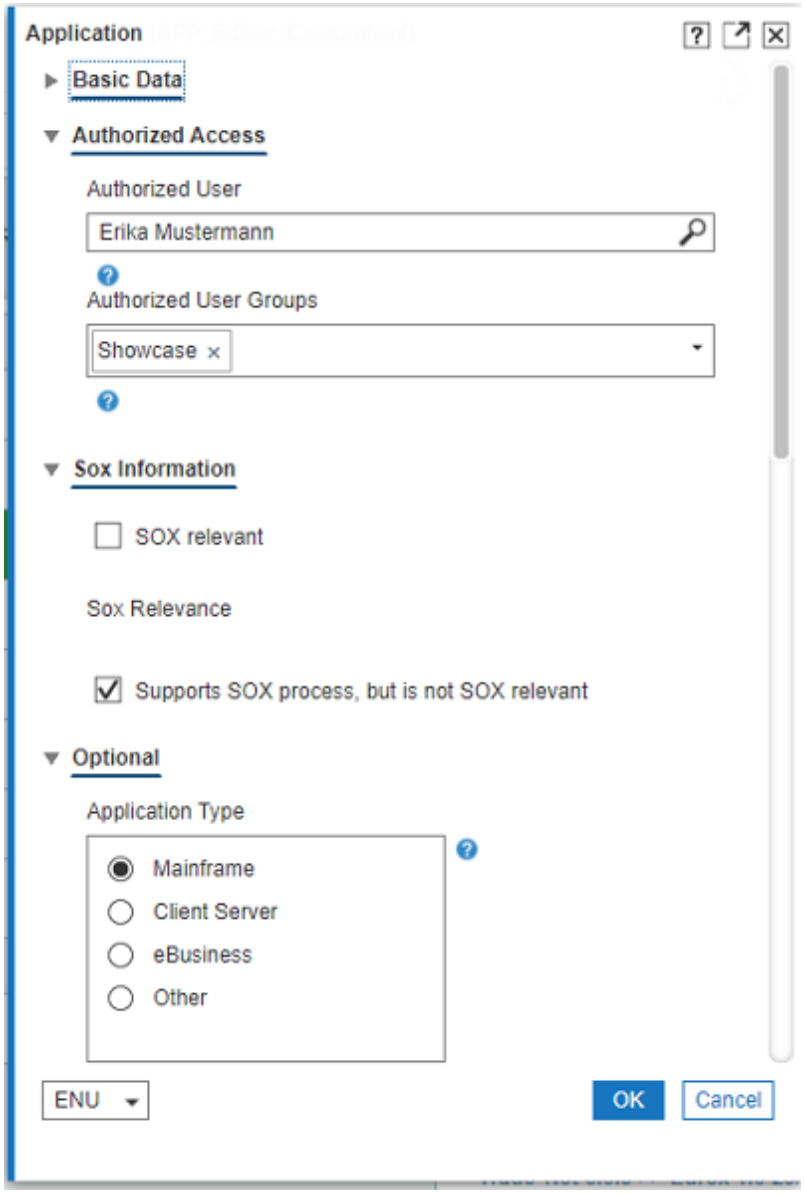


FIGURE: Editor above shows the editor with the stack rendering style

Stack Rendering Style: The stack rendering style greatly simplifies the task of designing the custom editor and provides a consistent and streamlined layout. In the stack rendering style, the interface controls are added to the custom editor but the solution designer does not need to pay attention to their placement or alignment in the editor. The placement and alignment of the interface controls is carried out at runtime. Please note however that when you add the interface controls to the custom editor, you must size them as needed for the data input. The size will not be automatically adjusted in the custom editor at runtime. Each tabbed page of the editor will be displayed as an expandable/collapsible section that, when expanded, displays the interface controls available in the tabbed page.

The stack rendering style has the advantage that if you configure conditional constraints for the interface controls and an interface control is not displayed as a result of a condition, the positioning of the visible interface controls will be automatically adjusted so that no blank spaces occur in the custom editor. Furthermore, it is also possible to group some interface controls in Group Box interface controls and maintain the explicit layout in the group box.

Please consider the following:

- **Editor Rendering Options** attributes are defined for GUI schemes and apply to all standard and custom editors in the solution. The **Editor Rendering Options** attributes are also available for class settings, allowing for the editors to be managed on a class-by-class basis. For example, stack rendering may be specified for the GUI scheme that is assigned to a user profile, but the traditional rendering style may be applied to the editors associated with the class settings for the classes Application and Component (or object class stereotypes of the classes Application and Component). If stack rendering is specified for a class setting, all standard and custom editors of the class will be displayed using the stack rendering style. Please note however that some standard editors that have buttons or presentation objects embedded (such as `APP_CopyFrom_Editor` or `APP_State_Editor`) will be rendered in the traditional layout, regardless of the **Editor Rendering Options** attributes defined for the GUI scheme (user profile) and class setting.

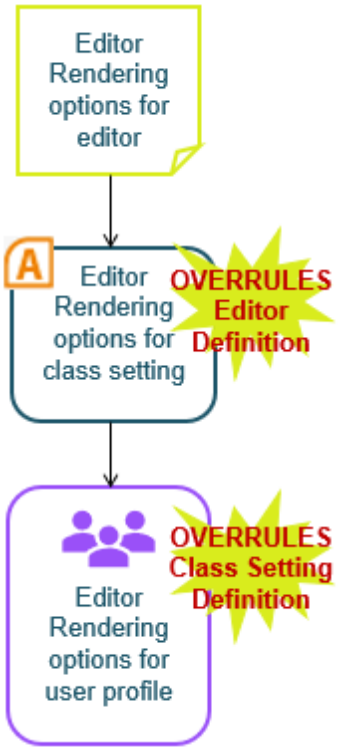


FIGURE: Precedence rules for the Editor Rendering attribute

- The **Editor Rendering Options** attributes allow you to specify whether the captions and interface controls are structured in one column that is left-aligned (as shown in the example above), or in two columns whereby the captions and interface controls are in one row with the captions in one column and the interface controls in a second column. The interface controls are ordered based on the **Tab Index** setting. You can also specify whether the help text defined in the **Hint** attributes of interface controls shall be displayed below the editor field in the custom editor or available in a pop-up that can be opened by clicking a help button next to the editor field.

- The stack rendering can be reviewed when you right-click the custom editor and select **Review Custom Editor**.
- For details about defining the GUI scheme, see the section Configuring GUI Scheme Definitions for the Alfabet Interface. For details about defining class settings, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes.

To define the rendering of standard and custom editors in your solution:

1) Go to the relevant GUI scheme below the **GUI Schemes** node in the **Presentation** tab.
2) In the **Application** section of the attribute window, expand the **Editor Rendering Options** attribute and define the following:

- **Rendering Style**: Select either **Traditional** to display standard and custom editors as the have been explicitly designed or select **Stack** to display standard and custom editors as a one- or two-columnar linear list. If you select **Stack**, define the attributes listed below.

> Please note that if you change the Rendering Style attribute, you must delete the value in the **Editor Rendering Options** field and click the **Save** button to update the attributes in the **Editor Rendering Options** section of the attribute window.

- **Stack Layout Type**: If **Stack** has been selected for the **Rendering Style** attribute: Select **One Column** if the captions and interface controls shall be displayed as a single left-aligned column in the editor. Select **Two Columns** if the captions and interface controls shall be displayed as two left-aligned columns in the editor. Please note that the sequence of the interface controls is determined by the **Tab Index** attribute defined for each interface control.
- **Preserve Layout in Group Boxes**: Select `True` if the position of interface controls in a Group Box interface control should follow a traditional rendering style and not be changed to have a stack rendering style. Select `False` if the interface controls in a Group Box interface control should have a stack rendering style.
- **Show Hint In-Line**: Select `True` if the help texts defined via the **Hint** attribute of the interface controls shall be displayed in the editor below the editor field. The hint icon specified in the **Interface Control Hint Icon** attribute will not be displayed if the **Show Hint In-Line** attribute is set to `True`. Select `False` if the help texts should not be displayed below the editor field.
- **Interface Control Hint Icon**: Displays the icon displayed to show the help texts defined for the **Hint** attribute of interface controls. It is recommended that this attribute is not changed. The hint icon will only be displayed if the **Show Hint In-Line** attribute is set to `False`.

3) Click the **Save** button to save the configuration.
4) To specify editor rendering options that are different than the editor rendering options defined for the GUI scheme, go to the relevant class setting below the **Class Settings** node in the **Presentation** tab.
5) Expand the **Editor Rendering Options** attribute and define the following:

- **Rendering Style**: Select either `Traditional` to display standard and custom editors as the have been explicitly designed or select `Stack` to display standard and custom editors as a one- or two-columnar linear list. If you select `Stack`, define the attributes listed below.

- **Stack Layout Type**: If **Stack** has been selected for the **Rendering Style** attribute: Select `OneColumn` if the captions and interface controls shall be displayed as a single left-aligned column in the editor. Select `TwoColumns` if the captions and interface controls shall be displayed as two left-aligned columns in the editor. Please note that the sequence of the interface controls is determined by the **Tab Index** attribute defined for each interface control.
- **Preserve Layout in Group Boxes**: Select `True` if the position of interface controls in a Group Box interface control should follow a traditional rendering style and not be changed to have a stack rendering style. Select `False` if the interface controls in a Group Box interface control should have a stack rendering style.
- **Show Hint In-Line**: Select `True` if the help texts defined via the **Hint** attribute of the interface controls shall be displayed in the editor below the editor field. The hint icon specified in the **Interface Control Hint Icon** attribute will not be displayed if the **Show Hint In-Line** attribute is set to `True`. Select `False` if the help texts should not be displayed below the editor field.
- **Override GUI Scheme Settings**: Select `True` if the class setting shall override the GUI scheme setting of the editor rendering options. Select `False` if the class setting shall not override the GUI scheme definition of the editor rendering options.

6) Click the **Save** button to save the configuration.

Configuring the Interface Controls in the Custom Editor

New interface controls can be added to the custom editor and edited as needed. Likewise, any interface controls that have been automatically added to the custom editor via the **Generate Custom Editor** option can also be further edited. You can determine the placement and size of the interface controls in the custom editor. Additionally, you can define the number of tabs that comprise a custom editor and choose which interface controls will appear on each tab.

If you delete a custom editor, you can recreate a default version of it again at any time. For more information, see Generating a Default Custom Editor.

To add new interface controls and design the layout of the custom editor, the custom editor must be displayed in the center pane of Alfabet Expand. To display the custom editor in the center pane:

1) Go to the **Presentations** tab and expand the **Custom Editors** folder.
2) Right-click the custom editor that you want to define and select **Design Editor**. The custom editor is displayed in the editor in the center pane. Depending on the method used to create the custom editor, the editor may be empty or may have a default layout displaying interface controls associated with custom object class properties.
3) If the number of controls that you plan to place in the custom editor is high, you will need to redefine the size of the standard custom editor by redefining its **Width** and **Height** values.
4) Above the custom editor, you will see a toolbar with the interface controls that can be added to the custom editor. Please note that not all interface controls are relevant for the configuration of custom editors.
5) Please note the following general information about the definition of interface controls:

- To add an interface control to the custom editor, click the interface control icon in the toolbar and click in the custom editor. The interface control is added to the custom editor. Depending on the interface control you select, various attributes can be defined in the attribute window.
- Each new interface control must be associated with a custom object class property via the **Property** attribute of the interface control. You must select a custom object class property that is relevant for the type of interface control (for example, a check box control is relevant for a Boolean property). For an overview of all interface controls and their corresponding property types, see the section Overview of the Interface Controls Available for Custom Editors.

6) Click the **Save**  button to save the configuration.

The following information is available:

- Overview of the Interface Controls Available for Custom Editors
- Adding Tabbed Pages to the Custom Editor
- Adding an Edit Field to the Custom Editor
- Adding a Text Box to Capture ASCII and HTML in the Custom Editor
- Adding a Text Box to Capture ASCII Format
- Adding a Text Box with an Embedded HTML Editor
- Adding a Field to Capture Dates in the Custom Editor
- Adding a Checked List Box to the Custom Editor
- Adding a Checkbox to the Custom Editor
- Adding a Combo Box to the Custom Editor
- Adding a Radio Button Group to the Custom Editor
- Adding an Edit Search Field to the Custom Editor
- Adding a Role Edit Search Field to the Custom Editor
- Adding Static Text to the Custom Editor
- Adding a List Box to Display Enumerations in the Custom Editor
- Adding Icons to the Custom Editor
- Adding an HTML Document Stored in the Internal Document Selector to the Custom Editor
- Adding a URL to the Custom Editor
- Adding HTML Text to the Custom Editor
- Adding a Color Selector to the Custom Editor
- Specifying the Interface Control To Be Mandatory
- Specifying an Editor Field To Be Non-Editable
- Specifying Conditional Constraints in the Custom Editor

Overview of the Interface Controls Available for Custom Editors

The following table lists all interface controls that can meaningfully be defined in a custom editor as well as the relevant custom object class property types that they can be associated with. Interface controls that are not relevant to custom editors are not included in the table.

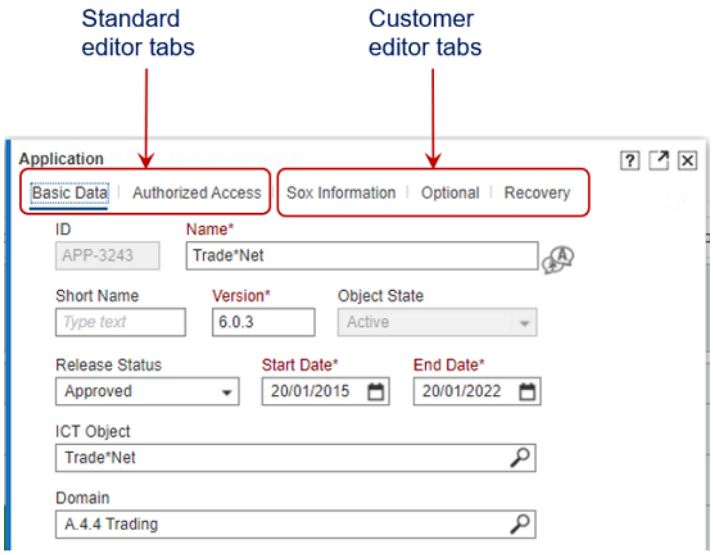
Interface Control	Description	Property Type
Flow Panel	Not relevant for custom editors.	
Table Layout Panel	Not relevant for custom editors.	
 Panel	Used to create a panel in the layout that can be used to visually frame and group a set of interface controls. The panel may have a different color than the rest of the custom editor and display a caption. Interface controls can be placed on the panel so that they appear to be grouped in the frame.	Not applicable.
 Page	Used to create a tabbed page in the custom editor. For more information, see Adding Tabbed Pages to the Custom Editor.	Not applicable.
 Static Text	Used to add static text to the customer editor in order to provide, for example, instructions to the users about capturing data. The static text interface control is not associated with a custom object class property. For more information, see Adding Static Text to the Custom Editor.	Not applicable

Interface Control	Description	Property Type
Edit Field	Used to create a one-line edit field in which users can enter data. If created for a property of type `Date` or `DateTime`, a **Calendar** icon will be displayed that opens a calendar to define a date. For more, see Adding an Edit Field to the Custom Editor and Adding a Field to Capture Dates in the Custom Editor. **NOTE**: An edit field can also display a Color button that allows users to select a color for an object's display. For more information, see the section Adding a Color Selector to the Custom Editor.	`String`, `Integer`, `Real`, `Text`, `URL`, `Date`, or `DateTime`
Memo	Used to display a large text field to capture longer texts. vertical scroll bar will be automatically displayed if the captured text exceeds the available space. For more, see Adding a Text Box to Capture ASCII and HTML in the Custom Editor.	`Text`
Check Box	Used to create a checkbox that allows the user to specify `True` (checked) or `False` (cleared) for the property. For more, see Adding a Checkbox to the Custom Editor.	`Boolean`
List Box	Used to display enumeration items in the box. Values cannot be defined via this interface control. For more information, see Adding a List Box to Display Enumerations in the Custom Editor.	Used for `String` and `Text` for which an enumeration has been defined.
Checked List Box	Used to allow multiple values to be selected. A vertical scroll bar will be automatically displayed if the list of available values (determined by the enumeration defined) is too long for the standard box. For more information, see Adding a Checked List Box to the Custom Editor.	Used for a `Text` property for which an enumeration has been defined or for a `ReferenceArray` property which will require the configuration of a query.
Checked Combo Box	Used to allow multiple values to be selected. Please note that a checked combo box is typically used as a filter field and is typically not meaningful for a custom editor.	
Combo Box	Used to create a drop-down list displaying the possible values. The user can select one value. For more information, see Adding a Combo Box to the Custom Editor.	Used for `String` and `Text` for which an enumeration has been defined or for `Reference` property which will require the configuration of a query.

Interface Control	Description	Property Type
 Radio Button Group	Used to display a list of buttons displaying the possible values. The user can select one value. For more information, see Adding a Radio Button Group to the Custom Editor.	Used for `String` and `Text` for which an enumeration has been defined or for `Reference` property which will require the configuration of a query.
 Edit Search Field	Used to display a search field search for properties of the type `Reference`. When the user clicks the **Search**  icon, a standard or custom selector will open in which the value can be selected. The user can select one value. For more information, see Adding an Edit Search Field to the Custom Editor.	`Reference`
 Role Edit Search	Used to specify the role for the relevant object directly in the custom editor rather than requiring the user to define the role in the *Responsibilities Page View*.	
 Icon	Used to display relevant icons or graphic images in the custom editor. For more information, see Adding Icons to the Custom Editor.	Not applicable.
 Group Box	Used as a visual frame that wraps a set of related interface controls. This is for purposes of visualization only. Please note it is not recommended that a Group Box interface control is set to ReadOnly. Instead, the controls included in the group box should individually be set to ReadOnly.	Not applicable.
 HTML Content	Used to add HTML text, URL, or a link that opens a document located in the **Internal Document Selector**. For more information, see Adding an HTML Document Stored in the Internal Document Selector to the Custom Editor, Adding a URL to the Custom Editor, and Adding HTML Text to the Custom Editor.	Not applicable.
Presentation Object	Not relevant for custom editors.	
Slider	Not relevant for custom editors.	

Adding Tabbed Pages to the Custom Editor

The Page interface control allows you to add tabbed pages to a custom editor.



While a custom editor can have multiple custom object class properties as well as multiple tabbed pages, for the sake of usability, it is recommended that you not create more than three tabbed pages per custom editor. When you create an automatically-generated custom editor that has more than ten custom object class properties, additional tabs are automatically created and the custom object class properties are automatically assigned to the tabs in the sequence in which they were created. If the number of controls placed on the custom editor is high, you will need to redefine the size of the standard custom editor by redefining its **Width** and **Height** attributes.

1) Go to the **Presentations** tab and expand the **Custom Editors** folder.
2) Right-click the custom editor that you want to define and select **Design Editor**. The custom editor is displayed in the editor in the center pane. Depending on the method used to create the custom editor, the editor may be empty or may have a default layout displaying interface controls associated with custom object class properties.
3) If the number of controls that you plan to place in the custom editor is high, you will need to redefine the size of the standard custom editor by redefining the **Width** and **Height** attributes.
4) Above the custom editor, you will see a toolbar with the interface controls that can be added to the custom editor. Please note that not all interface controls are relevant for the configuration of custom editors.
5) To add one or more tabs to a custom editor, click the **Page** button in the interface control toolbar and click in the editor. Add the necessary interface controls to the selected page. You may define additional tabs as needed. Alternatively, you can add all interface controls to a single tab page, and then copy and paste the interface controls to the tab pages where you want to display them.
6) In the **Caption** attribute, enter a short text that should be displayed on the tab for the Page interface control.
7) Click the **Save** button to save the configuration.

Adding an Edit Field to the Custom Editor

An Edit Field interface control allows the user to enter data in a one-line text field.

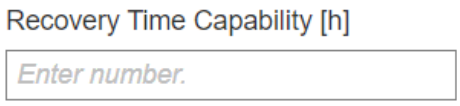


FIGURE: Edit field with placeholder text

Please consider the following:

- For properties of the type `Text`: A Memo interface control can be used instead of an Edit Field interface control to capture extensive text such is needed for **Description** properties. In this case, the text can be captured as ASCI text in a conventional text box or in HTML format using an embedded HTML editor. For more information, see the section Adding a Text Box to Capture ASCII and HTML in the Custom Editor.
- For properties of the type `Date`: An Edit Field interface control can be used to add a calendar field. For details about adding a field with a calendar, see the section Adding a Field to Capture Dates in the Custom Editor
- For properties of the type `Color`: An Edit Field interface control can be used to add a color picker. For details about adding a color picker, see the section Adding a Color Selector to the Custom Editor

To add and configure an Edit Field interface control:

1) Right-click the custom editor that you want to define and select **Design Editor**. The custom editor is displayed in the editor in the center pane.
2) In the toolbar of the design editor, click the **Edit** button and click in the editor where you want to place it. The Edit Field interface control is added to the editor.
3) Click the Edit Field interface control in the design editor to activate its attribute window. The **Control Type** attribute is automatically set to `Edit`.
4) In the **Property** attribute, specify the custom object class property of the types `String`, `Integer`, `Real`, `Text`, `URL`, or `Date` that should be captured via the interface control. The **Value Type** attribute should display either `String`, `Integer`, `Real`, `Text`, `URL`, or `Date`, depending on the property type selected in the **Property** attribute.
5) In the **Caption** attribute, enter the text that should be displayed for the Edit Field interface control. This could be the name of the custom object class property associated with the Edit Field interface control or more descriptive text.

   To use the ampersand (&) character in the caption, enter: &&

   A caption should not be more than 255 characters. If the field caption requires more characters or formatting, then you should leave the **Caption** attribute empty and create a static text interface control for which you can define the **Style** attributes. For more information, see the section Adding Static Text to the Custom Editor.
6) In the **Read-Only** attribute, select `False` if the field may be defined or select `True` if the field is read-only.
7) To provide cursory information about how to define the data, enter a short text in the **Placeholder** attribute. The text will be displayed as a hint in the text field. The text will be truncated if it is

longer than the field. The placeholder text will not be displayed if the **Read-Only** attribute is set to `True`. The display of the placeholder text can be configured via the GUI scheme attribute **Placeholders Text Color**. For more information, see the section *Overview of GUI Scheme Attributes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

8) **Validator**: In order to enforce values with a specific structure (for example, that version numbers should always be defined as two digits made up of a period and another digit), define a regular expression. Please note that specific enforcement mechanisms must be implemented for existing records or records that are imported into Alfabet from external systems. For more information about the syntax conventions for regular expressions, see http://msdn.microsoft.com/en-us/library/hs600312(VS.71).aspx

Alternatively, the **Validator** attribute may be defined for an object class property. In this case, it is enforced on **all** editors where the object class property is editable.

9) **Validator Message**: This attribute is only relevant if the **Validator** attribute is defined. Specify the message to help the user provide relevant data. If the validation message is specified, users will be prompted with the validation message if the value provided does not match the specified validator. If a validation message is not specified, an error message will state that the input value has an invalid format and will display the informational text specified in the **Hints** attribute for the interface control.

10) Specify the position of the interface control in the sequence of all interface controls. To do so, enter an integer in the **Tab Index** attribute. New interface controls will have a default value set to 0. Please note the following:

- The order that the interface controls are listed in the editor if stack layout is defined for rendering editors. For more information about stack vs. traditional rendering style for editors, see the section Specifying the Rendering Definition of the Custom Editor.
- The order that is used when the user moves the focus in the custom editor via the TAB key. The correct definition of tabbing is especially critical in order to provide barrier-free accessibility. For more information, see the section Configuring Barrier-Free User Profiles.
- The sequence that the help texts defined via the **Hint** attribute for an interface are listed in the modal Help window.

11) If a help text is required for the interface control, enter the text in the **Hint** attribute. If the **Hint** attribute is defined, the help text will be available in the modal Help window that opens when the editor's Help button is clicked. If stack rendering style is defined for editors, the help texts may be displayed below the editor field or via a Help button next to the editor field. For more information about stack rendering for editors, see the section Specifying the Rendering Definition of the Custom Editor.

12) In the toolbar, click the **Save**  button to save your changes.

Adding a Text Box to Capture ASCII and HTML in the Custom Editor

Custom object class properties of the type `Text` are usually configured to capture extensive comments or descriptions about an object. The Memo interface control provides a text box that can be sized as needed and will automatically display a vertical scroll bar if the captured text exceeds the available space. Per default, the text will be captured in ASCII format. Alternatively, you can configure the text box to embed an HTML editor so that users can capture the text in HTML format.

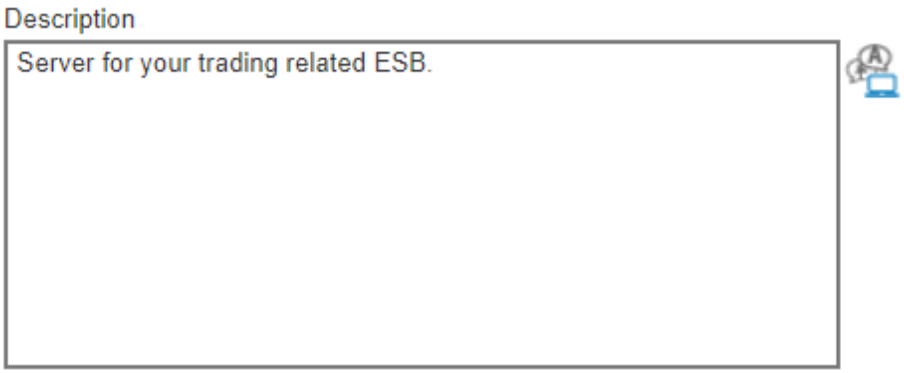


FIGURE: Text field to capture ASCII format

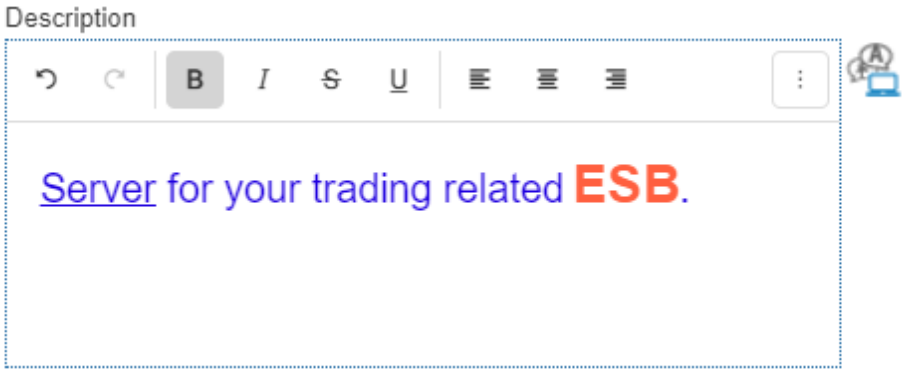


FIGURE: Text field with embedded HTML editor

The following information is available:

- Adding a Text Box to Capture ASCII Format
- Adding a Text Box with an Embedded HTML Editor

Adding a Text Box to Capture ASCII Format

A Memo interfaced control can be created for a property to capture text in ASCII format. This is the default format if no further configuration to enable HTML content is specified.

To add a Memo interface control to the custom editor:

1) Right-click the custom editor that you want to define and select **Design Editor**. The custom editor is displayed in the editor in the center pane.
2) In the toolbar of the design editor, click the **Memo** button and click in the editor where you want to place it. The Memo interface control is added to the editor.
3) Click the Memo interface control in the design editor to activate its attribute window. The **Control Type** attribute is automatically set to `Memo`.
4) In the **Property** attribute, specify the custom object class property of the type `Text` that should be captured via the interface control. The **Value Type** attribute should display `Text`.

5) In the **Caption** attribute, enter text for the caption that should be displayed for the Memo interface control. This could be the name of the object class property associated with the Memo interface control or more descriptive text.

To use the ampersand (&) character in the caption, enter: &&

A caption should not be more than 255 characters. If the field caption requires more characters or formatting, then you should leave the **Caption** attribute empty and create a static text interface control for which you can define the **Style** attributes. For more information, see the section Adding Static Text to the Custom Editor.

6) Specify the position of the interface control in the sequence of all interface controls. To do so, enter an integer in the **Tab Index** attribute. New interface controls will have a default value set to 0. Please note the following:

- The order that the interface controls are listed in the editor if stack layout is defined for rendering editors. For more information about stack vs. traditional rendering style for editors, see the section Specifying the Rendering Definition of the Custom Editor.
- The order that is used when the user moves the focus in the custom editor via the TAB key. The correct definition of tabbing is especially critical in order to provide barrier-free accessibility. For more information, see the section Configuring Barrier-Free User Profiles.
- The sequence that the help texts defined via the **Hint** attribute for an interface are listed in the modal Help window.

7) If a help text is required for the interface control, enter the text in the **Hint** attribute. If the **Hint** attribute is defined, the help text will be available in the modal Help window that opens when the editor's Help button is clicked. If stack rendering style is defined for editors, the help texts may be displayed below the editor field or via a Help button next to the editor field. For more information about stack rendering for editors, see the section Specifying the Rendering Definition of the Custom Editor.

8) To provide information about how to define the data, enter a text in the **Placeholder** attribute. The text will be displayed as a hint in the search field. Linebreaks can be configured in placeholder texts in order to structure long placeholder texts in Memo interface controls in editors. The display of the placeholder text can be configured via the GUI scheme attribute **Placeholders Text Color**. For more information, see the section *Overview of GUI Scheme Attributes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

9) In the toolbar, click the **Save**  button to save your changes.

Adding a Text Box with an Embedded HTML Editor

The following configuration is required to implement a HTML editor for the Memo interface control:

The **Can Have HTML Content** attribute must be set to `True` for the relevant property of type `Text`. For more information, see the section Configuring Custom Properties of the Type Text.

If the HTML editor is enabled and the **Enable Data Translation** attribute is set to `True`for the custom property, the XML attribute `TranslateContentExceedingHTMLLengthLimit` must be configured in the XML object ***AlfaTranslationServicesConfig***. For more information about

configuring the XML object ***AlfaTranslationServicesConfig***, see the section Configuring the Connection to the Translation Service

To add a Memo interface control to the custom editor:

1) Right-click the custom editor that you want to define and select **Design Editor**. The custom editor is displayed in the editor in the center pane.

2) In the toolbar of the design editor, click the **Memo** button and click in the editor where you want to place it. The Memo interface control is added to the editor.

3) Click the Memo interface control in the design editor to activate its attribute window. The **Control Type** attribute is automatically set to `Memo`. Define the following attributes:

- **Property**: Specify the custom object class property of the type `Text` that should be captured via the interface control. The **Value Type** attribute should display `Text`.
- **Caption**: Enter text for the caption that should be displayed for the Memo interface control. This could be the name of the object class property associated with the Memo interface control or more descriptive text.

  To use the ampersand (&) character in the caption, enter: &&

  A caption should not be more than 255 characters. If the field caption requires more characters or formatting, then you should leave the **Caption** attribute empty and create a static text interface control for which you can define the **Style** attributes. For more information, see the section Adding Static Text to the Custom Editor.

- **Enable HTML Content**: Set to `True` to display the HTML editor for the text field. If the **HTML Content** attribute has been set to `True` for the Memo interface control, the **Enable HTML Content** attribute will be automatically set to `True` for the property in the **Attributes** section of the standard and custom object profiles as well as for the Value Control interface control in an object cockpit. The **Enable HTML Content** attribute can be set to `False` for either the property in the custom object profile or the Value Control interface control if needed.

4) In the toolbar, click the **Save** button to save your changes.

5) Next, it is necessary to copy and generate the custom editors as HTF editors so that the Memo interface controls will be correctly sized and that the other interface controls will be retrofitted to the width of the embedded HTML editor. This is done for all editors containing Memo interface controls for either a specific object class or for all relevant object classes with Memo interface controls configured in the associated standard and custom editors. Go to the **Meta-Model** tab and navigate to the class node of the class that the custom editor has been created for (the class selected in the **Select Class** editor when the custom editor was created).

6) Right-click the object class and select **Generate HTF Editors**. Confirm the info message about how many HTF editors have been created. This number will include all custom editors with Memo interface controls as well as any private editors with Memo interface controls for the selected class. Please note the following:

- The original custom editors have been copied below the **HTF** node in the **Custom Editors** folder and appended with the suffix `_HTF`.
- Private editors as well as editors configured to replace standard editors will be displayed below the **HTF** node in the **Editors** folder. Editors configured to replace standard editors may be updated with HTF editors as described below.

Alternatively you can generate HTF editors for all relevant object classes with Memo interface controls configured in the associated standard and custom editors. Go to the **Presentation** tab and right-click **Generate HTF Editors**. Confirm the info message about how many HTF editors have been created.

7) Check the usage of the original custom editor in class settings, object views, view schemes, wizards, and workflows in order to decide whether to update the copied HTF editor to those usages. In the **Presentation** tab, expand the **Custom Editors** folder to view the generated HTF editors. Right-click the relevant HTF editor and select **Show Usage for Original Custom Editor**. The **Objects Usage** editor displays all usages of the original custom editor. Click **Close** to close the editor.

8) To replace the original custom editor in the class settings, object views, view schemes, wizards, and workflows displayed in the **Objects Usage** editor with the _HTF editor:

- To automatically update all relevant public class settings, object views, view schemes, wizards, and workflows: Right-click the HTF editor and select **Update All Possible Usages for Original Editor**. Confirm the info message about how many usages have been updated. The HTF editors will be specified instead of the original custom editor for the relevant class settings, object views, view schemes, wizards, and workflows. Please note that only public class settings, object views, view schemes, wizards, and workflows will be updated. Private configuration objects preconfigured by Software AG will not be updated.
- To manually update specific class settings, object views, view schemes, wizard, and workflows, navigate to the relevant configuration object listed in the **Objects Usage** editor and specify the relevant HTF editor as needed.

9) In the toolbar, click the **Save** button to save your changes.

Adding a Field to Capture Dates in the Custom Editor

If a custom object class property of the type `Date` is to be captured in the custom editor, an Edit Field interface control should be added to the custom editor. A calendar button will be automatically added to any edit field for which the **Value Type** attribute is set to `Date`. Users can enter a date directly in the edit field or click the **Calendar** button to open the calendar and select the date.

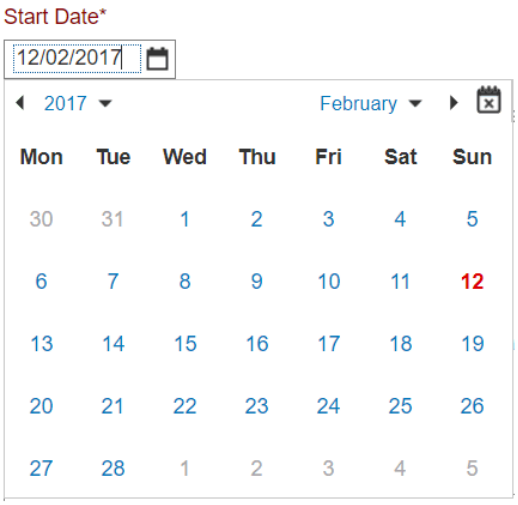


FIGURE: Clicking Calendar button opens date picker

A custom object class property of the type `DateTime` can also be captured in an Edit Field interface control. However, in contrast to an object class property of the type `Date`, a **Calendar** button is not available for properties of the type `DateTime`. If the edit field is associated with a custom object class property of the type `DateTime`, the user must explicitly enter the date and time in the appropriate format in the edit field.

The format used to display and capture the date and time information on the user interface is determined by the cultures configured by your enterprise for the Alfabet solution. For more information about configuring cultures, see the section Specifying the Cultures Relevant to Your Enterprise in the chapter Localization and Multi-Language Support for the Alfabet Interface.

To add a Edit Field interface control with a calendar button to the custom editor:

1) Right-click the custom editor that you want to define and select **Design Editor**. The custom editor is displayed in the editor in the center pane.

2) In the toolbar of the design editor, click the **Edit** button and click in the editor where you want to place it. The Edit Field interface control is added to the editor.

3) Click the Edit Field interface control in the design editor to activate its attribute window. The **Control Type** attribute is automatically set to `Edit`.

4) In the **Property** attribute, specify the custom object class property o the type `Date` or `DateTime` that should be captured via the interface control. The **Value Type** attribute should display `Date` or `DateTime`.

5) In the **Caption** attribute, enter the text that should be displayed for the Edit Field interface control. This could be the name of the object class property associated with the Edit Field interface control or more descriptive text.

   To use the ampersand (&) character in the caption, enter: &&

   A caption should not be more than 255 characters. If the field caption requires more characters or formatting, then you should leave the **Caption** attribute empty and create a static text interface control for which you can define the **Style** attributes. For more information, see the section Adding Static Text to the Custom Editor.

6) Specify the position of the interface control in the sequence of all interface controls. To do so, enter an integer in the **Tab Index** attribute. New interface controls will have a default value set to 0. Please note the following:

- The order that the interface controls are listed in the editor if stack layout is defined for rendering editors. For more information about stack vs. traditional rendering style for editors, see the section Specifying the Rendering Definition of the Custom Editor.
- The order that is used when the user moves the focus in the custom editor via the TAB key. The correct definition of tabbing is especially critical in order to provide barrier-free accessibility. For more information, see the section Configuring Barrier-Free User Profiles.
- The sequence that the help texts defined via the **Hint** attribute for an interface are listed in the modal Help window.

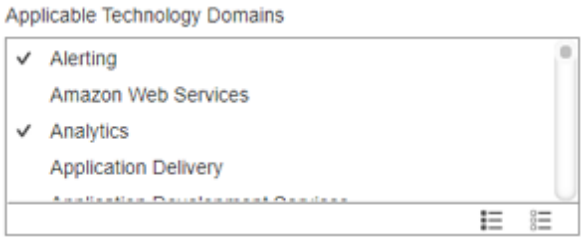
7) If a help text is required for the interface control, enter the text in the **Hint** attribute. If the **Hint** attribute is defined, the help text will be available in the modal Help window that opens when the editor's Help button is clicked. If stack rendering style is defined for editors, the help texts may be

displayed below the editor field or via a Help button next to the editor field. For more information about stack rendering for editors, see the section Specifying the Rendering Definition of the Custom Editor.

8) In the toolbar, click the **Save** button to save your changes. Please note that the **Calendar** button will not be displayed in the **Design Editor** mode that you are working in. However, if you right-click the custom editor and select **Review Custom Editor**, the **Calendar** button will be visible and can be clicked in order to open it.

Adding a Checked List Box to the Custom Editor

A Checked List Box interface control box allows a user to select multiple values by setting a checkmark next to each relevant value. The Checked List Box interface control automatically displays a vertical scroll bar if the list of available values (determined by the enumeration defined) is too long for the standard box.



The Checked List Box interface control can be defined for custom object class properties of the type `StringArray` for which an enumeration has been defined, or for custom object class properties of the type `ReferenceArray`, which will require the additional configuration of an Alfabet query or native SQL query.

If the editor is rendered in a stacked layout, an auto-complete function may be available if the checked list box field has a significant number of entries. Once a letter is entered in the search field, the auto-complete function will display a list of matching text. Users can hit the ENTER key or select the object in the drop-down menu to add it to the editor field. The editor field will increase in size to display all selected objects in the field. An X will be displayed next to each object to allow it to be cleared and removed from the editor field. The auto-complete function can be specified for checked list box editor fields in custom editors rendered in a stacked layout by setting the **Use Multi-Edit Search** attribute for the Checked List Box interface control to `True`. For more information about rendering editors in a stacked layout, see the section Configuring Styles and GUI Scheme Settings for Standard and Custom Editors.

To add and configure a checked list box:

1) Right-click the custom editor that you want to define and select **Design Editor**. The custom editor is displayed in the editor in the center pane.
2) In the toolbar of the design editor, click the **Checked List Box** button and click in the editor where you want to place it. The Checked List Box interface control is added to the editor.
3) Click the Checked List Box interface control in the design editor to activate its attribute window. The **Control Type** attribute is automatically set to `CheckedListBox`.
4) In the **Caption** attribute, enter the text that should be displayed for the Checked List Box interface control. This could be the name of the custom object class property associated with the Checked List Box interface control or more descriptive text.

To use the ampersand (&) character in the caption, enter: &&

A caption should not be more than 255 characters. If the field caption requires more characters or formatting, then you should leave the **Caption** attribute empty and create a static text interface control for which you can define the **Style** attributes. For more information, see the section Adding Static Text to the Custom Editor.

5) For properties of the type `StringArray`:

- In the **Property** attribute, specify the custom object class property of the type `StringArray` that should be captured via the interface control. The custom object class property must have an enumeration defined. The **Value Type** attribute should display `StringArray`. The **SubType** attribute should display `Enum`.

6) For properties of the type `ReferenceArray`:

- In the **Property** attribute, specify the custom object class property of the type `ReferenceArray` that should be captured via the interface control. The custom object class property must have an enumeration defined. The **Value Type** attribute should display `ReferenceArray`.
- In the **SubType** attribute, select `SqlEnum`.
- In the **Range** attribute, enter the Alfabet query or native SQL query to find the relevant objects for the `ReferenceArray`. The information about the object displayed in the caption of the respective entry in the checked list box is defined via the `SHOW` properties of the Alfabet query or the `SELECT` statement of the native SQL query. For details about defining a query, see the chapter Defining Queries.

In this example, a principle shall be defined for a project. A custom object class property of the type `ReferenceArray` has been created for the object class `Project`. The class `Principle` is defined in the **Type Info** attribute for the custom object class property. The following Alfabet query is specified in the **Range** attribute for the `Checked List Box` interface control:

```
Alfabet_QUERY_500
FIND Principle
SHOW Principle.Name
SORT Principle.Name
```

The resulting entries in the Checked List Box interface control display the name of the principles found by the Alfabet query via the `SHOW` property.

7) Specify the position of the interface control in the sequence of all interface controls. To do so, enter an integer in the **Tab Index** attribute. New interface controls will have a default value set to 0. Please note the following:

- The order that the interface controls are listed in the editor if stack layout is defined for rendering editors. For more information about stack vs. traditional rendering style for editors, see the section Specifying the Rendering Definition of the Custom Editor.
- The order that is used when the user moves the focus in the custom editor via the TAB key. The correct definition of tabbing is especially critical in order to provide barrier-free accessibility. For more information, see the section Configuring Barrier-Free User Profiles.
- The sequence that the help texts defined via the **Hint** attribute for an interface are listed in the modal Help window.

8) If a help text is required for the interface control, enter the text in the **Hint** attribute. If the **Hint** attribute is defined, the help text will be available in the modal Help window that opens when the editor's Help button is clicked. If stack rendering style is defined for editors, the help texts may be displayed below the editor field or via a Help button next to the editor field. For more information about stack rendering for editors, see the section Specifying the Rendering Definition of the Custom Editor.

9) In the toolbar, click the **Save** button to save your changes.

Adding a Checkbox to the Custom Editor

The Check Box interface control allows the user to define a custom object class property of the type `Boolean`. A user can click the checkbox to set a checkmark (= `True`) or clear the checkbox (= `False`).

Please note the following regarding the checkbox for a custom object class property of the type `Boolean`.

- A user will not be able to differentiate between a checkbox for which no value has been set (= `NULL`) vs. a checkbox that has been cleared (= `False`). Therefore, it is highly recommended that a default value be configured for a custom object class property of the type `Boolean` in order to ensure that the object class property is either `True` or `False`. For more information about configuring the default value, see the section Configuring Custom Properties of the Type Boolean in the chapter Configuring the Class Model.
- If the **Read Only** attribute is set to `True` for a `CheckBox` interface control, the checkbox will be disabled and the value defined in the **Hint** attribute of the associated custom object class property of the type `Boolean` cannot be displayed as a tooltip. It is recommended that you add the tooltip as static text to the editor if it should be visible for a non-editable checkbox. For more information about defining a non-editable checkbox displaying a default value, see the section Specifying an Editor Field To Be Non-Editable.

To add a Check Box interface control to the custom editor:

1) Right-click the custom editor that you want to define and select **Design Editor**. The custom editor is displayed in the editor in the center pane.
2) In the toolbar of the design editor, click the **Check Box** button and click in the editor where you want to place it. The Check Box interface control is added to the editor.
3) Click the Check Box interface control in the design editor to activate its attribute window. The **Control Type** attribute is automatically set to `CheckBox`.
4) In the **Property** attribute, specify the custom object class property of the type `Boolean` that should be captured via the interface control. The **Value Type** attribute should display `Boolean`.
5) In the **Caption** attribute, enter the text that should be displayed for the checkbox interface control. This could be the name of the object class property associated with the Check Box interface control or more descriptive text.

To use the ampersand (&) character in the caption, enter: &&

A caption should not be more than 255 characters. If the field caption requires more characters or formatting, then you should leave the **Caption** attribute empty and create a static text interface control for which you can define the **Style** attributes. For more information, see the section Adding Static Text to the Custom Editor.

6) **Validator**: In order to enforce values with a specific structure (for example, that version numbers should always be defined as two digits made up of a period and another digit), define a regular expression. Please note that specific enforcement mechanisms must be implemented for existing records or records that are imported into Alfabet from external systems. For more information about the syntax conventions for regular expressions, see http://msdn.microsoft.com/en-us/library/hs600312(VS.71).aspx

Alternatively, the **Validator** attribute may be defined for an object class property. In this case, it is enforced on **all** editors where the object class property is editable.

7) **Validator Message**: This attribute is only relevant if the **Validator** attribute is defined. Specify the message to help the user provide relevant data. If the validation message is specified, users will be prompted with the validation message if the value provided does not match the specified validator. If a validation message is not specified, an error message will state that the input value has an invalid format and will display the informational text specified in the **Hints** attribute for the interface control.

8) Specify the position of the interface control in the sequence of all interface controls. To do so, enter an integer in the **Tab Index** attribute. New interface controls will have a default value set to 0. Please note the following:

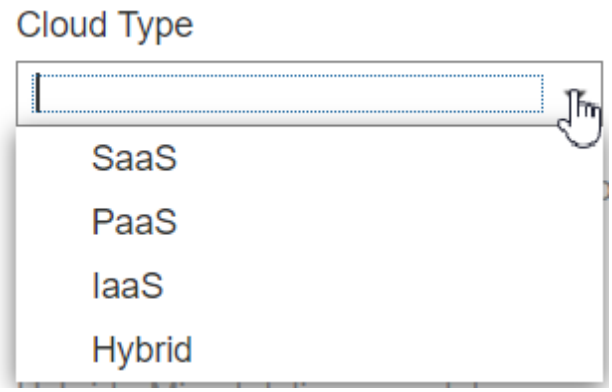
- The order that the interface controls are listed in the editor if stack layout is defined for rendering editors. For more information about stack vs. traditional rendering style for editors, see the section Specifying the Rendering Definition of the Custom Editor.
- The order that is used when the user moves the focus in the custom editor via the TAB key. The correct definition of tabbing is especially critical in order to provide barrier-free accessibility. For more information, see the section Configuring Barrier-Free User Profiles.
- The sequence that the help texts defined via the **Hint** attribute for an interface are listed in the modal Help window.

9) If a help text is required for the interface control, enter the text in the **Hint** attribute. If the **Hint** attribute is defined, the help text will be available in the modal Help window that opens when the editor's Help button is clicked. If stack rendering style is defined for editors, the help texts may be displayed below the editor field or via a Help button next to the editor field. For more information about stack rendering for editors, see the section Specifying the Rendering Definition of the Custom Editor.

10) In the toolbar, click the **Save** button to save your changes.

Adding a Combo Box to the Custom Editor

A Combo Box interface control provides a drop-down field that displays a list of values that can be selected for a custom object class property. The user can select only one value for the custom object class property. The first line of the combo box may be blank if no value may be defined.



Users can click the arrow to open the drop-down list or begin to type in the name of the object they are searching for. The auto-complete function will display a list of matching text. Please note that the auto-complete capability will not be implemented for a user profile for which the **Use WAI-ARIA** attribute is set to `True`.

The Combo Box interface control can be defined for custom object class properties of the type `String` or `Text` for which an enumeration has been defined, or for custom object class properties of the type `Reference`. However, this requires the additional configuration of an Alfabet query or a native SQL query.

To add a Combo Box interface control to the custom editor:

1) Right-click the custom editor that you want to define and select **Design Editor**. The custom editor is displayed in the editor in the center pane.
2) In the toolbar of the design editor, click the **Combo Box** interface control and click in the editor where you want to place it. The Combo Box interface control is added to the editor.
3) Click the Combo Box interface control in the editor to open its attribute window. The **Control Type** attribute is automatically set to `ComboBox`.
4) For properties of the type `String` or `Text` that have an enumeration defined: In the **Property** attribute, specify the custom object class property of the type `String` or `Text` that should be captured via the interface control. The custom object class property must have an enumeration defined. The **Value Type** attribute should display either `String` or `Text`.
5) For properties of the type `Reference`:

- In the **Property** attribute, specify the custom object class property of the type `Reference` that should be captured via the interface control. The custom object class property must have an enumeration defined. The **Value Type** attribute should display `Reference`.
- In the **SubType** attribute, select `SqlEnum`.
- In the **Range** attribute, enter the Alfabet query or native SQL query to find the relevant objects for the `Reference`. The information about the object displayed in the caption of the respective entry in the combo box is defined via the `SHOW` properties of the Alfabet query or the `SELECT` statement of the native SQL query. For details about defining a query, see the chapter Defining Queries.

In this example, a project group shall be defined as the approved scenario for a selected project portfolio (also a project group). A custom object class property of the type `Reference` has been created for the object class `ProjectGroup`. The class `ProjectGroup` is defined in the **Type Info** attribute for the custom object class property. The following native SQL query is specified in the **Range** attribute for the `Combo Box` interface control:

```
SELECT pg.REFSTR, pg.NAME
FROM PROJECTGROUP pg
WHERE pg.BELONGSTO = @BASE
ORDER BY pg.NAME
```

The resulting entries in the combo box display the name of the project groups found by the native SQL query via the `NAME` property in the `SELECT` statement. Please note that the `REFSTR` of the object must be defined as the first argument in the `SELECT` statement for technical reasons and is not displayed in the radio button caption.

6) In the **Caption** attribute, enter the text that should be displayed for the radio button group. This could be the name of the object class property associated with the combo box or more descriptive text.

   To use the ampersand (&) character in the caption, enter: &&

   A caption should not be more than 255 characters. If the field caption requires more characters or formatting, then you should leave the **Caption** attribute empty and create a static text interface control for which you can define the **Style** attributes. For more information, see the section Adding Static Text to the Custom Editor.

7) To provide cursory information about how to define the data, enter a short text in the **Placeholder** attribute. The text will be displayed as a hint in the combo box field. The text will be truncated if it is longer than the field. The display of the placeholder text can be configured via the GUI scheme attribute **Placeholders Text Color**. For more information, see the section *Overview of GUI Scheme Attributes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

8) **Validator**: In order to enforce values with a specific structure (for example, that version numbers should always be defined as two digits made up of a period and another digit), define a regular expression. Please note that specific enforcement mechanisms must be implemented for existing records or records that are imported into Alfabet from external systems. For more information about the syntax conventions for regular expressions, see http://msdn.microsoft.com/en-us/library/hs600312(VS.71).aspx

   Alternatively, the **Validator** attribute may be defined for an object class property. In this case, it is enforced on **all** editors where the object class property is editable.

9) **Validator Message**: This attribute is only relevant if the **Validator** attribute is defined. Specify the message to help the user provide relevant data. If the validation message is specified, users will be prompted with the validation message if the value provided does not match the specified validator. If a validation message is not specified, an error message will state that the input value has an invalid format and will display the informational text specified in the **Hints** attribute for the interface control.

10) Specify the position of the interface control in the sequence of all interface controls. To do so, enter an integer in the **Tab Index** attribute. New interface controls will have a default value set to 0. Please note the following:

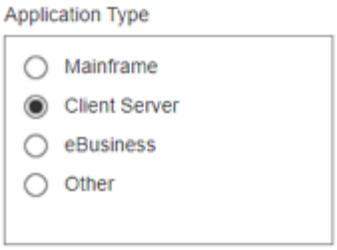
- The order that the interface controls are listed in the editor if stack layout is defined for rendering editors. For more information about stack vs. traditional rendering style for editors, see the section Specifying the Rendering Definition of the Custom Editor.
- The order that is used when the user moves the focus in the custom editor via the TAB key. The correct definition of tabbing is especially critical in order to provide barrier-free accessibility. For more information, see the section Configuring Barrier-Free User Profiles.
- The sequence that the help texts defined via the **Hint** attribute for an interface are listed in the modal Help window.

11) If a help text is required for the interface control, enter the text in the **Hint** attribute. If the **Hint** attribute is defined, the help text will be available in the modal Help window that opens when the editor's Help button is clicked. If stack rendering style is defined for editors, the help texts may be displayed below the editor field or via a Help button next to the editor field. For more information about stack rendering for editors, see the section Specifying the Rendering Definition of the Custom Editor.

12) In the toolbar, click the **Save** button to save your changes.

Adding a Radio Button Group to the Custom Editor

A Radio Button Group interface control provides a list of values that can be selected for a custom object class property by clicking a button. The user can select only one value for the custom object class property.



The Radio Button Group interface control can be defined for custom object class properties of the type `String` or `Text` for which an enumeration has been defined, or for custom object class properties of the type `Reference`, which will require the additional configuration of a query. You can configure either an Alfabet query or a native SQL query.

A Radio Button Group interface control is appropriate if only a few values are to be displayed for selection in the custom editor. If an excessive number of objects are expected to be found by the query defined for properties of the type `Reference`, you should define a `ComboBox` interface control where the query results can be listed in a drop-down field. For information about the configuration of a query for a `ComboBox` interface control for custom object class properties of the type `Reference`, see the section Adding a Combo Box to the Custom Editor.

To add a Radio Button Group interface control to the custom editor:

1) Right-click the custom editor that you want to define and select **Design Editor**. The custom editor is displayed in the editor in the center pane.

2) In the toolbar of the design editor, click the **Radio Button Group**  interface control and click in the editor where you want to place it. The Radio Button Group interface control is added to the editor.

3) Click the Radio Button Group interface control in the design editor to activate its attribute window. The **Control Type** attribute is automatically set to `RadioButtonGroup`.

4) For properties of the type `String` or `Text` that have an enumeration defined: In the **Property** attribute, specify the custom object class property of the type `String` or `Text` that should be captured via the interface control. The custom object class property must have an enumeration defined. The **Value Type** attribute should display either `String` or `Text`.

5) For properties of the type `Reference`:

- In the **Property** attribute, specify the custom object class property of the type `Reference` that should be captured via the interface control. The custom object class property must have an enumeration defined. The **Value Type** attribute should display `Reference`.
- In the **SubType** attribute, select `SqlEnum`.
- In the **Range** attribute, enter the Alfabet query or native SQL query to find the relevant objects for the `Reference`. The information about the object displayed in the caption of the respective radio button is defined via the `SHOW` properties of the Alfabet query or the `SELECT` statement of the native SQL query. For details about defining a query, see the chapter Defining Queries.

In this example, a project group shall be defined as the approved scenario for a selected project portfolio (also a project group). A custom object class property of the type `Reference` has been created for the object class `ProjectGroup`. The class `ProjectGroup` is defined in the **Type Info** attribute for the custom object class property. The following native SQL query is specified in the **Range** attribute for the `Radio Group Button` interface control:

```
SELECT pg.REFSTR, pg.NAME
FROM PROJECTGROUP pg
WHERE pg.BELONGSTO = @BASE
ORDER BY pg.NAME
```

The resulting radio buttons display the name of the project groups found by the native SQL query via the `NAME` property in the `SELECT` statement. Please note that the `REFSTR` of the object must be defined as the first argument in the `SELECT` statement for technical reasons and is not displayed in the radio button caption.

6) You should resize the radio button group interface control to ensure that all enumeration values or objects returned by the query will be visible. To do so, grab a handle on the radio button group interface control and drag it to the correct size.

7) In the **Caption** attribute, enter the text that should be displayed for the radio button group. This could be the name of the object class property associated with the radio button group or more descriptive text.

To use the ampersand (&) character in the caption, enter: &&

A caption should not be more than 255 characters. If the field caption requires more characters or formatting, then you should leave the **Caption** attribute empty and create

a static text interface control for which you can define the **Style** attributes. For more information, see the section Adding Static Text to the Custom Editor.

8) **Validator**: In order to enforce values with a specific structure (for example, that version numbers should always be defined as two digits made up of a period and another digit), define a regular expression. Please note that specific enforcement mechanisms must be implemented for existing records or records that are imported into Alfabet from external systems. For more information about the syntax conventions for regular expressions, see http://msdn.microsoft.com/en-us/library/hs600312(VS.71).aspx

Alternatively, the **Validator** attribute may be defined for an object class property. In this case, it is enforced on **all** editors where the object class property is editable.

9) **Validator Message**: This attribute is only relevant if the **Validator** attribute is defined. Specify the message to help the user provide relevant data. If the validation message is specified, users will be prompted with the validation message if the value provided does not match the specified validator. If a validation message is not specified, an error message will state that the input value has an invalid format and will display the informational text specified in the **Hints** attribute for the interface control.

10) Specify the position of the interface control in the sequence of all interface controls. To do so, enter an integer in the **Tab Index** attribute. New interface controls will have a default value set to 0. Please note the following:

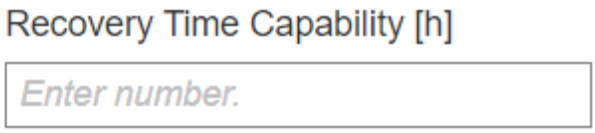
- The order that the interface controls are listed in the editor if stack layout is defined for rendering editors. For more information about stack vs. traditional rendering style for editors, see the section Specifying the Rendering Definition of the Custom Editor.
- The order that is used when the user moves the focus in the custom editor via the TAB key. The correct definition of tabbing is especially critical in order to provide barrier-free accessibility. For more information, see the section Configuring Barrier-Free User Profiles.
- The sequence that the help texts defined via the **Hint** attribute for an interface are listed in the modal Help window.

11) If a help text is required for the interface control, enter the text in the **Hint** attribute. If the **Hint** attribute is defined, the help text will be available in the modal Help window that opens when the editor's Help button is clicked. If stack rendering style is defined for editors, the help texts may be displayed below the editor field or via a Help button next to the editor field. For more information about stack rendering for editors, see the section Specifying the Rendering Definition of the Custom Editor.

12) In the toolbar, click the **Save**  button to save your changes.

Adding an Edit Search Field to the Custom Editor

An Edit Search Field interface control displays an edit field with a **Search**  icon that is automatically placed to the right of the field. When the user clicks the **Search**  icon, a standard or custom selector will open in which the value can be selected.



An Edit Search Field interface control can be defined for custom object class properties of the type `Reference`.

For more information about the configuration of custom selectors, see the section Configuring a Custom Selector for Search Functionalities.

To add an Edit Search Field interface control to the custom editor:

1) Right-click the custom editor that you want to define and select **Design Editor**. The custom editor is displayed in the editor in the center pane.

2) In the toolbar of the design editor, click the **Edit Search** button and click in the editor where you want to place it. The edit search interface control is added to the editor.

3) Click the Edit Search Field interface control in the design editor to activate its attribute window. The **Control Type** attribute is automatically set to `EditSearch`.

4) In the **Caption** attribute, enter the text that should be displayed for the edit search interface control. This could be the name of the object class property associated with the Edit Search Field interface control or more descriptive text.

   To use the ampersand (&) character in the caption, enter: &&

   A caption should not be more than 255 characters. If the field caption requires more characters or formatting, then you should leave the **Caption** attribute empty and create a static text interface control for which you can define the **Style** attributes. For more information, see the section Adding Static Text to the Custom Editor.

5) In the **Property** attribute, specify the custom object class property of the type `Reference` that should be captured via the interface control. The **Value Type** attribute should display `Reference`.

6) In the **Object Selector** attribute, select either the relevant standard selector or a custom selector that should open when the user clicks the **Search** icon. To find the name of the correct standard selector, navigate to the private class setting of the object class and copy the name of the standard selector displayed in the **Selector Definition** attribute. For more information about the configuration of custom selectors, see the section Configuring a Custom Selector for Search Functionalities.

7) To provide cursory information about how to define the data, enter a short text in the **Placeholder** attribute. The text will be displayed as a hint in the search field. The text will be truncated if it is longer than the field. The display of the placeholder text can be configured via the GUI scheme attribute **Placeholders Text Color**. For more information, see the section *Overview of GUI Scheme Attributes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

8) Specify the position of the interface control in the sequence of all interface controls. To do so, enter an integer in the **Tab Index** attribute. New interface controls will have a default value set to 0. Please note the following:

- The order that the interface controls are listed in the editor if stack layout is defined for rendering editors. For more information about stack vs. traditional rendering style for editors, see the section Specifying the Rendering Definition of the Custom Editor.
- The order that is used when the user moves the focus in the custom editor via the TAB key. The correct definition of tabbing is especially critical in order to provide barrier-free accessibility. For more information, see the section Configuring Barrier-Free User Profiles.
- The sequence that the help texts defined via the **Hint** attribute for an interface are listed in the modal Help window.

9) If a help text is required for the interface control, enter the text in the **Hint** attribute. If the **Hint** attribute is defined, the help text will be available in the modal Help window that opens when the editor's Help button is clicked. If stack rendering style is defined for editors, the help texts may be displayed below the editor field or via a Help button next to the editor field. For more information about stack rendering for editors, see the section Specifying the Rendering Definition of the Custom Editor.

10) In the toolbar, click the **Save** button to save your changes.

Adding a Role Edit Search Field to the Custom Editor

A Role Edit Search Field interface control allows a user to specify the role for the relevant object directly in the custom editor rather than requiring the user to define the role in the *Responsibilities Page View*.

A role type is configured in order to allow a functional role to be defined for objects in a specified object class. Role types can be configured to be explicitly available for a user or organization or both.

Only one role type can be captured per Role Edit Search Field interface control. Therefore, if several different role types may be possible for the specific objects captured in the custom editor, you must create a Role Edit Search Field interface control for each potential role type that may be captured. Typically, this configuration would be used if only a single role holder is relevant for the objects being captured in the custom editor. If many different role types may be captured for the objects, then users should specify the roles for the object directly in the *Responsibilities Page View*.

To add an Role Edit Search Field interface control to the custom editor:

1) Right-click the custom editor that you want to define and select **Design Editor**. The custom editor is displayed in the editor in the center pane.

2) In the toolbar of the design editor, click the **Role Edit Search Field** button and click in the editor where you want to place it. The role edit search interface control is added to the editor.

3) Click the Role Edit Search Field interface control in the design editor to activate its attribute window. The **Control Type** attribute is automatically set to `RoleEditSearch`.

4) In the **Caption** attribute, enter the text that should be displayed for the Role Edit Search Field interface control. Typically the caption should give the user some indication of which role type is being captured.

To use the ampersand (&) character in the caption, enter: &&

A caption should not be more than 255 characters. If the field caption requires more characters or formatting, then you should leave the **Caption** attribute empty and create

a static text interface control for which you can define the **Style** attributes. For more information, see the section Adding Static Text to the Custom Editor.

5) In the **Role Type** attribute, select the role type that may be captured via the interface control. All role types that have been configured in the **Reference Data** functionality will be displayed in the drop-down list. For more information about creating role types, see the section *Configuring Role Types to Define Roles in the Responsibilities Page View* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.

6) In the **Object Selector** attribute, the selector `StandardSelector:PERS_ORG_SelectorDef` will be displayed. If you want to specify a custom selector that should open when the user clicks the **Search** icon, select the custom selector in the **Object Selector** attribute. For more information about the configuration of custom selectors, see the section Configuring a Custom Selector for Search Functionalities.

7) To provide cursory information about how to define the data, enter a short text in the **Placeholder** attribute. The text will be displayed as a hint in the search field. The text will be truncated if it is longer than the field. The display of the placeholder text can be configured via the GUI scheme attribute **Placeholders Text Color**. For more information, see the section *Overview of GUI Scheme Attributes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

8) Specify the position of the interface control in the sequence of all interface controls. To do so, enter an integer in the **Tab Index** attribute. New interface controls will have a default value set to 0. Please note the following:

- The order that the interface controls are listed in the editor if stack layout is defined for rendering editors. For more information about stack vs. traditional rendering style for editors, see the section Specifying the Rendering Definition of the Custom Editor.
- The order that is used when the user moves the focus in the custom editor via the TAB key. The correct definition of tabbing is especially critical in order to provide barrier-free accessibility. For more information, see the section Configuring Barrier-Free User Profiles.
- The sequence that the help texts defined via the **Hint** attribute for an interface are listed in the modal Help window.

9) If a help text is required for the interface control, enter the text in the **Hint** attribute. If the **Hint** attribute is defined, the help text will be available in the modal Help window that opens when the editor's Help button is clicked. If stack rendering style is defined for editors, the help texts may be displayed below the editor field or via a Help button next to the editor field. For more information about stack rendering for editors, see the section Specifying the Rendering Definition of the Custom Editor.

10) In the toolbar, click the **Save**  button to save your changes.

Adding Static Text to the Custom Editor

The Static Text interface control allows you to add text to the custom editor to provide instructional text to provide the user community with specific information. Typically, this is used to provide instructions for users entering data in an Edit Field interface control. The size of Static Text interface controls will be adjusted automatically to the content to be displayed.

To add a Static Text interface control to the custom editor:

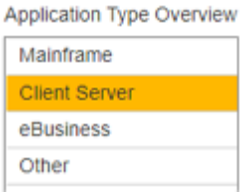
1) Right-click the custom editor that you want to define and select **Design Editor**. The custom editor is displayed in the editor in the center pane.
2) In the toolbar of the design editor, click the **Static Text** button and click in the editor where you want to place it. The Static Text interface control is added to the editor.
3) Click the Static Text interface control in the design editor to activate its attribute window. The **Control Type** attribute is automatically set to `StaticText`.
4) In the **Caption** attribute, enter the text that should be displayed for the Static Text interface control. This could be a short text such to elaborate a property caption or comprehensive information.
5) To format the font of the static text, expand the **Style** section of the attribute window by clicking the + symbol and define the formatting, as needed.
6) Specify the position of **Tab Index** attribute of the interface control. Please note the following:

- The order that the interface controls are listed in the editor if stack layout is defined for rendering editors. For more information about rendering editors in stack vs. traditional layout, see the section Specifying the Rendering Definition of the Custom Editor.
- The order that is used when the user moves the focus in the custom editor via the TAB key.

7) In the toolbar, click the **Save** button to save your changes.

Adding a List Box to Display Enumerations in the Custom Editor

The List Box interface control allows the user to view a list of values specified in an enumeration. The values are only listed as a source of information and cannot be selected by the user for the custom object class property. The value defined for the selected object is highlighted in the list box.



The List Box interface control is only available for properties of the type `String`, `Text`, or `StringArray` for which an enumeration has been defined.

To add a List Box interface control to the custom editor:

1) Right-click the custom editor that you want to define and select **Design Editor**. The custom editor is displayed in the editor in the center pane.
2) In the toolbar of the design editor, click the **List Box** button and click in the editor where you want to place it. The List Box interface control is added to the editor.
3) Click the List Box interface control in the design editor to activate its attribute window. The **Control Type** attribute is automatically set to `ListBox`.

4) In the **Property** attribute, specify the custom object class property of the types `String`, `Text`, or `StringArray` that should be captured via the interface control. The **Value Type** attribute should display `String` or `StringArray`.

5) In the **Caption** attribute, enter the text that should be displayed for the List Box interface control. This could be the name of the custom object class property associated with the List Box interface control or more descriptive text.

To use the ampersand (&) character in the caption, enter: &&

A caption should not be more than 255 characters. If the field caption requires more characters or formatting, then you should leave the **Caption** attribute empty and create a static text interface control for which you can define the **Style** attributes. For more information, see the section Adding Static Text to the Custom Editor.

6) Specify the position of the interface control in the sequence of all interface controls. To do so, enter an integer in the **Tab Index** attribute. New interface controls will have a default value set to 0. Please note the following:

- The order that the interface controls are listed in the editor if stack layout is defined for rendering editors. For more information about stack vs. traditional rendering style for editors, see the section Specifying the Rendering Definition of the Custom Editor.
- The order that is used when the user moves the focus in the custom editor via the TAB key. The correct definition of tabbing is especially critical in order to provide barrier-free accessibility. For more information, see the section Configuring Barrier-Free User Profiles.
- The sequence that the help texts defined via the **Hint** attribute for an interface are listed in the modal Help window.

7) If a help text is required for the interface control, enter the text in the **Hint** attribute. If the **Hint** attribute is defined, the help text will be available in the modal Help window that opens when the editor's Help button is clicked. If stack rendering style is defined for editors, the help texts may be displayed below the editor field or via a Help button next to the editor field. For more information about stack rendering for editors, see the section Specifying the Rendering Definition of the Custom Editor.

8) In the toolbar, click the **Save**  button to save your changes.

Adding Icons to the Custom Editor

The Icon interface control add any icons to the custom editor that are available in the icon gallery in Alfabet Expand. Any icon that you want to add to the custom editor first must be imported to the icon gallery.

FIGURE: Example of an icon displayed in a custom editor

For more information, see the section Adding and Maintaining Icons for the Alfabet Interface.

1) Right-click the custom editor  that you want to define and select **Design Editor**. The custom editor is displayed in the editor in the center pane.

2) In the toolbar of the design editor, click the **Icon** button and click in the editor where you want to place it. The Icon interface control is added to the custom editor.

3) Click the Icon interface control in the custom editor to open its attribute window. The **Control Type** attribute is automatically set to `Icon`.

4) In the **SubType** attribute, select the relevant option based on the size of the icon that you want to add:

- `Icon`: To add an icon that is 22x22 pixel in size.
- `IconLarge`: To add an icon that is 30x30 pixel in size.
- `IconFree`: To add an icon that has an arbitrary size.

5) In the **Icon** attribute, select the relevant icon that you want to display in the custom editor. The icons that you can choose from are based on the value you selected in the **SubType** attribute.

6) To adjust the Icon interface control to the general size of the icon, enter pixel values in the **Height** and **Width** attributes. For example, if you plan to add an icon that is 22x22 pixel in size, you could enter values 22 or larger in the **Height** and **Width** attribute to make handling the Icon interface control easier while designing the custom editor.

7) To provide cursory information about how to define the data, enter a short text in the **Placeholder** attribute. The text will be displayed as a hint in the icon field. The text will be truncated if it is longer than the field. The display of the placeholder text can be configured via the GUI scheme attribute **Placeholders Text Color**. For more information, see the section *Overview of GUI Scheme Attributes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

8) Specify the position of **Tab Index** attribute of the interface control. Please note the following:

- The order that the interface controls are listed in the editor if stack layout is defined for rendering editors. For more information about rendering editors in stack vs. traditional layout, see the section Specifying the Rendering Definition of the Custom Editor.
- The order that is used when the user moves the focus in the custom editor via the TAB key.

9) In the toolbar, click the **Save** button to save your changes.

Adding an HTML Document Stored in the Internal Document Selector to the Custom Editor

An HTML Content interface control can be added to a custom editor in order to provide a link to an HTML document stored in the **Internal Document Selector**. For information about adding documents to the **Internal Document Selector**, see the section *Uploading Documents and Managing User Permissions to Document Folders in the Internal Document Selector* in the reference manual *User and Solution Administration*.

Embedded images may not be included in the HTML code defined for a custom editor.

Adding an HTML Content interface control to the custom editor makes the link available only on the interface in the custom editor. Alternatively, a custom object class property of the type `URL` can be configured which allows a user to enter a URL or document link that will be available as a custom object class property and can therefore be displayed in the object profile or relevant

page view or configured report. For more information, see the section Configuring Custom Properties of the Type URL in the chapter Configuring the Class Model.

To add a document link via an HTML Content interface control to a custom editor:

1) Right-click the custom editor that you want to define and select **Design Editor**. The custom editor is displayed in the editor in the center pane.

2) In the toolbar of the design editor, click the **HTML Content** icon and click in the custom editor where you want to place the HTML Content interface control.

3) Click in the HTML Content interface control that you have added to the custom editor to open its attribute window.

4) In the **SubType** attribute, select `IDocument`.

5) In the **HTML Source** attribute, specify the full path to the HTML document in the **Internal Document Selector**. For example:

```
IDOC:\Documents\Links_Emergency_Plan.html
```

6) Specify the position of the interface control in the sequence of all interface controls. To do so, enter an integer in the **Tab Index** attribute. New interface controls will have a default value set to 0. Please note the following:

- The order that the interface controls are listed in the editor if stack layout is defined for rendering editors. For more information about stack vs. traditional rendering style for editors, see the section Specifying the Rendering Definition of the Custom Editor.
- The order that is used when the user moves the focus in the custom editor via the TAB key. The correct definition of tabbing is especially critical in order to provide barrier-free accessibility. For more information, see the section Configuring Barrier-Free User Profiles.
- The sequence that the help texts defined via the **Hint** attribute for an interface are listed in the modal Help window.

7) If a help text is required for the interface control, enter the text in the **Hint** attribute. If the **Hint** attribute is defined, the help text will be available in the modal Help window that opens when the editor's Help button is clicked. If stack rendering style is defined for editors, the help texts may be displayed below the editor field or via a Help button next to the editor field. For more information about stack rendering for editors, see the section Specifying the Rendering Definition of the Custom Editor.

8) In the toolbar, click the **Save** button to save your changes.

Adding a URL to the Custom Editor

An HTML Content interface control can be added to a custom editor to provide a URL that is external to the Alfabet components.

Embedded images may not be included in the HTML code defined for a custom editor.

Adding an HTML Content interface control to the custom editor makes the link available only in the custom editor. Alternatively, a custom object class property of the type `URL` can be configured which allows a user to enter a URL or document link to an object profile, object cockpit, page view or configured report. For more information, see the section Configuring Custom Properties of the Type URL in the chapter Configuring the Class Model.

To add a URL via an HTML Content interface control to a custom editor:

1) Right-click the custom editor that you want to define and select **Design Editor**. The custom editor is displayed in the editor in the center pane.
2) In the toolbar of the design editor, click the **HTML Content** icon and click in the custom editor where you want to place the HTML Content interface control.
3) Click the HTML Content interface control that you have added to the custom editor to open its attribute window.
4) In the **SubType** attribute, select `EmbeddedHTML`.

   The HTML Content interface control has additional options in the drop-down list for the specification of the `SubType` attribute. These other options are currently not activated.

5) In the **HTML Source** attribute, enter the HTML code. The HTML must be XML-conform HTML, compliant with HTML 5, and use standard HTML tags. The code must start with the tag `<html>` and end with the tag `</html>`. The definition of `<head>` and `<body>` specification is optional. The tags `<xhtml>`, `<html>`, `<body>`, and `<culture_>` are case-sensitive. The HTML code can include links to external URLs.

   For example, to add a link to a document containing additional information about the permissible property values that can be edited, the following code can be used to define the link. The example link calls a URL with parameters. Please note that the parameters must be defined XHTML conform using an entity definition for the character &. This example provides only HTML text in English. For information about providing HTML text in other languages, see the section Adding HTML Text to the Custom Editor.

```
<xhtml>
    <culture_1033>
        <html>
            <head>
                <title>Additional Information</title>
                <link type="text/css" rel="stylesheet"
                href="IDOC:\CSS\my_style_file.css"></link>
            </head>
            <body>
                <p class="hint">For information about the meaning
                of the available options, see<a
                href="http://companyserver/optionhelp/options.html
                ?option1=editorname&#83;option2=classname"
                target="_blank">Available Values</a>.</p>
            </body>
        </html>
```



```
        </culture_1033>
    </xhtml>
```

6) Specify the position of the interface control in the sequence of all interface controls. To do so, enter an integer in the **Tab Index** attribute. New interface controls will have a default value set to 0. Please note the following:

- The order that the interface controls are listed in the editor if stack layout is defined for rendering editors. For more information about stack vs. traditional rendering style for editors, see the section Specifying the Rendering Definition of the Custom Editor.
- The order that is used when the user moves the focus in the custom editor via the TAB key. The correct definition of tabbing is especially critical in order to provide barrier-free accessibility. For more information, see the section Configuring Barrier-Free User Profiles.
- The sequence that the help texts defined via the **Hint** attribute for an interface are listed in the modal Help window.

7) If a help text is required for the interface control, enter the text in the **Hint** attribute. If the **Hint** attribute is defined, the help text will be available in the modal Help window that opens when the editor's Help button is clicked. If stack rendering style is defined for editors, the help texts may be displayed below the editor field or via a Help button next to the editor field. For more information about stack rendering for editors, see the section Specifying the Rendering Definition of the Custom Editor.

8) In the toolbar, click the **Save** button to save your changes.

Adding HTML Text to the Custom Editor

An HTML Content interface control can be added to a custom editor to display text with HTML formatting (font and color). The HTML code can include text as well as links to external URLs and can reference stylesheets stored in the **Internal Document Selector**.

Embedded images may not be included in the HTML code defined for a custom editor.

To add HTML text via an HTML Content interface control to a custom editor:

1) Right-click the custom editor that you want to define and select **Design Editor**. The custom editor is displayed in the editor in the center pane.

2) In the toolbar of the design editor, click the **HTML Content** icon and click in the custom editor where you want to place the HTML Content interface control.

3) Click in the HTML Content interface control that you have added to the custom editor to open its attribute window.

4) In the **Sub-Type** attribute, select `EmbeddedHTML`.

The HTML interface control has additional options in the drop-down list for the specification of the `SubType` attribute. These other options are currently not activated.

5) In the **HTML Source** attribute, enter the HTML code. Please keep the following in mind:

- The HTML must be compliant with XHTML, XML-conform HTML, compliant with HTML 5, and use standard HTML tags. The HTML header implements standard HTML elements.
- The code must start with an `<xhtml>` tag and end with `</xhtml>`. Standard HTML elements must be written in lower-case letters in order to be correctly parsed: `<xhtml>`, `<html>`, `<head>`, `<body>`, and `<culture_>`. The definition of `<head>`, `<body>`, and `<culture_>` is optional.
- The formatting of the HTML can either be written explicitly in the `<body>` element or can be specified via a stylesheet that is stored in the **Internal Document Selector**. In this case, the HTML must refer to the target CSS file in a `<link>` element. The CSS file should contain all necessary styles to display the content of the HTML. Please note that the CSS file may not be stored in the root folder of the **IDOC** explorer. The CSS file stored in the **Internal Document Selector** must be located in a document folder that is subordinate to the root folder of the **IDOC** explorer. To upload a file to the **Internal Document Selector**, see the section *Uploading Documents and Managing User Permissions to Document Folders in the Internal Document Selector* in the reference manual *User and Solution Administration*. For example:

```
<xhtml>
   <culture_1033>
      <html>
      <body style="margin:4px;background:#3d4b60;overflow:hidden;">
         <link type="text/css" rel="stylesheet"
         href="IDOC:\CSS\my_style_file.css"></link>
         <p>This is the content for the header of the wizard
         step.</p>
         ...
      </body>
      </html>
   </culture_1033>
</xhtml>
```

- All CSS formatting instructions must end with `!important` to ensure that they are processed by the Alfabet application. For example:

```
<xhtml>
   <culture_1033>
      <html>
      <body style="margin:4px !important;background:#3d4b60
      !important;overflow:hidden !important;">
         <style type="text/css">
            p.header
            {
               font-family:verdana !important;
               font-size:18px !important;
               color:#ff0000 !important;
               text-align: Left !important;
```



```
                    }
                </style>
                <p class="header">This is the content for the header of the
                wizard step.</p>
                ...
            </body>
            </html>
        </culture_1033>
    </xhtml>
```

- If translation to a secondary language is required, you must provide the translation in the HTML specification. Please note that HTML texts cannot be translated via the **Translation Editor** available in Alfabet Expand. Please consider the following:
- If the HTML description is required in additional languages, each language text must be defined within language elements (For example, `<culture_1031>` for English, `<culture_1033>` for German, etc.)
- The element `<culture_xxx>` must be specified as a child of the root element `<xhtml>`. The element `<html>` must be specified as a child of the element `<culture_xxx>`. The element `<html>` contains the HTML specification in the relevant language. For example, to provide information in English and German:

```
<xhtml>
    <culture_1033>
        <html>
            <body>
                <h3>Glossary: Application</h3>
                <p>An application is a fully-functional integrated IT
                product that provides functionality to end users
                and/or to other applications. As such, an application
                supports the business to accomplish its mission.
                Applications operate on a platform made up of
                hardware and software components necessary to run the
                application.</p>
            </body>
        </html>
    </culture_1033>
    </culture_1031>
        <html>
            <body>
                <h3>Glossar: Applikation</h3>
                <p>Eine Applikation ist ein voll funktionsfähiges,
                integriertes IT-Produkt, das Funktionalitäten für
                Endanwender und/oder für andere Applikationen bietet.
                Eine Applikation unterstützt das Unternehmen bei der
                Zielerreichung. Applikationen werden auf einer
                Plattform betrieben, die aus den für die Ausführung
```



```
                der Applikation erforderlichen Hardware- und
                Software-Komponenten besteht.</p>
            </body>
        </html>
    </culture_1031>
</xhtml>
```

6) Specify the position of **Tab Index** attribute of the interface control. Please note the following:

- The order that the interface controls are listed in the editor if stack layout is defined for rendering editors. For more information about rendering editors in stack vs. traditional layout, see the section Specifying the Rendering Definition of the Custom Editor.
- The order that is used when the user moves the focus in the custom editor via the TAB key.
- The sequence that the help texts defined via the **Hint** attribute for an interface are listed in the modal Help window.

7) In the toolbar, click the **Save** button to save your changes.

Adding a Color Selector to the Custom Editor

A color selector may be added in order to assign a color to the object being created or edited in the custom selector. The color is then displayed for the object in all Alfabet views and configured reports.

A color selector can be added to any custom editor created for an object class that has a `Color` property of the type `String`. When the user creates or edits an object in the custom editor, he/she can change the color in the custom editor. The color will then be used for the display of that object in business graphics for all Alfabet views and configured reports for all users.

To add a color selector to a custom editor:

1) Right-click the custom editor that you want to define and select **Design Editor**. The custom editor is displayed in the editor in the center pane.
2) In the toolbar of the design editor, click the **Edit** icon and click in the custom editor where you want to place the Edit Field interface control.
3) Click in the Edit Field interface control that you have added to the custom editor to open its attribute window.
4) In the **Name** attribute, enter a technical name for the interface control.
5) In the **Caption** attribute, enter a caption for the interface control. The term **Color** is the conventional caption for the color selector in standard Alfabet views.
6) In the **Value Type** attribute, ensure that `String` is selected.
7) In the **Property** attribute, enter `Color`. Please note that this option cannot be selected but rather must be typed in.
8) In the **SubType** attribute, select `Color`.
9) To provide cursory information about how to define the data, enter a short text in the **Placeholder** attribute. The text will be displayed as a hint in the color field. The text will be truncated if it is

longer than the field. The display of the placeholder text can be configured via the GUI scheme attribute **Placeholders Text Color**. For more information, see the section *Overview of GUI Scheme Attributes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

10) Specify the position of the interface control in the sequence of all interface controls. To do so, enter an integer in the **Tab Index** attribute. New interface controls will have a default value set to 0. Please note the following:

- The order that the interface controls are listed in the editor if stack layout is defined for rendering editors. For more information about stack vs. traditional rendering style for editors, see the section Specifying the Rendering Definition of the Custom Editor.
- The order that is used when the user moves the focus in the custom editor via the TAB key. The correct definition of tabbing is especially critical in order to provide barrier-free accessibility. For more information, see the section Configuring Barrier-Free User Profiles.
- The sequence that the help texts defined via the **Hint** attribute for an interface are listed in the modal Help window.

11) If a help text is required for the interface control, enter the text in the **Hint** attribute. If the **Hint** attribute is defined, the help text will be available in the modal Help window that opens when the editor's Help button is clicked. If stack rendering style is defined for editors, the help texts may be displayed below the editor field or via a Help button next to the editor field. For more information about stack rendering for editors, see the section Specifying the Rendering Definition of the Custom Editor.

12) In the toolbar, click the **Save**  button to save your changes.

13) The color selector can be reviewed by right-clicking the custom editor in the **Custom Editors** folder and selecting **Review Custom Editor**. The **Color Selector** button will be displayed next to the edit field. When a user selects a color, that color will fill the edit field.

Specifying the Interface Control To Be Mandatory

You can define an interface control to display a red star next to the field indicating to users that the field is mandatory. Please note that mandatory fields are only enforced in the editor or wizard. If data is imported via other means (for example, via ADIF), it will be necessary to systematically ensure that mandatory fields are filled. To configure an interface control as mandatory, select `Always` in the **Presence** attribute.

Specifying an Editor Field To Be Non-Editable

In some cases, you may want an editor field to display a predefined default value that cannot be edited by users when creating a new object via the custom editor. In this case, could define an Edit Field interface control for a custom object class property of the type `String`, `Integer`, `Real`, and `URL` as well as a Check Box interface control of the type `Boolean` to be non-editable. An interface control that is defined as non-editable (ReadOnly) will be displayed in grey and thus disabled and therefore cannot be edited.

To specify an interface control to display a default value and not be edited, ensure that the **Default Value** attribute of the custom object class property has been defined. For more information about specifying the default value for a custom object class property, see the section Configuring Custom Properties for Protected or Public Object Classes in the chapter Configuring the Class Model.

In the design editor, click the relevant **Edit** or **Check Box** button. In the attribute window, set the **Read Only** attribute to `True`. In the toolbar, click the **Save** button to save your changes.

Please note that if the **Read Only** attribute is set to `True` for a Check Box interface control, the checkbox will be disabled, and the value defined in the **Hint** attribute of the associated custom object class property of the type `Boolean` cannot be displayed as help text. It is recommended that you add the help text as static text to the editor if it should be visible for a non-editable checkbox.

Specifying Conditional Constraints in the Custom Editor

Specifying conditional constraints in custom editors can provide a more dynamic user experience and ensure that users are seeing and defining the data that is pertinent to the task at hand. The display of fields that aren't relevant to the current context can be suppressed based on the definition of a value in a specified editor field. Likewise, whether a field is mandatory or read-only can also be controlled based on the value defined for a specified editor field. For example, in the case of an **Application** editor, you may want to only display the **Sox Description** field if the **Is Sox Relevant** checkbox is selected (=`True`) for the application. Furthermore, you may want the **Sox Description** field to be mandatory if the **Is Sox Relevant** checkbox is selected. In this case, you would specify a condition that makes the **Sox Description** field visible and mandatory if the **Is Sox Relevant** checkbox is selected.

If the visibility and editability of an interface control shall depend on the value defined in another interface control, the condition has to be defined in two steps:

- A condition configuration object has to be created that specifies the general condition. A condition defines one or multiple parameters, each parameter having a unique name and an operator and condition that shall apply to the parameter. One condition can be used in multiple interface controls in multiple editors.
- In the editor field that shall be managed by the condition, the condition must be defined by first choosing the relevant condition configuration object and then mapping each parameter name in the condition to the interface control in the editor that shall fulfill the condition defined for the parameter in the context of the selected editor.

You can configure conditions for interface controls in custom editors to determine the visibility of other interface controls as well as whether editor fields are mandatory or read-only. Conditions can be defined for any interface control for which the **Refresh on Submit** attribute is available and parameters in the condition can be mapped to any interface control for which the **Submit** attribute is available (such as Check Box, Checked Combo Box, and Radio Button Group interface controls.).

The following steps are required to configure conditional constraints for interface controls in custom editors:

- Create a condition and specify one or more expressions needed for the condition. Please note that the valid syntax that is permissible in expressions is described in the section Specifying Conditional Constraints in the Custom Editor.
- For the interface control whose value determines the visibility/mandatory/read-only behavior, set the **Submit** attribute to `True`. These are the fields mapped to the parameters in the condition.

- For the interface control(s) that are the target of the condition, set the **Refresh on Submit** attribute to `True`. Specify one or more of the following for the interface control(s) that are the target of the condition:

- To specify that visibility is determined by the condition, configure the condition in the **Visibility Condition** attribute of the interface control.
- To specify that mandatory behavior is determined by the condition, configure the condition in the **Presence Condition** attribute of the interface control.
- To specify that read-only behavior is determined by the condition, configure the condition in the **Read-Only Condition** attribute of the interface control.

To configure a condition for one of the above listed attributes, click in the attribute field and click the button to the right of the field to open the condition editor. Specify the following according to demand:

- Select a condition in the drop-down list of the **Select Condition** field.
- For each parameter defined in the condition, a row will be displayed in the table below the **Select Condition** field. For each parameter, select the interface control that shall be mapped to the parameter in the drop-down list of the **Parameter Value Reference** field. All interface controls in the same custom editor are displayed. When the condition is evaluated, the parameter will be substituted with the current value in the specified field. For example, if the condition is `@Type=='CloudService'` and the `Type` parameter is mapped to an interface control that returns the value `CloudService`, the condition will read `'CloudService'=='CloudService'` and the condition will have a positive result.

If the condition contains a parameter that cannot be mapped to an interface control in this editor, you can alternatively select a static value in the field **Parameter Value**. This value is then used to substitute the parameter in the condition instead of a value of an interface control. The setting of the **Parameter Value Reference** attribute supersedes the setting of a **Parameter Value** attribute.

Please note if a custom editor is embedded in a wizard, the conditions may only be relevant for the current wizard step. The conditions cannot refer to interface controls that are available on other wizard steps. This applies in particular to the configuration of a wizard with a single wizard step of the type `Editor` and that has the **Tab as Separate Steps** attribute set to `True`. For more information about configuring wizard steps with an embedded custom editor, see the section Configuring a Wizard Step to Display a Standard or Custom Editor.

Syntactically incorrect conditions in editors (or editors embedded in wizards) may limit or prevent in-line editing in object profiles or object cockpits. If a condition has an incorrect syntax, an error will be displayed if the user tries to open the editor/wizard.

The following information is available:

- Creating Conditions to Implement in a Custom Editor
- Assigning Conditions to Interface Controls in the Custom Editor

The following information is available:

- Creating Conditions to Implement in a Custom Editor
- Assigning Conditions to Interface Controls in the Custom Editor

Creating Conditions to Implement in a Custom Editor

Conditions can be used in the context of custom editors, object cockpits, and filters in configured reports. The conditions can be reused as needed. Conditions can be based on either expressions or queries defined in configured reports. Please note the following:

- Conditions based on expressions can be defined for custom editors, configured reports, and object cockpits. The expression is evaluated for a value defined for a specified interface control. Please note that only conditions based on expressions can be implemented for custom editors or configured reports.
- Conditions based on configured reports can be defined for object cockpits. The query defined for the configured report must search for the results in the database in order to provide a value to be evaluated by the condition.

When you expand the Conditions folder, you may see planned conditions  , active conditions  , and retired conditions  . The condition state must be set to `Active` in order to make the condition available in the **Visibility Condition**, **Presence Condition**, and **Read-Only Condition** attributes for custom editor fields, filter fields in configured reports, and object cockpits.

To create a condition:

1) Go to the **Presentation** tab and right-click the **Conditions** folder and select **New Condition**. The condition state will be set to `Plan` per default for the new condition.

2) Click the new condition  and define the following in the attribute window:

- **Technical Name**: Enter a name for the condition. It is recommended that the name indicates the meaning of the condition. All active conditions will be displayed in drop-down lists for the **Presence Condition**, **Read-Only Condition**, and **Visibility Condition** attributes.
- **Group**: Enter the name of a new condition folder that you want to store the condition in or select an existing condition folder.
- **Check Result Type**: Select either `Positive` or `Negative` to define what constitutes a fulfilled condition.
- `Positive` means that the condition is fulfilled if the defined expression is true (if **Type** = `Expression`), or if the query associated with the configured report delivers a result (if **Type** = `Report`).
- `Negative` means that the condition is fulfilled if the defined expression is false (if **Type** = `Expression`) or the query delivers no results (if **Type** = `Report`).

For example, if the expression defined for the condition is `@Type=='CloudSer-vice'` and the `Type` parameter is mapped to an interface control that returns the value `CloudService`, the expression will read `'CloudService'=='CloudSer-vice'` and the expression is true. If **Check Result Type** is set to `Negative`, the condition is not fulfilled. If **Check Result Type** is set to `Positive`, the condition is fulfilled.

- **Type**: Select `Expression`.
- **Expression**: Click the **Browse**  button to open the editor and enter the expression that shall be evaluated for the condition matching the following rules:

- The required syntax is: For some operators, like EMPTY, the definition of a condition is not required.

```
@ParameterName Operator Condition
```

```
@Type == 'CloudService'
```

- String values and boolean values must be placed in single quotes. Please note that the strings in conditions are case-sensitive.
- Multiple expressions may be combined via `AND` or `OR`, whereby individual expressions must be placed in parenthesis.

```
(@Type == 'CloudService') I ((@Type == 'OnPremise') &
(@CloudCandidate == 'True'))
```

- A condition cannot be a second paramter. It is not possible to compare the values of two interface controls within one condition.
- The following operators and conditions may be used in the expression:

equals	`@Param == 'STRING'` (or `'True'` and `'False'` for Boolean values)
does not equal	`@Param != 'STRING'`
in	`@Param IN ['STRING', 'STRING', 'STRING']` (used when the value is a list)
not in	`!(IN ['STRING', 'STRING', 'STRING']` (used when the value is a list)
begins with	`@Param BEGINS_WITH 'STRING'`
ends with	`@Param ENDS_WITH 'STRING'`
contains	`@Param CONTAINS 'STRING'`
does not contain	`!(CONTAINS 'STRING')`
is between	`(@Param > NUMERIC1) & (@Param < NUMERIC2)`
is empty	`@Param EMPTY`
is not empty	`!(@Param EMPTY)`
is greater than	`@Param > NUMERIC`

is greater than or equal to	`@Param >= NUMERIC`
is less than	`@Param < NUMERIC`
is less than or equal to	`@Param <= NUMERIC`
AND	`&`
OR	`|`
NOT	`!`

3) Once the condition is complete, the condition state must be set to `Active`. To do so, right-click the condition and select **Set Condition State to 'Active'**. The active condition will be available in the **Visibility Condition**, **Presence Condition**, and **Read-Only Condition** attributes for custom editor fields, filter fields in configured reports, and object cockpits.
4) Once the condition is set to `Active`, you can test the condition. To do so, right-click the condition and select **Test Condition**. In the editor, enter a valid value in the **Parameter Value** field and click **OK**. Information will be displayed indicating if the condition is valid based on the value entered or if an error has occurred.
5) In the toolbar, click the **Save** button to save your changes.

Assigning Conditions to Interface Controls in the Custom Editor

You can assign the configured conditions to the interface controls in custom editors to determine visibility as well as whether the interface controls are mandatory or read-only. Conditions can be defined for any interface control for which the **Submit** attribute is available (such as Check Box, Checked Combo Box, and Radio Button Group interface controls.). The interface control with the **Submit** attribute is the interface control whose value determines the visibility/mandatory/read-only behavior.

The **Refresh on Submit** attribute must be specified for the interface control(s) that are the target of the condition. The variables in the condition must be mapped to the source interface control whose value determines the visibility/mandatory/read-only behavior.

For example, in the case of an **Application** editor, you may want to only display the **Sox Description** field (Interface Control 2) if the **Is Sox Relevant** checkbox (Interface Control 1) is selected (`=True`) for the application. Furthermore, you want the **Sox Description** field to be mandatory if the **Is Sox Relevant** checkbox is selected In this case, you would specify a condition that makes the **Sox Description** field visible and mandatory if the **Is Sox Relevant** checkbox is selected.

To define conditions for editor fields:

1) Go to the **Presentations** tab and expand the **Custom Editors** folder.

2) Right-click the custom editor  that you want to define and select **Design**. The custom editor is displayed in the editor in the center pane.

3) Click the interface control whose value shall determine the visibility/mandatory/read-only behavior and set the **Submit** attribute to `True`. The interface control with the **Submit** attribute is the interface control that has the value that is evaluated for the condition. This attribute is mandatory to define a condition.

4) Click the interface control whose behavior will be determined by the condition. Define the following attributes.

- **Refresh on Submit**: Select `True`. This attribute is mandatory to define a condition. You must specify at least one the following attributes to define the behavior of the interface control:

- **Visibility Condition**: Specify that the interface control will be visible if the condition is fulfilled. You must map the variables defined in the condition to the source interface control that they reference. To define the **Visibility Condition** attribute, click the **Browse**  button and in the **Visibility Condition** editor, and define the following:

  For example, in the case of example about the **SOX Relevance** and **SOX Description** fields in the **Application** editor: the `SimpleBooleanOnTrue` condition has been created with the expression `@VALUE == 'True'`. You would define the attributes for the **Visibility Condition** as follows:

  - **Select Condition**: Select `SimpleBooleanOnTrue`.
  - **Parameter Name**: Displays `VALUE`.
  - **Parameter Value Reference**: Select `SOXRelevant`. This parameter represents the variable to be evaluated for the parameter in the **Parameter Name** column.

  Therefore, if the **SOX Relevance** interface control is set to `True`, then the `SimpleBooleanOnTrue` condition is fulfilled, and the **SOX Description** interface control will be visible.

  - **Select Condition**: Select the condition to specify the visibility of the interface control.
  - **Parameter Name**: Displays the variables used in the condition.
  - **Parameter Value**: This column is not relevant for custom editors.
  - **Parameter Value Reference**: Select the interface control that shall be visible if the variable in the **Parameter Name** column to the source interface control (Interface Control 1) of the condition.

Click **OK** to save the definition and close the editor.

- **Presence Condition**: Specify that the interface control will be mandatory if the condition is fulfilled. Click the **Browse**  button and in the **Presence Condition** editor, and define the following:

  - **Select Condition**: Select the condition to specify the interface control as a mandatory field.
  - **Parameter Name**: Displays the variables used in the condition.

- **Parameter Value**: This column is not relevant for custom editors.
- **Parameter Value Reference**: Select the interface control that shall be mandatory if the variable in the **Parameter Name** column to the source interface control (Interface Control 1) of the condition.

- **Read-Only Condition**: Specify that the interface control will be read-only if the condition is fulfilled. Click the **Browse** button and in the **Read-Only Condition** editor, and define the following:
  - **Select Condition**: Select the condition to specify the interface control as a read-only field.
  - **Parameter Name**: Displays the variables used in the condition.
  - **Parameter Value**: This column is not relevant for custom editors.
  - **Parameter Value Reference**: Select the interface control that shall be read-only if the variable in the **Parameter Name** column to the source interface control (Interface Control 1) of the condition.

5) In the toolbar, click the **Save** button to save your changes.

Specifying the Tab Order of the Interface Controls

The **Tab Index** attribute of each interface control determines the sequence when a user presses the TAB key to move the focus in the user interface as well as the order that the Help texts are listed in the modal window that opens when the **Help** button is clicked in the editor.

To define the tab index of each field in the editor, click each interface control and specify its position in the sequence by entering the relevant integer in the **Tab Index** attribute. Click the **Save** button after defining the **Tab Index** attribute of each interface control.

Adding Help Text to Interface Controls in the Custom Editor

Help text can be created for any interface control in order to provide the Alfabet user community with specific instructions relevant to the data captured via the interface control. The help text will be available in the modal window that opens when the editor Help is opened.

Please note the following:

- The text entered in the **Hint** attribute may not be longer than 600 characters. Texts exceeding 600 characters will not be saved to the Alfabet database. Please note that if the hint is translated, the translated text may also not exceed 600 characters.
- Please note that the value defined for the **Hint** attribute for the custom object class property associated with the interface control will not be displayed as help text in the custom editor. The help text must be explicitly defined via the **Hint** attribute of the interface control.

- The **Tab Index** attribute of the interface control determines the sequence when a user presses the TAB key to move the focus in the user interface as well as the order that the Help texts are listed in the modal window that opens when the **Help** button is clicked in the editor. For more information, see the section Specifying the Tab Order of the Interface Controls.

To define help text for an interface control:

1) Click the interface control in the custom editor to open the attribute window.

   If the **Read Only** attribute is set to `True` for a Check Box interface control, the value defined in the **Hint** attribute of the custom object class property of the type `Boolean` cannot be displayed as a help text in the editor. It is recommended that you add the help text as static text to the editor.

2) In the **Hint** attribute, click the **Browse** button to open the editor.
3) Enter the help text in the editor and click **OK**.
4) In the toolbar, click the **Save** button to save your changes.

Hiding Fields in the Custom Editor

It is possible to hide properties in the custom editor in the context of a particular user profile. Please note that if the custom editor is embedded in a wizard, this will also impact the wizard. For more information about the consequences of hiding standard or custom properties in editors as well as the procedure to do this, see the section Specifying the Visibility of Object Class Properties in Page Views in the chapter Configuring User Profiles for the User Community.

Reviewing the Visualization of an Existing Custom Editor

All existing custom editors can be reviewed in the Alfabet interface in order to check their visualization in the solution interface. Please note that the layout and design of the custom editor in the Alfabet interface when rendered via Alfabet Expand is not identical to the layout and design of the editor when rendered in the browser interface.

1) In the **Presentations** tab, expand the **Custom Editors** folder to display all existing custom editors.

   If you have just created or edited a custom editor, you must right-click the **Custom Editors** folder and select **Rescan Tree** to update the folder.

2) Right-click the custom editor and select **Review Custom Editor**. The new custom editor is displayed. Please note that the layout and design of the custom editor in the Alfabet interface when rendered via Alfabet Expand is not identical to the layout and design of the editor when rendered in the browser interface.

Configuring Styles and GUI Scheme Settings for Standard and Custom Editors

Various GUI scheme settings should be defined to determine the layout and visualization of editors. The GUI scheme settings that are defined will apply to all standard and custom editors associated with the user profile that the GUI scheme is assigned to. The following GUI scheme attributes are relevant for custom editors:

- Layout of interface controls for standard and custom editors. Specify the rendering style of the interface controls for all standard and custom editors via the **Editor Rendering Options** attribute available for the relevant GUI scheme. Two options are possible and the method you choose will determine how you go about designing the custom editors. You can choose the traditional rendering style, which reflects the explicit layout of all interface controls in custom editors as designed by the solution designer, or the stack rendering style which automatically positions the visible interface controls in a one- or two-column linear list. For detailed information about defining the layout of interface controls in editors, see the section Specifying the Rendering Definition of the Custom Editor.
- Specify the font color of the placeholders displayed in the editor fields in standard and custom editors via the **Placeholders Text Color** attribute available for the relevant GUI scheme. Placeholders represent the cursory information displayed in the editor field about how to define the data. Texts can be specified via the **Placeholder** attribute of the relevant interface control in the context of the custom editor.
- Specify the foreground and background for the primary and secondary button in standard and custom editors as well as wizards via the **Primary Button Skin** and **Secondary Button Skin** attributes available for the relevant GUI scheme. The primary button is the button that would typically be clicked by the user. For example, in an editor, the primary button is the **OK** button and the secondary button is the **Cancel** button. In a wizard, the primary button is the **Next** button and the secondary button is the **Previous** button. The skin styles allow issues like borders, box shadowing, and font color to be specified.
- Specify the font color and font style for the captions of editor fields when displayed in the **Help for Editor** dialog via the **Editor Help Control Caption Font** attribute available for the relevant GUI scheme. The **Help for Editor** dialog opens when the **Help** button is clicked in standard and custom editors. Expand the **Caption Styling** section of the relevant GUI scheme to define the **Editor Help Control Caption Font** attribute.

For an explanation and example of each attribute that can be defined for a GUI scheme, see the section *Overview of GUI Scheme Attributes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

Deleting a Custom Editor from the Custom Editors Folder

Before a custom editor is deleted, you should check to see if the custom editor is implemented in other configuration objects such as object views or workflow templates. To do so, right-click the custom editor  and click **Show Usage**. A window will be displayed showing the configuration objects using the selected custom editor. For more information regarding the **Show Usage** functionality, see the section Using the Show Usage Functionality in the chapter Getting Started with Alfabet Expand.

When you delete a custom editor from the **Custom Editors** folder, the custom object class properties displayed in the editor will not be deleted. These custom object class properties are still assigned to the respective class and can be viewed in the **Meta-Model** explorer tree in the **Classes** folder.

1) In the **Presentation** tab, click the + symbol next to the **Custom Editors** folder. You will see all existing custom editors listed in the tree.
2) Right-click the custom editor  that you want to delete and select **Delete**.
3) Confirm the warning by clicking **Yes**. The custom editor is deleted from the explorer.

Configuring Editors for the Mass Update of Data Capture Objects and Information Flows

Alfabet provides the configuration of custom editors that allow users to bulk update applications, components, devices, ICT objects, peripherals, and information flows. A button labelled **Mass Update** is displayed in the views for which a multi-editor is configured. In the configured mass update editors, users may modify standard and custom object class properties for multiple objects in the **Capture** <ObjectClass> functionalities and **Information Flows** page views. The multi-editors are configured via the respective XML objects available in the **MultiEditors** folder in the **Presentation** tab. The XML objects allow editors to be configured that support users to simultaneously update standard and custom object class properties for a set of objects found via a configured query.

Please note the following:

- If a user selects a user group node in the explorer and accesses the mass update editor from this node, the objects found by the configured query will be displayed and the user group node selected in the explorer will not be reflected in the set of objects in the editor.
- The query configured for a multi-editor will be executed when a user clicks the **Mass Update** button. If no objects are found by the query that has been configured for the multi-editor, an error message will be displayed indicating that there are no objects to be updated.

Please note that there is no enforcement mechanism in the **Mass Update** editor to ensure that the values users define are permissible. For example, the editor will not prevent users from defining start and end dates for business supports that are outside of the start and end dates of the provider or from setting a business support object state to **Active** when the provider's object state is set to **Plan**. Therefore, users must ensure that the values defined are correct and meaningful based on the object they are working with.

The following XML objects are available:

- The XML object ***APP_InformationFlows_ME*** specifies the multi-editor available in the *Information Flows Page View* (`APP_InformationFlows`) for applications and peripherals.
- The XML object ***COM_InformationFlows_ME*** specifies the multi-editor available in the *Information Flows Page View* (`COM_InformationFlows`) for components.
- The XML object ***APP_UserApplicaions_ME*** specifies the multi-editor available in the *Document Application Functionality* (`APP_UserApplication`).

- The XML object ***COM_UserComponents_ME*** specifies the multi-editor available in the *Document Components Functionality* (`COM_UserComponents`).
- The XML object ***DVC_UserDevices_ME*** specifies the multi-editor available in the *Document Devices Functionality* (`DVC_UserDevices`).
- The XML object ***ICTO_UserICTObjects_ME*** specifies the multi-editor available in the *Document ICT Objects Functionality* (`ICTO_UserICTObjects`).
- The XML object ***PRF_UserPeripherals_ME*** specifies the multi-editor available in the *Document Peripherals Functionality* (`PRF_UserPeripherals`).

To edit the XML objects:

1) Go to the **Presentation** tab, expand the **XML Objects** folder, and expand the **MultiEditors** node. You will see all XML objects for multi-editors that can be edited.
2) Right-click the relevant XML object and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.

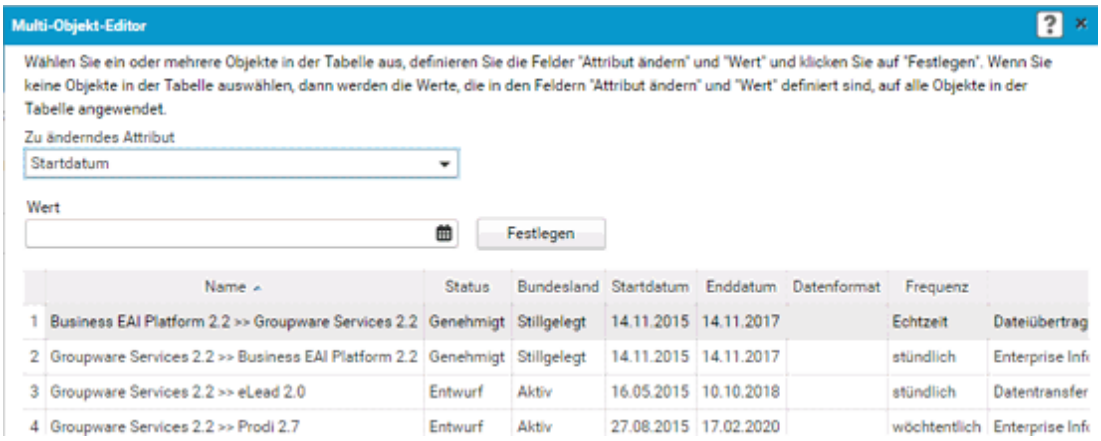


FIGURE: Example of the Multi-Object Editor the Information Flows page view

3) Define the XML attributes, as needed. The table below displays the XML attributes that can be edited for the relevant XML object:

XML Element (bold) / XML Attribute	Explanation
MO_Editor	
`ClassName`	Displays the object class that the custom editor targets. This should not be edited.
`CustomProperties`	Enter "`true`" if custom object class properties are included among the object class properties that may be changed for the selected object class.

XML Element (bold) / XML Attribute	Explanation
	NOTE: Please note that if the XML element `CustomProperties` is set to "`true`", not only will custom properties be included but you will no longer be able to specify displayed properties via the `ShowProperty` XML elements in the query. In other words, all properties will be displayed if the XML element `CustomProperties` is set to "`true`".
`CheckRights`	Enter "`true`" if access permissions should be honored in the editor. If you define "`true`", then an object that the currently logged on user has no access permissions for will not be displayed in the results found by the query
Query	Enter the Alfabet queries necessary to fill the dataset in the multi-editor. The query should specify the objects to be displayed in the editor's dataset and the `ShowProperty` XML elements should specify the object class properties displayed as columns in the tabular dataset. **NOTE**: If multiple queries are defined, the Show properties must be identical. Otherwise, an error will occur.XML restrictions do not apply when the Alfabet query is contained in the CDATA attribute. For more information, see the chapter ATO: Defining Queries. **EXAMPLE**:Query for `APP_UserApplications_ME`

Attribute	An XML element ***Attribute*** should be defined for each object class property that is to be displayed in the **Change Attribute** field in the **Multi-Object Editor** in the Alfabet interface.
	
	NOTE:Query for `APP_UserApplications_ME`
	
	`<Attribute Property="StartDate" CanBeNull="false" />`
	
	Query for `APP_InformationFlows_ME`
	
	`<Attribute Property="ConnectionDataFormat"`
	`CanBeNull="true">`
	`<Query><![CDATA[Alfabet_QUERY_500 FIND ConnectionDataFormat`
	`SHOW ConnectionDataFormat.Name] ]></Query>`
	`</Attribute>`

`Property`	Enter the name of the object class property that can be selected in the **Change Attribute** field in the **Multi-Object Editor** in the Alfabet interface. Scalar properties and object class properties of

XML Element (bold) / XML Attribute	Explanation
	type `Reference` are valid. Object class properties of type `ReferenceArray` are not supported.
`CanBeNull`	Enter "`true`" if the first value in the **Value** field in the **Multi-Object Editor** should be a blank line (null value). This ensures that a value is not selected per default. Please note that if the object class property is a mandatory property, you should enter "`false`" to ensure that a value is defined.
`Query`	For each XML element ***Attribute*** targeting an object class property of type `Reference`, a query must be defined that determines which objects must fill the **Value** field in the **Multi-Object Editor**. See the example for the query for `APP_InformationFlows_ME` described above for the XML element ***Attribute***.

4) In the toolbar, click the **Save** button to save the XML definition.

Modifying Preconfigured Editors Provided by Software AG

If you have been provided with preconfigured editors by Software AG, these will be displayed below the **Editors** node in Alfabet Expand. You can carry out any of the following on these editors:

- Right-click the relevant editor and select **Design Editor** to design and add additional interface controls to the editor. For more information, see the section Configuring the Interface Controls in the Custom Editor.
- Right-click the relevant editor and select **Review Editor** to view the visualization of the editor in the Web interface.

Chapter 7: Configuring Custom Selectors and Search Functionalities

Alfabet provides several methods to search for and find objects. Object classes and object class stereotypes may be configured to be searchable in the standard Alfabet search functionalities. Additionally, custom selectors can be created and implemented in the standard search functionalities as well as in page views, standard or custom editors, workflows, and configured reports. The solution designer can further configure whether a wildcard <*> should be automatically implemented when a user enters search criteria.

In addition to searching for objects, users can search for information about provided via configured reports.

The following information is available:

Configuring the Searchability of Alfabet Objects

If an object class is specified to be searchable, it can be searched via the **Simple Search**, **Full-Text Search** and **Glossary** functionalities as well as via object selectors available in page views, editors, wizards, and workflows. The searchability of an object class is specified via the class setting. In this way, the object class may be searchable in some user profiles but not in others. If object class stereotypes have been configured for an object class, the searchability of the object class stereotype is configured for the class setting of the object class stereotype.

If object class stereotypes have been defined for an object class, the solution designer can specify searchability on the level of the object class and/or object class stereotype. If the entire class is to be searchable irrespective of the object class stereotype, the **Searchable** attribute must also be set to `True`. The name of the object class will be displayed in the **Search for** filter in the search functionality. Please note that the caption of the object class should be different than the captions of its object class stereotypes.

Once the class setting is defined as searchable, objects in that object class or object class stereotype can be searched via the **Simple Search**, **Full-Text Search** and **Glossary** search functionalities as well as via object selectors available in page views, editors, wizards, and workflows. For an overview of the object classes that are searchable, see the chapter *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

For performance reasons, the return set for the standard search functionality is limited to 300 objects.

To configure an object class or object class stereotype as searchable:

1) Go to the **Presentation** tab, expand the **View Schemes** folder, and right-click the relevant view scheme  and select **Edit View Scheme**.
2) In the center pane, click the class setting that you want to define. You will see the attribute window in the right pane.
3) In the attribute window, ensure that the **Searchable** attribute is set to `True` to specify that users accessing Alfabet can search the object class. Select `False` if the object class should not be searchable in the search functionalities.
4) In the toolbar, click the **Save**  button to save the class setting definition.

Configuring a Custom Selector for Search Functionalities

Custom selectors can be configured in order to allow users to quickly and efficiently search for objects based on configured field filters and queries. Multiple custom selectors can be specified for an object class or object class stereotype and assigned to different user profiles in the user community, thus ensuring that only the relevant set of objects is available in the custom selector. The custom selectors that you configure can be specified to replace the standard search selectors in any of the following contexts:

- In the **Simple Search** functionality (`GenericSearch`)
- In page views in which an object selector is used to find an object in order to define a relationship. In this case, the configuration of the button operation used to open the selector will determine whether only a single object or multiple objects may be selected.

- In standard editors and custom editors in which an object selector is available for editor fields
- In workflows that are initiated for existing objects
- In configured reports for which the **Apply to Class** attribute is specified
- In any search selectors in which translated enumerations, release statuses, object states, or indicators are to be searched

Custom selectors assigned to a class setting will globally replace the standard selector except when a standard selector is required.

The **Selectors** node in the **Presentation** tab in Alfabet Expand allows you to create and configure custom selectors. Custom selectors may consist of a single page with one or two filter fields or they may contain multiple tabbed pages that allow users to search for objects in a specified object class using different search options.

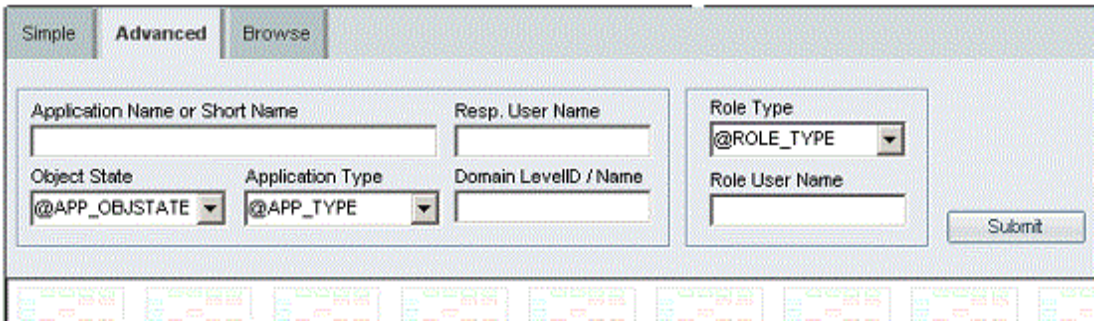


FIGURE: Example of a custom selector with multiple search options

In the example above, the custom selector is configured to include a **Simple** tab with a filter in which an application's name is entered to find the application as well as an **Advanced** tab with several filters that allow the search to be more defined in more detail. The filters allow, for example, the user to search for an application based on its object state, responsible user, and associated domain. The **Browse** tab is configured to allow the application to be searched by navigating through different explorer hierarchies including, for example, the organization structure containing the organization that owns the application or the domain model containing the domains that the application is associated with.

For each tabbed page configured for the custom selector, a search query must be written to find the objects to be searched. The search queries can be written in AQL or SQL. When a user opens the custom selector, the search query will be automatically triggered and the results immediately displayed. If the user modifies the search criteria, he/she must click the **Submit** button in order to update the results.

You can configure two optional buttons to be displayed in the filter areas. The **Collapse Filter** button allows users to close the filter area of the selector in order to increase the amount of space available to display the search results. The **Clear Search Patterns** button allows users to clear all filter settings with a single click.

Furthermore, an **Add to Clipboard** button is available in the toolbar as a convenient means to allow users to copy multiple objects and save them to the clipboard in order to easily define them as reference objects later in the user session. Data can be saved to the clipboard even if the user has read-only access to a standard page view or configured report. Objects saved to the clipboard will be available in custom selectors if the **Add My Objects Tab** attribute has been set to `True` for the class entry definition of the custom selector. The columns displayed in the **My Objects** tab are determined based on the **Preview Properties** attribute of the associated class setting for the base object class.

Once the custom selector has been defined, it can be assigned as the default selector for an object class or object class stereotype. The is done by selecting the custom selector in the **Selector Definition** attribute of the class setting of the relevant object class or object class stereotype.

Please note that new objects may be created in the context of class-based standard and custom object selectors with a minimal amount of navigation required. When ad-hoc adding of new objects is enabled, the **Add Missing <Class.Caption>** button will be available in the selector and uses can create objects needed to define a reference in the context of the selector. When the user clicks the **Add Missing <Class.Caption>** button, the configured functionality/explorer, configured report, or standard view in which the user can create the new object will open in a separate browser tab. Once the missing object has been created, the user can return to the original browser tab with the object selector and search in the selector for the new object. The the **View for Missing Object Button** attribute available for the class setting must be specified if the **Add Missing <Class.Caption>** button shall be available in the custom selector. For more information about specifying the **Selector Definition** and **View for Missing Object Button** attributes, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes.

The following information is available:

- Creating a Custom Selector
- Adding a Full-Text Search Tabbed Page to the Custom Selector
- Configuring the Buttons Displayed in the Custom Selector
- Configuring a Query for the Auto-Fill Function in the Search Field
- Implementing the Custom Selector in Alfabet
- Implementing a Custom Selector in the Simple Search Functionality
- Implementing a Custom Selector in a Page View
- Implementing a Custom Selector in a Standard or Custom Editor
- Implementing a Custom Selector in a Workflow
- Implementing a Custom Selector in a Configured Report
- Testing the Custom Selector
- Deleting a Custom Selector

Creating a Custom Selector

Please keep the following in mind when configuring the custom selector:

- For each custom selector, one class entry should be created per relevant object class. The class entry defines which object class is selectable in the custom selector. For example, in a **Browse** search, a hierarchy of application groups and applications can be displayed. If the class entry for which the search is configured defines the object class Application as searchable class, only applications can be selected in the hierarchy. Application groups will be displayed for structuring purposes, but they cannot be selected.

- A graphic view is automatically created for each class entry and displayed in the center pane. The graphic view will have a header panel and a presentation area. The graphic view allows you to configure multiple tabbed pages with their respective filter fields. A query must be defined for each tabbed page in order to find the specified dataset or browse hierarchy.

In order for an object class to be searchable in object selectors as well as the **Simple Search** functionality, you must ensure that the **Searchable** attribute is set to `True` for the relevant class settings implemented in the user profiles that require the selector. For more information about configuring the searchability of an object class in a class setting, see the section Configuring the Searchability of Alfabet Objects.

To create a new selector:

1) Go to the **Presentations** tab, right-click the **Selectors** folder, and select **New Selector**. The new selector is displayed below the **Selectors** folder.
2) Click the new selector to activate the attribute window and enter a name for the selector in the **Name** attribute. It is recommended that you provide a name that makes the selector easy to identify. This name will be displayed, for example, when configuring the implementation of custom selectors for search fields in standard editors, custom editors, or configured reports.
3) In the **Caption** attribute, enter the caption to display in the title bar of the selector. For example, the caption will be displayed in the Alfabet interface if the custom selector is implemented in a page view to add new objects in the relevant object class.
4) Next, define the object class for which the selector should be implemented. Right-click the selector and select **New Class Entry**. In the editor that opens, select the object class for which you want to create the selector and click **OK**. The user will only be able to select objects in the defined object class via the selector, even if objects of other object classes are displayed in the selector window (for example, objects classes that structure the search results).

   You can also select object class stereotypes in the editor. An object class stereotype must be defined in the format ObjectClassName:ObjectClassStereotypeName. When you define a class entry for a stereotype, the queries defined for the pages in the class entry must have a valid syntax. For more information, see below or see the section Enabling a Stereotype-Specific Search in Custom Selectors in the chapter Defining Queries.

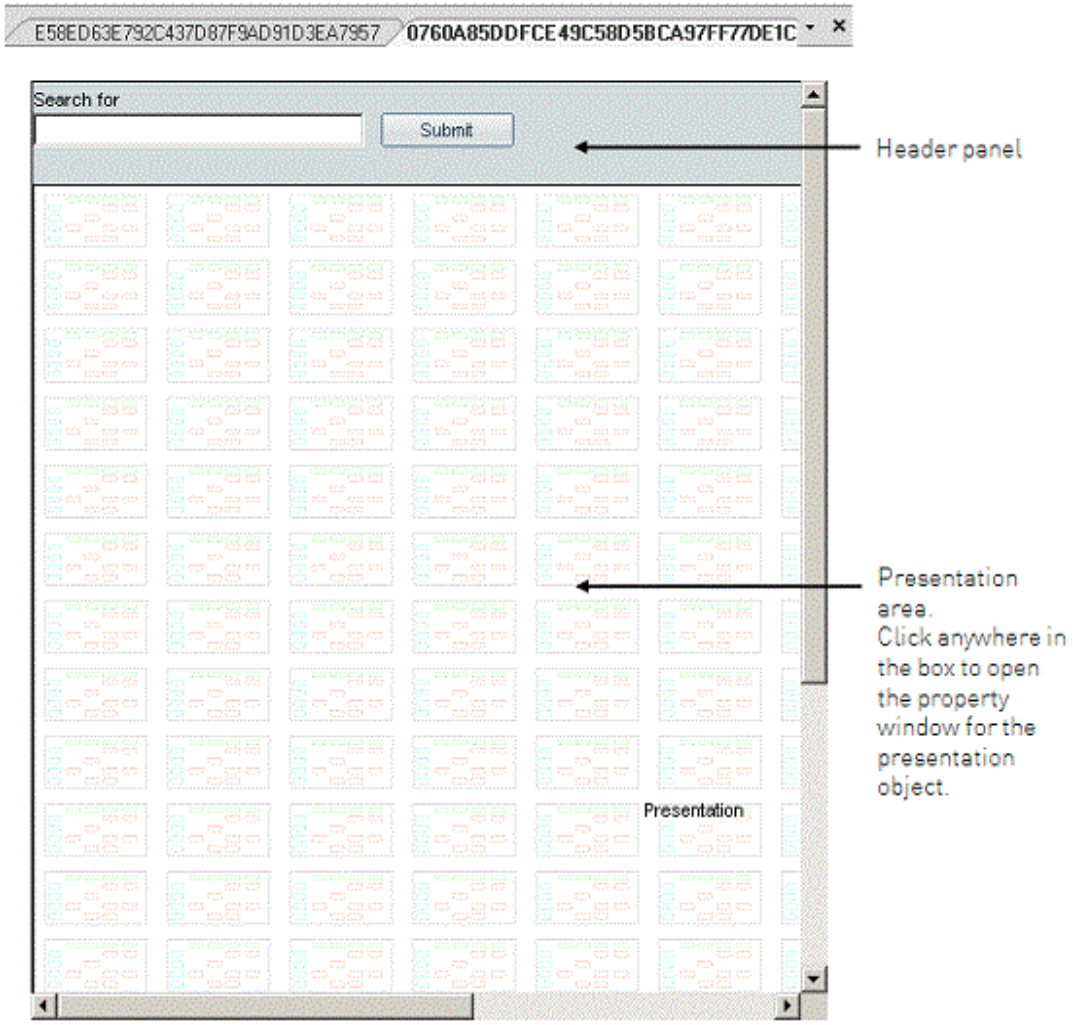
5) The class entry is displayed below the custom selector. Click the class entry and define the following attributes in attribute window:

- **Class**: Displays the object class selected in the selector when the class entry was created.
- **Max. Record Count**: Enter the number of objects found by the query that should be displayed. If all objects that are found by the query should be displayed, enter -1.
- **Add My Objects Tab**: Select `True` to include the **My Objects** tab in the custom selector. For custom selectors without tabs, a root tab will be added to the dataset of returned objects and the existing controls will be shown in a **Basic Data** tab. The **My Objects** tab will be dynamically added to the custom selector. For more information about the clipboard capability, see the section *Using the Clipboard Capability* in the reference manual *Getting Started with Alfabet*.

Please note the following about the clipboard functionality:

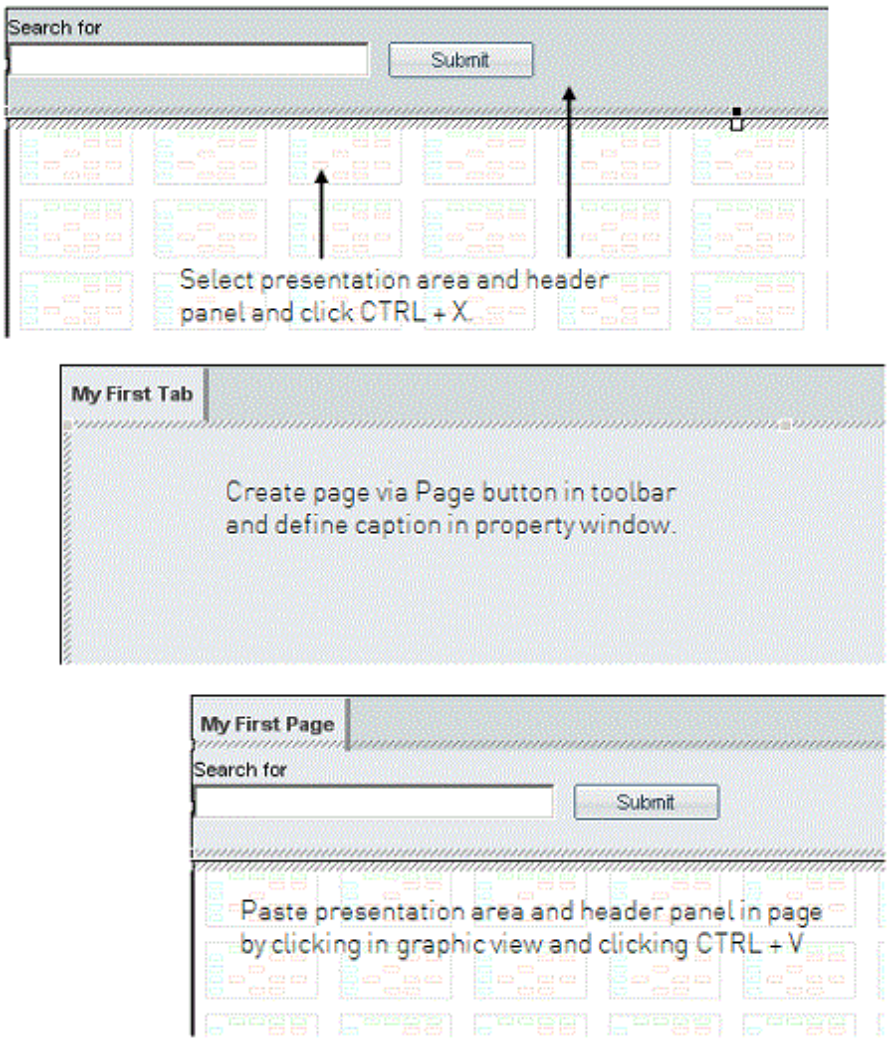
- Data added to the clipboard will remain in the clipboard during the running sub-session. If the user opens another browser tab to open Alfabet or if the user logs out, the data will be removed from the clipboard. Changing the user profile will not empty the clipboard.
- Each time a user adds objects to the clipboard within a running session, the data will be added to the existing data in the clipboard.
- The **My Objects** tab will only display the sub-set of objects in the clipboard that may be selected via the selector. For example, if components and applications have been added to the clipboard, only applications will be displayed in an application selector, only components will be displayed in a components selector, and none of the objects will be displayed in a device selector.

6) A presentation object is automatically generated for the class entry and is displayed below it. In the design editor in the center pane, the graphic view of the selector is displayed. The graphic view allows you to design the selector by adding tabbed pages, filters, as well as static text or other interface controls that might support the users when working with the selector. The graphic view automatically includes a header panel with a **Search for** field and a **Submit** button and a presentation area.



7) Next, define the tabbed pages required for the search selector. This will depend largely on the purpose of the search selector. If you require only a simple search selector with one view, then you can skip this step. However, if you need to define a search selector with multiple search options, then you will need to define a tabbed page for each search query that you plan to implement for the class entry.

   To create a page (tab) in the selector for each query you plan to define for the class entry.



- Click both the panel and presentation area while pressing the Shift key and cut the panel and presentation area out of the graphic view by clicking CTRL + X. The graphic view is now empty.
- Click in the graphic view to display the toolbar with interface controls that can be added to the graphic view. Click the **Page** control in the toolbar and click in the graphic view again. The page with a tab will be added. In the attribute window, enter a caption for the page's tab in the **Caption** attribute. Click the **Save** button to update the caption to the tab. Repeat until all pages have been created and their caption defined.
- For each tabbed page created, click in the graphic view and paste the panel by clicking CTRL + V. The panel will be added to the view. If the tabbed page is to be used to search a browse hierarchy, the header panel can be deleted.
- Create a page for all required search queries, pasting the header panel in each new tabbed page.

Please note the following:

- Do not place two presentation areas on top of one another.
- Do not right-click a tab and select **Cut** or **Delete**.

- If you have created multiple tabbed pages and want to move a tabbed page to the left or right, you must select all interface controls in a tabbed page by pressing the Shift key. Cut the panel and presentation area out of the graphic view by clicking CTRL + X. The graphic view is now empty. Right-click in the graphic view and select either **Move to Left** or **Move to Right**. Once the tab is in the correct position. Select CTRL + V to place the interface elements back in the tabbed page.

8) Next, you should add any additional filter fields required for each tabbed page. The filter fields are defined by adding the relevant interface element to the panel. For more information about the configuration of individual filter fields, see the section Defining Filters for Configured Reports and Selectors in the chapter Defining Queries.

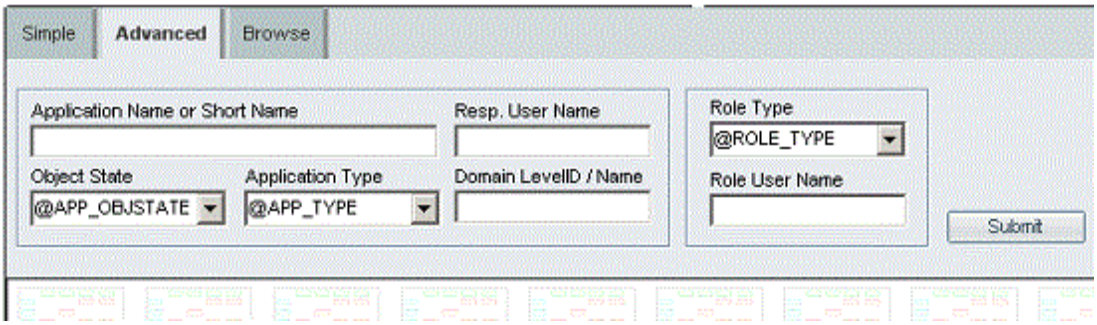


FIGURE: Example of a complex page with many filter fields

Please note the following:

- Each interface element that you add as a filter field should have a **Caption** attribute defined so that users can identify the purpose of the filter.
- Each filter field may have a tooltip explaining the purpose of the filter. Users will see the tooltip when they mouse over the field caption. The tooltip is defined in the **Hint** attribute.
- The filter fields you add to the tabbed page must be referenced by the search query that you will define for the page. Therefore, the filter fields MUST have the **Name** attribute defined and the syntax of the name must be written according to the conventions of the query language you intend to write the query in (AQL or SQL).
- In contrast to configured reports, filter controls cannot be automatically generated. They must be manually defined or copied from standard reports and pasted to the graphic views for custom selectors. References to parameters in filter controls are handled similarly as in configured reports. For more information about the procedure to create and specify filters, see the section Defining Filters for Configured Reports and Selectors in the chapter Defining Queries.
- If you plan to include a query for an explorer hierarchy (for example, similar to a browse search), you do not need to define a filter field. The field in which users will select the explorer will be automatically generated via the query definition.
- If the search selector is being defined to search for translated enumerations, release statuses, object states, or indicators, you must configure a Combo Box interface control for the search filter.

You can write the query first and then create a filter for each filter referenced in the query. Please note that if a filter referenced in the query is not defined, an error will occur.

9) Click the presentation area in the graphic view to open the attribute window. In the **Sub-Type** attribute, select `DataSet`.

10) Next, you need to define the search query that determines the set of objects to be searched in the tabbed page. This could be, for example, a simple data set determined by one or two filters, a complex data set determined by multiple filters, or an explorer browse hierarchy. A search query must be specified in the **XML Definition** attribute for each page:

    Ensure that you have clicked in the presentation area so that the attribute window displayed is for the graphic view. In the attribute window, click the **Browse** button in the **XML Definition** field. An editor opens in which you can define the search queries needed to find the objects to be returned when the search is triggered. The **XML Definition** attribute is filled automatically with the string `<DataSetDef Query="" />`. Define the query in the attribute `Query`.

    The search query may be specified in the Alfabet query language or native SQL and must match the following criteria:

- The query must return objects of the class defined in the class entry. In an Alfabet query, the class must be defined as a FIND class. In native SQL, the `REFSTR` of the class must be placed as the first `SELECT` property to be returned by the query.
- When you define a class entry for an object class stereotype, the queries defined for the tabbed pages in the class must return the stereotype of the objects in a column of the result dataset. The query must also include the instruction `SetRowsStereotypeIndex` ("ColumnName") that specifies the column in the dataset listing the stereotype information. For more information, see the section Enabling a Stereotype-Specific Search in Custom Selectors in the chapter Defining Queries.
- The classes specified for the custom selector must be valid according to the meta-model for the context in which the selector is used. An error message will be displayed in the user interface if the class specified for the custom selector does not match the context that is specified via, for example, a relevant button operation or Edit Search interface control.
- If you have defined filter fields for the selector page, the query must include a WHERE clause referencing the filter field via a parameter definition. When the query is executed, the parameter is substituted with the current value entered in the filter field. For more information about defining queries that reference filters, see Defining Filters for Configured Reports and Selectors in the chapter Defining Queries.
- The parameter `BASE` that references the current object can only be used if the selector opens after selecting a base object. The base object can be the object that the user is currently working with or the object the user has currently selected in a table in a view. For example, a selector that allows an application to be selected in order to assign it to an application group can refer to the current application group with the parameter `BASE`. Selectors that are used to perform a search in the **Simple Search** functionality cannot refer to a base object because a search is not performed in the context of working with a specific object. Selectors that are opened via an editor cannot refer to a base object because when the editor is used to create a new object, the base object is under creation and not available in the database yet.

For general information about writing Alfabet queries, see the chapter Defining Queries.

For general information about the rules that apply when using native SQL for configuring Alfabet, see the section Defining Native SQL Queries in the chapter Defining Queries.

Please note that the definition of a query inside an XML element requires a special syntax described in the section Defining Filters for Configured Reports and Selectors in the chapter Defining Queries.

For example, the following query is written for a simple search with only one filter field. This query is written in Alfabet query language and references the filter field using syntax required in AQL (for example,:`NameOrSN`):

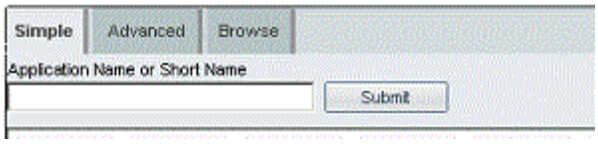


FIGURE: Query below specifies the field displayed in the selector

```
<DataSetDef Query="ALFABET_QUERY_500 FIND Application

    WHERE (OR
        Application.Name LIKE:NameOrSN
        Application.ShortName LIKE:NameOrSN)
  SHOW Application.Name Application.ShortName
  Application.Version Application.Status" />
```

For more information about writing queries, see the chapter Defining Queries.

For example, the following query is written for a complex search with multiple filter fields. This query is written in native SQL and references the filter field using the syntax required in SQL (for example, `@APP_NAM`, `@APP_OBJSTATE`, `@APP_TYPE`, etc):

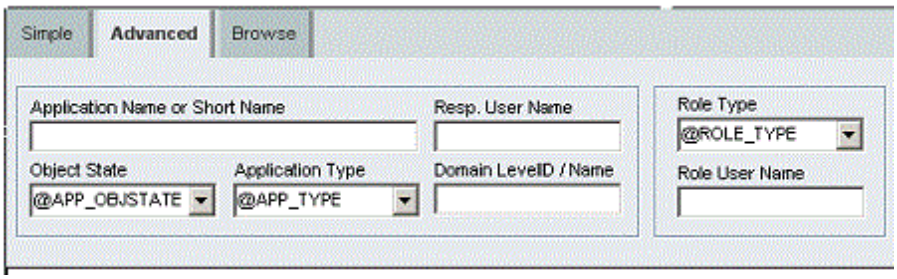


FIGURE: Query below specifies the filters displayed in the selector

```
<DataSetDef Query="
   SELECT app.REFSTR, app.ID,
      app.NAME + ' ' + app.VERSION AS 'Name and Version',
      app.SHORTNAME AS 'Short Name',
      app.OBJECTSTATE AS 'Object State',
      app.APPLICATIONTYPE AS 'Appl. Type',
      per.TECH_NAME AS 'Resp. User'
      dom.LEVELID + ' ' + dom.NAME AS 'Domain'
   FROM APPLICATION app
      LEFT JOIN PERSON per ON app.RESPONSIBLEUSER =
      per.REFSTR
      LEFT JOIN DOMAIN dom ON app.DOMAIN = dom.REFSTR
```



```
WHERE (app.NAME LIKE(@APP_NAME) OR app.SHORTNAME
LIKE(@APP_NAME))
    AND app.OBJECTSTATE = @APP_OBJSTATE
    AND app.APPLICATIONTYPE = @APP_TYPE
    AND per.TECH_NAME LIKE(@APP_USER_NAME)
    AND (dom.NAME LIKE(@APP_DOM_NAME) OR dom.LEVELID
    LIKE(@APP_DOM_NAME))
    AND (EXISTS(SELECT r.REFSTR
        FROM ROLE r, PERSON p
        WHERE r.OBJECT = app.REFSTR
        AND r.ROLETYPE = @ROLE_TYPE
        AND r.RESPONSIBLE = p.REFSTR
        AND p.TECH_NAME LIKE(@ROLE_USER_NAME))
    OR ('dummy' = @ROLE_TYPE AND 1=1))
" />
```

11) If the tab is to be used for a browse hierarchy:

- Click the presentation area in the graphic view to open the attribute window. In the **Sub-Type** attribute, select `ExplorerObject`.
- The string in the **XML Definition** attribute must be replaced with `<Explorers> </Explorers>`.
- Within the XML element ***Explorers***, there may be an arbitrary number of explorer definitions. If the tab is to be used to search a browse hierarchy, the header panel can be deleted.
- Within the XML element ***Explorers***, each explorer must be defined in an XML element ***ExplorerDef***. The required syntax for the definition of an explorer is explained in detail in the chapter Configuring Standard Business Functions and Custom Explorers.

When multiple explorers are defined, the user will see the explorers as sub-nodes in an **Explorer** root node when he/she selects the **Browse** search:

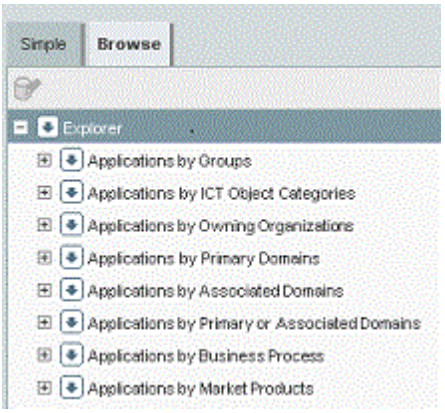


FIGURE: Multiple browse hierarchies to search for applications

For reasons of space, only the first explorer hierarchy in the example above is defined in the query below:


```
<Explorers>
    <ExplorerDef
        Name="Applications by Groups"
        BaseClasses = "ApplicationGroup,Application" >
        <Query ClassName="ICTObjectCategory"
        Query="BelongsTo IS NULL" />
        <ClassEntry ClassName="ICTObjectCategory"
        ShowProps="Name" SortProps="Name" >
            <Query ClassName="ICTObjectCategory"
            Query="BelongsTo Contains:BASE" />
            <Query ClassName="ICTObject" Query="Category
            Contains:BASE" />
        </ClassEntry>
        <ClassEntry ClassName="ICTObject"
        ShowProps="Name" SortProps="Name" >
            <Query ClassName="Application" Query="(AND
            ICTObject Contains:BASE VariantOf IS NULL)"
            />
        </ClassEntry>
        <ClassEntry ClassName="Application"
        ShowProps="Name,Version"
        SortProps="Name,Version" >
            <Query ClassName="Application"
            Query="VariantOf Contains:BASE"
            /></ClassEntry>
    </ExplorerDef>
</Explorers>
```

12) Once a query has been defined for each tabbed page in the custom selector has been defined, click the **Save**  button in the toolbar.

13) Below the class entry in the explorer, click the presentation object  . In the toolbar above the attribute grid, click the **Events**  button to open the event window. In the `OnGetExplorerNodes` event, select `GetExplorerNodes`. Click the **Save**  button in the toolbar and click the **Attributes**  button to return to the attribute window.

14) Click the class entry node below the selector node in the explorer tree and in the **Max. Record Count** field, enter the number of objects found by the query that should be displayed. If all objects that are found by the query should be displayed, enter `-1`.

15) You should now test the layout and search results of the custom selector. To do so, right-click the selector node in the explorer and select **Review Selector**. You can view the selector in the context of the Alfabet interface. If the results are not as expected, you can edit the XML definition until the output of the selector meets your expectations.

Adding a Full-Text Search Tabbed Page to the Custom Selector

If a full-text search should be available in the custom selector, you must explicitly create an additional tabbed page entailing the full-text search functionality.

You must ensure that a search group has been created for the relevant object class in the XML object ***SearchManager***. For more information, see the section Configuring an Object-Centric Search Group.

To configure a tabbed page with the full-text search functionality:

1) Add the page (tab) to the custom selector by clicking the **Page** interface control in the toolbar and clicking in the graphic view again. The page with a tab will be added. In the attribute window, enter a caption for the page's tab in the **Caption** attribute. Click the **Save** button to update the caption to the tab.
2) Create a **Panel** interface control in the tab. Click the panel and set the **Dock** attribute to `Top`.
3) Add a presentation object to the page by clicking the **Presentation Object** interface control in the toolbar and clicking in the graphic view below the panel.
4) Click the Presentation area in the graphic view to open the attribute window. Define the following attributes:

- **SubType**: Select `DataSet`.
- **Refresh on Submit**: Select `True`.
- **Dock**: Select `Fill`.

A validation mechanism checks for inconsistencies in the layout and docking definition of interface controls in guide views, configured reports, and other configured views. Please note that an error message will be displayed when the user attempts to open an incorrectly configured view if two or more interface controls have no docking defined in the view. The error message will provide detailed information about the faulty configuration.

- **XML Def**: Click the **Browse** button to open the editor in which you can define the relevant search group and the fuzziness factor for the full-text search. The **XML Def** attribute is filled automatically with the string `<DataSetDef Query="" />`. Delete the string and enter `<FullTextSearchDef SearchGroup="<SearchGroupName>" Fuzziness="<NumberFrom0To10>" />` and click the **Save** button to save the definition.

For example:

```
<FullTextSearchDef SearchGroup="ApplicationSearch"
Fuzziness="3" />
```

5) Add the edit field to the custom selector by clicking the **Edit** interface control in the toolbar and clicking in the graphic view again. In the attribute window, define the following attributes for the Edit interface control:

- **Name**: Enter `@aName`.
- **Caption**: Enter a caption for the page's tab.

6) Click the **Save** button to update the Edit interface control.
7) Next, click the **Submit** button next to the edit field in order to activate its attribute window. In the **Submit** attribute, select `True`. Click the **Save** button to save the definition.
8) Click the Presentation area and click the **Events** icon above the attribute window. In the **OnGetPresentationData** attribute, select `OnGetFullTextData`.
9) Click the **Save** button to save the definition.

Configuring the Buttons Displayed in the Custom Selector

Two buttons may be displayed in the header panel of a custom selector. The **Collapse Filter** button allows users to close the filter area of the selector in order to increase the amount of space available to display the search results. The **Clear Search Patterns** button allows users to clear all filter settings with a single click.

You can optionally configure individual filter fields to be excluded from the **Clear Search Patterns** functionality. To prevent a specified filter field to be cleared:

1) Click the filter field that you want to exclude from the **Clear Search Patterns** functionality to open its attribute window.
2) In the **Allow Clear Value** attribute, select `False` to prevent the value set by a user from being cleared when the user clicks the **Clear Search Patterns** button.

To include the button in the header panel of the custom selector:

1) Click in the header panel of the graphic view to open its attribute window. Define the following attributes:

- **Has Clear Button**. Select `True` to implement the **Clear Search Patterns** button.
- **Collapsible**. Select `True` to implement the **Collapse Filter** button

2) In the toolbar, click the **Save** button to save your changes.

Configuring a Query for the Auto-Fill Function in the Search Field

An auto-fill function is automatically available that provides potential matches when users enter text in a search field in standard selector. Potential matches for the text entered by the user are displayed in a drop-down list and must be explicitly selected to be entered in the search field. An auto-fill function may also be configured for custom selectors. This is done via an Alfabet query or native SQL query that finds potential matches for the selector field.

The auto-fill function is not available for a `Person` selector that is configured to access an external source. Please note that the auto-fill function for standard selectors is defined via the **Image Properties** attribute of the relevant class setting. For more information, see the section Creating a Custom Class Setting for a Protected Object Class or Object Class Stereotype.

1) Click the custom selector to activate the attribute window.

2) Next, you need to define the search query that determines the set of objects that may qualify as potential matches for the auto-fill function in the search field in the custom selector. In the **Lookahead Query** attribute, click the **Browse** button to open the editor in which you can define the query. Please note the following regarding the search query definition:

- The search query may be specified in the Alfabet query language or native SQL. The query must be written in SQL if multiple object classes are to be searched. Please note however that a query in native SQL requires that the `REFSTR` of the class is placed as the first `SELECT` property. For more information about syntax and guidelines for defining queries, see the chapter Defining Queries.
- The number of results to display in the drop-down list can be defined in the query. This number should not be excessively high.
- The parameter `SEARCH_PATTERN` can be used in the query. The parameter `SEARCH_PATTERN` is required to refer to the value entered in the filter field.
- The variables `CURRENT_USER`, `CURRENT_LANGUAGE`, `CURRENT_MANDATE`, and `TODAY` can be used in the query.. Please note that this does not enable the use of `BASE` variables. The selector does not have any knowledge about the underlying view, editor or presentation object.
- Show and sort properties define the visualization of the matching data in the drop-down list.

For example, the following query is written in native SQL. The auto-fill function will display the first 10 organizations based on the organization stereotype `External Organization` found matching the letters entered in the search field. The matches will be displayed alphabetically based on their names:

```
SELECT TOP 10 REFSTR, NAME
FROM ORGAUNIT
WHERE NAME LIKE @SEARCH_PATTERN
   AND STEREOTYPE = 'ExternalOrganization'
ORDER BY NAME
```

3) Save the query by clicking the **OK** button.

4) In the toolbar, click the **Save** button to save your changes.

Implementing the Custom Selector in Alfabet

The custom selectors that you configure can be specified to replace the standard search selectors or object selectors in any of the following contexts:

- In the **Simple Search** functionality (`GenericSearch`).
- In page views in which an object selector is used to find an object in order to define a relationship

- In standard editors and custom editors in which an object selector is available for editor fields
- In workflows that are initiated for existing objects
- In configured reports for which the **Apply to Class** attribute is specified

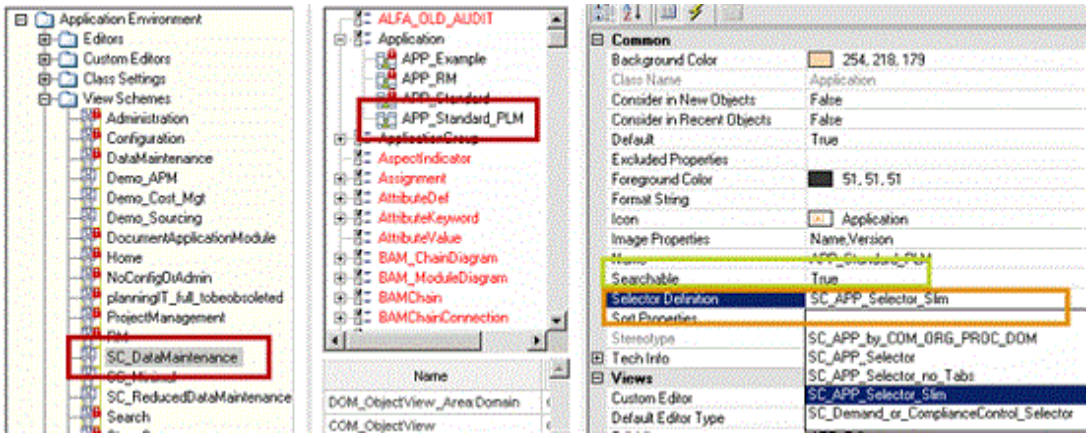
The following information is available:

Implementing a Custom Selector in the Simple Search Functionality

To implement a custom selector for the **Simple Search** (`GenericSearch`) functionality, you must explicitly specify the custom selector for each relevant class setting defined for the view scheme assigned to the relevant user profile.

To implement the custom selector in the **Simple Search** functionality:

1) Go to he **Presentation** tab, click the relevant view scheme and select **Edit View Scheme**.
2) In the center pane, select the class setting that you want to define. You will see the attribute window in the right pane.



3) In the attribute window, ensure that the **Searchable** attribute is set to `True` to specify that users accessing Alfabet can search the object class.
4) In the **Selector Definition** attribute, select the custom selector that should be displayed for the selected object class in the **Simple Search** functionality.

> All custom selectors that have been configured for your solution will be displayed. You must ensure that you select a custom selector that is meaningful for the view and defined for the relevant object class.

5) In the toolbar, click the **Save** button to save the class setting definition.

Implementing a Custom Selector in a Page View

You can implement a custom selector to replace the standard object selector in the page views accessible via a specified user profile.

Only a subset of the standard Alfabet page views can be configured. for some standard page views, the selector definition is fixed, and the configuration described below is not available.

To configure the custom selector for a page view:

1) Go to the **Admin** tab, right-click the user profile that you want to configure and select **Configure User Profile**. The Alfabet interface opens.
2) Navigate to the relevant view in which you want to specify the custom selector.
3) Click the **Configure** button in the upper right corner of the view to open the **View Configuration** editor.
4) In the **Custom Selector** column, select a custom selector in the drop-down menu in order to replace the standard custom selector displayed in the **Default Selector** column.

   All custom selectors that have been configured for your solution will be displayed. You must ensure that you select a custom selector that is meaningful for the view and defined for the relevant object class.

5) Click **OK** to close the **View Configuration** editor and save the selector definition.
6) Close the browser and reopen it to see the changes you specified in the **View Configuration** editor.
7) Repeat for other page views that should be configured for the selected user profile.
8) In the toolbar in Alfabet Expand, click the **Save** button to save the configuration

Implementing a Custom Selector in a Standard or Custom Editor

You can implement a custom selector to replace the standard object selector in standard and custom editors available in object views and page views that are accessed by a specified user profile.

To configure the custom selector for a standard or custom selector:

1) Go to the **Admin** tab, right-click the user profile that you want to configure and select **Configure User Profile** in order to access Alfabet. The Alfabet interface opens.
2) Navigate to the relevant object view or page view with the editor for which you want to specify the custom selector.
3) Click the **Edit** button in the view to open the editor.

4) In the upper right corner of the editor, click the **Configure** button.

5) In the **Custom Selector** column, select a custom selector in the drop-down menu in order to replace the standard custom selector displayed in the **Default Selector** column.

> All custom selectors that have been configured for your solution will be displayed. You must ensure that you select a custom selector that is meaningful for the view and defined for the relevant object class.

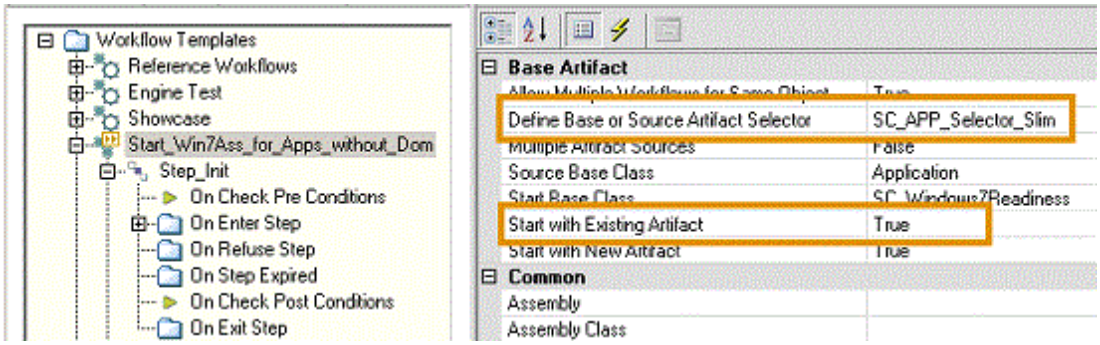
6) Click **OK** to save the selector definition.

7) Repeat for other editors that should be configured for the selected user profile.

8) In the toolbar in Alfabet Expand, click the **Save** button to save the configuration

Implementing a Custom Selector in a Workflow

A custom selector can be implemented in place of the standard selector for a workflow that is specified to start with an existing object.



To implement the custom selector in a workflow:

1) Go to the **Workflows** tab, click the relevant workflow template .

> The workflow template must have the state `Plan` in order to be edited. To change the state of the workflow template, right-click the workflow template and select **Set State to Plan**.

2) In the attribute window, ensure that the **Start with Existing Artifact** attribute is set to `True` to specify that the workflow should begin with existing objects.

3) In the **Define Base or Source Artifact Selector** attribute, select the custom selector that should be implemented for users to select the base object for the workspace.

4) In the toolbar, click the **Save** button to save the workflow template definition.

Implementing a Custom Selector in a Configured Report

A custom selector can be implemented for the standard object selector in configured reports for which the **Apply to Class** attribute is specified. Custom selectors can also be implemented for configured reports for which filters containing Edit Search interface controls are implemented. Depending on how the filter field is generated, custom selectors can be implemented using one of the following methods:

- Filter fields that are automatically added when the **Apply to Class** attribute of the configured report is set: The custom selector will be implemented if it is defined in the **Selector Definition** attribute of the class setting of the object class defined in the **Apply to Class** attribute of the configured report.
- Filter fields in configured reports of the type `Query`: The custom selector will be implemented if it is defined in the **Selector Definition** attribute of the class setting of the object class targeted by the Edit Search interface control.
- Filter fields in configured reports of the type `Native SQL` or `Custom`: Open the configured report view and click the Edit Search interface control for which you have defined a custom selector. In the attribute window, enter the name of the custom selector in the attribute **Object Selector**.

Testing the Custom Selector

You can test the custom selector to ensure that the query is returning the results you expect. To do so, right-click the selector and select **Review Selector**. The custom selector is opened in a browser tab. Test the configuration of the custom selector as needed.

Deleting a Custom Selector

Before a custom selector is deleted, you should check to see if the custom selector is implemented in other configuration objects such as custom editors or configured reports. To do so, right-click the custom selector and click **Show Usage**. A window will be displayed showing the configuration objects using the selected custom selector. For more information regarding the **Show Usage** functionality, see the section Using the Show Usage Functionality in the chapter Getting Started with Alfabet Expand.

1) Go to the **Presentation** tab and expand the **Selectors** folder. You will see all existing custom selectors listed in the tree.
2) Right-click the custom selector that you want to delete and select **Delete**.
3) Confirm the warning by clicking **Yes**. The custom selector is deleted from the explorer.

Configuring the Full-Text Search Capability

Alfabet supports full-text search within data in the Alfabet database. Full-text search is based on search groups defined by a solution designer in the XML object ***SearchManager***. A search group defines a scope for the full-text search. For each search group, the user can search in defined object class property values of all or a subset of objects of defined object classes. The full-text search can be conducted in all secondary languages supported by Alfabet. To enable full-text search based on a configured search group, a search index must be created for the search group for each supported language. Search indexes must be updated in regular intervals to include recent changes to the data in the Alfabet database.

There are two different methods of full text search in Alfabet:

- The global *Full-Text Search* functionality provides access to search groups defining full-text search for objects of an object class or a set of object classes such as applications and ICT objects or business processes and organizations. The search index for this search is created and updated centrally by an administrative user as a separate functionality. For each combination of search group and supported language a separate index is generated.
- Object-centric full-text search can be conducted via the **Full-Text Search** page view in the **Search** workspace available in an object profile of an object class. Search groups for object centric full-text search are configured to find objects related to the object the user is currently working with when accessing the **Full-Text Search** page view. The search index for an object centric full-text search is created and updated by the user conducting the search directly on the **Full-Text Search** page view available in the object profile of the object class. For each combination of search group, base object and supported language a separate index is generated.

  The following functionalities rely on object-centric full-text search groups. For these special use cases, the definition of the object-centric search group is described separately in the scope of the documentation of the functionality, to take functionality specific configuration requirements for the search groups into account:

  - An object-centric search group for the object class `ALFA_REPORT` is required to activate semantic search in the **Business Problem Statement** attribute of configured reports for the AlfaBot functionality. Semantic search enhances the ability of the AlfaBot to find a configured report that a user wants to access via the AlfaBot.
  - Object-centric search groups can be defined for the glossaries defined in the *Glossary Functionality* (`Glossary`) in the **Search** module. If a glossary is defined, a search group must be configured in order to search the glossary. For more information about the configuration of a search group for the **Glossary** functionality, see the section Configuring the Glossary Search Functionality.
  - Object-centric search groups are required for the implementation of the domain glossary defined in the *Domain Glossary Page View* (`DOM_Glossary`) in Alfabet. If a domain glossary is defined, a search group must be available in order to search the glossary. For more information about the configuration of a search group for domain glossaries, see the section Configuring the Domain Glossary Capability in the chapter Configuring Domain Models and Domain Planning.

The following configuration is required to activate full-text search for the full-text search functionalities:

- Full-text search indexes are not part of the Alfabet database but stored on the local file system. A search index directory must be defined for storage of the search indexes on the local file system.

This is done by a system administrator in the server alias of the Alfabet Web Application. For more information, see the section *Creating a Server Alias for the Alfabet Web Application* in the reference manual *System Administration*.

- Full-text search must be activated and general search parameters for all search groups must be specified in the XML object ***SearchManager***. For more information, see the section Specifying General Parameters for Index Creation.
- Search groups must be defined for the full-text search in the XML object ***SearchManager***:
  - For information about the definition of global search groups for the global Full-Text Search functionality, see the section Configuring a Global Search Group.
  - For information about the definition of object-centric search groups for object-centric full-text search, see the section Configuring an Object-Centric Search Group.
  - For information about the definition of object-centric search groups for the **Glossary** functionality, see the section Configuring the Glossary Search Functionality.
  - For information about the definition of search groups for the **Domain Glossary** functionality, see the section Configuring the Domain Glossary Capability in the chapter Configuring Domain Models and Domain Planning.
- The required views must be made available to the user via the user profile configuration. The following views or functionalities are required for the different full-text search functionalities:
  - **Global full-text search**: The **Full-Text Search** (`GlobalFullTextSearch`) functionality must be added to the user profile of the users that shall conduct the search. For more information about adding functionality to a user profile, see Making Functionalities Accessible to a User Profile in the chapter Configuring User Profiles for the User Community.
  - **Object centric full-text search including glossary and domain glossary search**: Object centric full-text search is limited to the object classes with a **Search** (`SRCH_GenericObject`) workspace with a **Full-Text Search** page view in the standard object view. For an overview of object classes with activated object-centric full-text search, see *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*. The workspace will only be visible to the user if the object class is the base class for an object centric search group in the XML object **SearchManager**. Therefore, it is not required to generate a custom object view to actively exclude the workspace if no object centric search shall be provided. The **Search** workspace is available for the object class Glossary to conduct full-text search for glossaries and for the object class Domain to conduct full-text search in domain glossaries and in any object centric search group for the base class Domain.
- For the global full-text search only, one of the functionalities to create and update the full-text search index must be configured and executed in regular intervals. The following options are available to update the search indexes for global search groups:
  - An administrative user can create and update the full-text search indexes for globally-defined search groups in the administrative **Full-Text Search** functionality on the Alfabet user interface. The functionality allows creation and update of indexes for all supported languages to be performed.

The view `ADMIN_FullTextSearch` must be added to the user profile of the administrative user to enable manual creation and update of the globally-defined full-text search. For information about how to manually create and update search indexes, see the section *Creating an Index for the Full-Text Search* in the reference manual *User and Solution Administration*.

- Via the **Job Schedule** functionality on the Alfabet user interface, administrative users can create job schedules for the automatic creation and update of search indexes for globally-defined full-text search groups in defined time intervals. A job schedule triggers the creation and update of the search index of a single full-text search group. Please note that the job schedule functionality currently supports only definition of full-text search indexes for the language English (United States).

For information about how to implement and configure the **Job Schedule** functionality, see the section Activating the Job Schedule Functionality. For information about how to schedule full-text search index update via a job schedule, see the section *Creating a Job Schedule for the Generation of a Full-Text Search Index* in the reference manual *User and Solution Administration*.

- Via the executable `FullTextSearchUtil.exe`, system administrators can configure a batch process to regularly create and update search indexes for globally-defined full-text search groups. The command line tool can create and update indexes for all supported languages. It can either update an index for a defined search group or for all globally-defined full-text search groups.

For more information about the configuration of the batch process, see the section *Updating Indexes with the FullTextSearchUtil.exe* in the reference manual *System Administration*.

The following information is available:

- Specifying General Parameters for Index Creation
- Configuring an Object-Centric Search Group
- Configuring a Global Search Group

Specifying General Parameters for Index Creation

Regardless of whether an object-centric search or a global search has been configured, a few general configurations are required in order to ensure that indexes can be created for all search groups that are defined in the XML object ***SearchManager***. This includes the activation of the search index and the configuration of the maximum number of matching words to display for the search results.

To edit the XML object ***SearchManager***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder. You will see all XML objects that can be edited.
2) Right-click the XML object ***SearchManager*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) If not already defined in the XML object, add the root element `SearchManager` to the XML object. If defined, edit the existing root element:

```
<SearchManager HighlightCount = "20000" AutoWildCard="false">
    ...
</SearchManager>
```

4) In the XML attribute `HighlightCount`, enter an integer to define the maximum number of matching words to highlight in the display of search results. The default count is 10,000.

The XML attribute `AutoWildCard` has no effect on the full-text search. For information about the configuration of the XML attribute `AutoWildCard`, see the section Configuring the Wildcard for Standard and Custom Search Functionalities.

5) In the toolbar, click the **Save** button to save the XML definition.

Configuring an Object-Centric Search Group

An object-centric search allows users to enter search terms to find objects related to a base object, like for example applications that are assigned to a selected application group. This search is indexed and executed by means of the *Full-Text Search* page view (`SRCH_FullTextSearch`) in the workspace **Search** (`SRCH_GenericObject`) for the base object.

An object centric search group must be configured if any of the following is to be implemented in your Alfabet solution:

- The implementation of one or more glossaries defined in the *Glossary Functionality* (`Glossary`) in the **Search** module. If a glossary is defined, a search group must be configured in order to search the glossary. For more information about the configuration of a search group for the **Glossary** functionality, see the section Configuring the Glossary Search Functionality.
- The implementation of the domain glossary defined in the *Domain Glossary Page View* (`DOM_Glossary`) in Alfabet. If a domain glossary is defined, a search group must be configured in order to search the glossary. The configuration of the search capabilities for the domain glossary are addressed separately in the section Configuring the Domain Glossary Capability in the chapter Configuring Domain Models and Domain Planning.

To edit the XML object ***SearchManager*** for an object-centric search:

1) Click the **Presentation** tab in the tab bar and expand the **XML Objects** folder by clicking the + symbol. You will see all XML objects that can be edited.
2) Right-click the XML object ***SearchManager*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) Enter an XML element `SearchGroup` with the required child XML elements and XML attributes as child XML element of the root XML element `SearchManager` and define the XML attributes as needed. The table below displays the XML attributes that should be edited for an object-centric search in the XML object ***SearchManager***.

The following example displays a code for an object-centric search group searching within all applications in an application group:

```
<SearchGroup Name="Applications in Application Group"
BaseClass="ApplicationGroup" Type="Object">

   <ClassEntry ClassName="Application"
   ShowProps="Application.Name,Application.Version"

   ExportProps="Application.Name,Application.Description"
```



```
    ImageProps="Application.Name,Application.Version" />
    <Query Name="Applications in Group"
    ClassName="Application"
    Query="ALFABET_QUERY_500 FIND Application WHERE
    Application.ApplicationGroups CONTAINSOR:BASE" />
</SearchGroup>
```

XML Element (bold) / XML Attribute	Explanation
SearchGroup	
`Name`	Enter text that defines the name of the search group. This is the name that users will see when they select the search group in order to execute a search.
`Type`	Enter `Object` to define an object-centric search. **NOTE**: This XML attribute is mandatory for the definition of an object-centric search. If the XML attribute is not defined, the search is by default a global search.
`BaseClass`	Enter the name of the object class that is the base class for the search. **EXAMPLE**:To search for applications in an application group, enter `ApplicationGroup` for the XML attribute `BaseClass`. **NOTE**: Please note the following: • This XML attribute is mandatory for the definition of an object-centric search. • Only an object class that is predefined to provide the **Search** workspace in their object profile can be defined as base class. For an overview of the object classes for which an object-centric full-text search can be configured via the XML object ***SearchManager***, see *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.
`Profiles`	Enter the user profiles that have access permissions to the search group. If no value is specified for the XML attribute `Profiles`, then all configured user profiles will have access permissions to the search group.
ClassEntry	A definition for the XML element ***ClassEntry*** should be created for the object class that is to be found by the Alfabet query defined for the object-centric search.

XML Element (bold) / XML Attribute	Explanation
ClassName	Enter the name of the object class associated with the base class that is to be queried. **EXAMPLE**: To search for applications in an application group, enter ApplicationGroup for the BaseClass attribute and Application for the XML attribute ClassName. The **Full-Text** tab will be automatically added to the standard object selector implemented for the object class specified in the ClassName attribute. If the **Full-Text** tab is required for a custom selector, additional configuration is required. For more information, see the section Configuring a Custom Selector for Search Functionalities.
ExportProps	Enter a comma-separated list of object class properties to define all object class properties of the object class to be indexed. Each object class property must be defined as: ObjectClassName.PropertyName The XML attribute ExportProps must include all object class properties defined in the XML attribute ShowProps and XML attribute ImageProps. The export properties are the object class properties written into the full-text search index file. The index file will only contain values for a language version if a translation exists in the Alfabet database. That means that the index only contains a value if the object class property is configured to be translatable and a translation is available for the current object. If full-text search shall be conducted in multiple languages, you should make sure that all object class properties that shall be included into the search are translatable and translations are provided. Please note that if the subset of object class properties defined both in the XML attributes ExportProps and ImageProps is not translated, the information displayed about the object in the headline of the record-by-record view as well as in the data set view will fall back to the primary language to ensure that the user can identify the object. Nevertheless, these object class properties are excluded from search.
ImageProps	Enter a comma-separated list of object class properties to create a concatenation that serves as the title line for each search result. The sequence of the object class properties listed in the XML attribute ImageProps determines the sequence displayed in the concatenated title. Each object class property must be defined as: ObjectClassName.PropertyName

XML Element (bold) / XML Attribute	Explanation
	The object class properties defined in the XML attribute `ImageProps` are read from the Alfabet database when the index is created. Results of a full text search can be displayed either record-by-record or as a data set. In the record-by-record view, the title of each record is the name of the object class defined with the attribute `ClassName` and the concatenation of the `ImageProps`. Object class properties defined in the `ExportProps` are listed as additional information if their value matches the search or if they are defined as `ShowProps`. **Application: BLOOMBERG 6.6.3** **Description** Application providing **marketdata** In the data set, the name of the object class defined with the attribute `ClassName` and the concatenation of the `ImageProps` are displayed in two different columns **Class** and **Object**. No other information is provided. Because the title line should be brief, the object class property `Description` should not be included in the XML attribute `ImageProps`.
`ShowProps`	Enter a comma-separated list of object class properties to be displayed in the search results for the objects found by the search. Each object class property must be defined as: `ObjectClassName.PropertyName` Please keep the following in mind: • The results will display matches for any of the object class properties defined in the XML attribute `ExportProps`. For example, if `Application.Description` is defined in the XML attribute `ExportProps` and the search term is found in an application's `Description` property, then the description will be displayed in the results with the matching words highlighted, even if `Application.Description` is not defined in the XML attribute `ShowProps`. • Any object class properties defined in the XML attribute `ImageProps` should not be redefined again in the XML attribute `ShowProps` since the object class properties defined in the XML attribute `ImageProps` will already be displayed as title line. • The object class properties defined in the XML attribute `ShowProps` are read from the Alfabet database when the index is created. The object

XML Element (bold) / XML Attribute	Explanation
	class properties are visualized in the results displayed record-by-record only.
`Export-Reference`	If you want the navigation functionality of the full text search not to open the object that was found by the search, but an object referenced by the found object, enter the object class property that determines which referenced object should be the target of the navigation. The reference must be established via an object class property of the type `Reference` of the object class defined with the XML attribute `ClassName`. Please note that the object class name displayed in the search results both in the record-by-record view and the data set will display the name of the object class that is the target of the navigation, although all information that is displayed informs about the object class that is searched.  If the search group is defined with the XML attribute `ClassName` set to `ICTObject` and the XML attribute `ExportReference` is defined as `ICTObject.Owner`, the search criteria will be applied to search for the ICT object, but the object view of the organization that is the owner of the ICT object will be opened if the user navigates from the search result to the underlying object. If the ICT object has no owner organization, then the ICT object itself will be the navigation target.
Query	A search group may contain multiple queries.
`Name`	Enter a name for the Alfabet query that shall be implemented in the object-centric search.
`ClassName`	Enter the name of the object class defined in the XML element ***ClassEntry***.  The **Full-Text** tab will be automatically added to the standard object selector implemented for the object class specified in the XML attribute `ClassName`.
`Query`	Enter the Alfabet query to search for the objects that the index should be generated for (via the object class properties defined in the XML attribute `ExportProps` of the XML element ***ClassEntry***). The Alfabet Query must match the following rules: • The `FIND` class must be the class defined with the XML attribute `ClassName`.

XML Element (bold) / XML Attribute	Explanation
	• A WHERE condition must be defined that specifies the relation with the current object of the base class defined with the XML attribute `BaseClass` of the XML element `SearchGroup`. The Alfabet query language parameter BASE can be used in the WHERE clause. At runtime, it will be substituted with the REFSTR of the current base object. • Show Properties shall not be defined. For example to search for applications in application groups, the query can be defined as: `ALFABET_QUERY_500 FIND Application WHERE Application.ApplicationGroups CONTAINSOR:BASE` For more information about defining an Alfabet query, see the chapter Defining Queries. Please note that only queries written in the Alfabet query language may be used in the XML object ***SearchManager***. Queries written in native SQL are not supported.

4) In the toolbar, click the **Save** button to save the XML definition.

Configuring a Global Search Group

A global search allows users to enter search terms for any objects defined via the global search groups that are configured in the XML object ***SearchManager***. This search is executed by means of the *Full-Text Search Functionality* (`GlobalFullTextSearch`) in the **Search** module.

To edit the XML object ***SearchManager*** for a global search:

1) Go to the **Presentation** tab and expand the **XML Objects** folder. You will see all XML objects that can be edited.
2) Right-click the XML object ***SearchManager*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) Enter an XML element `SearchGroup` with the required child XML elements and XML attributes as child XML element of the root XML element `SearchManager` and define the XML attributes as needed. The table below displays the XML attributes that can be edited for the XML object ***SearchManager***:

The following example displays a code for a globally-defined search group searching within all applications in an application group:

```
<SearchGroup Name="Organizations and Business Processes"
Profiles="Viewer">
   <Query Name="Organization" ClassName="OrgaUnit"
   ShowProps="Name" ExportProps="Name,Description"
   ImageProps="Name"/>
   <Query Name="Business Process" ClassName="BusinessProcess"
   ShowProps="LevelID,Name"
   ExportProps="LevelID,Name,Description"
   ImageProps="LevelID,Name"/>
</SearchGroup>
```

XML Element (bold) / XML Attribute	Explanation
SearchGroup	
`Name`	Enter text that defines the name of the search group. This is the name that users will see when they select the search group in order to execute a search. The `Name` attribute also defines the name of the file containing the search index.
`Profiles`	Enter the user profiles that have access permissions to the search group. If no value is specified for the XML attribute `Profiles` then all configured user profiles will have access permissions to the search group.
Query	You should create an XML element ***Query*** for each object class that should be included in the XML element ***SearchGroup***.
`Name`	Enter a unique name for the query.
`ClassName`	Enter the name of the object class to be queried. The **Full-Text** tab will be automatically added to the standard object selector implemented for the object class specified in the XML attribute `ClassName`.
`ExportProps`	Enter a comma-separated list of object class property names to define all object class properties of the object class to be indexed. The XML attribute `ExportProps` must include all object class properties defined in the XML attribute `ShowProps` and XML attribute `ImageProps`.

XML Element (bold) / XML Attribute	Explanation
	 The export properties are the object class properties written into the full-text search index file. The index file will only contain values for a language version if a translation exists in the Alfabet database. That means that the index only contains a value if the object class property is configured to be translatable and a translation is available for the current object. If full-text search shall be conducted in multiple languages, you should make sure that all object class properties that shall be included into the search are translatable and translations are provided. Please note that if the subset of object class properties defined both in the XML attributes `ExportProps` and `ImageProps` is not translated, the information displayed about the object in the headline of the record-by-record view as well as in the data set view will fall back to the primary language to ensure that the user can identify the object. Nevertheless, these object class properties are excluded from search.
`ImageProps`	Enter a comma-separated list of the names of the object class properties to be displayed in the search results for the objects found by the search. Please keep the following in mind: • The results will display matches for any of the object class properties defined in the XML attribute `ExportProps`. For example, if `Description` is defined in the XML attribute `ExportProps` and the search term is found in an application's `Description` property, then the description will be displayed in the results with the matching words highlighted, even if `Description` is not defined in the XML attribute `ShowProps`. • Any object class properties defined in the XML attribute `ImageProps` should not be redefined again in the XML attribute `ShowProps` since the object class properties defined in the XML attribute `ImageProps` will already be displayed as title line. • The object class properties defined in the XML attribute `ShowProps` are read from the Alfabet database when the index is created. The object class properties are visualized in the results displayed record-by-record only.
`ShowProps`	Enter a comma-separated list of names of object class properties to create a concatenation that serves as the title line for each search result. The sequence of the object class properties listed in the XML attribute `ImageProps` determines the sequence displayed in the concatenated title. The object class properties defined in the XML attribute `ImageProps` are read from the Alfabet database when the index is created.

XML Element (bold) / XML Attribute	Explanation
	Results of a full text search can be displayed either record-by-record or as a data set. In the record-by-record view, the title of each record is the name of the object class defined with the attribute `ClassName` and the concatenation of the `ImageProps`. Object class properties defined in the `ExportProps` are listed as additional information if their value matches the search or if they are defined as `ShowProps`. **Application: BLOOMBERG 6.6.3** **Description** Application providing **marketdata** In the data set, the name of the object class defined with the attribute `ClassName` and the concatenation of the `ImageProps` are displayed in two different columns **Class** and **Object**. No other information is provided. Because the title line should be brief, the object class property `Description` should not be included in the XML attribute `ImageProps`.
`Query`	Optionally, you can define an Alfabet query with the XML attribute `Query` to search only for a sub-set for the objects of the object class that is defined with the XML attribute `ClassName`. • The `FIND` class must be the class defined with the XML attribute `ClassName`. • Show Properties shall not be defined. These are defined directly in the XML attribute `ShowProps`. For example to search only for applications that are currently active, define the following query: `ALFABET_QUERY_500 FIND Application WHERE Application.ObjectState = 'Active'` For more information about defining an Alfabet query, see the section Defining Queries. **NOTE**:The Alfabet query defined in the XML attribute `Query` should not include the definition of show properties. Please note that only queries written in the Alfabet query language may be used in the XML object ***SearchManager***. Queries written in native SQL are not supported.

4) In the toolbar, click the **Save**  button to save the XML definition.

Configuring Faceted Semantic Search for Information in Configured Reports

Users can search for content in configured reports in the **Faceted Semantic Search** (`SRCH_FacetedSearch`) functionality. The user can enter a description of the information he/she is looking for in a search field to search for the information in configured reports. In addition to the phrase entered by the user, synonyms and related words are taken into account.

The question the user entered into the search field will provide results via two mechanisms:

- **Search within available configured reports**:

In addition to the **Caption**, **Description**, and **Business Problem Statement** attributes of a configured report, the search will also use the **Apply To** attribute and the information from the **Semantic Analysis** sub-node of the configured report to find relevant reports. Strings in the **Alias** attribute of object classes and object class properties are taken into account to identify object classes.

- **Ad-hoc generation of new configured reports**:

Simple tabular datasets providing the information will be generated on the fly and presented to the user in addition to the search results derived from searching available configured reports defined by your company. The generation of ad-hoc reports is limited to a sub-set of user entries and will currently only be generated if the user looks for objects of an object class or object class stereotype with

- a specific value for an object class property and the object class property is based on an enumeration
- a value for a specific indicator
- a specific role assigned to a specific user

There are two ways users can access the **Faceted AlfaBot Semantic Search** functionality:

- The **Faceted Semantic Search** (`SRCH_FacetedSearch`) functionality can be made available via the user profile configuration. The user can access the functionality for example via a menu or a link in the guide page or guide view.

   For information about providing access to a functionality via the user profile configuration, see Making Functionalities Accessible to a User Profile.

- Users can access the view via the `Analyze` intent of the AlfaBot. The AlfaBot is available to the users via the slide-in toolbar.

The **Faceted Semantic Search** functionality is only available if the AlfaBot is activated and the `Analyze` intent of the AlfaBot is active. For information about activating the AlfaBot, see Implementing the AlfaBot for Navigation via a Full-Text Command in the chapter Configuring Alfabet Functionalities Implemented in the Solution Environment.

In addition, the functionality requires activation and regular maintenance. The complete procedure for configuration and maintenance is described in the scope of the AlfaBot functionality in the section Define AlfaBot Search Capabilities for Configured Reports.

Configuring the Glossary Search Functionality

Alfabet provides a **Glossary** functionality in which one or more enterprise-specific glossaries can be created with glossary entries (glossary term and description) and then searched in the **Full-Text Search** functionality. Before a defined glossary can be implemented however, a search group must be configure for the class Glossary in the XML object ***SearchManager***.

- The glossary and glossary items must be created in the *Glossary Functionality* (`Glossary`) in the **Search** module in Alfabet and a user or solution administrator must create and update the index for the glossary after the search group has been defined. For more information about these aspects of implementing the ***Glossary*** functionality, see the section *Searching an Enterprise Glossary via the Full-Text Search Capability* in the reference manual *Getting Started with Alfabet*.
- For more information about activating the *Full-Text Search Functionality* (`GlobalFullTextSearch`) which is necessary in order to create an index for the search group, see the section Specifying General Parameters for Index Creation.
- In order to create an index and execute a search, a path must be defined by your system administrator describing where the search indexes needed for the full-text search functionalities are found. For more information, see the section *Basic Configuration of the Alfabet Components* in the chapter *Installation* in the reference manual *System Administration* or contact your system administration.
- Alfabet also provides a domain glossary functionality. A domain glossary can be created in order to define and search for domain-specific terminology for objects assigned to a selected domain. The implementation of the domain glossary defined in the *Domain Glossary Page View* (`DOM_Glossary`) in Alfabet. If a domain glossary is defined, a search group must be configured in order to search the glossary. The configuration of the search capabilities for the domain glossary are addressed separately in the section Configuring the Domain Glossary Capability in the chapter Configuring Domain Models and Domain Planning.

Please note that only queries written in the Alfabet query language may be used in the XML object ***SearchManager***. Queries written in native SQL are not supported.

To edit the XML object ***SearchManager*** in order to create a search group for the Glossary functionality:

1) Go to the **Presentation** tab and expand the **XML Objects** folder. You will see all XML objects that can be edited.
2) Right-click the XML object ***SearchManager*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.

The following displays an example definition of a search group for the **Glossary** functionality in the XML object ***SearchManager***.

```
<SearchGroup Name="Glossary Search" BaseClass="Glossary"
Type="Object" >

   <ClassEntry ClassName="GlossaryItem"
   ShowProps="GlossaryItem.Alias, GlossaryItem.Description"
```



```
ExportProps="GlossaryItem.Alias,GlossaryItem.Description"
ImageProps="GlossaryItem.Alias" />

<Query Name="All Glossary Items" ClassName="GlossaryItem"
ImageProps="GlossaryItem.Alias" Query="ALFABET_QUERY_500
FIND GlossaryItem WHERE GlossaryItem.Glossary
CONTAINSOR:BASE" />

</SearchGroup>
```

3) Define the remaining attributes, as needed. The table below provides general information about the attributes that can be edited for the XML object ***SearchManager***:

XML Element (bold) / XML Attribute	Explanation
SearchGroup	
`Name`	Enter a name of the search group (for example, "`Glossary`"). This is the name that users will see when they select the search group in order to execute a search.
`Type`	Enter "`Object`" to define the object-centric search.
`BaseClass`	Enter "`Glossary` " as the base class for the search.
`ImageProps`	**NOTE**: Because the title line should be brief, the object class property `Description` should not be included in the XML attribute `ImageProps`. The attribute specifies the object class properties to display in the title line of the search results. Enter a comma-separated list of object class properties to create a concatenation that serves as the title line for each search result. The sequence of the object class properties listed in the XML attribute `ImageProps` determines the sequence displayed in the concatenated title. The object class properties defined in the XML attribute `ImageProps` are read from the Alfabet database when the index is created. The concatenation is displayed in the results displayed via a data set as well as record-by-record.
ClassEntry	The XML element ***ClassEntry*** should be created for the object class that is to be found by the Alfabet query defined for the object-centric search.
`ClassName`	Enter "`GlossaryItem`".
`ShowProps`	This attribute specifies the defined object class properties for all objects found by the search. Enter a comma-separated list of object class properties to be displayed in the search results for the objects found by the search. For

XML Element (bold) / XML Attribute	Explanation
	example, you may want `Application.Version` or `Application.ShortName` to be displayed for all applications displayed in the results. The object class properties are visualized in the results displayed via a data set as well as record-by-record. Please keep the following in mind. • Any object class properties defined in the XML attribute `ImageProps` should not be redefined again in the XML attribute `ShowProps` since the object class properties defined in the XML attribute `ImageProps` will already be displayed. • The results will display matches for any of the object class properties defined in the XML attribute `ExportProps`. For example, if `Application.Description` is defined in the `ExportProps` and the search term is found in an application's description, then the description will be displayed in the results with the matching words highlighted, even if `Application.Description` is not defined in the XML attribute `ShowProps`.
`ExportProps`	The attribute specifies the object class properties to be indexed. Enter a comma-separated list of object class properties to define all object class properties of the object class to be indexed. The XML attribute `ExportProps` must include all object class properties defined in the `ShowProps` and XML attribute `ImageProps`.
Query	
`Name`	Enter the name of the Alfabet query that shall be implemented in the object-centric search.
`ClassName`	Enter "`GlossaryItem`".
`Query`	Enter the Alfabet query to search for the objects that the index should be generated for (via the object class properties defined in the XML attribute `ExportProps`). For more information about defining an Alfabet query, see the section Defining Queries. Please note that only queries written in the Alfabet query language may be used in the XML object ***SearchManager***. Queries written in native SQL are not supported.

4) In the toolbar, click the **Save** button to save the XML definition.

Configuring the Wildcard for Standard and Custom Search Functionalities

The XML object ***SearchManager*** allows you to configure whether a wildcard <*> should be automatically implemented when a user enters search criteria for a relevant class in the following Alfabet search functionalities:

- *Simple Search Functionality* (`GenericSearch`, `Simple_Search`)
- explorer search capability
- standard and configured object selectors available in standard page views
- Any Edit Search interface controls implemented in custom editors, custom selectors, standard page views, or configured reports.

The wildcard capability is not implemented in the *Full-Text Search Functionality* (`Global-FullTextSearch`) nor in filters (such as **Name** fields) in standard Alfabet page views.

The implementation of the automatic wildcard ensures that for any search criteria entered in the search criteria field of a search functionality, the term will be wrapped automatically by the wildcard when the search is triggered. Thus, if a user enters Trade in the search pattern, the search criteria *Trade* will be applied and objects with Trade in any of the object class properties defined for the search will be returned in the result list.

If the `AutoWildCard` attribute is set to "`true`", a message will be displayed in the *Simple Search Functionality* (`GenericSearch`, `Simple_Search`) and selectors for standard page views stating "A wildcard is not required for the search."

Please note that this automatic text will not be displayed in configured reports. In the case of configured reports:

- If the search function is configured to set wildcards automatically via the XML object ***SearchManager***, you can configure individual filter fields in a configured report or custom selector to be excluded from the automatic wildcard setting. Depending on the underlying query, the setting of wildcards might be undesirable (for example, when comparing the entry in the filter field with a string using an equal to operator). For more information about configuring the wildcard for filter fields in configured reports, see the section Automatically Adding Wildcards to Search Strings in the chapter Defining Queries as well as the chapter Configuring Reports.
- A static text can be configured for the selector field informing the user that a wildcard is not required for the search. For more information about the configuration of search fields in configured reports, see the section Defining Filters for Configured Reports and Selectors in the chapter Defining Queries as well as the chapter Configuring Reports.

To configure the implementation of the wildcard via the XML object ***SearchManager***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder. You will see all XML objects that can be edited.
2) Right-click the XML object ***SearchManager*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see the section Working with XML Objects.

3) In the XML object ***SearchManager***, enter "`true`" in the XML attribute `AutoWildCard` of the root XML object `SearchManager` to implement the automatic wildcard.

```
<SearchManager HighlightCount="20000" AutoWildCard="true">
    ...
</SearchManager>
```

If "`false`" is defined, the wildcard will not be implemented in the search functionalities available in the *Simple Search Functionalities* (`GenericSearch`, `Simple_Search`), the explorer search capability, as well for the standard and configured object selectors and quick selectors available in standard page views as well as any Edit Search interface controls implemented in custom editors, custom selectors, or configured reports configured in Alfabet. In this case, the user must manually enter the wildcard.

4) In the toolbar, click the **Save**  button to save the XML definition.

Chapter 8: Configuring Wizards

Software AG provides a number of standard private wizards for commodity classes in which a large number of objects are typically documented as well as the means to configure custom wizards that allow users to capture a specified set of data for an object class or object class stereotype.

A wizard is an assistant that consists of a configured set of standard editors, custom editors, standard views, and configured reports. The wizard will typically guide a user through a linear multi-step process to capture data for an object that the user has access permissions to.

Standard data capture wizards are available for commodity classes in which a large number of objects are typically documented. An unlimited number of custom wizards may be configured for an object class in order to allow different users working in different contexts to capture the object data that they are responsible for. However, only one wizard may be available for an object class per user profile.

Software AG also provides a workflow that allows multiple users to collaborate in the context of a configured process in which roles and responsibilities are distributed. For more information about configuring a task-oriented process in which multiple users should be involved, see the chapter Configuring Workflows.

Private wizards provided by Software AG can be implemented in your enterprise as-is or copied and used as a basis for configuring your own custom wizards. Click the + symbol next to the private wizard to display the predefined wizard steps. To access a private wizard, go to the **Presentations** tab and expand the **Wizards** node. The following private wizards are available:

- `APP_Wizard` (for the object class `Application`)
- `BSPR_Wizard` (for the object class `Business Appraisal`)
- `COM_Wizard` (for the object class `Component`)
- `CNRT_Wizard` (for the object class `Contract`)
- `DEM_Wizard` (for the object class `Demand`)
- `DOM_Wizard` (for the object class `Domain`)
- `DVC_Wizard` (for the object class `Device`)
- `ICTO_Wizard` (for the object class `ICT Object`)
- `ITMPM_Wizard` (for the object class `Master Plan Map`)
- `PRF_Wizard` (for the object class `Peripheral`)
- `RISKOBJ_Wizard` (for the object class `Risk Object`)
- `SPL_Wizard` (for the object class `Standard Platform`)
- `User_Wizard` (for the object class `User`)

Custom wizards may also be configured by your enterprise for most object classes. Customized wizards are used primarily in three different contexts in Alfabet:

- Wizards are typically used to create new objects and define existing objects. You can configure the wizard to be used in place of an editor so that the wizard opens instead of the editor in object

profiles, views, etc. For information about additional configuration requirements to implement a wizard for data capture and definition in page views, see Implementing a Wizard Instead of an Editor to Capture and Define Data.

- A wizard may also be used in the context of the workflow capability to capture or edit objects targeted by a workflow step. For information about additional configuration requirements to implement a wizard in workflows, see Implementing a Wizard in the Workflow Capability. For more information about the configuration of workflows, see the chapter Configuring Workflows.

The general process of creating a wizard and configuring wizard steps is the same for all wizards, regardless of whether the wizard will be implemented in the context of creating or editing an object or in a workflow.

For an overview of the object classes for which custom wizards can be configured, see the chapter *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*. Please note the special considerations regarding the specification of wizards for specific object classes or their object class stereotypes:

- **Interface System**: If a wizard is implemented to capture data for objects of the class **Interface System**, the editor `ISystem_Editor` must be specified for the first wizard step.
- **Business Support**: If you are implementing the **Business Support** editor in the wizard step, you should implement the `BSP_Editor`. All other business support editors (for example, `BSP_APP_PROC_ORG_Editor`) are used in the context of views and may not be configured for a wizard step.
- **Tactical Business Support**: If a tactical business support is to be created, the relevant wizard(s) should be implemented in the context of a master plan map. The editor to implement is determined by the relevant combination of business support provider, X-dimension object, and Y-dimension object to be defined for the tactical business support
- **Strategic Business Support**: If a strategic business support is to be created, the wizard should be implemented in the context of an IT strategy map. The editor to implement is determined by the relevant combination of business support provider, X-dimension object, and Y-dimension object to be defined for the strategic business support
- **Object class stereotypes**: To create a custom wizard for an object class stereotype, you must first create the wizard for the base class and then assign the wizard to the relevant class setting configured for the object class stereotype. For more information about configuring class settings, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes. For example:
  - **Project Stereotype**: If a wizard is configured for a project stereotype, you must set the **Class Name** attribute to `Project`. The wizard must then be selected in the **Wizard** attribute for the class setting for the relevant project stereotype. For specific information about configuring a wizard for the Project Management functionality, see the section Configuring Wizards for Project Stereotypes.
  - **Domain Stereotype**: If a wizard is configured for a domain stereotype, you must set the **Class Name** attribute to `Domain`. The wizard must then be selected in the **Wizard** attribute for the class setting for the relevant domain stereotype.

The following information is available:

- Implementing a Wizard in the Workflow Capability
- Testing the Wizard in the Alfabet User Interface

Understanding How Wizards Work

Various interface elements are preconfigured by Software AG whereas other interface elements can be customized. For example, each wizard step can be configured to display information about the task at hand that will be displayed in the header of the respective wizard step.

Each wizard step typically constitutes one screen with preconfigured buttons that control the wizard process. The following information explains the general behavior that is triggered by the buttons as well as additional information that will help you understand possible consequences if post-conditions are configured for a wizard step. The **Next** button and **Exit** button are preconfigured by Software AG and will be displayed per default. The **Back** button and **Go to Step** button are optional and must be configured to be displayed in the wizard.

A wizard step may have pre-conditions or post-conditions defined in order to validate whether data has been entered for a wizard step. Thus, depending on the configuration, it may be mandatory or desired that specific data is completed for a wizard step. Please note that any post-conditions that have been configured for that wizard step will be executed AFTER the data has been saved in the wizard step. This is of particular relevance for post-conditions defined for editors. The first database transaction will attempt to save the data that has been defined in the wizard step. However, this may fail if mandatory or uniqueness conditions are violated. Such errors must first be resolved before the system can check the post-conditions. A customized error message will be displayed providing the user with information about the data that is required.

Please note the following issues regarding the behavior of the wizard:

- **Mandatory Fields**: Mandatory fields in an editor must be completed in order to save the data and proceed to the next wizard step. If a wizard cannot advance to the next wizard step because mandatory properties have not been defined, an error message will list the mandatory properties that must be defined in order for the wizard to proceed to the next step.
- **NextButton**: The **Next** button allows the user to save the data entered during the wizard step and proceed to the next wizard step. If the automated translation is supported for the object class, the automated data translation capability will be triggered when the **Next** button is clicked. Please note the following:

- If a pre-condition is defined for the last wizard step and the pre-condition is not fulfilled, then the wizard will close when the user clicks the **Next** button in the wizard step before the last wizard step.
- If the post-condition specifies that the data is required (mandatory), a customized error message will be displayed when the user clicks the **Next** button. The user will not be able to advance to the next wizard step until the required data is provided. If the post-condition specifies that the data is desired (optional), a customized information message will be displayed if the data is missing. However, the user will not be prevented from advancing to the next wizard step.
- If multiple post-conditions have been configured for a wizard step, only one error message at a time will be displayed when the user attempts to save the data by clicking the **Next** button. Once the user corrects the first violation and attempts to advance to the next wizard step, a customized error message for the next violation will be displayed. This process will continue until all violations have been

corrected. The user will only be able to advance to the next wizard step when no violations have been found via the data entry prompts.

- **ExitButton**: The **Exit** button allows the user to save the data entered during the wizard step and close the wizard. The next time the user opens the wizard for the selected object, the wizard will automatically open to the wizard step that the user most recently exited. Likewise, if the user session is terminated while the wizard is open, the wizard will open to the wizard step that was last open when the session was terminated. Please note the following:

- If a user clicks the **Exit** button during the creation of a new object on a first wizard step, the new object will be deleted!

- The caption displayed on the **Exit** button is configurable and therefore, a different caption (such as **Save**) may be displayed on the button.

- **BackButton**: A **Back** button may be displayed that allows the user to return to the preceding wizard step. By clicking the **Back** button, the user saves the data entered during the wizard step and returns to the preceding wizard step. The **Back** button will be greyed out and inactive on the first step of a wizard.

- **Go to StepButton**: A **Go to Step** drop-down box may be displayed that will allow users to select another wizard step in the workflow. This enables the user to leave the linear sequence of steps and return to a subsequent or preceding wizard step in the wizard process. The **Go to Step** field is only available when an existing object is being edited. It is not available when a new object is being created. If the wizard is configured to include the **Go to Step** field, users will be able to traverse to any subsequent or proceeding wizard step. If post-conditions are configured for the wizard step that the user is leaving, the post-conditions will be executed and, if necessary, required data must be entered in order to leave the current wizard step. Please be aware, however, that post-conditions for any intermediary steps between the wizard step the user is leaving and the wizard step the user is going to will not be checked. For example, if a user goes from Wizard Step 2 to Wizard Step 4, the post-conditions for Wizard Step 2 will be checked and must be fulfilled, but post-conditions for Wizard Step 3 will not be triggered and must not be fulfilled in order to go to Wizard Step 4.

- **Close Button**: The wizard can be closed by clicking the **X** button in the upper right corner of the wizard. All data that has been saved via the **Next** button will be saved to the Alfabet database.

- **Online Help**: The standard and custom online help will be available to users working in the wizard:

- For wizard steps of the type `Editor`, users can point to a field caption to display a tooltip providing information about a standard Alfabet property or custom object class property (if the **Hint** attribute has been defined for the custom property field).

- For wizards steps of the type `View`, a link to the context-sensitive Help will be available.

- For wizard steps of the type `Report`, a link to your enterprise's custom Help may be available if custom Help has been configured. For more information about configuring a custom report, see the section Assigning Custom Help and Automated Help Assistants to Configured Reports.

- HTML text may be configured for the header to provide additional instructional content directly in a wizard step. For more information, see the section Configuring the Header Text of a Wizard Step.

Conceptualizing the Custom Wizard

The following displays the steps relevant to configuring a custom wizard.

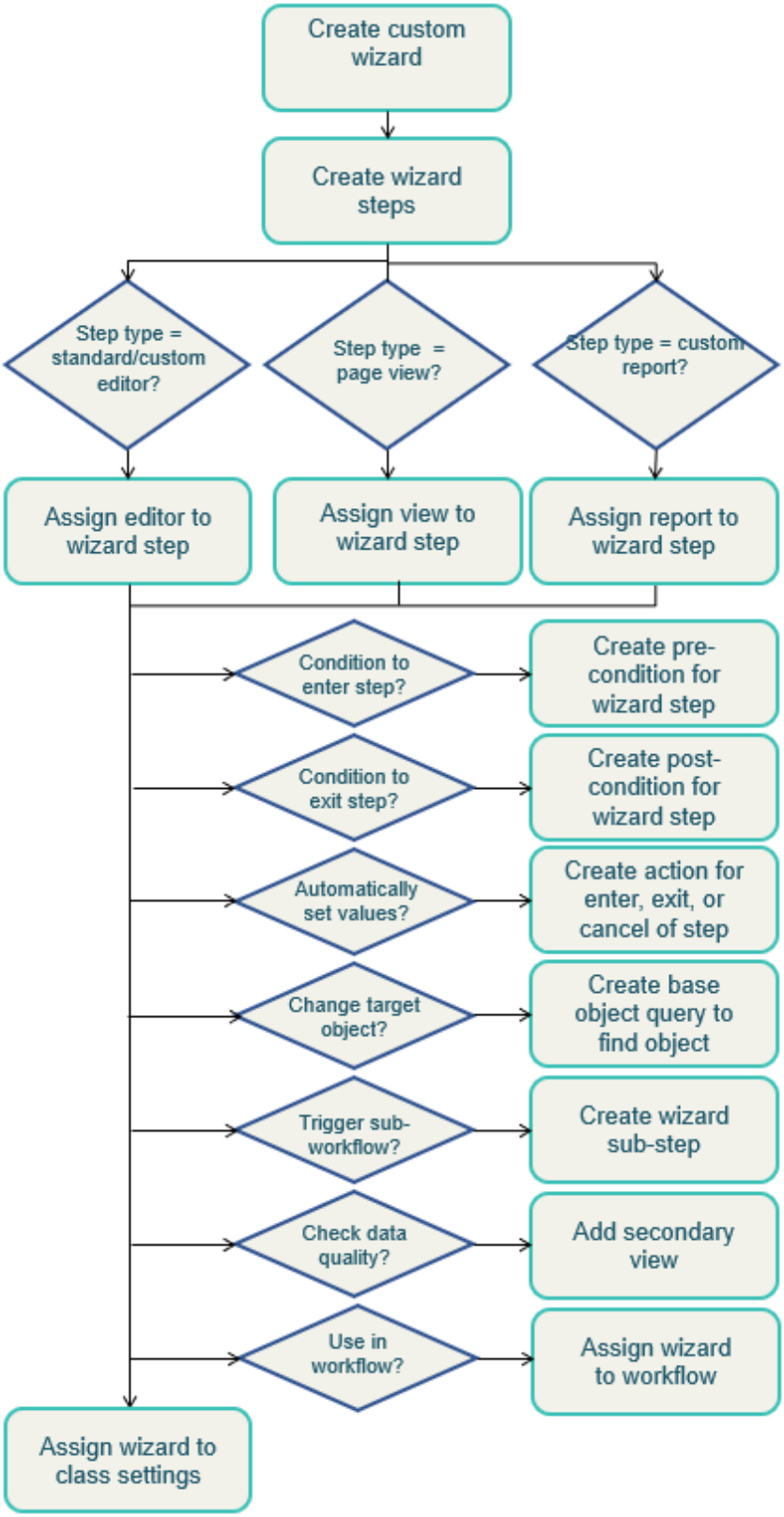


FIGURE: Steps to configure a custom wizard

Please keep the following in mind regarding the general configuration of a wizard:

- You can create multiple wizards for an object class. This allows you to create wizards for users with different roles in the enterprise or different levels of maturity working with Alfabet. Each wizard defined for an object class should be assigned to a different class setting. For an overview of the object classes for which custom wizards can be configured, see the chapter *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.
- If object class stereotypes are configured for an object class, you can specify a different wizard for each object class stereotype via its class setting. This allows you, for example, to assign a wizard as the mechanism for data capture for one object class stereotype and assign a standard or custom editor for data capture in another object class stereotype.
- Only one wizard can be assigned to a class setting. Thus, only one wizard is allowed per object class (or object class stereotype) per user profile
- A custom wizard can be further refined for the user profile(s) in which it is implemented. In this case, you can hide object class property in the editors, page views, and configured reports embedded in the wizard as well as hide functionalities available in the toolbars in page views and configured reports.

The configuration of a custom wizard will apply to all instances of the custom wizard in Alfabet. For example, if you hide object class properties or functionalities in a custom wizard, the object class properties or functionalities will be hidden for all instances of the custom wizard. In other words, a selected wizard cannot be differently configured in the context of different object class stereotypes. The wizard configuration will apply to each object class stereotype that the custom wizard is assigned to. If different configurations are required for a custom wizard, it is recommended that you create a new wizard via the copy functionality and modify the new wizard as needed. Once the wizard is configured, it can be selected in the **Wizard** attribute for the class setting that you create for the object class stereotype.

The wizard designer should consider the following issues when defining a wizard

- Which object class is the wizard process targeting?
- Will the user need to define data for objects in other object classes as well? If object classes other than the base class are required, a query must be defined for the relevant wizard step to find the new target object.
- What task is to be completed in the wizard and which user profile(s) will be accessing the wizard? This information will help you to determine which wizard steps should be included in the wizard.

Please note that if a new object is to be created in the wizard, the first wizard step must implement an editor. Therefore, for a wizard in which a new object is to be created, the **Step Type** attribute of the first wizard step must be set to `Editor`.

- Does a standard or custom wizard already exist for the relevant object class that can be copied and used as a template for the new wizard, or do you need to create a new wizard from scratch?
- Which editors, views, and reports are required by the user to complete the tasks? A wizard step may consist of any of the following:

- a standard editor
- a custom editor
- a standard Alfabet page view
- a tabular query-based configured report
- a query-based configured report displaying graphics (such as a tree-map or layered diagram report)

- If a custom editor is implemented for a wizard step:

- Do you want the user to complete all editor tabs in thee editor at once or do you want to split the data collection in each tab in the editor across several wizard steps. By configuring each tabbed page in the custom editor as a separate wizard step, the user can enter data in one tab and complete a wizard step and therefore close the editor without losing data.
- Do you want to hide properties in the custom editor? If this is the case, you can customize the custom editor in the context of the wizard to show only properties relevant for the current wizard step.

- Do you want the user to be able to go to any step in the wizard at any time or should the user be required to navigate through the configured sequence of steps? By including the **Go to Step** field, a user will be able to navigate to any wizard step.
- Do you want to specify a subordinate wizard step for a wizard step? Should the subordinate wizard step also have subordinate wizard steps? By creating a branch of subordinate wizard steps, you can direct the user to complete a set of tasks if a pre-condition is fulfilled for the top-level subordinate wizard step in the hierarchy. Once the branch of subordinate wizard steps is complete, the user would continue with the next wizard step in the wizard process. If the pre-condition is not fulfilled for the subordinate wizard step, the wizard will ignore the branch of subordinate wizard steps.
- Do you want to specify pre-conditions for a wizard step? By defining one or more pre-conditions via queries, you can stipulate the criteria that must be fulfilled in order to enter a wizard step or subordinate wizard step.
- Do you want to specify post-conditions for a wizard step? By defining one or more post-conditions via queries, you can determine the object class properties or references that must be defined before a wizard step an be completed. You can also define the messages prompting the user to provide the relevant data.
- Do you want an action when a wizard step is entered, exited, or cancelled? Actions such as the update of properties, calculation of indicators, or triggering of events can be configured. For example an event can be triggered as an alternative to batch jobs which may typically be initiated daily. In this case, a configured ADIF scheme will be triggered to process the data.
- Do you want to provide instructions on the wizard interface in order to guide users through the wizard process? If so, do you want to provide plain text in the header or do you want to implement HTML with the option of using hyperlinks to additional information that is relevant for the wizard step?
- Do you want to hide properties in an editor that is embedded in a wizard step?
- Do you want to implement specific custom selectors in an editor that is embedded in a wizard step?
- Do you want to disable specific functionalities such as create, copy, edit, navigate, etc. in views and reports embedded in a wizard step?

Creating a Wizard

Multiple wizards can be created for an object class. You can copy a private wizard to create a new custom wizard, create a new wizard from scratch, or copy an existing custom wizard to create a new custom wizard. The new wizard can be created for any permissible object class and configured as needed. For an overview of the object classes for which wizards can be implemented, see *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

Please note the following about the creation of wizards:

- A custom wizard can be assigned to one or more class settings that have been defined for an object class or its object class stereotypes.
- The configuration of a custom wizard will apply to all implementations of the custom wizard. Therefore, if you assign a custom wizard to a class setting for stereotype 1 but need a slight modification for stereotype 2, then you must create a new custom wizard for stereotype 2. This can be done by copying the original custom wizard to create a new custom wizard that can be modified for stereotype 2.
- To create a custom wizard for an object class stereotype, you must first create the wizard for the base class and then assign the wizard to the relevant class setting configured for the object class stereotype. For example, if a wizard is required for the domain stereotype Technological Domains, then you must create the wizard for the object class `Domain`. Once the wizard is configured, it can be selected in the **Wizard** attribute for the class setting that you create for the domain stereotype Technological Domains. For more information about configuring class settings, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes.

To create a new wizard:

1) Go to the **Presentations** tab, navigate to the **Wizards** node and carry out one of the following procedures:

- To create a new wizard based on an existing wizard, right-click the private wizard  that you want to copy and select **New Wizard as Copy**. The entire wizard configuration including all wizard steps and their view definitions, post-conditions, pre-conditions, etc. will be copied to the new wizard. These can be modified or deleted, as needed
- To create a new wizard from scratch, right-click the **Wizards** folder and select **New Wizard**. In the **Select Class** editor that opens, select a valid object class for which you want to create the wizard and click **OK**. If you are creating a wizard that is to be implemented for an object class stereotype, you must select the base object class. For an overview of the object classes for which wizards can be implemented, see *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

2) The new custom wizard  is displayed below the **Wizards** node in the explorer. Click the new wizard to open its attribute window and define the following attributes, as needed:

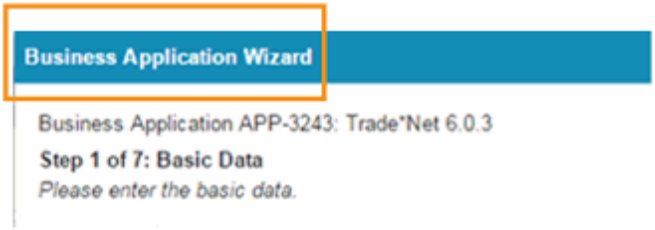
- **Class Name**: Displays the object class that you created the wizard for.
- **Name**: Enter a technical name for the wizard. This name is not displayed in the Alfabet user interface.

A validation mechanism checks for correct syntax when defining a technical name. The technical name of a configuration object may not begin with an empty space nor include any of the following special characters: **\ / * ? " > < | ':**

Furthermore, please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.

Furthermore, if the technical name of a configuration object is changed, the name will be correctly updated in other configuration objects referencing the changed object during design time. Please note that the name of a changed configuration object will not be updated in guide pages nor if the guide pages reference a configuration object listed in the **Show Usage** functionality. If you plan to change the name of a configuration object, the reference in the guide page should be changed prior to changing the configuration object name.

- **Caption**: Enter the caption that should be displayed in the header bar of the wizard in the Alfabet interface.



The text defined in the **Caption** attribute of a wizard can be translated to the other languages of your Alfabet interface. For more information, see the section Modifying, Translating and Managing the Vocabularies in the chapter Localization and Multi-Language Support for the Alfabet Interface

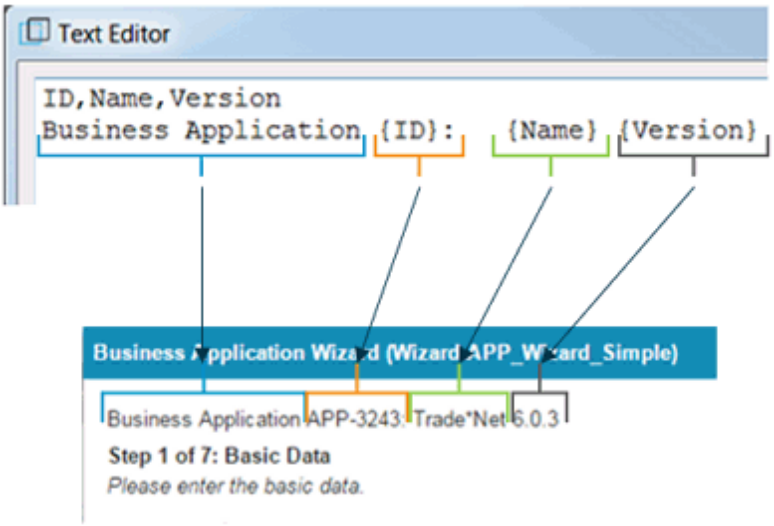
- **Group**: If the wizard should be stored in a wizard folder below the Wizards node, select an existing wizard folder in the drop-down list or enter a new folder name to create a new wizard folder. The wizard will be moved to the wizard folder.
- **Check Rights**: Enter `True` if access permissions should be checked for the object managed by the wizard when a user attempts to open the wizard. Enter `False` if the wizard should not check the access permissions for the objects managed by the wizard.

For example, it may be useful to set the **Check Rights** attribute to `False` in the context of a configured report which already verifies the access permissions and only shows objects the current user has Read/Write permissions for. Setting this attribute to `False` can improve performance.

- **Show Comments Panel**: Select `True` if you want to display a header that provides information about the object being edited as well as any instructions configured for the header for each wizard step. If you select `False`, no information specified for the Format String will be displayed about the object being processed and no instructional text will be available for the wizard step. Furthermore, no context-sensitive Help link will be available for views processed a wizard step.
- **Format String**: If you have selected `True` in the **Show Comments Panel** attribute, then you can specify a format string to determine the information to display in the header panel of the wizard steps. This attribute allows you to specify the information displayed in the header panel about the object targeted by the wizard step. For example, the identification number, name, and version of the object could be displayed for all wizard steps. The following example shows the configuration of the format string in the text editor and its display in the Alfabet interface:

To define the **Format String** attribute, click the **Browse** button to define the information to display about the object in the header of the wizard. Define the format string as follows:

- On the first line of the text editor, enter a comma-separated list of the standard or custom object class properties that shall be displayed. For example: `ID,Name,Version`
- On the second line of the text editor, define the content that should be displayed in the header of the wizard. This could include a combination of text, punctuation, and attributes. All attributes in the second line must be enclosed in brackets and spelled correctly. For example: <Text> {`ID`}: {`Name`} {`Version`}).



Alternatively, HTML text can be configured for the header text for one or more wizard steps. This allows you to format the text and provide more complex instructional information for a wizard step. In this case, the definition of HTML header text for a wizard step will override the specification in the **Format String** attribute of the wizard. For more information about the definition of instructional text to display in the header of the wizard step, see the section Configuring the Header Text of a Wizard Step.

- **Has Back Button**: Select `True` if the **Back** button should be displayed in the wizard. The **Back** button allows the user to return to the preceding wizard step. By clicking the **Back** button, the user saves the data entered during the wizard step and returns to the preceding wizard step. The **Back** button will be greyed out and inactive on the first step of a wizard.
- **Exit Button Alias**: If necessary, change the default caption on the **EXIT** button. Enter a new caption in the cell (for example, SAVE).
- **Show Wizard Steps Combo-Box**: Select `True` if the **Go to Step** field should be displayed in the wizard. The **Go to Step** field allow users to select another wizard step in the wizard via a drop-down list. This enables the user to leave the linear sequence of steps and return to any subsequent or preceding wizard step in the wizard process. The **Go to Step** field is only available in a wizard step when an existing object is being edited. It is not available in the first wizard step when a new object is being created. For more information about the impact of post-conditions on the **Go to Step** field as well as the impact of the **Execute Post-Conditions on Back Moves** attribute, see the section Defining a Post-Condition for a Wizard Step.
- **Type**: This attribute is relevant if subordinate wizard steps shall be specified.

- Select `Tree` to number all subordinate wizard steps so that subordinate wizard step 1, subordinate wizard step 2, and subordinate wizard step 3 that are subordinate to wizard step 1 will be numbered step 1.1, 1.2, and 1.3 in the wizard header. Please note that if the **Show Wizard Steps Combo-Box** attribute is set to `True` for the wizard, then the subordinate wizard steps will be displayed in the **Go to Step** field.
- Select `Straight` to number all subordinate wizard steps so that all subordinate wizard step that are subordinate to wizard step 1 will all have the same number as their ascendant wizard step. In this case, subordinate wizard step 1, subordinate wizard step 2, and subordinate wizard step 3 would all be numbered 1 of 7 in the wizard header. Please note that if the **Show Wizard Steps Combo-Box** attribute is set to `False` for the wizard, then the subordinate wizard steps will not be displayed in the **Go to Step** field.
- **Height** and **Width**: If necessary, change the default height and weight of the wizard. The default height of 700 pixels and width of 800 pixels should not be decreased. Any value you enter below these minimums will be ignored. Wizards are displayed as modal windows and can be resized by users as needed.
- **Automated Assistant URL**: If custom help content should be available via the automated assistant capability, enter the URL or server variable that targets the content to display in the assistant. Please note that if an automated assistant is configured for views embedded in the wizard, the automated assistant will be displayed for those views. If no automated assistant is defined for the individual views, the automated assistant specified for the wizard will be displayed. For more information about additional requirements to implement the automated assistant capability, see the chapter Providing Custom Online Help to the User Community.

3) After the wizard has been created, click the **Save**  button to save the wizard.

Specifying Wizard Steps for a Custom Wizard

A wizard step is an editor, view, or report in a wizard. The wizard step constitutes one aspect of data entry or review that is required in the data collection process. A wizard step may be reused for multiple wizards for the same object class.

Each wizard step may have instructions configured that are displayed in the header of the wizard step. The current step that the user is currently processing vs. the total number of wizard steps are automatically generated and displayed in the header of the wizard step.

A wizard step may consist of any of the following:

An editor, view, or report can be reused in the configuration of multiple wizard steps in the same wizard, thus enabling the user to enter the data in multiple steps rather than having to complete the data entry process all at once. Furthermore, each wizard step can be configured so that only the relevant standard and custom object class properties are displayed for a wizard step.

A wizard step may have an unlimited number of subordinate wizard steps that can constitute a branch in the wizard process. Any wizard step or subordinate wizard steps may be controlled by pre-conditions that determine whether the wizard step should be entered or skipped. The pre-conditions are configured as Alfabet queries or SQL queries that determine the criteria that must be fulfilled in order to enter a specified step. Multiple post-conditions may be defined per wizard step in order to prompt users to provide data for specified standard or custom object class properties before proceeding to the next wizard step.

A pre-condition may NOT be configured for the first wizard step NOR for the final wizard step or subordinate wizard step. If you do so and the pre-condition is not fulfilled, an error will occur. In this case, for example, the wizard could stagnate and not advance to the next wizard step. If a pre-condition is required for the last logical step or subordinate wizard step in the wizard, it is recommended that you add another final concluding wizard step with no pre-condition and only static text defined (for example, with a message such as "You have completed the wizard").

The following information is available:

Creating a Wizard Step

You can create a new wizard step from scratch or copy a wizard step from another wizard. To do so, you must create an empty wizard step for the wizard you are defining, copy an existing wizard step (via right-click > **Copy**), and paste the wizard step on the empty wizard step (right-click > **Paste**). The entire wizard step definition is copied to the new wizard step and can be modified, as needed.

The initial creation of a wizard step includes the specification of the technical name as well as the caption and description for the wizard step. Please note that instead of creating a caption and description in plain text for a wizard step, you could alternatively configure formatted HTML text as described in the section Configuring the Header Text of a Wizard Step.

The **Caption** and **Description** attributes will be available in the vocabularies and can be translated. For more information about the translation capability, see the section Modifying, Translating and Managing the Vocabularies the chapter Localization and Multi-Language Support for the Alfabet Interface.

Once a wizard step has been created, you can specify the content of the wizard, any pre- or post-conditions that must be fulfilled for a wizard step, and any actions such as the update of properties, calculation of indicators, or triggering of events that shall be associated with a wizard step.

To define wizard steps for the selected wizard:

1) Go to the **Presentations** tab, expand to the **Wizards** node, right-click the custom wizard and select **New Wizard Step**. A wizard step labelled `View` is added to the wizard step.

2) Click the wizard step to activate the attribute window. Define the following attributes as needed:

- **Name**: Enter a technical name for the wizard step. This name is not displayed in the Alfabet interface.

  A validation mechanism checks for correct syntax when defining a technical name. The technical name of a configuration object may not begin with an empty space nor include any of the following special characters: **\ / * ? " > < | ':**

  Furthermore, please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.

  Furthermore, if the technical name of a configuration object is changed, the name will be correctly updated in other configuration objects referencing the changed object during design time. Please note that the name of a changed configuration object will not be updated in guide pages nor if the guide pages reference a configuration object listed in the **Show Usage** functionality. If you plan to change the name of a configuration object, the reference in the guide page should be changed prior to changing the configuration object name.

- **Caption**: Enter the caption to display in the header panel of the wizard step in the Alfabet interface. If the wizard step is an editor, the caption will be displayed as well as the tab name (<Caption / Tab Name>). The caption you define here will replace the default name of standard and custom editors, standard Alfabet views, as well as the caption defined for configured reports. The caption will not be displayed if the **Header HTML** attribute is defined for the header. For more information about the

configuration of HTML text in the header of the wizard step, see the section Configuring the Header Text of a Wizard Step.

If the **Show Wizard Steps Combo-Box** attribute is set to `True` for the wizard, any wizard step captions that are longer than the **Go to Step** field will be truncated in the drop-down list. For more information about including the **Go to Step** field in the wizard, see the section Creating a Wizard.

- **Caption Step Color**: The captions of wizard steps can be colored to indicate, for example, that the wizard step is mandatory, important, optional, etc. The color will be applied to the text specified in the **Caption** attribute as well as in the **Go to Step** drop-down menu at the bottom of the wizard.
- **Caption Step Suffix**: The captions of wizard steps can be provided with a caption suffix to indicate, for example, that the wizard step is mandatory, important, optional, etc. The suffix will be appended to the text specified in the **Caption** attribute as well as in the **Go to Step** drop-down menu at the bottom of the wizard.
- **Close (X) Button Hint**: Specify a hint to replace the standard hint for the **Close (X)** button in the upper right corner of the editor/wizard which states " **Click here to exit without saving data.** " You can specify a differentiated hint to be displayed if, for example, an editor is embedded in the wizard step and the **Tab as Seperate Step** attribute is set to `True`. In this case, you could specify a text such as " **Click here to exit without saving data for the current wizard step.** " to let the user know that only the data in the current wizard step would be lost if the wizard were to be closed via the **Close (X)** button.
- **Description**: Enter a description to display in header panel of the wizard step. The description could provide the user with instructions about what to do in the wizard step. The description will not be displayed if the **HTML Header** attribute is defined for the header. For more information about the configuration of HTML text in the header of the wizard step, see the section Configuring the Header Text of a Wizard Step.

The text defined in the **Caption** and **Description** attributes of a wizard step can be translated to the other languages of your Alfabet interface. For more information, see the section Modifying, Translating and Managing the Vocabularies in the chapter Localization and Multi-Language Support for the Alfabet Interface

3) In the toolbar, click the **Save**  button to save your changes.

4) Next, you must specify the content of the wizard step. The following is possible:

- Specify an editor for the wizard step.

Please note that if a new object is to be created in the wizard, the first wizard step must implement an editor. Therefore, for a wizard in which a new object is to be created, the **Step Type** attribute of the first wizard step must be set to `Editor`.

- Specify a standard page view for the wizard step.
- Specify a configured report for the wizard step.

The following information is available:

- Configuring a Wizard Step to Display a Standard or Custom Editor
- Controlling the Visibility of Tabbed Pages in Wizard Steps

- Configuring a Wizard Step to Display a Standard Page View
- Configuring a Wizard Step to Display a Configured Report

Configuring a Wizard Step to Display a Standard or Custom Editor

If the wizard step is to be based on an editor, you must assign a standard editor to the wizard step. Once the standard editor has been defined, you can add a custom editor to be included in the wizard step.

Online help is available for wizard steps of the type `Editor` for all standard object class properties and custom object class properties for which the **Hint** attribute has been defined. It is also possible to configure instructional text in the header of the wizard step. For more information about configuring HTML text in the header of the wizard step, see the section Configuring the Header Text of a Wizard Step.

There are a number of configuration possibilities for a wizard step of the type `Editor`:

- A standard editor is displayed for the wizard step
- A standard editor and custom editor is displayed for the wizard step
- All tabbed pages in the editor are processed for the wizard step.
- Each tabbed page in the editor is processed as a separate wizard step.
- Only one tabbed page in the editor is visible for the wizard step.
- Only some properties are visible in the editor and other properties are hidden. This issue is not addressed in the description below. For more information about customizing the visibility of properties in the wizard, see the section Hiding Object Class Properties and Functionalities in the Wizard for a Specific User Profile.
- One or more properties must be defined in order to complete the wizard step and advance to the next wizard step. This issue is not addressed in the description below. For more information about defining post-conditions for a wizard step, see the section Defining a Post-Condition for a Wizard Step.

> Please note if a custom editor is embedded in a wizard and the **Tab as Separate Steps** attribute set to `True`, any conditions configured for the custom editor will only be relevant for the current tab/wizard step. The conditions cannot refer to interface controls that are available on other tabs/wizard steps. For more information about the configuration of conditions for custom editors, see the section Specifying Conditional Constraints in the Custom Editor.

To configure a wizard step of the type `Editor`:

1) Expand the custom wizard and click the wizard step to open its attribute window.
2) Define the following attributes, as needed:

- **Wizard Step Type**: Select `Editor` to specify a standard and, if relevant, custom editor for the wizard step.
- **View**: Select the standard editor relevant for the object class targeted by the wizard step.

Typically, this will be the object class that the wizard has been created for. However, it is also possible to configure the wizard step so that an object in a different object class is processed in the wizard step. In this case, you would select the base class of the query defined in the **Base Object via Query** attribute. For more information specifying a different object class for a wizard step, see the section Configuring a Different Target Object for a Wizard Step.

- **Custom Editor**: Select a custom editor if you want to include a custom editor in the wizard step.
- **Tab as Separate Step**: If the editor has multiple tabs, then you can specify that each tabbed page is treated as a separate wizard step in the wizard process. For example, if a standard editor with 2 tabs and a custom editor with 1 tab is assigned to the wizard step, the first tab of the standard editor would be step 1, the second tab would be step 2, and the custom editor would be step 3. To specify that each tab is treated as a separate step in the wizard process, select `True`. Select `False` if the tabbed pages in the standard and custom editors should constitute one wizard step.
- If you plan to implement post-conditions for the wizard step and the **Tab as Separate Step** attribute has been set to `True`, the post-conditions will be checked after the last tab of the standard or custom editor has been reached. In this case, you must ensure that the attribute **Execute Post-Conditions on Back Moves** for the wizard is set to `False`. For more information about the configuration requirements for wizard steps with post-conditions, see the section Defining a Post-Condition for a Wizard Step.
- **Page**: If you have defined a standard or custom editor with multiple tabbed pages for the wizard step, you can configure the wizard step to display only one of the tabbed pages and hide all other tabbed pages that are not relevant. To specify that only a selected tabbed page in the standard editor is displayed for the wizard step, select the relevant page for the wizard step. All other tabbed pages will be hidden.

  For example, if you have a situation where you have 5 tabbed pages in a standard editor and you want to display 2 of the tabbed pages, you could assign the standard editor to two different wizard steps, defining the visibility of one of the pages for each wizard step.

- **Is Read-Only**: Typically, users accessing a wizard step that is associated with an editor should have ReadWrite access permissions. Therefore, in order for users to be able to capture data and edit existing data for the selected wizard step, you must ensure that the **Is Read-Only** attribute is set to `False` If users should have read-only access permissions to the selected wizard step, set the **Is Read-Only** attribute to `True`.

3) If properties have been specified as excluded for a class setting, you may explicitly define them to be included in the context of a wizard step. To do so, define the following attributes:

- **Class Name**: Specify the class targeted by the wizard step to be defined. Once a class is specified in the **Class Name** attribute.
- **Override Property Exceptions**: Select the properties in the **Local Class Settings** editor that shall be included in the scope of the wizard step. Only the properties that have been excluded via the class settings of the class specified in the **Class Name** attribute will be displayed in the **Local Class Settings** editor.

4) In the toolbar, click the **Save** button to save your changes.

Controlling the Visibility of Tabbed Pages in Wizard Steps

Please note that visibility rules will not be further developed by Software AG. For reasons of backward compatibility, the following section is available regarding the definition of visibility rules for custom editors. However, wizard step pre-conditions and post-conditions are a much more flexible mechanism to control the visibility of tabbed pages in an editor as well as other types of wizard steps. It is highly recommended that you reconfigure existing visibility rules as pre-conditions or post-conditions for a wizard step. For more information, see the section Defining Pre-Conditions and Post-Conditions for a Wizard Step.

It is possible to configure a wizard so that the visibility of specified tabs in a custom editor that is assigned to a wizard step is determined by a set of visibility rules. Depending on the value selected by the user in the first wizard step, the user will proceed to the specified tab in the next wizard step that meets the condition specified by the visibility rule. In this way, the tabs displayed for a wizard step are controlled by the value the user defines for the custom object class property.

Visibility rules are defined for the custom editor embedded in the wizard. The visibility rules are applied to the custom editor after data has been entered for the custom object class property of the type `Boolean`, `String`, or `Integer` determining the visibility and the data has been saved to the database. This is either when the custom editor is closed and then reopened or when the user clicks the **Next** button in the wizard. Please note the following:

- A custom object class property of the type `Boolean`, `String`, or `Integer` must be created for the relevant object class. If the custom object class property is of the type `String` or `Integer`, then an enumeration with enumeration items must also be defined. The enumeration items represent the values that can be selected in the custom editor.
- The custom editor should have multiple pages (tabs). You must create one page on which the custom object class property is displayed and one additional page for every property value that can be selected for the custom object class property. A visibility rule for every option (`True`/`False`) available for the relevant `Boolean` property or every enumeration item available for the relevant custom object class property. For example, for `Boolean` properties, a page should be created of Yes is selected and a page should be created if No is selected; for properties with an enumeration, one page should be created for each enumeration item defined for the enumeration.
- The visibility of tabs can be controlled for a specified wizard step only.

To define visibility rules for a custom editor with multiple tabs:

1) Go to the **Presentation** tab, expand the **Custom Editors** folder, right-click the relevant custom editor  nd select **New Visibility Rule**. The visibility rule  is displayed below the custom editor.
2) Click the visibility rule to open the attribute window and define the following values:

- **Name**: Enter a unique name for the visibility rule.
- **Property**: Select the custom object class property that the visibility rule is based on.
- **Property Value**: Select the value of custom object class property for which the visibility rule is valid. You can select either `True` or `False` for a custom object class property of the type `Boolean` or an enumeration item for a custom object class property of the types `String` or `Integer`.

- **Check Result Type**: Select `Positive` if the value defined in the **Property Value** attribute indicates that the visibility rule is fulfilled if the value is selected. Select `Negative` if the value defined in the **Property Value** attribute indicates that the visibility rule is fulfilled if the value is not selected.
- **Description**: Enter text in order to provide information about the visibility rule. This text is not displayed in the Alfabet interface.

Instead of defining the visibility rule for a specified custom object class property, you can alternatively define an Alfabet query to determine the condition specified in the visibility rule via the **Alfabet Query** attribute. For more information about configuring an Alfabet query, see the section Defining Queries.

3) In the toolbar, click the **Save** button to save your changes.
4) Next, you must associate the page that the visibility rule is assigned to in the **Visibility Rules** attribute of the tabbed page. Expand the **Custom Editors** node, right-click the relevant custom editor and select **Design Editor**. The custom editor is displayed in the editor designer.
5) Click the tabbed page (not the tab) that should be displayed if the visibility rule is valid. In the attribute window, select the visibility rule that determines the visibility of the tabbed page in the **Visibility Rules** attribute.
6) Continue this procedure until all required visibility rules have been defined for the various tabbed pages of the custom editor.
7) In the toolbar, click the **Save** button to save your changes.
8) As a final step, you must ensure that the wizard step is correctly defined in order to allow for the execution of the visibility rules. Expand the **Wizards** folder and right-click the relevant wizard step that you want to associate the custom editor and its visibility rules with.
9) In the attribute window for the wizard step, ensure that the following attributes are specified:

- **Wizard Step Type**: Select `Editor`.
- **View**: Select the standard editor for the respective object class.
- **Custom Editor**: Select the relevant custom editor that has been defined with visibility rules.
- **Is Read-Only**: Select `False`.
- **Tab as Separate Step**: Select `False`.

10) In the toolbar, click the **Save** button to save your changes.

Configuring a Wizard Step to Display a Standard Page View

A wizard step can constitute a standard Alfabet view.

A link will be displayed per default in the header that opens the context-sensitive Help. You can replace the preconfigured text "Click to access Help for this wizard step." with a customized HTML text for the header. It is also possible to configure instructional text in the header of the wizard step. For more information

about configuring HTML text in the header of the wizard step, see the section Configuring the Header Text of a Wizard Step.

There are a number of configuration possibilities for a wizard step of the type `View`:

- You can configure the wizard step so that users may have ReadOnly access permissions or Read/Write access permissions.
- You can require that specific data is defined in the page view in order to complete the wizard step and advance to the next wizard step. This issue is not addressed in the description below. For more information about defining post-conditions for a wizard step, see the section Configuring Wizard Step Actions for a Wizard Step.

Please note that the **Show Collaboration Panel** button which opens the **Collaboration** capability is not available in page views embedded in a wizard step.

1) Expand the custom wizard and click the wizard step to open its attribute window.
2) Define the following attributes, as needed:

- **Step Type**: Select `View` to specify a standard Alfabet page view for the wizard step.
- **View**: Select the page view to display for the wizard step. All permissible standard page views in Alfabet are listed in the drop-down list. Typically, this will be a page view associated with the object class that the wizard has been created for. However, it is also possible to configure the wizard step so that an object in a different object class is processed in the wizard step. In this case, you would select the base class of the query defined in the **Query** attribute (or **Query as Text** attribute). For more information specifying a different object class for a wizard step, see the section Configuring a Different Target Object for a Wizard Step.
- **Is Read-Only**: If users accessing the selected wizard step should only have ReadOnly access permissions, select `True`. If users should have Read/Write access permissions, select `False`.

3) If properties have been specified as excluded for a class setting, you may explicitly define them to be included in the context of a wizard step. To do so, define the following attributes:

- **Class Name**: Specify the class targeted by the wizard step to be defined. Once a class is specified in the **Class Name** attribute.
- **Override Property Exceptions**: Select the properties in the **Local Class Settings** editor that shall be included in the scope of the wizard step. Only the properties that have been excluded via the class settings of the class specified in the **Class Name** attribute will be displayed in the **Local Class Settings** editor.

4) In the toolbar, click the **Save** button to save your changes.

Configuring a Wizard Step to Display a Configured Report

A wizard step can constitute a configured report. You can configure the wizard step so that users may view a tabular query-based custom report or a query-based custom report displaying graphics (such as a tree-map or layered diagram report). Please note that a report can only be viewed in a wizard. A report cannot be opened from within a wizard.

For wizard steps of the type `Report`, a link to your enterprise's custom Help may be available if custom Help has been configured. For more information about configuring a custom report, see the section Assigning Custom Help and Automated Help Assistants to Configured Reports. However, if you plan to implement custom HTML text in the header, you can embed your custom Help in the HTML text as a hyperlink. For more information about configuring HTML text in the header of the wizard step, see the section Configuring the Header Text of a Wizard Step.

1) Expand the custom wizard and click the wizard step to open its attribute window.
2) Define the following attributes, as needed:

- **Step Type**: Select `Report` to specify a configured report for the wizard step.
- **View**: Select the configured report to display for the wizard step. All reports configured for your enterprise are listed in the drop-down list.

3) If properties have been specified as excluded for a class setting, you may explicitly define them to be included in the context of a wizard step. To do so, define the following attributes:

- **Class Name**: Specify the class targeted by the wizard step to be defined. Once a class is specified in the **Class Name** attribute.
- **Override Property Exceptions**: Select the properties in the **Local Class Settings** editor that shall be included in the scope of the wizard step. Only the properties that have been excluded via the class settings of the class specified in the **Class Name** attribute will be displayed in the **Local Class Settings** editor.

4) In the toolbar, click the **Save** button to save your changes.

Defining a Subordinate Wizard Step or a Branch of Subordinate Wizard Steps

A subordinate wizard step is a wizard step that is subordinate to a wizard step and forms a branch of wizard steps in the wizard process. A wizard step can have a hierarchy of subordinate wizard steps, each of which could also have subordinate wizard steps. For each branch of subordinate wizard steps that you define, the root wizard step may have a pre-condition defined in order to determine whether the user should enter the wizard step and its branch of subordinate wizard steps. The pre-conditions are configured as Alfabet queries or SQL queries.

The logical condition would determine whether the wizard step is entered or whether it should be skipped. If the pre-conditions for the root wizard step are not met, the wizard step and its subordinate wizard steps will be skipped, and the wizard will advance to the next wizard step for which the pre-conditions are fulfilled. If the pre-condition for the root wizard step is met, the wizard will advance through the branch of wizard steps. When the leaf-level sub-step is reached, the wizard would then advance to the next wizard step on the root level of wizard steps.

Post-conditions may be defined for any subordinate wizard step in order to prompt users to provide data for specified standard or custom object class properties before proceeding to the next wizard step.

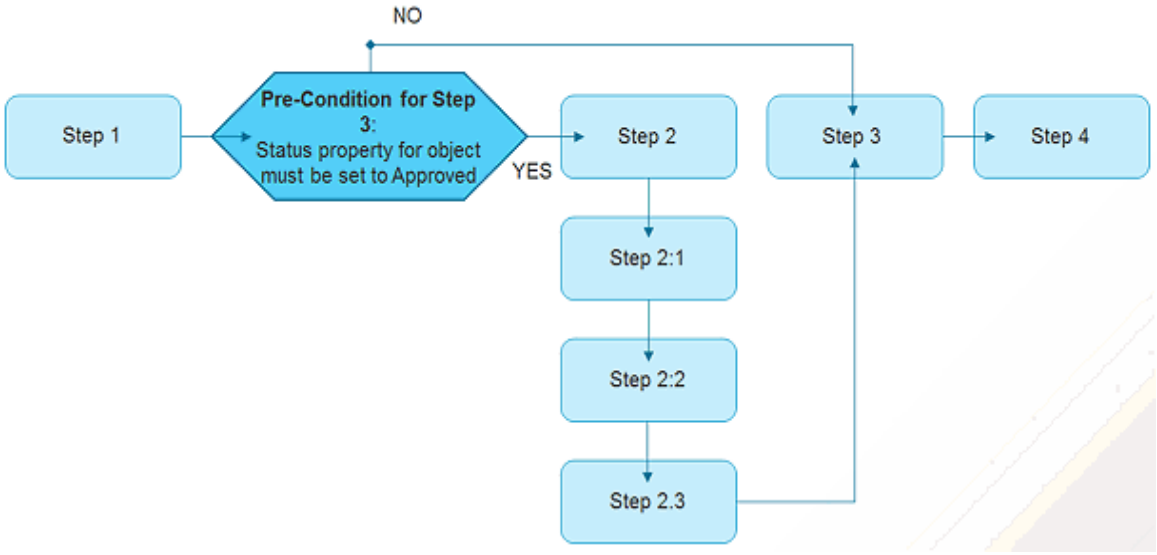


FIGURE: Wizard process with a branch of subordinate wizard steps

In the example above, if the pre-condition for wizard step 2 is not fulfilled, the wizard would ignore the Wizard Step 2 and all subordinate wizard steps and continue to Wizard Step 3. If the pre-condition is fulfilled, the wizard will enter Wizard Step 2. The wizard would then advance to the next subordinate wizard step in the branch. When Wizard Step 2.3 is reached at the leaf-level of the branch, the wizard would advance to Wizard Step 3.

Once a subordinate wizard step is created, the process of defining the subordinate wizard step and its attributes is similar to that of a conventional wizard step. Each subordinate wizard step may have instructions configured that are displayed in the header of the wizard step and like a wizard step, the subordinate wizard step may be an editor, view, or report in a wizard. A subordinate wizard step may be reused within a wizard as well as in other wizards.

The final wizard step in a wizard may not have a pre-condition defined. If a pre-condition is defined for the final wizard step or subordinate wizard step and the pre-condition is not fulfilled, an error will occur (for example, the wizard will not be able to advance to a next wizard step and the wizard will stagnate). If a pre-condition is required for the last logical step or subordinate wizard step in the wizard, it is recommended that you add another final concluding wizard step with no pre-condition and, for example, only static text defined (with a message such as "You have completed the wizard").

For example, the wizard scenario requires a branch at the end of the wizard where the last wizard step may be Wizard Step A or Wizard Step B; if the pre-condition for Wizard Step A is fulfilled, then the final wizard step will be Wizard Step A but if the pre-condition for Wizard Step B is fulfilled, then the final wizard step will be Wizard Step B. However, only one wizard step (for example, Wizard Step A) can be defined as the final wizard step which has the **Exit** button that allows the wizard process to be completed.

In the example, however, the wizard ends in a fork with two possible wizard steps as a potential last step. If the last wizard step is Wizard Step B, the **Exit** button will not be available. In this case, the wizard will stagnate and cannot be completed.

For a wizard scenario like this one, you must define a final wizard step that has no pre-condition configured to follow Wizard Step A and Wizard Step B. The final wizard step would require no data input but would simply allow the user to complete the wizard step via the **Exit** button.

If one or more wizards are configured to open as sub-branches of a base wizard, each new wizard that is triggered to open may be configured to open in a modal window. The user will not be able to return to a preceding wizard until the current wizard has been completed. The configuration of wizards stacked in modal windows is specified in the **Stacked Wizards** attribute in the server alias configuration. For more information, see the section *Configuration Attributes for the Alfabet Components* in the reference manual *System Administration*.

To define subordinate wizard steps for a selected wizard:

1) Go to the **Presentations** tab, expand the custom wizard and right-click the relevant wizard step that you want to define subordinate wizard steps for and select **New Wizard Sub-Step**. A wizard step labelled `Step_<Number>` is added to the wizard view.

   You can copy an existing subordinate wizard step and modify it as needed. To do so, you must create an empty wizard step for the wizard you are defining, copy the existing wizard step (via **right-click >Copy**), and paste the wizard step on the empty wizard view (**right-click >Paste**). The entire wizard step definition is copied and can be modified, as needed.

2) Define the wizard step as defined in the section Specifying Wizard Steps for a Custom Wizard. In addition, you must define the following:

- Go to the custom wizard node and click to open the attribute window. Specify the **Type** attribute for the selected wizard to determine the numbering to display in the wizard header for the subordinate wizard steps:
- Select `Tree` to number all subordinate wizard steps so that subordinate wizard step 1, subordinate wizard step 2, and subordinate wizard step 3 that are subordinate to wizard step 1 will be numbered step 1.1, 1.2, and 1.3 in the wizard header. Please note that if the **Show Wizard Steps Combo-Box** attribute is set to `True` for the wizard, then the subordinate wizard steps will be displayed in the **Go to Step** field.
- Select `Straight` to number all subordinate wizard steps so that all subordinate wizard step that are subordinate to wizard step 1 will all have the same number as their ascendant wizard step. In this case, subordinate wizard step 1, subordinate wizard step 2, and subordinate wizard step 3 would all be numbered 1 of 7 in the wizard header. Please note that if the **Show Wizard Steps Combo-Box** attribute is set to `False` for the wizard, then the subordinate wizard steps will not be displayed in the **Go to Step** field.

  If the **Go to Step** field is configured to be displayed in the wizard (**Show Wizard Steps Combo-Box** attribute = `True` for the wizard), then the subordinate wizard steps will be displayed in the **Go to Step** field if `Tree` is selected in the **Type** attribute. The subordinate wizard steps will not be displayed in the **Go to Step** field, if `Straight` is selected in the **Type** field.

- Define a pre-condition for the parent wizard step in the branch. This is required in order to provide the criteria that must be fulfilled to enter the subordinate wizard step. For more information about defining a pre-condition, see the section Defining Pre-Conditions and Post-Conditions for a Wizard Step.
- Optional: Create another subordinate wizard step or branch of sub-steps for each subordinate wizard step. In this way, you can create a hierarchy of subordinate wizard steps in the order in which they should be processed. To create a subordinate wizard step of an existing subordinate wizard step, right-

click the subordinate wizard step and select **New Wizard Sub-Step**. Continue this process until the branch of subordinate wizard steps is complete.

3) In the toolbar, click the **Save**  button to save your changes.

Configuring the Header Text of a Wizard Step

Instructional text can be configured for the header text for one or more wizard steps. You can either specify plain text by configuring the **Caption** and **Description** attributes of the wizard step or configure HTML text in the **Header HTML** attribute. The size of the header is static and will NOT adjust its size to display all text you define. Therefore, the header text should be short and succinct.

Please not the following advantages and disadvantages for each option:

- Plain text:

- Plain text is configured via the **Caption** and **Description** attributes of a wizard step. The configuration of the **Caption** and **Description** attributes is simple. For more information about configuring plain text via the **Caption** and **Description** attributes of a wizard step, see the section Creating a Wizard.
- The text is plain text and cannot be formatted.
- The **Caption** and **Description** attributes will be available in the vocabularies and can be translated using the standard translation capabilities available.
- The specification of the **Format String** attribute configured for the wizard will also be displayed. For more information about the configuration of the format string, see the section Creating a Wizard.
- The availability of the online Help for views and configured reports is not impacted. F

- HTML text:

- HTML text is configured via the **Header HTML** attribute of a wizard step. You must follow the guidelines described below to ensure the HTML is valid.
- The HTML must be XML-conform HTML, compliant with HTML 5, and use standard HTML tags. The HTML may contain basic elements to capture information about the wizard step as well as the object targeted by the wizard step.
- You can include text formatting and color, hyperlink to URLs, and multiple translations of the text.
- Alfabet proprietary elements may be included to retrieve data from the database.
- A stylesheet that specifies the layout and visualization of the data may be stored in the **Internal Document Selector**.
- The online Help for views and configured reports will not be available unless explicitly linked in the customized HTML text.
- The HTML text will not be available in the vocabularies and cannot be translated using the standard translation capabilities available. If translation to a secondary language is required, you must provide the translation in the HTML specification.

The specification in the **Header HTML** attribute for a wizard step supersedes the definition of the **Caption** or **Description** attribute of the wizard steps as well as the definition of the **Format String** attribute for the wizard.

To configure HTML text to display in the header of the wizard step:

1) Expand the custom wizard and click the wizard step to open its attribute window.
2) In the **HTML Header** attribute, configure HTML text for the header. Please note the following:

- The HTML may be defined in a standard XHTML editor and pasted in the **HTML Header** attribute.
- The HTML must be compliant with XHTML, XML-conform HTML, compliant with HTML 5, and use standard HTML tags. The HTML header implements standard HTML elements.
- The code must start with an `<xhtml>` tag and end with `</xhtml>`. Standard HTML elements must be written in lower-case letters in order to be correctly parsed: `<xhtml>`, `<html>`, `<head>`, `<body>`, and `<culture_>`. The definition of `<head>`, `<body>`, and `<culture_>` is optional.
- The formatting of the HTML can either be written explicitly in the `<body>` element or can be specified via a stylesheet that is stored in the **Internal Document Selector**. In this case, the HTML must refer to the target CSS file in a `<link>` element. The CSS file should contain all necessary styles to display the content of the HTML. Please note that the CSS file may not be stored in the root folder of the **IDOC** explorer. The CSS file stored in the **Internal Document Selector** must be located in a document folder that is subordinate to the root folder of the **IDOC** explorer. To upload a file to the **Internal Document Selector**, see the section *Uploading Documents and Managing User Permissions to Document Folders in the Internal Document Selector* in the reference manual *User and Solution Administration*. For example:

```
<xhtml>
   <culture_1033>
      <html>
      <body style="margin:4px;background:#3d4b60;overflow:hidden;">
         <link type="text/css" rel="stylesheet"
         href="IDOC:\CSS\my_style_file.css"></link>
         <p>This is the content for the header of the wizard
         step.</p>
         ...
      </body>
      </html>
   </culture_1033>
</xhtml>
```

- All CSS formatting instructions must end with `!important` to ensure that they are processed by the Alfabet application. For example:

```
<xhtml>
   <culture_1033>
      <html>
      <body style="margin:4px !important;background:#3d4b60
      !important;overflow:hidden !important;">
```



```
            <style type="text/css">
                p.header
                {
                    font-family:verdana !important;
                    font-size:18px !important;
                    color:#ff0000 !important;
                    text-align: Left !important;
                }
            </style>
            <p class="header">This is the content for the header of the
            wizard step.</p>
            ...
        </body>
        </html>
    </culture_1033>
</xhtml>
```

- If translation to a secondary language is required, you must provide the translation in the HTML specification. Please note that HTML texts cannot be translated via the **Translation Editor** available in Alfabet Expand. Please consider the following:
- If the HTML description is required in additional languages, each language text must be defined within language elements (For example, `<culture_1031>` for English, `<culture_1033>` for German, etc.)
- The element `<culture_xxx>` must be specified as a child of the root element `<xhtml>`. The element `<html>` must be specified as a child of the element `<culture_xxx>`. The element `<html>` contains the HTML specification in the relevant language. For example, to provide information in English and German:

```
<xhtml>
    <culture_1033>
        <html>
            <body>
                <h3>Glossary: Application</h3>
                <p>An application is a fully-functional integrated IT
                product that provides functionality to end users
                and/or to other applications. As such, an application
                supports the business to accomplish its mission.
                Applications operate on a platform made up of
                hardware and software components necessary to run the
                application.</p>
            </body>
        </html>
    </culture_1033>
    </culture_1031>
        <html>
```



```
            <body>
                <h3>Glossar: Applikation</h3>
                <p>Eine Applikation ist ein voll funktionsfähiges,
                integriertes IT-Produkt, das Funktionalitäten für
                Endanwender und/oder für andere Applikationen bietet.
                Eine Applikation unterstützt das Unternehmen bei der
                Zielerreichung. Applikationen werden auf einer
                Plattform betrieben, die aus den für die Ausführung
                der Applikation erforderlichen Hardware- und
                Software-Komponenten besteht.</p>
            </body>
        </html>
    </culture_1031>
</xhtml>
```

- Each Alfabet proprietary element `<AlfaHtmlElement>` should be included in a standard HTML tag such as `<body>`, `<td>`, `<span>`, `<div>`, etc. in order to retrieve data from the database. The following Alfabet query language parameters can be used in the `ApplyTo` attribute to define the object whose data is to be displayed:
- `@WIZARDBASE`: to retrieve data about the object that the wizard step references (if the `ShowProps` attribute has been added to the element `<AlfaHtmlElement>` without a child `SqlText` element)
- `@PREVIOUS_STEP`: to retrieve data about the previous wizard step
- `@WIZARD`: to retrieve data about the current wizard
- The attributes `ApplyTo`, `ShowProps`, `PropName`, and `FormatString` can also be used to retrieve and display data. For example, to display the object state of the object targeted by the current step, you would specify `@WIZARDBASE` in the `ApplyTo` attribute and the object class properties `ObjectState` in the `ShowProps` attribute:

```
<span><AlfaHtmlElement ApplyTo="@WIZARDBASE"
ShowProps="Description"/></span>
```

- The attribute `SqlText` may also be included in the element `<AlfaHtmlElement>`. If the attribute `SqlText` is included as a child of the element `<AlfaHtmlElement>`, then the object reference is available via the parameter `@BASE`. This attribute allows you to include an Alfabet query or native SQL query to retrieve data. Please note that if the attribute `SqlText` is included in the element `<AlfaHtmlElement>`, the attribute `ShowProps` will be ignored.

In the example below, multiple application groups will be returned for the query. The `MaxCount` attribute allows you to limit the number of rows (results) displayed. If the number of results is greater than the value in the `MaxCount` attribute, only the first results found in the database will be displayed. If the query includes a sorting instruction (for example, in the query below: `ORDER BY`), the first results found in the sort order will be displayed:

For example:

```
<AlfaHtmlElementMaxCount ="5" ApplyTo="@BASE">
    <SqlText> <![CDATA[
        SELECT appg.REFSTR, appg.Name
        FROM APPLICATIONGROUP appg, RELATIONS rel
```



```
                WHERE rel.FROMREF = appg.REFSTR
                    AND rel.PROPERTY = 'Applications'
                    AND rel.TOREF = @BASE
                ORDER BY appg.NAME
                ]]>
            </SqlText>
        </AlfaHtmlElement>
```

- The attribute `Filter` may be included in the element `<AlfaHtmlElement>`. This attribute allows you to include color instruction to color properties based on a specified value.

For example:

```
<AlfaHtmlElement ApplyTo="@WIZARDBASE"
ShowProps="ObjectState">
    <Filter> <![CDATA[COLORASSIGNMENT(EqualTo, "Active",
    #c0c000, Transparent);]]> </Filter>
    <Filter> <![CDATA[COLORASSIGNMENT(EqualTo, "Retired",
    #00c000, Transparent);]]> </Filter>
    <Filter> <![CDATA[COLORASSIGNMENT(EqualTo, "Draft",
    #c00000, Transparent);]]> </Filter>
</AlfaHtmlElement>
```

3) In the toolbar, click the **Save**  button to save your changes.

Defining Pre-Conditions and Post-Conditions for a Wizard Step

For each wizard step, you may define one or more pre-conditions and post-conditions that must be fulfilled before the wizard step can be entered (pre-condition) or exited (post-condition). Each pre-condition and post-condition require an Alfabet query or native SQL query to be defined that checks whether the pre-/post-condition has been fulfilled. In the case of a post-condition, you can additionally define a message to display information to a user if the post-condition has not been fulfilled. Each wizard step may have multiple pre-conditions and post-conditions defined.

A pre-condition may NOT be configured for the first wizard step NOR for the final wizard step or subordinate wizard step. If you do so and the pre-condition is not fulfilled, an error will occur. In this case, for example, the wizard could stagnate and not advance to the next wizard step. If a pre-condition is required for the last logical step or subordinate wizard step in the wizard, it is recommended that you add another final concluding wizard step with no pre-condition and only static text defined (for example, with a message such as "You have completed the wizard").

The following information is available:

- Defining a Pre-Condition for a Wizard Step
- Defining a Post-Condition for a Wizard Step
- Determining the Sequence of Pre-Conditions and Post-Conditions

Defining a Pre-Condition for a Wizard Step

Pre-conditions can be defined for a wizard step in order to determine if the wizard step should be entered. A pre-condition is based on an Alfabet query or a native SQL query that checks the minimum conditions that should apply for the wizard step to be entered. If certain conditions are not met for a wizard step, then that wizard step will be skipped and the wizard will advance to the next wizard step for which the pre-conditions are met or for which no pre-conditions are defined. Each wizard step may have multiple pre-conditions defined. All pre-conditions must be fulfilled before the wizard step may be entered.

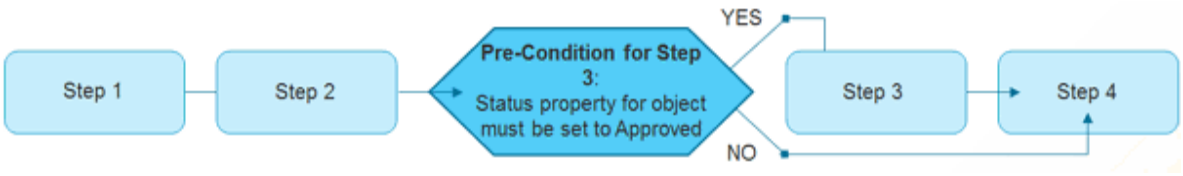


FIGURE: Approval of object is required to enter Step 3

If pre-conditions are not fulfilled for a wizard step, the user may enter the next permissible wizard step. This may be the next wizard step for which the pre-conditions are met or for which no pre-conditions are defined.

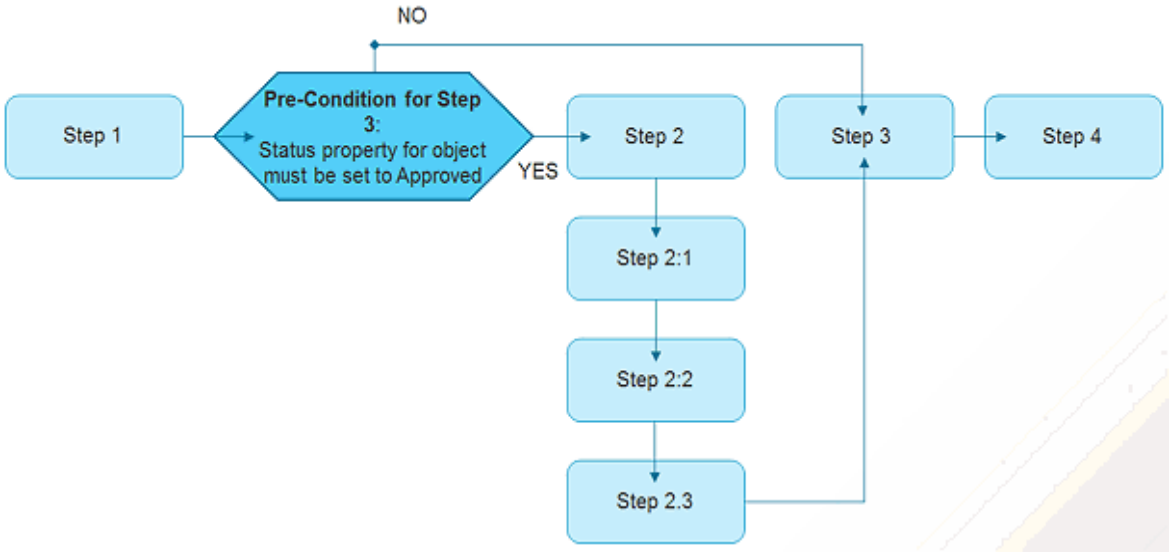


FIGURE: Wizard with a branch of subordinate wizard steps

If subordinate wizard steps are defined for a wizard step, the subordinate wizard steps will only be entered if the pre-conditions are met for the root wizard step, as displayed in the figure above. If the pre-conditions for the root wizard step are not met, the wizard step and its subordinate wizard steps will be skipped and the wizard will advance to the next permissible wizard step.

In the example above, if the pre-condition for wizard step 2 is not fulfilled, the wizard would ignore the wizard step 2 and all subordinate wizard steps and continue to wizard step 3. If the pre-condition is fulfilled, the wizard will enter step 2. The wizard would then advance to the next subordinate wizard step in the branch. When step 2.3 is reached at the leaf-level of the branch, the wizard would advance to wizard step 3.

You can define multiple pre-conditions per wizard step. Once all pre-conditions have been created for a wizard step, the order that they are to be executed should be defined. For more information, see the section Determining the Sequence of Wizard Steps in the Wizard.

A pre-condition may NOT be configured for the first wizard step NOR for the final wizard step or subordinate wizard step. If you do so and the pre-condition is not fulfilled, an error will occur. In this case, for example, the wizard could stagnate and not advance to the next wizard step. If a pre-condition is required for the last logical step or subordinate wizard step in the wizard, it is recommended that you add another final concluding wizard step with no pre-condition and only static text defined (for example, with a message such as "You have completed the wizard").

To define a pre-condition for a selected wizard step or subordinate wizard step:

1) Expand the custom wizard and click the wizard step to open its attribute window.
2) Right-click the **Pre-Conditions** node and select **New Pre-Condition**. You will see the new pre-condition below the relevant node.

A pre-condition can be created based on a copy of an existing condition. To do so, select the pre-condition you want to copy and right click and select **Copy**. Click the wizard step or subordinate wizard step that you want to copy the pre-condition to and select **Paste**. The values define for the pre-condition's attributes are copied and can be modified, as needed.

3) Define its attributes in the attribute window as needed:

- **Technical Name**: Enter a name for the pre-condition.
- **Technical Comment**: Enter any relevant information about the pre-condition.
- Define the query that is executed to ascertain whether the pre-condition has been fulfilled. You can define either an Alfabet query or an SQL query. For information about defining an Alfabet query or SQL query, see the chapter Defining Queries. Define one of the following:
- **Query**: Click the **Browse** button to open the **Alfabet Query Builder** and define an Alfabet query.
- **Query as Text**: Paste in an Alfabet query or native SQL query that you have defined in a text editor.

The following is an example of a pre-condition that specifies that the **Release Status** property of the application must be set to `Approved` in order to enter the subordinate wizard step:

```
ALFABET_QUERY_500
FIND
APPLICATION
WHERE
    (AND
       Application.REFSTR CONTAINS:BASE
       Application.Status LIKE 'Approved'
    )
```

Data update is not allowed in a wizard step pre-condition. If a data update is required, you should define a wizard step action of the type **Action on Exited Step**. For more information, see the section Defining Property Values To Be Automatically Updated When a Wizard Step Is Exited.

- **Check Result Type**: Select either `Positive` or `Negative` to define what constitutes a fulfilled pre-condition.
- `Positive` means that the pre-condition is fulfilled if the query has delivered a result (at least one row.)
- `Negative` means that the pre-condition is fulfilled if the query doesn't deliver any results.
- **Message**: Specify a warning message to display if the pre-condition is not fulfilled. The text should provide the user with information about what is required to fulfill the pre-condition. If no warning message is configured, then the standard warning message preconfigured by Alfabet will be displayed to the user if the pre-condition is not fulfilled.

4) Click the **Save**  button to save your definitions.

Defining a Post-Condition for a Wizard Step

A post-condition is a mechanism that reviews the definition of an object class property for a specified standard object class property or custom object class property for the object being edited in the wizard step. The post-conditions of a wizard step specify the minimum conditions that should apply for the wizard step to be exited. For example, post-conditions might be configured to validate that a specific object class property is defined before proceeding to the next wizard step, or that an object class property is filled with a specified value (for example, a specific release status), or that values are aligned with other values (for example, that start and end dates are aligned with a specified referenced object.)

You can configure whether it is optional or mandatory for the user to fill in the value for the specified standard or custom object class property. If the validation performed on the standard or custom object class property specified in the post-condition fails, a customized prompt message will appear. If the value targeted by the post-condition is optional, the user will be able to continue to the next wizard step without entering the respective object class property. If the value targeted by the post-condition is required, the user cannot continue to the next wizard step until a valid value has been entered for the object class property and the post-condition is validated.

Please consider the following when configuring a post-condition:

- You must ensure that the object class property that is being validated by the post-condition can actually be defined by the user in the wizard step. For example, will the user have the relevant access permission to define or edit the object class property? Or, if the object class property `Status` must be changed, is the release status that is to be modified by the user editable?
- If standard editors are defined for the wizard, the standard object class properties that are preconfigured by Software AG as mandatory properties must be filled in for the user to proceed to the next wizard screen. If any custom object class properties are configured as mandatory for a custom editor, these fields will also display the mandatory red star and must be filled in for the user to proceed.

Please note that you must ensure that the configuration of post-conditions for a wizard step and the configuration of pre-conditions for a subsequent wizard step are not contradictory.

Please note that the post-condition is executed AFTER the database transaction is performed. This is of particular relevance for post-conditions associated with standard or custom editors because the first attempt will be to save the user's data which may fail, for example, due to

mandatory or uniqueness conditions being violated. Once this save has been successful, the post-conditions will be executed.

If the attribute **Execute Post-Conditions on Back Moves** is set to `False`, the post-conditions will not be executed if the user clicks the **Back** button. However, the validation of editor-related constraints such as preconfigured mandatory properties or uniqueness constraints for a set of properties will continue to be performed if the user clicks the **Back** button.

Post-conditions may also be implemented in the Alfabet solution to validate the definition of objects. The following options are possible:

- Post-conditions specified for a wizard step of the type `Editor` will be applied if a user edits a relevant scalar or reference property in-line in an object profile or object cockpit. If a post-condition is not fulfilled after a value has been entered for a permissible property in the object profile or object cockpit, the error message configured for the post-condition will be displayed. Permissibility of inline editing can be specified by your solution designer on the level of an object view, class setting, or GUI scheme. For more information about configuring in-line editing, see the section Configuring Inline Editing of Attributes in the Object View.

  Please note the following regarding the inline editing functionality:

  - Only scalar and reference properties that are editable in the editor/wizard available in the object profile or object cockpit can be edited via inline editing.
  - All unique constraints defined for the class as well as any post-conditions configured for the wizard associated with the object profile/object cockpit will be applied to the data entered.

    Please note that data is saved to the Alfabet database prior to the post-condition being validated. Therefore, data entered via inline editing will be saved even if the post-condition is not fulfilled. This is because the data must be available in the Alfabet database in order to be evaluated for the post-condition. If the data entered does not fulfill the post-condition, the configured warning message will be displayed explaining that the input must be corrected in order to fulfill the post-condition.

  - Inline editing may be limited or prevented if the syntax of the post-condition is not correct. In this case, an error will be displayed if the user tries to open the wizard.

- A section displaying violated wizard post-conditions may be displayed in a **Personal Information** section of the object cockpit. The wizard validations section will display the messages configured for all wizard post-conditions that have been violated for the object displayed in the object cockpit. To include this information, you must add a Value Interface control of sub-type `PersonalInfo` to the flow panel of the object cockpit. For more information about configuring the display of wizard validation information in the object cockpit, see the section Adding Workflow, Assignment, Microsoft Teams Meeting Information and Object Validation Information to the Object Cockpit in the chapter Configuring Object Views.

- Object validation rules based on wizard step post-conditions may be configured. In this context, any object in the object class that the post-condition is associated with is validated when a user accesses the object in the object profile. Violations of the validations rules will be displayed in the object view and the user can then correct the cause of the violations. If the object validation functionality is enabled, all violations of each post-condition will be displayed in the header of the object profile of the currently selected object. The icon and message displayed if the post-condition is not fulfilled will depend on the configuration of the post-condition.

Please note that implementing the execution of the object validation functionality **may significantly impact performance** of the Alfabet solution since each upload to the Alfabet database will trigger the execution of all validation rules. Therefore, it is highly recommended that validation rules are configured for object class properties via a check entry definition in the context of a custom object cockpit. For more information about the configuration of a check entry definition for object validation, see the section Adding a Check Entry to the Object Cockpit in the chapter Configuring Object Views.

To enable validation rules associated with the post-conditions specified for a wizard:

- In order for the validation rules to be executed, one of the following is required:
- The object view must have a custom button for which the **Operation** attribute is set to `WizardEdit` or `Edit` button and the relevant wizard with the post-conditions must be specified in the **View** attribute. For more information about configuring buttons for a custom object view, see the section Configuring Custom Buttons for the Toolbar of a Custom Object View in the chapter Configuring Object Views.
- The wizard with the post-conditions must be specified for the class setting of the base object or object class stereotype. If no wizard is defined for this class setting, the wizard defined for the default class setting of the object class stereotype or base object class will be used. If no post-conditions should be displayed for users accessing the object view, you must ensure that no post-conditions are defined for the wizards specified in the class setting or default class setting specified for the relevant view scheme. For more information about specifying class settings, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.
- The specification of an object view with a relevant custom button will have precedence over a wizard assigned to the class settings.
- The user must specify the execution of object validation in his/her user settings by selecting the **Execute Object Validation on Access** checkbox in the **User Settings** editor. For more information about the implementation of this functionality in Alfabet, see the section *Defining Your User Settings in Alfabet* in the reference manual *Getting Started with Alfabet*.

Multiple post-conditions can be configured for a wizard step. Once all post-conditions have been created for a wizard step, the order that they are to be executed should be defined. For more information, see the section Determining the Sequence of Wizard Steps in the Wizard.

To define a post-condition for a selected wizard step:

1) Expand the custom wizard and click the wizard step to open its attribute window.
2) Right-click the **Post-Conditions** node and select **New Post-Condition**. You will see the new post-condition below the relevant node.

A post-condition can be created based on a copy of an existing condition. To do so, select the post-condition you want to copy and right click and select **Copy**. Click the wizard step or subordinate wizard step that you want to copy the post-condition to and select **Paste**. The values define for the post-condition's attributes are copied and can be modified, as needed.

3) Define its attributes in the attribute window as needed:

- **Technical Name**: Enter a name for the post-condition.
- **Technical Comment**: Enter any relevant information about the post-condition.
- **Presence**: Select `Desirable` if a violation of the post-condition triggers a warning message informing the user about the requested property value. In this case, the user must confirm a warning message by clicking either a **Yes** or **No** button. The user can continue to the next wizard step without entering the respective property. Select `Always` if a violation of the post-condition triggers an error message informing the user about the required property value. In this case, the user must confirm the error message by clicking the **OK** button. The user cannot continue to the next wizard step until the required value has been entered for the object class property.
- Define the object class property or set of object class properties targeted by the post-condition:
- If a standard or custom object class property is targeted by the post-condition: In the **Properties** attribute, define the standard or custom object class property that should be validated in the wizard step. Depending on the configuration of the **Presence** attribute, an informational or error message will be displayed. Contact Software AG Support for more information about the object class properties associated with individual views. You should leave this field blank if you have defined an Alfabet query or native SQL query for the post-condition.
- If a referenced property or set of properties is targeted by the post-condition: Define the **Query** or **Query as Text** attribute:
    - **Query**: Define an Alfabet query using the **Alfabet Query Builder** to return a set of object class properties.
    - **Query as Text**: Enter an Alfabet query or a native SQL query to return a set of object class properties. For more information about defining Alfabet queries or about the special conditions that apply to native SQL queries used in Alfabet configurations, see the chapter Defining Queries.

The following is an example of an Alfabet query defined for a post-condition that validates whether an ICT object is assigned to the selected application. The Alfabet query validates that the ICT object's lifecycle is within the timeframe of the application's lifecycle:

```
ALFABET_QUERY_500

FIND

    Application

    InnerJoin ICTObject ON (And
    Application.ICTObject = ICTObject.REFSTR
    Application.StartDate >= ICTObject.StartDate
    Application.EndDate <= ICTObject.EndDate)

WHERE

    Application.REFSTR =:BASE
```


The following is an example of an Alfabet query defined for a post-condition that validates whether the release statuses of the demands associated with a selected project are set to Approved:

```
ALFABET_QUERY_500
FIND
    Project
    InnerJoin Demand ON Demand.AssignedToProject =
    Project.REFSTR
    WHERE
        (AND Project.REFSTR CONTAINS:BASE
        Demand.Status <> 'Approved')
```

- **Check Result Type**: Select either `Positive` or `Negative` to define what constitutes a validated result.
- `Positive` means that the post-condition is fulfilled if the selected property has a value or the query has delivered a result (at least one row.)
- `Negative` means that the post-condition is fulfilled if the selected property has no value or the query doesn't deliver any results.

For example, to validate whether an object class property is set, an Alfabet query is specified that returns the value for the object class property. The post-condition is that the Alfabet query returns a result. In this case, the **Check Result Type** attribute must be set to `Positive`.

On the other hand, to validate that an object class property is not set to a specific value because that value is not allowed in the current environment, an Alfabet query must be specified that searches for the property value. The post-condition is only met if the Alfabet query returns no result. In this case, the **Check Result Type** attribute must be set to `Negative`.

- **Message**: Specify a warning message to display if the post-condition is not fulfilled. The text should provide the user with information about what is required to resolve the post-condition. If no warning message is configured, then the standard warning message preconfigured by Software AG will be displayed to the user if the post-condition is not fulfilled.
- **Apply to Page**: If the wizard step has the **Step Type** attribute set to `Editor` and the **Tab as Separate Step** attribute set to `True`, you must define which editor tab the post-conditions should apply to. To do so, select the relevant tab that should be validated by the post-condition.

Please note that if the **Tab as Separate Step** attribute is set to `False` on the wizard step, the **Apply to Page** attribute must be empty otherwise the post-conditions will NOT be executed.

4) To specify additional behavior for the execution of post-conditions on the level of the wizard, go to the custom wizard  and define the following attributes:

- **Execute Post-Conditions on Back Moves**: Select `True` if post-conditions must be validated for the current wizard step before navigating to a preceding step via the **Back** button or a selected step via the **Go to Step** field. Select `False` if the post-conditions for the current wizard step do not need to be validated in order to navigate to the target wizard step. The default value is `False`.

The **Back** button is automatically displayed in the wizard. The **Go to Step** field is optional and will only be displayed if the **Show Wizard Steps Combo-Box** attribute is set to `True` for the wizard. For more information about configuring the buttons in the wizard, see the section Creating a Wizard.

The **Go to Step** field is only available in the wizard when an existing object is being edited. The field is not available in the first step when a new object is being created. **Please note that data for intermediary steps will not be checked!** For example, if a user goes from Step 2 to Step 4, the post-conditions for Step 2 will be checked and must be fulfilled, but post-conditions for Step 3 are not triggered and must not be fulfilled in order to go to Step 4. a **Go to Step** field can be displayed at the bottom of the wizard in the Alfabet interface.

Please note that the post-condition is executed AFTER the database transaction is performed. This is of particular relevance for post-conditions associated with standard or custom editors because the first attempt will be to save the user's data which may fail, for example, due to mandatory or uniqueness conditions being violated. Once this save has been successful, the post-conditions will be executed.

If the attribute **Execute Post-Conditions on Back Moves** is set to `False`, the post-conditions will not be executed if the user clicks the **Back** button. However, the validation of editor-related constraints such as preconfigured mandatory properties or uniqueness constraints for a set of properties will continue to be performed if the user clicks the **Back** button.

- **Execute Post-Condition in Transaction of Wizard Step**: Wizard step actions performed in an editor may be rolled back in case of failing post-conditions. Set to `True` if the transaction shall be rolled back and the data shall not be saved to the Alfabet database if a configured post-condition for a wizard step of the type `Editor` fails. The wizard will return to the wizard step as it was before the transaction was initiated and the message specified for the failed post-condition will be displayed. Select `False` if the wizard step actions performed in an editor shall not be rolled back in case post-conditions fail for the wizard step.

5) Click the **Save** button to save your definitions.

Determining the Sequence of Pre-Conditions and Post-Conditions

If you define multiple pre-conditions or post-conditions, the conditions will be executed in the order that they are specified for the wizard step. Please keep the following in mind when sequencing the post-conditions:

- Once any preconfigured mandatory properties have been validated as well as mandatory custom object class properties, the post-conditions will be validated in the sequence specified. If multiple post-conditions are violated, the error message for the first violated condition will be displayed when the user attempts to click the **Next** button or close the wizard. Once the error has been corrected and the user attempts to advance to the next wizard step, the error message for the second violated condition will be displayed, etc. The user must proceed to correct all violations in this manner. Once all violations have been corrected, the user will be able to save the current wizard step and advance to the next wizard step or close the wizard without losing data:
- If the wizard is configured to include the **Go to Step** field displayed at the bottom of the wizard view, users will be able to traverse to any step in the wizard process. Any post-conditions configured for

the wizard step that the user is leaving will be checked and, if necessary, required data must be entered in order to conclude the current wizard step. Data for intermediary steps, however, will not be checked. For example, if a user goes from Wizard Step 2 to Wizard Step 4, the post-conditions for Wizard Step 2 will be checked and must be fulfilled, but post-conditions for Wizard Step 3 are not triggered and must not be fulfilled in order to go to Step 4.

Please note that the post-condition is executed AFTER the database transaction is performed. This is of particular relevance for post-conditions associated with standard or custom editors because the first attempt will be to save the user's data which may fail, for example, due to mandatory or uniqueness conditions being violated. Once this save has been successful, the post-conditions will be executed.

If the attribute **Execute Post-Conditions on Back Moves** is set to `False`, the post-conditions will not be executed if the user clicks the **Back** button. However, the validation of editor-related constraints such as preconfigured mandatory properties or uniqueness constraints for a set of properties will continue to be performed if the user clicks the **Back** button.

If the wizard step has post-conditions defined and the **Tab as Separate Step** attribute has been set to `True`, the post-conditions will be checked after the last tab of the standard or custom editor has been reached. In this case, you must ensure that the attribute **Execute Post-Conditions on Back Moves** for the wizard is set to `False`. Otherwise, a situation may occur in which the user cannot navigate forward or backward if a mandatory post-condition has been defined for an object class property that can only be edited on a tab other than the last tab of the editor. For more information about the definition of the attributes associated with post-condition, see the section Defining a Post-Condition for a Wizard Step.

To order the sequence of pre-conditions or post-conditions:

1) To define the sequence of the wizard step actions, do one of the following as needed:

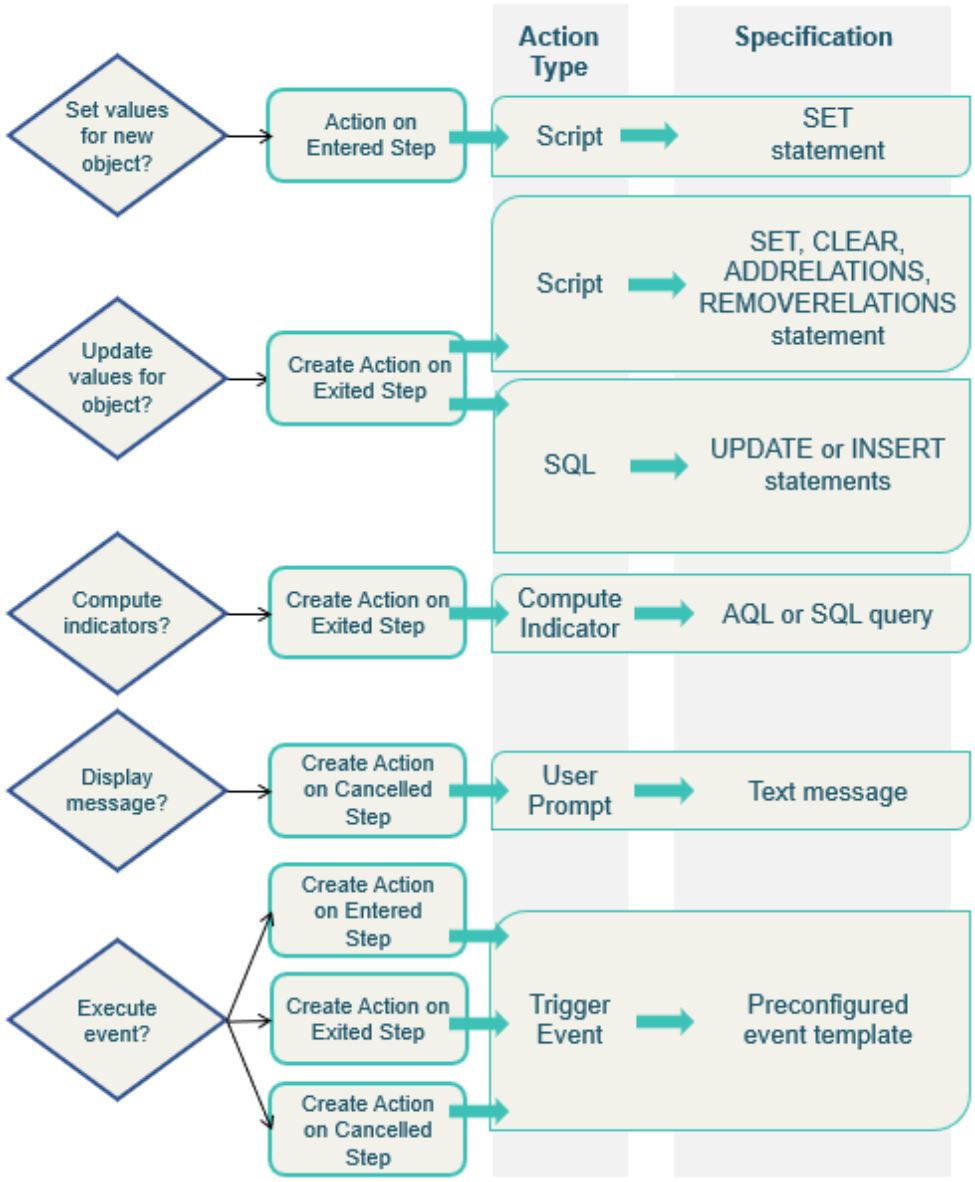
- Click the **Pre-Conditions** node below the relevant wizard step to open its attribute window. In the **Pre-Conditions** field, click the **Browse** button to open the editor and sequence the pre-conditions by clicking the **Up/Down** arrows.
- Click the **Post-Conditions** node below the relevant wizard step to open its attribute window. In the **Post-Conditions** field, click the **Browse** button to open the editor and sequence the post-conditions by clicking the **Up/Down** arrows.

2) In the toolbar, click the **Save** button to save your changes.

If you have completed the definition of the wizard, see Testing the Wizard in the Alfabet User Interface.

Configuring Wizard Step Actions for a Wizard Step

A wizard step action is a configured operation that is automatically executed when a wizard step is entered, exited, or cancelled. You can configure wizard step actions that update property values, create new objects, calculate indicators, or trigger events.



The following can be configured via wizard step actions:

- Automatically enter values for properties for a new object that is created in the context of a first wizard step. For example, you could configure that a status value and start date is automatically set when an object is created in the first wizard step. The command SET must be specified
- Automatically enter or clear properties for an object when a wizard step is exited. In this case, conditions can be specified that describe if the property update shall occur. For example, you could configure that a specified property is cleared when a wizard step is exited. For complex property updates, the SQL statements INSERT and UPDATE may be used.

- Automatically create a new object when a wizard step is exited. Upon creation of the object, you can also automatically enter values for the new object as well as specify references to the wizard in which the object was created.
- Automatically compute indicators for the current object when a wizard step is exited.
- Automatically trigger an event when a wizard step is entered, exited, or cancelled. Events allow for real-time automated updates to external systems as well as to data in Alfabet. Events allow, for example, a configured action to be triggered when a user closes the wizard window in the middle of the wizard process. A configured ADIF scheme will be triggered to process the data.

Wizard step actions are executed each time the wizard step is traversed. If this should not occur, a pre-condition should be configured for the wizard step.

There is no mechanism to verify the syntax or content of the statements. The following syntax is permissible for statements specified for wizard step actions:

- A value for a scalar property or object class property of type `Reference` may be targeted via the commands `SET` and `CLEAR`.
- A value for an object class property of type `ReferenceArray` may be targeted via the commands `ADDRELATIONS` and `REMOVERELATIONS`.
- The object class properties to be updated must be enclosed in parenthesis.
- Each statement must be terminated by a semi-colon.
- The following parameters may be used in the statement:
  - In the specification of the target property to be set:
  - `@BASE` (notation for the object targeted by a wizard step),
  - `@WIZARDBASE` (notation for the base object of the wizard)
  - `@NEW` (notation for a new object created via a `CREATE` statement. `@NEW` refers to the latest created object in the order of the specification.)
  - In the specification of the source property: `@BASE`, `@WIZARDBASE`, `@CURRENT_USER`, `@CURRENT_MANDATE`, `@CURRENT_PROFILE`, `@TODAY`.

The following information is available:

- Defining Property Values To Be Automatically Set for New Objects
- Defining Property Values To Be Automatically Updated When a Wizard Step Is Exited
- Defining a Wizard Step Action of the Action Type Script
- Defining a Wizard Step Action of the Type SQL
- Computing Indicators When a Wizard Step Is Exited
- Creating a New Object When a Wizard Step Is Exited
- Triggering an Event When a Wizard Step Is Exited
- Specifying Wizard Step Actions When a Wizard Step Is Cancelled

- Determining the Sequence of Wizard Step Actions

Defining Property Values To Be Automatically Set for New Objects

When a new object is created in the first wizard step, the object class property fields in the wizard can be pre-populated with default values. A value for a scalar property or object class property of type `Reference` may be specified via the command `SET`. This is only possible for the first wizard step. If multiple statements are required, a separate wizard step action must be created for each statement.

The wizard step action **Action on Entered Step** for which the **Action Type** attribute is set to `Script` may only be specified for the first wizard step. For all other wizard steps, the update of properties must be specified via the wizard step action **Action on Exited Step**. For more information see, Defining Property Values To Be Automatically Updated When a Wizard Step Is Exited.

To define an object class property to be set for a new object created in the context of the first wizard step:

1) Expand the custom wizard and expand the first wizard step .
2) Right-click the **Action on Entered Step** node and select **New Action On Enter Step**.
3) Define the following attributes in the attribute window:

- **Technical Name**. Enter a name for the wizard step action. This name is not displayed in the Alfabet interface.
- **Action Type**: Select `Script`.
- **Technical Comment**: Enter any relevant information about the wizard step action.
- **Statements**: Click the **Browse** button to open the editor. In order to provide an initial value for an object class property when the object is created in the first wizard step, Specify a `SET` statement for the wizard step action. Please note the following:

There is no mechanism to verify the syntax or content of the statements.

- The statement must refer to the class model. Property names must be spelled in accordance with the class model.
- The following syntax is required for the `SET` statement:
  - Specify a value for a scalar property or object class property of type `Reference` that may be targeted.
  - The object class properties to be updated must be enclosed in parenthesis.
  - A statement must be terminated by a semi-colon.
  - The following parameters may be used in the statement:
  - In the specification of the target property to be set: `@BASE` (notation for the object targeted by a wizard step), `@WIZARDBASE`(notation for the base object of the wizard)

- In the specification of the source property: `@BASE`, `@WIZARDBASE`, `@CURRENT_USER`, `@CURRENT_MANDATE`, `@CURRENT_PROFILE`, `@TODAY`.

The following displays the correct syntax to use for statements:

```
SET(PROPERTY_NAME,"Value");
```

For example, to set the `Application Type` property to `Client Server` for new applications:

```
SET(@BASE.SC_ApplicationType, "Client Server");
```

4) Click the **Save**  button to save your definitions.

Defining Property Values To Be Automatically Updated When a Wizard Step Is Exited

You can update property values when a wizard step is exited. To do so, you can either specify an **Action on Exited Step** of type `Script` which allows you to use the commands `SET`, `CLEAR`, `ADDRELATIONS`, and `REMOVERELATIONS`, or of type `SQL` which allows you to specify more complex statements via the commands `UPDATE` and `INSERT`.

If an **Action on Exited Step** wizard step action is a configured for a wizard step of **Type** = `Editor` and the **Tab as Separate Step** attribute for the wizard is set to `True`, the wizard step action will only be executed on the last tab of the editor (which is the last sub-step of the wizard step). Please note that if the user moves backward through the sub-steps of the wizard step via the **Previous** button, for example, the wizard step action will be executed for each sub-step of the wizard step.

The following information is available:

- Defining a Wizard Step Action of the Action Type Script
- Defining a Wizard Step Action of the Type SQL

Defining a Wizard Step Action of the Action Type Script

You can define one or more wizard step actions of the type **Action on Exited Step**. Wizard step actions for which the **Action Type** attribute is set to `Script` can be created and statements using the commands `SET`, `CLEAR`, `ADDRELATIONS`, and `REMOVERELATIONS` can be defined. If multiple statements are required, a separate wizard step action must be created for each statement.

Optionally, you can also define a condition to execute the wizard step action depending on the availability of specific data returned by a query.

To define a wizard step action of the type `Script`:

1) Expand the custom wizard  and expand the first wizard step  .
2) Right-click the **Action on Exited Step** node and select **New Action On Exited Step**.
3) Define the following attributes in the attribute window:

- **Technical Name**. Enter a name for the wizard step action. This name is not displayed in the Alfabet interface.
- **Action Type**: Select `Script`.
- **Technical Comment**: Enter any relevant information about the wizard step action.
- **Statements**: Click the **Browse** button to open the editor. Enter one statement for the wizard step action. Please note the following:

  There is no mechanism to verify the syntax or content of the statements.

- The statement must refer to the class model. Property names must be spelled in accordance with the class model.
- You may use the commands `SET`, `CLEAR`, `ADDRELATIONS`, and `REMOVERELATIONS`. The following syntax is required for statements:
  - `SET` and `CLEAR`: A value for a scalar property or object class property of type `Reference` may be targeted.
  - `ADDRELATIONS` and `REMOVERELATIONS`: A value for an object class property of type `ReferenceArray` may be targeted.
  - The object class properties to be updated must be enclosed in parenthesis.
  - A statement must be terminated by a semi-colon.
  - The following parameters may be used in the statement:

In the specification of the target property to be set: `@BASE`, `@WIZARDBASE`

In the specification of the source property: `@BASE`, `@WIZARDBASE`, `@CURRENT_USER`, `@CURRENT_MANDATE`, `@CURRENT_PROFILE`, `@TODAY`.

The following examples display the correct syntax to use for statements:

```
CLEAR(PROPERTY_NAME);
ADDRELATIONS(Reference.ToProperty,FromReference);
REMOVERELATIONS(Reference.ToProperty,FromReference);
```

If a newly created application that is the base object of the wizard should be added to an application group that is the base object of the current wizard step, the statement would be:

```
ADDRELATIONS(@BASE.Applications,@WIZARDBASE);
```

4) Optionally, you can define a condition to execute the wizard step action depending on the availability of specific data returned by a query. For example, you could specify that the wizard step action may only be executed if a property of the base object of the wizard is set to a specific value. Edit the following attributes in the attribute window to define the condition:

- **Check Query** / **Check Query as Text**: Define a query that will return a result only if a condition is met. The query can be defined either as an Alfabet query in the **Check Query** attribute or as a native SQL query in the **Check Query as Text** attribute.

- **Result Type**: Select `Positive` if the wizard step action shall only be executed if the query returns a result. Select `Negative` if the wizard step action shall only be executed if the query does not return a result.

  For example, a wizard step action to update an application's indicators when a wizard step is exited shall be triggered if the application's release status is set to the value "Approved". The following query will only return a result if the application that is the base application of the wizard step has the correct release status:

  ```
  SELECT REFSTR
  FROM APPLICATION
  WHERE APPLICATION.STATUS = 'Approved'
  AND APPLICATION.REFSTR = @WIZARDBASE
  ```

  The **Result Type** is set to `Positive` so that the wizard step action is only executed if the query returns a result.

5) Click the **Save**  button to save your definitions.

Defining a Wizard Step Action of the Type SQL

You can define one or more wizard step actions of the type **Action on Exited Step**. Wizard step actions for which the **Action Type** attribute is set to `SQL` can be created and the commands `UPDATE` and `INSERT` can be defined.

Please note that the `INSERT` statement is only allowed to insert new relation entries in the `RELATIONS` table. If you attempt to insert new relation entries in class tables, an error will occur because the `GUID` and `REFSTR` are not automatically created and result in a `NULL` value.

Optionally, you can also define a condition to execute the wizard step action depending on the availability of specific data returned by a query.

To define a wizard step action of the type `SQL`:

1) Expand the custom wizard  and expand the wizard step  .
2) Right-click the **Action on Exited Step** node and select **New Action On Exit Step**.
3) Define the following attributes in the attribute window:

- **Technical Name**. Enter a name for the wizard step action. This name is not displayed in the Alfabet interface.
- **Action Type**: Select `SQL`.
- **Technical Comment**: Enter any relevant information about the wizard step action.
- **Statements**: Click the **Browse**  button to open the editor. Enter one statement for the wizard step action. If multiple statements are required, a separate wizard step action must be created for each statement. You may use the commands `UPDATE` or `INSERT`. The parameters `@BASE` (notation for the object targeted by a wizard step) and `@WIZARDBASE` (notation for the base object of the wizard) can be used.

The following examples display the correct syntax to use for statements:

```
UPDATE APPLICATION
SET APPLICATION.APPLICATIONTYPE='Mainframe'
WHERE APPLICATION.REFSTR=@BASE
```

4) Optionally, you can define a condition to execute the wizard step action depending on the availability of specific data returned by a query. For example, you could specify that the wizard step action may only be executed if a property of the base object of the wizard is set to a specific value. Edit the following attributes in the attribute window to define the condition:

- **Check Query** / **Check Query as Text**: Define a query that will return a result only if a condition is met. The query can be defined either as an Alfabet query in the **Check Query** attribute or as a native SQL query in the **Check Query as Text** attribute.
- **Result Type**: Select `Positive` if the wizard step action shall only be executed if the query returns a result. Select `Negative` if the wizard step action shall only be executed if the query does not return a result.

For example, a wizard step action to update an application's indicators when a wizard step is exited shall be triggered if the application's release status is set to the value "Approved". The following query will only return a result if the application that is the base application of the wizard step has the correct release status:

```
SELECT REFSTR
FROM APPLICATION
WHERE APPLICATION.STATUS = 'Approved'
AND APPLICATION.REFSTR = @WIZARDBASE
```

The **Result Type** is set to `Positive` so that the wizard step action is only executed if the query returns a result.

5) Click the **Save** button to save your definitions.

Computing Indicators When a Wizard Step Is Exited

You can configured to automatically compute all indicators for the current object targeted by the wizard step. The indicators will be recalculated when the wizard step is exited. Optionally, you can also define a condition to execute the wizard step action depending on the availability of specific data returned by a query.

For more information about the configuration of evaluation types and indicator types, see the section *Configuring Evaluations, Prioritization Schemes, and Portfolios* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.

To define a wizard step action of the type `ComputeIndicators`:

1) Expand the custom wizard and expand the wizard step .
2) Right-click the **Action on Exited Step** node and select **New Action on Exited Step**.

3) Define the following attributes in the attribute window:

- **Technical Name**. Enter a name for the wizard step action. This name is not displayed in the Alfabet interface.
- **Action Type**: Select `ComputeIndicators`.
- **Technical Comment**: Enter any relevant information about the wizard step action.

4) Optionally, you can define a condition to execute the wizard step action depending on the availability of specific data returned by a query. For example, you could specify that the wizard step action may only be executed if a property of the base object of the wizard is set to a specific value. Edit the following attributes in the attribute window to define the condition:

- **Check Query** / **Check Query as Text**: Define a query that will return a result only if a condition is met. The query can be defined either as an Alfabet query in the **Check Query** attribute or as a native SQL query in the **Check Query as Text** attribute.
- **Result Type**: Select `Positive` if the wizard step action shall only be executed if the query returns a result. Select `Negative` if the wizard step action shall only be executed if the query does not return a result.

For example, a wizard step action to update an application's indicators when a wizard step is exited shall be triggered if the application's release status is set to the value "Approved". The following query will only return a result if the application that is the base application of the wizard step has the correct release status:

```
SELECT REFSTR
FROM APPLICATION
WHERE APPLICATION.STATUS = 'Approved'
AND APPLICATION.REFSTR = @WIZARDBASE
```

The **Result Type** is set to `Positive` so that the wizard step action is only executed if the query returns a result.

5) Click the **Save**  button to save your definitions.

Creating a New Object When a Wizard Step Is Exited

A new object can be created in the context of a wizard step via the command `CREATE`. The command `ADDRELATIONS` may also be specified in order to specify the base object and add values to the new object. This is of particular relevance when a questionnaire or survey object shall be created in the context of a wizard. The command `SET` may be specified to set property values for the new object.

Optionally, you can also define a condition to execute the wizard step action depending on the availability of specific data returned by a query.

1) Expand the custom wizard  and expand the wizard step  .
2) Right-click the **Action on Exited Step** node and select **New Action On Exited Step**.
3) Define the following attributes in the attribute window:

- **Technical Name**. Enter a name for the wizard step action. This name is not displayed in the Alfabet interface.
- **Action Type**: Select `Script`.
- **Technical Comment**: Enter any relevant information about the wizard step action.
- **Statements**: Click the **Browse** button to open the editor. In order to create a new object, a `CREATE` command should be defined. Enter one statement for the wizard step action. If multiple statements are required, a separate wizard step action must be created for each statement. Please note that wizard step actions of the type `Script` must always refer to the class model and hence property names must be spelled in accordance with the class model.

The following displays the correct syntax to use for statements:

```
CREATE(OBJECTCLASS);
ADDRELATIONS (@NEW.OBJECTCLASSPROPERTY,@WIZARDBASE);
```

For example, to create a new object for the survey class `SC_APP_CSAT` whereby the property `BASEOBJECT` is set to the wizard object:

```
CREATE(SC_APP_CSAT);
ADDRELATIONS(@NEW.BASEOBJECT,@WIZARDBASE);
```

4) Optionally, you can define a condition to execute the wizard step action depending on the availability of specific data returned by a query. For example, you could specify that the wizard step action may only be executed if a property of the base object of the wizard is set to a specific value. Edit the following attributes in the attribute window to define the condition:

- **Check Query** / **Check Query as Text**: Define a query that will return a result only if a condition is met. The query can be defined either as an Alfabet query in the **Check Query** attribute or as a native SQL query in the **Check Query as Text** attribute.
- **Result Type**: Select `Positive` if the wizard step action shall only be executed if the query returns a result. Select `Negative` if the wizard step action shall only be executed if the query does not return a result.

For example, a wizard step action to update an application's indicators when a wizard step is exited shall be triggered if the application's release status is set to the value "Approved". The following query will only return a result if the application that is the base application of the wizard step has the correct release status:

```
SELECT REFSTR
FROM APPLICATION
WHERE APPLICATION.STATUS = 'Approved'
AND APPLICATION.REFSTR = @WIZARDBASE
```

The **Result Type** is set to `Positive` so that the wizard step action is only executed if the query returns a result.

5) Click the **Save** button to save your definitions.

Triggering an Event When a Wizard Step Is Exited

You can configure that an event is triggered when a wizard step is entered, exited, or cancelled. Events allow for real-time automated updates to external systems as well as to data in Alfabet. Events allow, for example, a configured action to be triggered when a user closes the wizard window in the middle of the wizard process. A configured ADIF scheme will be triggered to process the data.

Additional configuration is required in order to trigger events. For more information about the configuration required to implement the event capability, see the chapter Configuring Events.

Optionally, you can also define a condition to execute the wizard step action depending on the availability of specific data returned by a query.

To configure the wizard step to trigger an event:

1) Expand the custom wizard  and expand the wizard step  .
2) Right-click the node of the relevant wizard step action that shall trigger the event and select either **Action on Entered Step**, **Action on Exited Step**, or **Action on Cancelled Step**.
3) Define the following attributes in the attribute window:

- **Technical Name**. Enter a name for the wizard step action. This name is not displayed in the Alfabet interface.
- **Action Type**: Select `TriggerEvent`.
- **Technical Comment**: Enter any relevant information about the wizard step action.
- **Parameter**: Select the relevant event template from the drop-down list.

4) Optionally, you can define a condition to execute the wizard step action depending on the availability of specific data returned by a query. For example, you could specify that the wizard step action may only be executed if a property of the base object of the wizard is set to a specific value. Edit the following attributes in the attribute window to define the condition:

- **Check Query** / **Check Query as Text**: Define a query that will return a result only if a condition is met. The query can be defined either as an Alfabet query in the **Check Query** attribute or as a native SQL query in the **Check Query as Text** attribute.
- **Result Type**: Select `Positive` if the wizard step action shall only be executed if the query returns a result. Select `Negative` if the wizard step action shall only be executed if the query does not return a result.

For example, a wizard step action to update an application's indicators when a wizard step is exited shall be triggered if the application's release status is set to the value "Approved". The following query will only return a result if the application that is the base application of the wizard step has the correct release status:

```
SELECT REFSTR
FROM APPLICATION
WHERE APPLICATION.STATUS = 'Approved'
AND APPLICATION.REFSTR = @WIZARDBASE
```


The **Result Type** is set to `Positive` so that the wizard step action is only executed if the query returns a result.

5) Click the **Save** button to save your definitions.

Specifying Wizard Step Actions When a Wizard Step Is Cancelled

You can configure the action that should occur for a wizard step if a user closes a wizard by clicking the **Close** (X) button at the top of the wizard. You can optionally specify a message in order to intercept the cancel action and allow the user to confirm or negate the impending cancel action.

To configure an event for a cancelled wizard step:

1) Expand the custom wizard and expand the wizard step .
2) Right-click the **Action on Cancelled Step** node and select **New Action on Cancelled Step**.
3) Define the following attributes in the attribute window:

- **Technical Name**. Enter a name for the wizard step action. This name is not displayed in the Alfabet interface.
- **Action Type**: Select `TriggerEvent`.
- **Technical Comment**: Enter any relevant information about the wizard step action.
- **Parameter**: Select the relevant event template from the drop-down list.

4) A message can be configured for wizard step actions of type **Action on Cancelled Step** in order to intercept the cancel action and allow the user to confirm or negate the impending cancel action. To configure a message, create another wizard step action by selecting **New Action on Cancelled Step**. Define the following attributes:

- **Technical Name**. Enter a name for the wizard step action. This name is not displayed in the Alfabet interface.
- **Action Type**: Select `UsePrompt`.
- **Technical Comment**: Enter any relevant information about the wizard step action.
- **Prompt**: Enter the message to display if the user attempts to cancel the wizard step.
- **Cancel Step Message Prompt Buttons**: Select the relevant buttons to show in the message box.

5) Click the **Save** button to save your definitions.

Determining the Sequence of Wizard Step Actions

If you define multiple wizard step actions, the actions will be executed in the order that they are specified for the wizard step.

To order the sequence of actions:

1) To define the sequence of the wizard step actions, do one of the following as needed:

- Click the **Action on Entered Step** node below the relevant wizard step to open its attribute window. In the **Actions on Entered Step** field, click the **Browse** button to open the editor and sequence the actions by clicking the **Up/Down** arrows.
- Click the **Action on Exited Step** node below the relevant wizard step to open its attribute window. In the **Actions on Exited Step** field, click the **Browse** button to open the editor and sequence the actions by clicking the **Up/Down** arrows.
- Click the **Action on Cancelled Step** node below the relevant wizard step to open its attribute window. In the **Actions on Cancelled Step** field, click the **Browse** button to open the editor and sequence the actions by clicking the **Up/Down** arrows.

2) In the toolbar, click the **Save** button to save your changes.

If you have completed the definition of the wizard, see Testing the Wizard in the Alfabet User Interface.

Configuring a Different Target Object for a Wizard Step

When a wizard is created, the wizard designer will specify the object class for which the wizard is configured. The object class is then displayed in the **Class Name** attribute of the wizard. Typically, the wizard will process a single object in the object class that the wizard was created for.

However, it is possible to specify that another object in a different object class must also be processed in the context of capturing the base object of the wizard. To do this, a query must be defined on the wizard step in order to find the related target object.

For example, a wizard has been configured in order to capture application data. The enterprise requires that the ICT object that the application is assigned also be defined at the time that the application data is captured. In this case, you can configure a query to find the object in the object class `ICTObject` that should be defined in the context of the wizard

The query must be entered in the **Base Object Query** field for each relevant wizard step that targets the object found by the query. Therefore, if the object found by the query shall be the target object for multiple wizard steps, the query must be entered in the **Base Object Query** field for each of those wizard steps. If the query is not entered in the **Base Object Query** field of a wizard step, the wizard will return to the original object defined in the wizard.

You must ensure that the object being searched for by the query exists and can be found by the query. If an object cannot be found by the query, an error will occur in the wizard. For example, if an ICT object that the application is assigned to must be found by the query, then you could make the ICT object property a mandatory property that is controlled by a post-condition. Or, you could define a pre-condition for the next wizard step that skips the step if the reference to ICT object does not exist.

To configure a wizard step to find a different target object:

1) Expand the custom wizard and expand the wizard step .

2) In the **Base Object Query** field, enter an Alfabet query or native SQL query to find the relevant object to be processed in the wizard step.

Keep the following in mind when configuring the base object query for a wizard step:

- The query to find the base object for the wizard step is evaluated before the evaluation of other queries associated with the wizard step such as the queries for pre-conditions or wizard step actions. Therefore, when using the parameter `@BASE` in queries related to a wizard step, `@BASE` returns the `REFSTR` of the base object of the wizard step. To refer to the initial base object of the wizard, use the parameter `@WIZARDBASE`.
- The result of the query entered in the **Base Object Query** attribute should have the `REFSTR` of the object which should be used as base object in the first column. Therefore, the `REFSTR` of the object must be defined as the first `SELECT` property in native SQL queries. For Alfabet queries, `SHOW` properties do not need to be defined. The `REFSTR` of the object found by the query is selected automatically as the query result.
- The query should be defined to find only one object. If the query defined in the **Base Object Query** attribute of a wizard step returns several results (rows), then the wizard step will be executed for the first result. If no result is returned by the query, then an error will be displayed stating that no instance has been found.
- If you are implementing an SQL query, the query must be of the type `SELECT`.

The following example illustrates an Alfabet query that searches for the ICT object that the application is assigned to.

```
ALFABET_QUERY_500
FIND
ICTObject
    InnerJoin Application
        ON Application.ICTObject = ICTObject.REFSTR
WHERE
Application.REFSTR CONTAINS:BASE
```

3) Once you have defined the base object query for the selected wizard step, define the **View** attribute as well as the rest of the wizard step attributes as you would for any wizard step as described in the section Creating a Wizard Step.

4) If the object searched for in the query is to be the target object of the next wizard step, then you must also enter the query in the **Base Object Query** attribute for that wizard step. Otherwise, the wizard will return to the original object defined in the wizard.

5) In the toolbar, click the **Save** button to save your changes.

Determining the Sequence of Wizard Steps in the Wizard

Typically, a wizard user will move through the wizard process in a linear manner. In this case, the user starts with the first wizard step and once the required data has been entered, the user can click the **Next** button and advance to the next wizard step.

The order of the wizard steps is determined in the **Steps** attribute of the wizard. Here you must list the wizard steps from top to bottom in the sequence in which they should be processed. Please note the following:

- If you have defined one or more branches of subordinate wizard steps for a particular wizard step, these subordinate wizard steps do not need to be explicitly sequenced since they are created by definition as a hierarchy. For more information about creating a branch of subordinate wizard steps, see the section Defining a Subordinate Wizard Step or a Branch of Subordinate Wizard Steps.
- If the **Tab as Separate Step** attribute is set to `True` for wizard step of the type `Editor`, you cannot change the sequence of the tabbed pages in the editor. In this case, the tabbed pages in an editor are treated as a single unit. The workflow steps based on the editor's tabbed pages will be displayed in the order that they appear in the custom editor. A more granular sequence is not possible. For more information about defining tabbed pages as separate wizard steps, see the section Configuring a Wizard Step to Display a Standard or Custom Editor.

  Alternatively, you could assign the editor to two wizard steps, hide the tabbed pages that are not relevant for each wizard step, and position the two wizard steps as needed in the sequence.

- You can specify the wizard so that users can circumvent the linear process prescribed by the specified sequence. In this case, a **Go to Step** field can be displayed at the bottom of the wizard in the Alfabet interface. This is specified by setting the **Show Wizard Steps Combo-Box** attribute to `True` for wizard. If the **Execute Post- Conditions on Back Moves** attribute is also set to `True` for the wizard, any post-conditions configured for the wizard view that the user is leaving will be checked and, if necessary, the user will be required to provide the missing data in order to conclude the current wizard step. For more information about configuring post-conditions, see the section Defining a Post-Condition for a Wizard Step.

  The completeness of data for intermediary wizard views will not be checked. For example, if a user goes from Wizard Step 2 to Wizard Step 4, the data entry prompts for Wizard Step 2 will be checked and must be fulfilled, but data entry prompts for Step 3 are not triggered and must not be fulfilled in order to go to Wizard Step 4.

A pre-condition may NOT be configured for the first wizard step NOR for the final wizard step or subordinate wizard step. If you do so and the pre-condition is not fulfilled, an error will occur. In this case, for example, the wizard could stagnate and not advance to the next wizard step. If a pre-condition is required for the last logical step or subordinate wizard step in the wizard, it is recommended that you add another final concluding wizard step with no pre-condition and only static text defined (for example, with a message such as "You have completed the wizard").

To define the sequence of the wizard steps:

1) Expand the custom wizard  .
2) Define the following attributes as needed:

- **Steps**: Click the **Browse**  button to open the **Sort Entries** editor to order the wizard steps. You will see all wizard views assigned to the wizard. Subordinate wizard steps defined for a particular wizard

step will not be displayed in the **Sort Entries** editor. To define the sequence of the wizard steps, select a wizard step in the list and click the **Up/Down** button in the upper right corner of the editor to move the view up or down in the list. Repeat this for all wizard steps until they are listed in the correct order. Click **OK** to save the sequence definition.

- **Show Wizard Steps Combo-Box**: Select `True` to display the **Go to Step** field at the bottom of the wizard in order to allow wizard users to go forward or backward to any step in a wizard.

3) If one or more branches of subordinate wizard steps have been specified for a particular wizard step, specify the type of numbering to display in the wizard header for the subordinate wizard steps created for a wizard step. In the **Type** attribute, specify one of the following:

- Select `Tree` to number all subordinate wizard steps so that subordinate wizard step 1, subordinate wizard step 2, and subordinate wizard step 3 that are subordinate to wizard step 1 will be numbered step 1.1, 1.2, and 1.3 in the wizard header. If the **Show Wizard Steps Combo-Box** attribute is set to `True` for the wizard, then the subordinate wizard steps will be displayed in the **Go to Step** field.
- Select `Straight` to number all subordinate wizard steps so that all subordinate wizard step that are subordinate to wizard step 1 will all have the same number as their ascendant wizard step. In this case, subordinate wizard step 1, subordinate wizard step 2, and subordinate wizard step 3 would all be numbered 1 of 7 in the wizard header. Subordinate wizard steps will not be displayed in the **Go to Step** field if the **Type** field is set to `Straight`.

4) In the toolbar, click the **Save** button to save your changes.

Hiding Object Class Properties and Functionalities in the Wizard for a Specific User Profile

The custom wizard will typically have wizard steps in which editors, page views, and configured reports are embedded. Each wizard that you create can be further configured to hide specific data (properties) from users working with the custom wizard. The content of the **Customization Editor** will differ, depending on whether you are configuring an editor, page view, or configured report in the wizard step. If secondary views (data quality widgets) are configured for the wizard, these will also be displayed when you access the Alfabet user interface via the **Configure Wizard** functionality.

Please note the following:

- You can hide specific fields (object class properties) in embedded editors. Additionally, you can specify custom selectors to replace standard selectors for fields in which a reference to an object must be defined.
- You can hide functionalities (for example, create, copy, edit, navigate, etc.) available in the toolbar of page views and configured reports. However, this configuration cannot be done in the context of a wizard, but rather in the context of the user profile configuration. The customization of a page view or configured report is then applied to ALL instances of the page view or report in the configured user profile. In this case, a page view or report cannot be differently configured for specific object views. For more information, see the section Hiding Functionalities in a Page View or Configured Report in the chapter Configuring User Profiles for the User Community.
- You cannot configure the columns in page views and configured reports in the context of a wizard.

The configuration of a custom wizard will apply to all instances of the custom wizard in Alfabet. For example, if you hide object class properties or functionalities in a custom wizard, the object class properties or functionalities will be hidden for all instances of the custom wizard. In other words, a selected wizard cannot be differently configured in the context of different object class stereotypes. The wizard configuration will apply to each object class stereotype that the custom wizard is assigned to. If different configurations are required for a custom wizard, it is recommended that you create a new wizard via the copy functionality and modify the new wizard as needed. Once the wizard is configured, it can be selected in the **Wizard** attribute for the class setting that you create for the object class stereotype.

Please note that if changes are made to a wizard after the **Configure Wizard** functionality has been closed, the settings in the **Configure Wizard** functionality will be lost. Therefore, the following procedure is recommended:

1) Design the wizard
2) Save the design
3) Refine the visibility of tabs, properties, custom selectors in the embedded editors and hide functionalities in embedded page views and configured reports via the **Configure Wizard** functionality
4) Close and reopen the database.

The following information is available:

- Hiding Properties in an Editor Embedded in the Wizard
- Hiding Functionalities in Page Views and Configured Reports Embedded in the Wizard
- Resetting the Configuration of Visibility to the Default Settings

Hiding Properties in an Editor Embedded in the Wizard

You can hide standard or custom object class properties in the editors embedded in a wizard. Please note the following about hiding properties in wizards:

- Any properties that you hide in an editor cannot be defined or edited by users accessing Alfabet with the associated user profile.
- A red star in an editor indicates that the object class property is a mandatory property. It is possible to exclude a mandatory property in an editor or wizard as long as the mandatory property is not part of a class key definition. A class key definition specifies one or a combination of object class properties that must have unique values when the object is created. For more information about the configuration of class keys, see the section Configuring Class Keys for Object Classes in the chapter Configuring the Class Model.
- In conjunction with the task of hiding properties in editors, you can also specify a custom selector that is to be used in place of the standard object selector in editor fields that require a reference to an object to be defined. For more information about the configuration of a custom selector, see the section Configuring a Custom Selector for Search Functionalities.
- Properties that are hidden in an editor embedded in a wizard are excluded in that wizard for all user profiles that the wizard is assigned to.

The configuration of a custom wizard will apply to all instances of the custom wizard in Alfabet. For example, if you hide object class properties or functionalities in a custom wizard, the object class properties or functionalities will be hidden for all instances of the custom wizard. In other words, a selected wizard cannot be differently configured in the context of different object class stereotypes. The wizard configuration will apply to each object class stereotype that the custom wizard is assigned to. If different configurations are required for a custom wizard, it is recommended that you create a new wizard via the copy functionality and modify the new wizard as needed. Once the wizard is configured, it can be selected in the **Wizard** attribute for the class setting that you create for the object class stereotype.

- If the object class properties should not be displayed at all in the Alfabet interface, then you must ensure that they are completely excluded from the view scheme and therefore not displayed in the **Attributes** section of a custom object view, page views, and configured reports.

1) Go to the **Presentations** tab, right-click the custom wizard  and select **Configure Wizard**. The wizard is displayed in the browser.

2) If the wizard step is of the type `Editor`, click the **Configure View**  button in the upper right corner of the first wizard step. The **Customization Editor** opens and displays the relevant interface elements for the view you are working with.

3) In the **Choose One Settings Mode** menu, select **Explicit Settings from Configuration Object**. In the dataset that is displayed, you can configure the visibility of tabs and properties and specify the custom selector in the editor that is embedded in a wizard. The **Customization Editor** displays the following columns for an editor:

- **Name**: Displays the technical name of a tab or object class property in a tab.
- **Caption**: Displays the caption defined for the tab or object class property in a tab.
- **Default Selector**: Displays the name of the default selector used to find objects if an object selector is required to define the object class property. This field will only by filled in if an object selector is actually implemented.
- **Excluded**: Click in the cell to set an X to specify that the respective element shall be hidden in the wizard. Click the X to clear the exclusion setting and make the element visible in the editor.
- **Custom Selector**: Select a custom selector to replace the default selector displayed in the **Default Selector** column. A drop-down menu will display all custom selectors configured for your solution. You must ensure that you select a custom selector relevant to the context for which it will be used. For more information about the configuration of a custom selector, see the section Configuring a Custom Selector for Search Functionalities.

4) Click the **OK** button to save the changes in the **Customization Editor** or click **Cancel** to exit without saving the changes.

5) To view the changes, you must close the wizard and return to Alfabet Expand and click the **Save**  button. When you reopen the wizard, the excluded tabs and properties will no longer be displayed in the editor.

Hiding Functionalities in Page Views and Configured Reports Embedded in the Wizard

In a page view or configured reports that is embedded in a wizard, you can configure the visibility of menus, menu options and action buttons displayed in the toolbar, thus hiding various functionalities in the wizard. Please note that you cannot configure the columns in page views and configured reports in the context of a wizard.

1) Go to the **Presentations** tab, right-click the custom wizard and select **Configure Wizard**. The wizard is displayed in the browser.

2) Click the **Configure View** button in the upper-right corner of the first wizard step. The **Customization Editor** opens and displays the relevant interface elements for the view you are working with.

3) If the wizard step is of the type `View` or `Report` to click the **Configure View** button in the upper right corner of the first wizard step. The **Customization Editor** opens and displays the relevant interface elements for the view you are working with.

4) To specify the visibility of functionalities, select `Toolbar` in the **Select Configuration Object** field.

5) In the **Choose One Settings Mode** menu in the toolbar of the dataset, select **Explicit Settings from Selected Configuration Object** to display a dataset that allows visibility to be explicitly specified for the selected object view.

6) In the **Excluded** column, click in the cell to set an X to specify that the respective menu or action button is hidden in the view.

   If a button includes a drop-down menu providing one or multiple options, you can hide one or more of these functionalities. Please note that if you hide all functionalities available for a button, you must explicitly exclude the parent button in order to ensure that it is hidden from the toolbar.

7) Click the **OK** button to save the changes in the **Customization Editor** or click **Cancel** to exit without saving the changes.

8) To view the changes, you must close the wizard and return to Alfabet Expand and click the **Save** button. When you reopen the wizard, the excluded functionalities will no longer be displayed in the relevant page view or configured report.

Resetting the Configuration of Visibility to the Default Settings

You can reset the configuration to its default setting. You must save your changes by clicking the **Save** button before you can further configure the wizard.

1) Go to the **Presentations** tab, right-click the custom wizard that you want to reset and select **Reset Wizard Configuration**.

2) You must save your changes by clicking the **Save** button before you can further configure the wizard.

Adding Data Quality Widgets to Wizards or Wizard Steps

The completeness of data in a wizard can be displayed in data quality widgets that provide information to the user about the status of data completion in the wizard. Secondary views based on configured data quality widget reports or similar complementary information as represented in Gantt charts or business charts can be assigned to a wizard as a whole or individual wizard steps to provide information to the user such as the completeness of the underlying data of an object and may even provide a link to a view that allows incomplete data to be updated. The secondary views will be attached to the right-hand edge of the wizard view and will stay attached to the wizard if it is resized or repositioned.

A typical use case for quality widgets is a widget report that opens to provide information to the user about the quality of the data in a wizard step. For example, the user could be provided with information about the completeness of the properties or indicators specified for the object in the wizard.

The following additional configuration is required for secondary views:

- The configured reports that are implemented as quality widgets must first be configured before they can be assigned to an object cockpit, wizard, or wizard step. For more information about the configuration of the configured reports, see the section Integration of Quality Widgets in Configured Reports, Object Cockpits and Wizards.
- The visualization of secondary window skins is configured via the **Quality Widget** attributes for a GUI scheme. For more information, see the section Configuring GUI Scheme Definitions for the Alfabet Interface as well as the section *Attributes Displayed in the Grid Section: Secondary Window Skins* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

To assign a quality widget to an existing object cockpit:

1) Right-click the wizard or wizard step and select **Add Secondary View**. A secondary view is added to the explorer and the attributes of the secondary view are displayed in the attribute window.
2) Define the following attributes of the secondary view:

- **Name**: Enter a unique name for the secondary view. The name is a technical name and shall not contain special characters or whitespaces. The technical name is not displayed in the user interface.
- **Caption**: Enter a caption for the secondary view that shall be displayed in the title bar of the secondary view in the Alfabet user interface and in the tooltip displayed for the icon of the secondary view in the slide-in toolbar. Please note that the caption should be short. It will not be displayed in the title bar if it is too long.
- **Role**: Select `QualityWidget`.
- **View Type**: Select `Report`.

- **View Name**: Select the configured report that shall be displayed as secondary view. The drop-down list displays the names of all available configured reports of the allowed report types widget report, Gantt chart report, and business chart report.
- **Show Automatically**: Select `True` if you would like the secondary view to be fully displayed for a few seconds when the user accesses the object cockpit or configured report that the secondary view is defined for. Select `False` if only the caption of the secondary view shall be displayed in a title bar for a few seconds when the user accesses the object cockpit or configured report that the secondary view is defined for. The secondary view will then open if the user clicks the title bar.
- **Height**: Define the inner height of the quality widget window in pixel. This is the height available for the display of the configured report opened in the window.
- **Width**: Define the inner width of the quality widget window in pixel. This is the width available for the display of the configured report opened in the window.

3) In the toolbar, click the **Save** button.

Deleting a Wizard

Before a wizard is deleted, you should check to see if the wizard is implemented in other configuration objects such as object views or workflow templates. To do so, right-click the wizard and click **Show Usage**. The **Objects Usage** editor will display the configuration objects using the selected wizard. For more information regarding the **Show Usage** functionality, see the section Using the Show Usage Functionality in the chapter Getting Started with Alfabet Expand.

1) Right-click the custom wizard that you want to delete and select **Delete**.
2) Confirm the warning by clicking **Yes**. The wizard is deleted from the explorer.
3) In the toolbar, click the **Save** button to save your changes.

Implementing a Wizard Instead of an Editor to Capture and Define Data

In Alfabet, either standard/custom editors or configured wizards can be implemented to capture data about a new object. The implementation of a configured wizard allows you to standardize the process of data capture. Unlike an editor in which only standard or custom object class properties can be defined upon object creation, a configured wizard allows you to configure a set of views in which reference data can be defined for a new object. Thus, in addition to the editor in which basic data is captured, the wizard can also include views capturing, for example, object indicators, reference documents, and relevant references to other objects in the IT architecture. If a configured wizard is not specified in the class setting, the default editor will be implemented for object creation.

Whether an editor or wizard will be implemented to capture data when creating a new object is configured in the class setting of the relevant object class. The configuration will impact all views in which an object governed by the class setting can be created.

For example, if you want to implement a wizard for the creation of an application, the configured wizard would be available in the **Applications** page view in the **Application Group** object profile, **ICT Object** object profile, and **Domain** object profile, the **Application Variants** page view in the **Application** object profile, and in the **Document Applications** functionality and **Capture Applications** functionality. The wizard would be available to create a new application, a new application variant, a new application version, as well as to edit an existing application.

Please note that some standard page views in Alfabet implement a predefined editor when creating and editing objects. For an overview of the page views with a predefined editor, see the section *Alfabet Page Views with Preconfigured Editing and Navigation Behavior* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

If object class stereotypes are implemented for an object class, you can determine whether the wizard or editor is implemented for each object class stereotype. In this case, for example, you could specify that a wizard is implemented in all relevant views for the object class stereotype Technical Application and that the conventional editor is used in all relevant views for the object class stereotype Business Applications.

To specify that a standard or custom wizard is to be implemented for object creation, you must do the following:

- Go to the **Presentation** tab, expand the **Class Settings** node, expand the folder for the relevant object class, click the class setting  and define the following attributes:
- **Wizard**: Select the wizard that you want to implement in the class setting.
- **Default Editor Type**: Select `Wizard`.
- Ensure that the relevant class setting is selected for the view scheme that is assigned to the relevant user profile. For more information about user profile configuration, see the chapter Configuring User Profiles for the User Community.

Implementing a Wizard in the Workflow Capability

A wizard can be implemented in the workflow capability to capture the initial data required for a new object or to provide a structured process for the definition of references, cost data, project planning, etc. A wizard that is to be implemented in the workflow capability must first be configured in the **Wizards** node. Once the wizard has been configured and tested, it must be assigned to the relevant workflow step.

If a wizard is only implemented in a workflow, it does NOT need to be assigned to a class setting. For more information about assigning a wizard to a workflow, see the section Defining the View Implemented for a Workflow Step in the chapter Configuring Workflows.

To specify that a wizard is to be implemented in a workflow, go to the **Workflows** tab, expand the relevant workflow template  , click the workflow step  and define the following attributes:

- **Type**: Select `Wizard`.
- **View**: Select the wizard that you want to implement in the workflow step.

Testing the Wizard in the Alfabet User Interface

After you have defined the wizard, you can test the wizard in the Alfabet interface and make it available to the user community. To do this, you must assign the wizard to the relevant class setting, assign the class setting to a view scheme, assign the view scheme to a user profile, and then review the wizard in the Alfabet interface. If a wizard has been configured for the workflow capability, it does not need be assigned to a class setting. For more information about implementing wizards in workflows, see Creating a Workflow Step in the chapter Configuring Workflows.

Once the wizard is assigned to the relevant user profile, it will be available to users accessing Alfabet with the corresponding user profile. For more information about configuring a user profile to access and test the wizard, see the chapter Configuring User Profiles for the User Community:

Chapter 9: Configuring Standard Business Functions and Custom Explorers

The **Functions** tab displays all standard business functions that may be implemented for a user profile as well as all custom explorers that have been configured for your enterprise. Expand the **Business Functions** root node to display the **Standard Business Functions** node and the **Custom Explorers** node.

The **Standard Business Functions** node contains all protected standard business functions and standard explorers that may be assigned to a user profile. The standard business functions and standard explorers are bundled in folders based on their functionality. You can specify a custom help for the standard business functions. The **Custom Explorers** node allows you to create and configure custom explorers that address the specific needs of your enterprise. Custom explorers can be assigned to one or more user profiles.

The following information is available:

- Implementing Standard Business Functions in Your Solution
- Configuring a Custom Explorer
- Defining the XML Def Attribute for a Customized Explorer
- Defining a Name for the Explorer
- Defining the Content of the Explorer Root Node
- Defining the Nodes of the Explorer Tree

Implementing Standard Business Functions in Your Solution

The standard business functions and standard explorers displayed below the **Standard Business Functions** node are bundled in folders based on their functionality. These are protected business functions and can be made available to the user profiles that you configure for your solution configuration. Any of the standard business functions that you see listed here may be made available to a user profile via the configuration of menu items or guide pages/guide views. This is carried out in the **Admin** tab in Alfabet Expand. For more information about making business functions available to the user community, see the section Making Functionalities Accessible to a User Profile.

Configuring a Custom Explorer

An explorer allows users to work with a subset of objects in Alfabet. Via the **Custom Explorers** node, you can build custom explorers and assign them to user profiles as Alfabet functionalities.

The following is important to understand in order to build customized explorers:

- Each explorer has a root node icon. Below the root node, an explorer can display one or multiple levels of nodes.
- Each explorer has a base class. The first node level in the explorer will display only objects in the specified base class.

- An object class stereotype can be configured as the base class of a custom explorer. In this case, the root node level in the explorer will display objects of the specified object class stereotype. By configuring an object class stereotype based on the object class `GenericReferenceData`, it is thus possible to specify multiple root nodes in a custom explorer. Each object based on the object class stereotype created in the **Generic Reference Data** functionality would serve as a root node in the custom explorer. The query specified in the **XML Def** attribute of the custom explorer must be specified to search for the objects that are subordinate to the root nodes.

To specify generic reference data, see the section *Configuring Objects for the Object Class Generic Reference Data* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.

- Each subordinate level in the explorer may display one or more object classes. For example, stacks and deployments can be child nodes of a node representing an application. Or, applications and application groups can be child nodes of a node representing an application group.
- An object displayed in an explorer must have a defined relationship to the parent object and the subordinate objects in the explorer hierarchy.
- Each node in the explorer except the root node represents an object in the Alfabet database. If you click a node in the explorer, the object view for the object will open.

The object views displayed in Alfabet are configurable in Alfabet Expand. Whether the standard object profile or a customized object view opens when an object in an explorer is clicked is determined by the configuration of the class settings. A user profile can have only one class setting per object class assigned to it. However, multiple class settings can be configured for an object class and each class setting can thus be assigned to a different user profile in your enterprise. Once all class settings that are relevant for a user profile are configured, they must be assigned to the view scheme that is configured for and assigned to a user profile. Regardless of the custom or standard explorer associated with the user profile, the content of the object view that opens when an object in the explorer is clicked will always be determined by the class setting configured for that object class. For more information about configuring class settings and view schemes for a user profile, see the chapter Configuring User Profiles for the User Community.

The following must be included in the configuration of a custom explorer:

- The view that opens in the work area when a user clicks the root node.
- The base class of the explorer that is displayed directly below the root node. If a subordinate node is to be displayed in the explorer hierarchy, the relation to the object class associated with the subordinate level must also be defined.
- All subordinate levels of nodes in the explorer hierarchy and their relation to the parent level and the subordinate level, if relevant.

For an overview of the behavior of standard explorers in Alfabet, see the section *Working with Explorers* in the reference manual *Getting Started with Alfabet*. For an overview regarding the standard page views available for each standard explorer, see *Page Views Assigned to Explorer Root Nodes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

To create a new custom explorer:

1) Go to the **Functions** tab, right-click the **Custom Explorers** node and select **New Custom Explorer**. The new custom explorer is displayed in the explorer tree and the attribute window is visible on the right.

2) In the attribute window, define the attributes, as needed:

- **Name**: Enter a name for the custom explorer. This name is not displayed in the Alfabet interface.

  If the **Name** attribute of a business function is changed, the Alfabet Server must be restarted if the **Validate Access** attribute is set to `True`.

  A validation mechanism checks for correct syntax when defining a technical name. The technical name of a configuration object may not begin with an empty space nor include any of the following special characters: **\ / * ? " > < | ':**

  Furthermore, please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.

  Furthermore, if the technical name of a configuration object is changed, the name will be correctly updated in other configuration objects referencing the changed object during design time. Please note that the name of a changed configuration object will not be updated in guide pages nor if the guide pages reference a configuration object listed in the **Show Usage** functionality. If you plan to change the name of a configuration object, the reference in the guide page should be changed prior to changing the configuration object name.

- **Caption**: Enter the caption to display on the root node of the custom explorer.
- **Comments**: Enter information explaining the purpose of the custom explorer. This information is not displayed on the Alfabet interface.

  The **Tech Info** attribute displays technical information such as the date that the custom explorer was created and last updated as well as who last updated the custom explorer. This information is not displayed on the Alfabet interface.

- **Group**: You can either create a new explorer folder to store the custom explorer in or assign it to an existing explorer folder. To create a new explorer folder, enter the name of the new explorer folder in the field and click the Return key. To assign the explorer to an existing explorer folder, click the drop-down arrow and select an explorer folder to store the custom explorer in. All explorer folders that you have created as well as the standard explorer folders predefined by Software AG will be displayed.
- **Restrict to Administrative User Profiles**: Set to `True` if only users with an administrative user profile may access the custom explorer. Set to `False` if the custom explorer may be accessible to users with relevant access permissions.

  Administrative user profiles are user profiles for which the **Is Administrative User Profile** attribute has been set to `True`. Business functions that are relevant for an administrative user profile must be explicitly added to the user profile. For more information about configuring user profiles, see the section Creating User Profiles for the User Community.

- **Validate Access**: Enter `True` if access permissions should be checked by the system when accessing the custom explorer via a link in a guide page, guide view, express view, etc. If set to `True`, the custom explorer must be assigned to the user profile of the user attempting to access the functionality. Please note that if the **Validate Access** attribute is set to `False` but the **Restrict to Administrative User Profiles** attribute is set to `True`, then only users with an adminstrative access may access the custom explorer.

- **Function Type**: Displays `CustomExplorer`. This attribute cannot be edited.
- **Usage Type**: Displays `Functional`, which specifies the level of complexity relevant for purchasing contracts. This attribute cannot be edited.
- **XML Def**: Click the **Browse** button to open the text editor and define as described below and click **OK** to save your changes. See the section Defining the XML Def Attribute for a Customized Explorer.
- **Automated Assistant URL**: Enter the URL or server variable that targets the content to display in the Assistant assigned to the custom explorer. For more information about configuring the automated assistant capability, see the chapter Providing Custom Online Help to the User Community.
- **Custom Context-Sensitive Help URL**: Enter the URL or server variable that targets the custom context-sensitive help. For more information about configuring a custom context-sensitive help file, see the chapter Providing Custom Online Help to the User Community.
- **Standard Context-Sensitive Help Index**: This field is not relevant for a custom explorer.
- **Standard Context-Sensitive Help Visible**: Select `True` to display both the standard and custom context-sensitive help file in the **Help Selector** Select `False` if the link to the standard context-sensitive help should not be available and should not be displayed in the **Help Selector**.

3) In the toolbar, click the **Save** button to save your changes.

Defining the XML Def Attribute for a Customized Explorer

To define or edit the explorer, you must define the XML object in the **XML Def** attribute of the custom explorer. Please note the following when configuring the XML definition of a custom explorer:

- It is not possible to specify XML elements or XML attributes other than the ones described in the section below. For an overview of the permissible syntax for XML definitions, see the sections Working with XML Objects and Defining Queries in XML Elements.
- XML is case-sensitive.
- You can leave some definitions out if they are not required. For example, it is possible to specify explorers that do not open a view at the root node.

The content and layout of the explorer is defined in the XML element `ExplorerDef`. The following table lists the `ExplorerDef` element and all of its sub-elements that can be configured to specify the content of the report and provides information about the purpose of each sub-element:

XML Element	Required to Configure:	Maximum Number of Elements Allowed
ExplorerDef	General layout of the explorer	1 as top element

XML Element	Required to Configure:	Maximum Number of Elements Allowed
Query	Defines the Alfabet objects that are the content of the explorer's first-level nodes when specified as a subordinate XML element of the XML element ***ExplorerDef***. Defines the Alfabet objects that are the content of the explorer's lower-level nodes when specified as a subordinate element of the XML element ***ClassEntry***.	1 as child of `ExplorerDef` multiple as child of ***ClassEntry***
ClassEntry	Definition of the object classes included in the explorer and their relation to the objects of the subordinate nodes.	Multiple as child of ***ExplorerDef***
AlfaExplorerNode	Definition of the root node of the explorer and definition of the behavior of the object nodes in the explorer tree.	Multiple as child of ***ExplorerDef***

The following example displays a complete explorer specification for an explorer that displays applications for which the current user is the authorized user and all application variants, their stacks as well as the stacks of the application variants and the deployments of the stacks.

On the root node, the Alfabet page view listing all applications for which the current user is the authorized user is displayed.

```
<ExplorerDef>
   <AlfaExplorerNode
      ClassName="CntxExplorerRoot"
      UpdateViews="APP_UserApplicaions"
      NodeView="GraphicView:APP_UserApplicaions"
   />
   <Query

      ClassName="Application"
      Query="ALFABET_QUERY_500 FIND Application
      INNERJOIN Person ON Application.ResponsibleUser=Person.REFSTR
      WHERE
      (AND Person.REFSTR =:CURRENT_USER
      Application.VariantOf IS NULL
      SHOW Application.Name Application.Version
      SORT Application.Name Application.Version" />
   <ClassEntry
```



```
            ClassName="Application"
            ShowProps="Name,Version"
            SortProps="Name,Version" >
            <Query

               ClassName="Application"
               Query="VariantOf CONTAINS:BASE" />
            <Query

               ClassName="Stack"
               Query="(AND Owner CONTAINS:BASE UpgradeFrom IS NULL)" />
         </ClassEntry>
         <ClassEntry

            ClassName="Stack"
            ShowProps="Name,Version"
            SortProps="Name,Version" >
            <Query

               ClassName="Stack"
               Query="UpgradeFrom Contains:BASE" />
            <Query

               ClassName="Deployment"
               Query="Stack CONTAINS:BASE" />
         </ClassEntry>
         <ClassEntry

            ClassName="Deployment"
            ShowProps="Name"
            SortProps="Name" >
         </ClassEntry>
      </ExplorerDef>
```


Defining a Name for the Explorer

You can specify a name for the explorer in the XML element `ExplorerDef` with the XML attribute `Name`. The name is only used for internal identification purposes and is not displayed in the Alfabet interface.

Defining the Content of the Explorer Root Node

The view that opens when clicking the root node of the explorer, is defined in an XML element ***AlfaExplorerNode***. The XML element must contain an XML attribute `ClassName` with the value `CntxExplorerRoot` and a specification of the view displayed when the root node of the explorer is clicked.

Optionally, you can define that the explorer is updated during the current session when a user performs changes in a defined page view or set of page views when the changes impact the objects of one of the object classes displayed in the explorer.

```
<AlfaExplorerNode ClassName="CntxExplorerRoot"
NodeView:"GraphicView:APPG_Root" UpdateViews="APPG_Root"/>
```

The table lists all XML attributes that can be specified for the XML element ***AlfaExplorerNode*** and its child elements.

XML Attribute	Allowed Values or Data Types	Default Values / Mandatory	Description
`ClassName`	`"CntxExplorerRoot"`	Mandatory	The XML attribute `ClassName` must be set to `CntxExplorerRoot` to ensure that the XML attribute specifies the root node of the custom explorer.
`NodeView`	`GraphicView: ViewName` `Report: ViewName`	Mandatory	Specify which page view or configured report you want to add to the root node. Standard page views must be specified as `"GraphicView:"` followed by the **Name** attribute of the page view. For a list of all page views with their name and caption and a short description of the view, see *Page Views Assigned to Explorer Root Nodes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*. Configured reports must be specified as `"Report:"` followed by the **Name** attribute of the configured report. **NOTE:** If the custom explorer is configured to display workflow activities (as an alternative to **Workflow Activities Explorer** (`WFS_Explorer`), then you must specifiy `NodeView="GraphicView:CustomExplorerHtmlView"` in order to call the relevant view required to display the configured HTML templates for the workflow

XML Attribute	Allowed Values or Data Types	Default Values / Mandatory	Description
			activites. For more information, see the section Specifying the Customized Workflow Activities Explorer (WFS_Explorer) or a Custom Explorer.
`UpdateViews`	*ViewName,ViewName*	Mandatory	Enter the **Name** attribute of the graphic view to trigger the update of the explorer tree. If changes are made in the view by creating, deleting, or editing objects of the object classes displayed in the explorer, the explorer tree will be updated to reflect the current changes. Multiple views can be defined in a comma-separated list. Only standard Alfabet views can be specified as the update view.

Defining the Nodes of the Explorer Tree

Each level of nodes in the explorer displays objects of one or multiple object classes that are related to the object class of the parent node.

The objects will be displayed in the explorer with their relevant icon. Please note that if a custom icon has been defined for an object displayed in the custom explorer, the custom icon will be displayed. If the object in the custom explorer is based on an object class stereotype, the custom icon will have precedence over the icon defined for the object class stereotype. If no custom icon or object class stereotype icon has been defined, the standard icon will be displayed for the object class.

For each node, the objects in the node must be specified by an XML element ***ClassEntry*** that specifies the object class and the default display of the nodes for the object class. The child XML element ***Query*** of the XML element ***ClassEntry*** specifies the relation of the object in the node to objects in child nodes.

The following must be taken into account when defining explorer nodes:

- To define which objects are displayed as top level explorer nodes, an XML element ***Query*** is specified directly as child element of the ***ExplorerDef*** element because this node level does not have a parent node that can contain the XML element ***Query***.
- The XML element ***ClassEntry*** for the leaf level does not contain a child XML element ***Query*** because no objects for a next layer may be specified. Nevertheless, the ***ClassEntry*** element must be specified for the leaf level of all explorer nodes.
- The XML element ***ClassEntry*** can contain multiple child XML elements ***Query***. This allows to display nodes for multiple different object classes under the same parent node.

- Only one ***ClassEntry*** element can be specified for an object class but the object class can be referenced multiple times by different XML elements ***Query*** in any ***Class Entry*** element in the explorer definition.
- A XML element ***ClassEntry*** can contain a XML element ***Query*** that references the same XML element ***ClassEntry***. For example, the XML element ***ClassEntry*** of the class `Application` can contain an Alfabet query with applications as a base class. The same XML elements `ClassEntry` and `Query` are then evaluated at each level of the explorer tree. This specifies that the explorer will display a hierarchy of parent and child objects.
- When building the explorer tree, Alfabet starts evaluating the Alfabet query defined directly in the XML element ***ExplorerDef*** and displays all objects of the base class of the Alfabet query that match the search criteria in the first level of explorer nodes. The mechanism then searches for the XML element ***ClassEntry*** of that class and reads the information from the child XML elements ***Query*** of that class regarding which objects to display. For the next level of nodes, the XML elements ***Query*** in the XML element ***ClassEntry*** of the classes displayed in the second level are evaluated. This means that the order of processing the specification is not necessarily the order written in the XML definition. For example, the first level displays applications, and the XML elements ***Query*** in the XML element ***ClassEntry*** for the class Application specify that application variants and local components are displayed on the second level. In the XML element ***ClassEntry*** for application variants, local components are allowed as third level elements. The XML element ***ClassEntry*** used for local components on the third level is the same as the one for local components in the second level. If a ***Query*** child element is defined in the local component's XML element ***ClassEntry***, objects found by the defined Alfabet query are displayed as child elements in the third and fourth level nodes under local components.
- A XML element ***ClassEntry*** can have XML elements ***Query*** that searches for the same class. This allows a hierarchy of parent and child elements to be constructed. For example, an explorer can be constructed with application groups that have no parent application group on the first level. In this case, the XML element ***ClassEntry*** for application groups has a XML elements ***Query*** with an Alfabet query searching for application groups that are assigned to the current application group. The same XML element ***ClassEntry*** is evaluated for each node level in the explorer until no more child elements are found for an application group.

The table below lists all available attributes in the XML element ***ClassEntry*** and XML element ***Query***.

XML Elements (Bold) and Attributes	Allowed Values or Data Types	Default Values / Mandatory	Description
ClassEntry			
`ClassName`	Name of Alfabet object class	**MANDATORY**	Defines the object class specified in this XML element ***Query***. For the object classes that allow stereotype specifications (for example `Project` and `Domain`), the XML attribute `ClassName` can specify an object of a specific stereotype. To define an XML element

XML Elements (Bold) and Attributes	Allowed Values or Data Types	Default Values / Mandatory	Description
			Query for a stereotype, the XML attribute `ClassName` must be *`ClassName:StereotypeName`* For example: *`Project:StatementOfWork`*

`ShowProps`	Name of a property of the object class	**MANDATORY**	Defines the text displayed in the explorer on an explorer node. The content of the selected property for the object is displayed in the explorer. Multiple properties can be selected in comma separated format. A typical specification of `ShowProps` is or
			`ShowProps="Name"`
			`ShowProps="Name, Version"`

`SortProps`	Name of a property of the object class	Optional	Defines the sort order of explorer nodes in a node level. The content if the selected property is used for sorting in ascending alphanumerical order. Multiple properties can be selected in comma separated format. Sorting is then performed in the order of specification of the properties.
Query			
`ClassName`	Name of Alfabet object class	Class specified in the Alfabet query	Defines the object class for the node level below the current level. This attribute is mandatory in case the XML attribute ***Query*** contains a `WHERE` clause only.
`Query`	Valid Alfabet query or `WHERE` statement of Alfabet query language	**MANDATORY**	Defines the content and number of the objects in the next node level. Either a valid Alfabet query or a `WHERE` statement has to be written into the parameter that finds the relevant objects.

Take the following into account when defining the Alfabet query in the XML element ***Query***:

Basic knowledge about how to define a valid Alfabet query is assumed in this description. If you are not familiar with alfabet queries, see the chapter Defining Queries.

- The Alfabet query to define subordinate node levels must include a `WHERE` statement that contains the parameter `:BASE`. This parameter specifies the object in the current level.

  The use of the `:BASE` parameter is not allowed for the root node objects.

- Show and sort properties are not allowed in the Alfabet query. The show and sort properties for an explorer node are read from the attributes of the XML element ***ClassEntry*** of the object class of the node.
- `JOINS` can be specified if the `WHERE` conditions require comparisons with objects of other classes. For example, a `WHERE` condition can specify that only applications that offer business services are displayed. This requires a `JOIN` to the class `Business Service`.
- The `FIND` statement is optional, because the base class for the Alfabet query is defined separately in the XML element ***Query*** with the XML attribute `ClassName`. Therefore, if no `JOINS` are required to specify the `WHERE` conditions, you can write a `WHERE` statement into the XML attribute ***Query*** instead of a complete Alfabet query.
- When the query contains special characters (for example, a greater then (>) or lesser then (<) symbol) in the `WHERE` clause, you have to replace it with the HTML code for angle brackets:
- `>` for >
- `<` for <
- `&quot;` for "
- `&#091;` for [
- `&#093;` for]
- The instruction `SetRowsStereotypeIndex` can be used in the query to display stereotype-specific icons for objects in the explorer. The query must return one column in the dataset that finds the object class stereotype of the objects. The row is used by the instruction to evaluate which object icon shall be displayed for each object in the explorer. The `RemoveColumns` instruction can be used to hide the stereotype information from display in the object information in the explorer.

Specifying the Content of Explorer Nodes and the Update of Explorer Content

Optionally, you can define the following for a node view:

- The automatic update of relevant objects in the explorer during the current session if a user performs changes in a defined page view or set of page views.
- The automatic display of a defined view in the working area when the user clicks a node in the explorer. By default, the object profile of the object represented by the explorer node is displayed. When multiple object profiles and object cockpits are defined for an object class, the object profile/object cockpit that opens when the node is clicked will depend on the view scheme assigned to the user profile that the user is logged in with when working with the explorer. The explorer nodes can be configured to ignore the view scheme definition and display a defined graphic view or object view. This can be useful, for example, in order to configure an explorer focused on a defined task only. For example, the explorer might only display the **Business Support Map** page view of

applications in order to provide a functionality that is limited to editing business support for applications.

An XML element ***AlfaExplorerNode*** must be added for each object class node in the explorer tree that is to be updated. The XML element must contain an XML attribute `ClassName` identical to the XML attribute `ClassName` of the XML element ***ClassEntry*** defining the explorer nodes for the object class. The XML element can contain a definition of the graphic view that is to trigger the update of the explorer tree content on the current node level and/or a definition of the view that opens when a user clicks the node.

```
<AlfaExplorerNode ClassName="Application"
NodeView="ObjectView:APP_ObjectView_Customized"
UpdateViews:"GraphicView:APPG_Applications" />
```

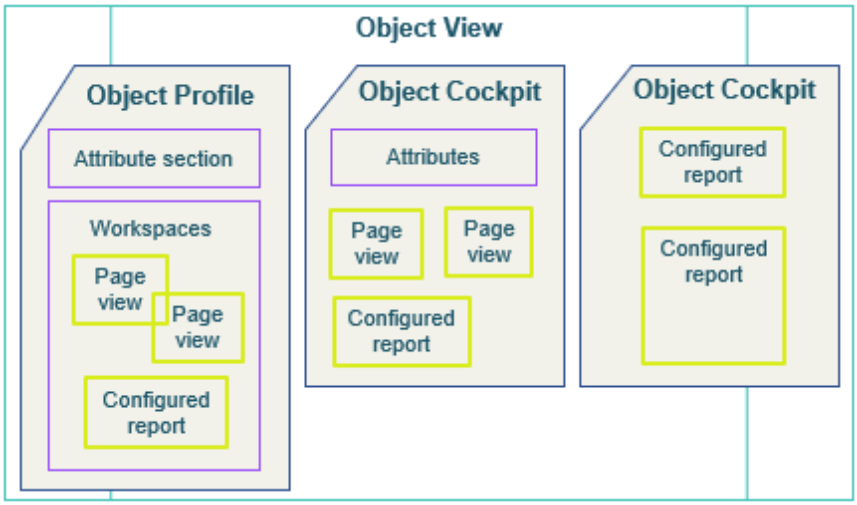
The table lists all attributes of the XML element ***AlfaExplorerNode*** and its child elements.

XML Attribute	Allowed Values or Data Types	Default Values / Mandatory	Description
`ClassName`	*ClassName*	Mandatory	The XML attribute `ClassName` must be identical to the XML attribute `ClassName` of the XML element ***ClassEntry*** defining the explorer nodes that shall be updated.
`UpdateViews`	*ViewName,ViewName*	Optional	Enter the **Name** attribute of the graphic view that is to trigger the update of the explorer tree. If changes are made in the view by creating, deleting, or editing objects of the object class defined in the XML attribute `ClassName`, the explorer tree will be updated to reflect the current changes. Multiple views can be defined in a comma-separated list. **NOTE:** Only a standard Alfabet view can be specified as the update view.

`NodeView`	`GraphicView: ViewName`
	
	`ObjectView: ObjectViewName`
		Optional	Specify which view you want to add to the class node. Page views and object views can be added to a class node. Standard page views must be specified as `"GraphicView:"` followed by the **Name** attribute of the page view. For a list of all page views with their name, caption, and a short description of the view, see *Page Views Assigned to Explorer Root Nodes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*. Object views must be specified as `"ObjectView:"` followed by the **Name** attribute of the object view.

Chapter 10: Configuring Object Views

An object view is the configuration object that typically consists of an object profile as well as any object cockpits that have been configured for the object view.



A standard object view is available for all object classes that require an object profile. The object profile is a standard overview that displays the basic information about an object with topical workspaces grouping all standard page views available for the object class. Object cockpits are not available in standard object views. Multiple custom object views may be configured for an object class or object class stereotype, whereby only one object view can be assigned per class setting per user profile.

The solution design can specify which standard and custom object class properties, page views, configured reports, and object cockpits should be visible or hidden for a custom object view. Each custom object view may include the standard object profile or a customized object profile as well as multiple configured object cockpits. The object cockpit is typically a focused and abbreviated set of data for a particular topic that includes pertinent attribute information as well as a few relevant page views or configured reports that are embedded in the object cockpit.

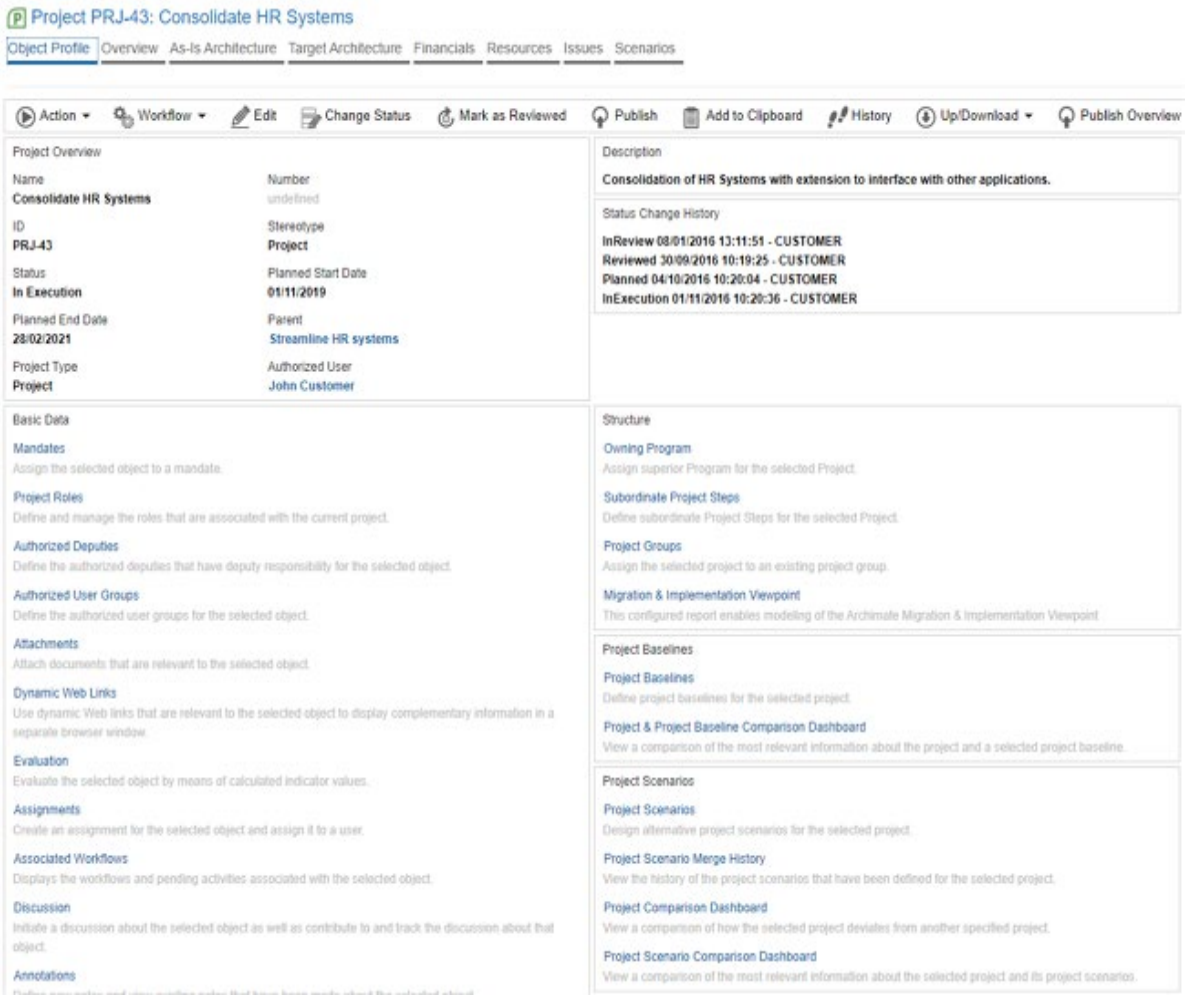


FIGURE: Object view configured for a project stereotype

The object profile and associated object cockpits are accessible via tabs in the header of the object view in the Alfabet user interface. The tabs of the object profile and object cockpits are displayed below the object header and the user can switch back and forth as needed. The caption of the object view is displayed on the object profile tab and the captions of the object cockpits are displayed on the cockpit tabs. The object profile can be hidden in the Alfabet user interface, if necessary.

The display of standard attributes and page views available for a standard object view is determined by Software AG.

A custom object view should be created and made available to a user profile if any of the following is relevant:

- Object class stereotypes have been configured for a relevant class.
- The standard object profile must be modified. The following is possible:
  - Specify a customized **Attributes Section** that includes custom properties
  - Specify a customized **Attributes Section** that regroups or hides standard properties
  - Rename or resequence workspaces that group page views
  - Remove individual page views or entire workspaces with their page views

- Include configured reports
- One or more object cockpits are required. An object cockpit provides users with an immediate and transparent overview of data for the selected object. It is an abbreviated and focused presentation of object data, typically from a particular perspective such as an architecture, technology, or strategy perspective. The display of data in an object cockpit can be highly customized. The following is possible:
- Add attributes with custom captions organized in group boxes.
- Specify that an attribute is highlighted with color or displayed as an icon based on its value.
- Display a URL or document link or properties that are fetched via a query.
- Embed views or configured reports directly in the object cockpit so that users can immediately see data without having to navigate to another page. In the case of views with a significant amount of data, you can include a placeholder that allows the user to navigate to a full display of the page view or configured report.
- Specify conditional restraints that control the information displayed in the object cockpit based on the value of an attribute.
- Customized context-sensitive help or an automated assistant providing specific help for the custom object view.

An overview of the workflow to configure custom object views is displayed in the section Conceptualizing the Object View.

Please keep the following in mind when conceptualizing the object views, class settings, and view schemes that you plan to implement in the user profiles you configure for your user community:

- Multiple custom object views can be defined for an object class, but only one object view can be assigned to a class setting.
- Multiple class settings can be defined for an object class, but only one class setting can be assigned per object class to a view scheme.
- Only one view scheme can be assigned to a user profile.

The following information is available:

- Understanding the Object View Node
- Conceptualizing the Object View
- Creating a New Custom Object View
- Creating a Custom Object View Based on an Existing Object View
- Creating a Custom Object View for an Object Class Stereotype
- Creating a New Custom Object View for a Custom Object Class
- Configuring an Object Profile for a Custom Object View
- Configuring the Attributes Section of the Object Profile
- Adding Standard or Custom Properties to the Attributes Section

Understanding the Object View Node

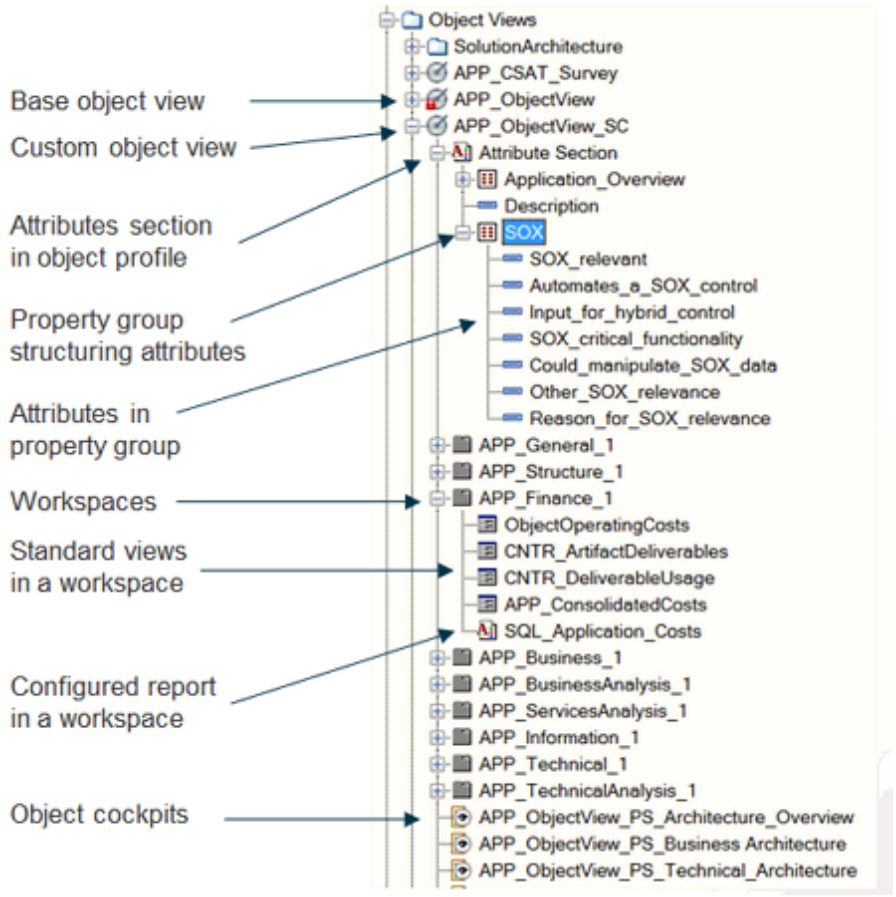


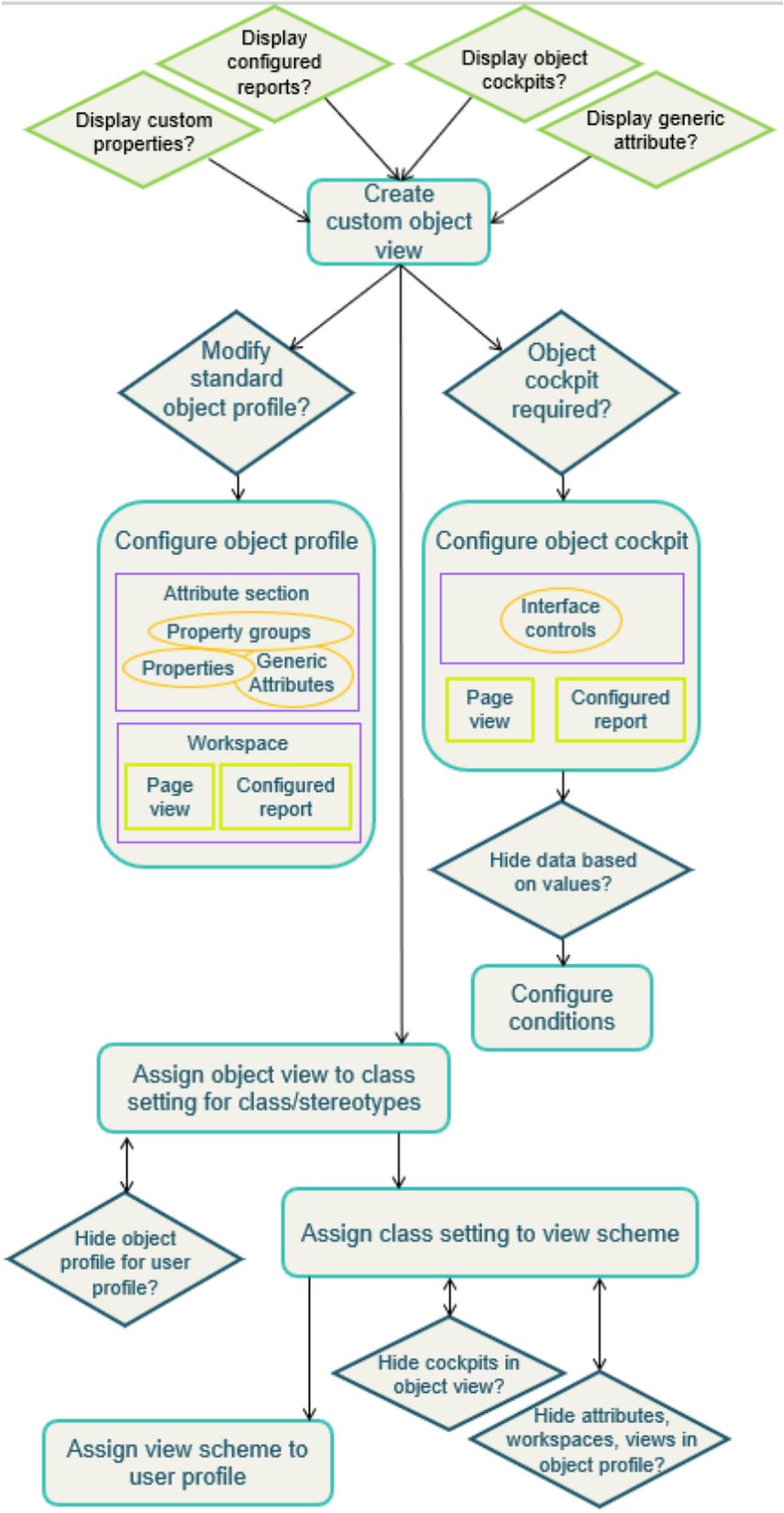
FIGURE: Object Views node with the objects relevant for the configuration of an object view

To view existing standard object views that have been preconfigured by Software AG, go to the **Presentation** tab and expand the **Object Views** node. You may see any of the following:

- Standard private object views hat can be used as the base object view to create and configure custom object views. Private object views cannot be edited and are only available as a template. They can be copied to create a custom object view.
- Standard protected object views that can be edited in a limited way. Protected object views typically include object views that require controlled user access.
- Public (custom) object views that have been created by your enterprise.

Conceptualizing the Object View

The following image illustrates the steps to configure an object view:



Creating a New Custom Object View

An object view is the configuration object that typically consists of an object profile as well as any object cockpits that have been configured for the object view. A standard object view is available for all object classes that require an object profile. The object profile is a standard overview that displays the basic information about an object with topical workspaces grouping all standard page views available for the object class. Object cockpits are not available in standard object views. Multiple custom object views may be configured for an object class or object class stereotype, whereby only one object view can be assigned per class setting per user profile.

The solution design can specify which standard and custom object class properties, page views, configured reports, and object cockpits should be visible or hidden for a custom object view. Each custom object view may include the standard object profile or a customized object profile as well as multiple configured object cockpits. The object cockpit is typically a focused and abbreviated set of data for a particular topic that includes pertinent attribute information as well as a few relevant page views or configured reports that are embedded in the object cockpit.

The following information is available:

- Creating a Custom Object View Based on an Existing Object View
- Creating a Custom Object View for an Object Class Stereotype
- Creating a New Custom Object View for a Custom Object Class

Creating a Custom Object View Based on an Existing Object View

You can create a new custom object view by copying either a private object view (a preconfigured standard Alfabet object view) or an existing custom object view that has already been configured by your enterprise. The copied object view can be modified as needed. For example, you could add or remove properties in the **Attributes** section, rename or remove workspaces, add or remove page views in the workspaces, and configure one or more object cockpits.

A few protected object views are available for limited configuration (`USERG_RO_ObjectView`, `USER_RO_ObjectView` and <ObjectClass>`_LE_ObjectView`). These object views require controlled user access and are thus preconfigured by Software AG to have limited access permissions. Please note the following:

- The name of the protected object view cannot be changed. You can create a custom object view based on one of the protected object views but the custom object view will not be available in the Alfabet functionalities where the base protected object view is typically implemented.
- New workspaces, page views and configured reports can be added to protected object views.
- Visibility can be configured for the standard workspaces and page views included in the protected object view.
- Inline-editing in not possible in read-only object views.

- Future changes by Software AG to standard protected object views must be manually updated to modified or customized protected object views for each future release. **Customized protected object views will not be migrated to future releases of Alfabet and must be manually reconfigured in such cases**.

Click the + symbol to expand a private object view in order to view the attributes, workspaces, and page views that have been preconfigured for the object view.

To create a new custom object view based on a copy of an existing object view:

1) Go to the **Object Views** folder, right-click the private object view or custom object view that you want to base the custom object view on and select **New Custom Object View as Copy**. The new custom object view is displayed in the explorer. The new object view has an underscore and automatically generated number appended to the name. The **Attributes** section including its properties as well as the preconfigured workspaces with their page views have been copied to the new object view.

   Please note that if a new custom object view based on an existing custom object view is created via the copy and paste functionalities, the workspaces configured for the source custom object view will be reused. If a reused workspace in the source or new custom object view is deleted, the reused workspace will be deleted from all other object views in the same database.

2) Right-click the new custom object view to activate its attribute window. The **Class Name** attribute displays the object class for which the custom object view was created. Define the following attributes, as needed:

- **Name**: Enter a name for the object view. This name is not displayed in the Alfabet interface.

  A validation mechanism checks for correct syntax when defining a technical name. The technical name of a configuration object may not begin with an empty space nor include any of the following special characters: **\ / * ? " > < | ':**

  Furthermore, please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.

  Furthermore, if the technical name of a configuration object is changed, the name will be correctly updated in other configuration objects referencing the changed object during design time. Please note that the name of a changed configuration object will not be updated in guide pages nor if the guide pages reference a configuration object listed in the **Show Usage** functionality. If you plan to change the name of a configuration object, the reference in the guide page should be changed prior to changing the configuration object name.

- **Caption**: Enter the caption to display on the object profile tab in the user interface.
- **Format String**: Specify the standard and custom properties to display in the header of the object view. The **Name** property will be displayed per default if no value is defined. The properties must be listed comma-separated on the first line. The formatting of the properties is specified as variables enclosed in curly brackets. For example, for the object class stereotype **Banking Application**, the following definition in the text editor displays the following results in the object profile in the Alfabet interface.

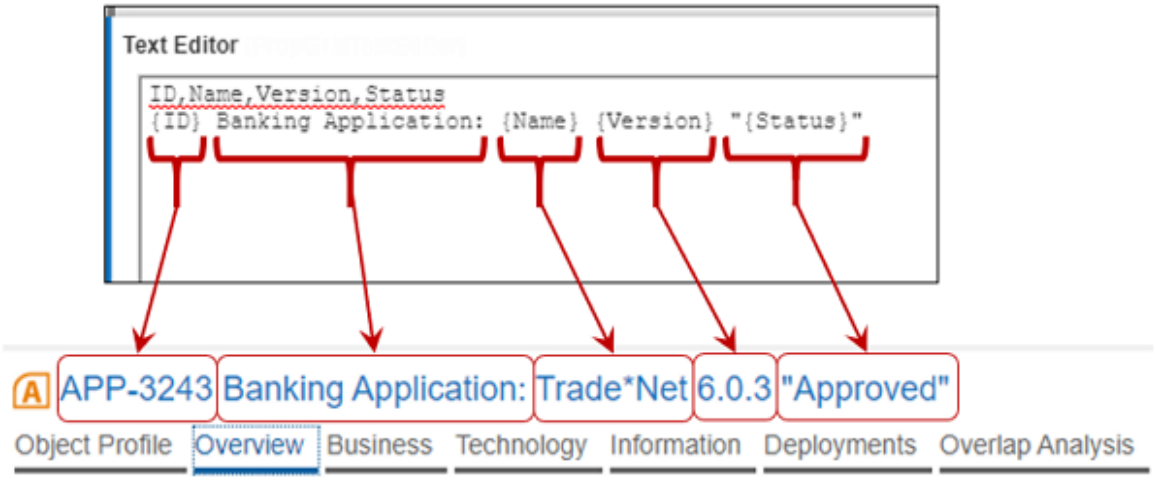


FIGURE: Format string for an application stereotype

If the **Format String** attribute is not defined for the object view, the **Format String** attribute for the class setting that the object view is assigned to will be displayed. The **Format String** attribute for an object view will have precedence over the **Format String** attribute for the class setting. If a **Format String** attribute is not specified for neither the object view nor the class setting, the following will be displayed: `<Object.Class.Caption|Object.Stereotype.Caption + Object.ID + ":" + Object.ClassSetting.ImageProps>`. If no image properties are defined in the class setting, the **Name** property will be used. If neither image properties nor the **Name** property are defined in the class setting, the header will be defined as follows: `<Object.Class.Caption|Object.Stereotype.Caption + Object.ID + ":">`.

- **Can Create Bookmark**: Select `True` if bookmarks can be created for the object view, or select `False` if bookmarks cannot be created for the object view. Please note that the custom object view may only be added to a user's splash screen if the **Can Create Bookmark** attribute is set to `True` for the respective object view. For more information about the splash screen capability, see the chapter *Creating a Splash Screen As Your Start Page* in the reference manual *Getting Started with Alfabet*.
- **Can Create Express View**: Select `True` if express views can be created for the object view, or select `False` if express views cannot be created for the object view.
- **Prevent Inline Editing in Object Views**: You must specify whether inline editing is permissible for scalar and reference properties in the selected object view. Select `True` if inline editing shall be prevented in the object view. Select `False` if inline editing shall be allowed in the object view. If no value is selected, inline editing shall be allowed per default (= `False`). For more information about the configuration required to allow inline editing of scalar and reference properties, see the section Configuring Inline Editing of Attributes in the Object View.
- **Comments**: Enter information explaining the purpose of the object view. This information is not displayed on the Alfabet interface.

The **Tech Info** attribute displays technical information such as the date that the object view was created and last updated as well as who last updated the object view. This information is not displayed on the Alfabet interface.

3) In the toolbar, click the **Save** button to save your changes.

Creating a Custom Object View for an Object Class Stereotype

Custom object views for object class stereotypes are created via the base class as described in the section Creating a Custom Object View Based on an Existing Object View. Multiple custom object views may be created per object class stereotype. The custom object view must be assigned to the relevant class setting defined for the object class stereotype. For more information about creating a class setting for an object class stereotype, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes.

Creating a New Custom Object View for a Custom Object Class

If you have configured a custom object class, you must explicitly create an object view for that object class. To do so, right-click the **Object Views** folder and select **New Object View** in the context menu. In the **Select Class** editor, select the custom object class that you want to create the object view for and click **OK**. A new object view will be added to the **Object Views** folder. Define the object view as described in the section Creating a Custom Object View Based on an Existing Object View.

Please note that if custom object views required for a survey class are created for the survey in the Surveys tab, additional configuration is required. For more information about working with surveys, see the chapter Configuring Surveys for Data Capture Campaigns.

Configuring an Object Profile for a Custom Object View

The object profile summarizes information that is relevant to a selected object. The object profile typically displays object class properties and their values for the object as well as workspaces with views and reports relevant to the selected object. Users can open the available views and define or edit the data about the selected object.

Typically, object profiles are configured specifically for the needs of the user community. Thus, various object profiles with different data and functionality may be available for the same object class, depending on the user profile with which users access Alfabet. Each object profile may have multiple object cockpits associated with it that have been configured to visualize a specific set of data about the object.

The object profile can be configured in the context of a custom object view. It is possible to hide the object profile in the context of the view scheme configuration. In this case, only the object cockpits visible for the view scheme will be available to the associated user profile.

When a new custom object view is created, the standard object profile associated with the private, protected, or public object view that the new object view is based on will be automatically copied to the new object view. The object profile can be modified or hidden in the Alfabet user interface, if necessary.

Consider the following in the configuration of a custom object view:

- If `SYS_VIEW_CustomRelationsManagement` (*Custom Relation Page View*) is included in a custom object view that is created as a copy of an existing object view, you must ensure that `SYS_VIEW_CustomRelationsManagement` is placed in a separate workspace that contains no other views. If `SYS_VIEW_CustomRelationsManagement` is in a workspace with other views, these views will no longer be visible in the object view at runtime.

- If `ObjectReportsDataSet` (*Configured Reports Page View*) is included in a custom object view, you must ensure that `ObjectReportsDataSet` is placed in a separate workspace that contains no other views. If `ObjectReportsDataSet` is in a workspace with other views, these views will no longer be visible in the object view at runtime.
- If these views are **NOT** put in their own workspace and there are other views in the workspace, all other views in the workspace or the entire workspace may be hidden if the visibility constraints for `ObjectReportsDataSet` / `SYS_View_CustomRelationsMgmt` are not fulfilled.
- Only ReadOnly versions of configured reports of the type `Custom` that are based on the template `EvaluationReport` should be implemented in a custom object view. You should not implement a ReadWrite version of these reports in the custom object view.

Please note that the font, hyperlink colors, and other style issues displayed in the object profile are determined by the configuration of the GUI schemes implemented in your enterprise. For more information, see the section Configuring GUI Scheme Definitions for the Alfabet Interface.

The following information is available:

Configuring the Attributes Section of the Object Profile

Once a new custom object view has been created, you can configure the **Attributes Section** of the object view. The **Attributes Section** preconfigured for the base object view will be copied to the new object view and any exiting property groups or properties will be displayed below the **Attributes Section** node.

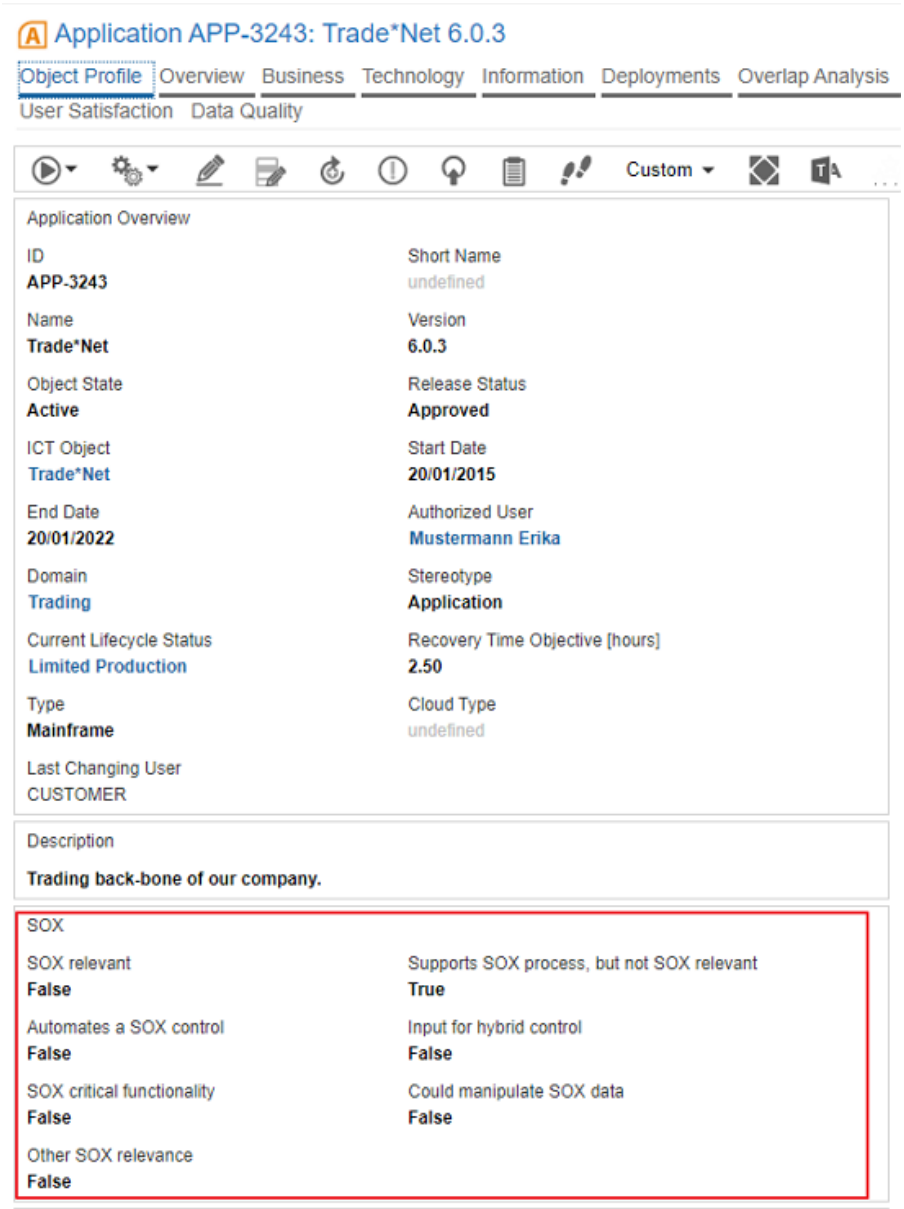


FIGURE: Attributes section in a custom object view

The following is possible in terms of the configuration of the **Attributes Section**:

- You can display the properties in the **Attributes Section** as a flat list or you can group sets of properties in subordinate property groups. Each property group can have a caption that is displayed as a heading for the group.

- You can add any standard or custom properties configured for the associated object class directly to the **Attributes Section** or to a property group defined for the **Attributes Section**. If you add a custom object class property of the type `URL`, the URL defined by the user in the relevant custom editor will be displayed as a hyperlink in the **Attributes Section**.
- You can add a property that is not a standard or custom object class property of the associated object class. Such properties require the configuration of an Alfabet query.
- You can remove any unwanted standard or custom properties that were copied to the custom object view from the base object view.
- You can display important page views or configured reports directly above or below of the **Attributes Section** so that they are easily accessible to the user community.

Please consider the following:

- Inline editing of scalar and reference properties may be configured so that users can edit data directly in the object profiles and object cockpits. If inline editing is supported, users can edit scalar attributes for an existing object directly in the **Attributes** section of an object profile or in an object cockpit without needing to open the object's editor or wizard. For more information about configuring inline editing, see the section Configuring Inline Editing of Attributes in the Object View.
- Users can determine whether only properties that have a value defined are displayed in the **Attributes Section** or whether properties that have no value defined are included in the **Attributes Section**. This setting is defined by a user in the **Show Empty Values in Object Profile** attribute in the **User Settings** editor in Alfabet.

Please note that if a scalar or reference property is displayed in an object profile/object cockpit and is editable in the user's wizard/editor for the associated object class/object class stereotype and the **Show Empty Values in Object Profile** attribute is set to `False` in the **User Settings** editor in Alfabet, the setting will be ignored so that the user can edit the property and provide a value using in-place editing. For more information about how a user can define the display of properties with no value defined, see the section *Defining Your User Settings in Alfabet* in the reference manual *Getting Started with Alfabet*.

The following information is available:

- Adding Standard or Custom Properties to the Attributes Section
- Copying and Pasting Properties to the Attributes Section
- Configuring Property Groups to Bundle a Set of Properties in the Attributes Section
- Adding a Generic Attribute to the Attributes Section
- Determining the Sequence of the Information Displayed in the Attributes Section

Adding Standard or Custom Properties to the Attributes Section

1) Depending on where you want to place the new property, do one of the following:

- To add properties to a property group, right-click the property group  and select **New Property**.

- To add a property directly to the **Attributes Section** without placing it in a subordinate grouping, right click the **Attributes Section** node  in the explorer and select **New Property.**

2) Click the new property to open the attribute window. Define the following:

- **Property**: Select the standard or custom object class property that you want to display.

Please note that if the property that you are defining is of the type `Reference`, you must specify the **Show Properties** otherwise a defined value for the property will not be displayed in the **Attributes** section.

If you want to add one or more properties that are not standard or custom properties of the associated object class, then you should leave the **Property** attribute empty and specify an Alfabet query or native SQL query. To do so, in the **SQL Type** field, select either `Query` to specify an Alfabet Query

or `NativeSql` to specific a native SQL query. Select `None` if no query will be defined. Enter the relevant query in the **Query** attribute. You will also need to define the **Show Properties** and **Sort Properties** attributes in order to determine what will be displayed in the **Attributes Section**. If the display of properties in the **Attributes Section** is determined via a query, the exclusion of properties configured for a class setting will not be applied.

- **Caption**: Enter the caption for the object class property selected in the object class property attribute. If no value is entered here, then the object class property will not be displayed in the **Attributes** section.
- **Mark as Mandatory**: Select `True` if a mandatory property should be marked with a red star as a mandatory property in the **Attributes Section**. Select `False` if no red star should be displayed next to the object class property in the **Attributes Section**.
- **Enable Data Translation**: Select `True` if the caption should be available in the vocabulary for translation.
- **Enable HTML Content**: If the **HTML Content** attribute is set to `True` for the Memo interface control in the editor where the property is defined, the attribute will be automatically set to `True` and the HTML formatting for the property will be displayed in the object profile. The HTML editor will also be available to edit the property directly in the user interface of the object profile. The **Enble HTML Content** attribute can be set to `False` if HTML format shall not be displayed.
- **Navigate**: Select `True` if the object class property is a reference to another object and the user should be able to navigate to that referenced object. In this case, the object class property will be displayed as a blue hyperlink that the user can click to navigate. Select `False` if the user should not be able to navigate to the referenced object.
- **Show in Foreign Mandates**: Specify `True` if the property should be shown when the object view is displayed for a user that does not have mandate rights on the current object.

3) Continue this procedure to add all properties to the **Attributes Section** as required.

4) In the toolbar, click the **Save**  button to save your changes.

Copying and Pasting Properties to the Attributes Section

You can copy a property or an entire property group configured in one object view and paste it to another object view.

1) Right-click the object class property or property group that you want to copy and select **Copy**.
2) Right-click the node in the object view that should be the ascendant object of the object class property or property group you are adding and select Paste.

For example, if the object class property is to be placed in a property group, you must select the object class property group node and click **Paste**. If the object class property is to be placed directly in the **Attributes Section**, click the **Attributes Section** node and click **Paste**.

3) Remove any unnecessary properties by right-clicking the object class property and selecting **Delete**.
4) In the toolbar, click the **Save**  button to save your changes.

Configuring Property Groups to Bundle a Set of Properties in the Attributes Section

You can display the properties in the **Attributes Section** as a flat list or you can group sets of properties in subordinate property groups. Each property group can have a caption that is displayed as a heading for the group.

1) Right-click the **Attributes Section** node  in the explorer and select **New Property Group**. A new property group  is added below the **Attributes Section** node.
2) Click the object class property group  to activate the attribute window. Define the following:

- **Caption**: Enter a caption for the header of the object class property group.
- **Attribute Sequence**: To order the sequence of properties in a property group, click the object class property group node and click the **Browse**  button. The **Sort Entries** editor opens. To define the sequence of the data, select an object in the list and click the up/down arrows in the upper right corner of the dialog. Repeat this for all properties until they are listed in the correct order. Click **OK** to close the editor.
- **Show in Foreign Mandates**: Specify `True` if the property group should be shown when the object view is displayed for a user that does not have mandate rights on the current object.

3) In the toolbar, click the **Save**  button to save your changes.

Adding a Generic Attribute to the Attributes Section

This section is only relevant for object profiles configured for the object classes `Application`, `Component`, `Deployment`, `Deployment Element`, `Standard Platform`, `Standard Platform Element`, `Stack`, `Stack Element`, and `Stack Item`.

If generic attributes have been created for an object in the *Generic Attributes Page View* (`ObjectGenericAttributes`), you can configure the **Attributes** section of the object profile to display the generic attributes and their values. For more information about working with generic attributes, see the section Implementing the Generic Attribute Concept Instead of Custom Properties in the chapter Configuring the Class Model.

To display generic attributes in the object profile:

1) Depending on where you want to place the generic attribute, do one of the following:

- Add a property to a property group by right-clicking the property group and selecting **New Property**.
- Add a property directly to the **Attributes Section** without placing it in a subordinate grouping by right-clicking the **Attributes Section** node in the explorer and selecting **New Property.**

2) Click the new property to open the attribute window. Define the following:

- **SQL Type**: Select `AttributeQuery`.
- **Query**: Enter a query to find the generic attributes for the object class. The query should be specified for the class `GenericAttribute` and return the relevant `SHOW` and `SORT` properties.

The following example displays an Alfabet query specified to display generic attributes for a configured object profile:

```
ALFABET_QUERY_500
FIND GenericAttribute
WHERE(AND GenericAttribute.Owner CONTAINS:BASE)
QUERY_XML
   <QueryDef></QueryDef>
      <ShowProperty Type="Property"
      ClassName="GenericAttribute" Name="Name" />
      <ShowProperty Type="Property"
      ClassName="GenericAttribute" Name="Value" />
      <ShowProperty Type="Property"
      ClassName="GenericAttribute" Name="Type" />
      <SortProperty Type="Property"
      ClassName="GenericAttribute" Name="Name" />
```

3) In the toolbar, click the **Save** button to save your changes.

Determining the Sequence of the Information Displayed in the Attributes Section

Once you have defined the content to display in the object profile, you can determine the order of the information that is displayed. You will be able to define the sequence of any page views, configured reports, individual properties, and property groups displayed in the **Attributes Section**. By default, properties in a property group are listed in two columns.

1) Click the new custom object view and define the following:

- To order the sequence of all property groups as well as any individual properties not grouped in a property group, configured reports, and page views displayed in the **Attributes Section**, click the **Attribute Section** node and click the **Browse** button in the **Attribute Sequence** attribute.
- To order the sequence of properties in a property group, click the object class property group node and click the **Browse** button in the **Attribute Sequence** attribute.

2) In the **Sort Entries** editor, select an object in the list and click the up/down arrows in the upper right corner of the dialog. Repeat this for all elements assigned to the **Attributes Section** until they are listed in the correct sequence. Select the **Sort Lexicographically** checkbox if the object class property groups and individual properties should be sorted alphabetically. Click **OK** to close the editor.

3) In the toolbar, click the **Save** button to save your changes.

Adding Configured Reports to the Object Profile Independent of a Workspace

You may decide that particular configured reports have data that is significant to users accessing the object profile and that you therefore want the reports to be easily and immediately accessible to the user community. In this case, you can add one or more configured reports directly to the object profile. This allows you to display them independent of a workspace and thus sequence them to be available at the top of the object profile after the **Attributes Section**.

Configured reports can also be assigned to a workspace. For more information, see the section Configuring Workspaces for an Object Profile.

Keep the following in mind when assigning configured reports to custom object views:

- Only configured reports of the type `Query` or `Custom` can be assigned to custom object views.
- A configured report with the **Report State** attribute set to `Planned` or `Retired` will be displayed in the custom object view, but an error message will be displayed if the user attempts to open the configured report.
- The text entered in the **Caption** attribute of the configured report is displayed as the title of the configured report in the configured report itself and as the default caption for the configured report in the object view. The report caption displayed in the object view can be overwritten by defining the **Caption** attribute for the respective report object assigned to the object view. This has no effect on the caption displayed in the configured report itself. Therefore, the caption can be different for the object view and the configured report itself.
- The text entered in the **Description** attribute of the configured report is displayed as the page view description in the object view of the report.
- You can provide customized context-sensitive help to your user community in order to understand the content of a configured report that you have configured. The Help file is stored externally and the URL of the help file is defined in the configured report. When the

user views the configured report in the Alfabet interface, he/she can click the **Help** button that opens the Alfabet online Help. The context-sensitive help screen that provides access to the context-sensitive online Help will display an option that can be clicked to open the hyperlinked help document.

- The **Can Create Bookmark** and **Can Create Express View** attributes are configured for configured reports in the context of the report configuration in the **Reports** tab. Please note that a configured report may only be added to a user's splash screen if the **Can Create Bookmark** attribute is set to `True` for the respective report. For more information about the splash screen capability, see the chapter *Creating a Splash Screen As Your Start Page* in the reference manual *Getting Started with Alfabet*.

For more information about configuring a configured report, see the chapter Configuring Reports.

To add a configured report to the object profile of a custom object view:

1) Right-click the new custom object view and select **Add Report**. The **Select Report** editor opens.
2) Select one or more configured reports that you would like to add to the object profile and click **OK**. Simultaneously select multiple configured report by holding down the CTRL key while you select the reports.
3) To sequence the configured reports (and page views), click the object view to activate its attribute window. In the **Workspace Sequence** attribute, click the **Browse** button to open the **Sort Entries** editor. In the **Sort Entries** editor, select a report in the list and click the up/down arrows in the upper right corner of the dialog. Repeat this for all reports until they are listed in the correct sequence. Select the **Sort Lexicographically** checkbox if the reports should be sorted alphabetically. Click **OK** to close the editor.

   Please note that the **Attributes Section** is not listed in the editor. The **Attributes Section** is always the first item in the object view. Therefore, the first item listed in the **Sort Entries** editor will be the first page view/configured report/workspace to follow the **Attributes Section**.

4) In the toolbar, click the **Save** button to save your changes.

Adding Page Views to the Object Profile Independent of a Workspace

You may decide that particular page views have data that is significant to users accessing the object profile and that you therefore want the page view to be easily and immediately accessible to the user community. In this case, you can add one or more page views directly to the object profile. This allows you to display them independent of a workspace and thus sequence them to be available at the top of the object profile after the **Attributes Section**.

Captions defined for page views in object profiles and object cockpits may only be changed via the localization method described in the section Modifying the Original Strings Provided by Software AG . in the chapter Localization and Multi-Language Support for the Alfabet Interface.

Page views can also be assigned to a workspace. For more information, see the section Configuring Workspaces for an Object Profile.

To add a page view to the object profile:

1) Right-click the new custom object view and select **Add View**. The **Select View** editor opens. All permissible page views not yet added to the object view will be displayed.
2) Select one or more page views that you would like to add to the object profile and click **OK**. Simultaneously select multiple configured report by holding down the CTRL key while you select the page views.
3) To sequence the page views (and configured reports), click the object view to activate its attribute window. In the **View Sequence** attribute, click the **Browse** button to open the **Sort Entries** editor. In the **Sort Entries** editor, select a page view in the list and click the up/down arrows in the upper right corner of the dialog. Repeat this for all page views until they are listed in the correct sequence. Select the **Sort Lexicographically** checkbox if the reports should be sorted alphabetically. Click **OK** to close the editor.

   Please note that the **Attributes Section** is not listed in the editor. The **Attributes Section** is always the first item in the object view. Therefore, the first item listed in the **Sort Entries** editor will be the first page view/configured report/workspace to follow the **Attributes Section**.

4) In the toolbar, click the **Save** button to save your changes.

Configuring Workspaces for an Object Profile

When you configure the workspaces for an object profile, all of the workspaces associated with the standard object profile will typically be included in the creation of the new custom object view. These workspaces can be modified, as needed, by adding and removing page views.

Once a custom workspace has been configured, you can copy that workspace and reuse it in other custom object views for the same object class. The copied workspace can then be further modified by adding and removing page views.

Consider the following in the configuration of a custom object view:

- If `SYS_VIEW_CustomRelationsManagement` (*Custom Relation Page View*) is included in a custom object view that is created as a copy of an existing object view, you must ensure that `SYS_VIEW_CustomRelationsManagement` is placed in a separate workspace that contains no other views. If `SYS_VIEW_CustomRelationsManagement` is in a workspace with other views, these views will no longer be visible in the object view at runtime.
- If `ObjectReportsDataSet` (*Configured Reports Page View*) is included in a custom object view, you must ensure that `ObjectReportsDataSet` is placed in a separate workspace that contains no other views. If `ObjectReportsDataSet` is in a workspace with other views, these views will no longer be visible in the object view at runtime.
- If these views are **NOT** put in their own workspace and there are other views in the workspace, all other views in the workspace or the entire workspace may be hidden if the visibility constraints for `ObjectReportsDataSet` / `SYS_View_CustomRelationsMgmt` are not fulfilled.
- Only ReadOnly versions of configured reports of the type `Custom` that are based on the template `EvaluationReport` should be implemented in a custom object view. You should not implement a ReadWrite version of these reports in the custom object view.

The following information is available:

- Creating a New Workspace for an Object Profile
- Adding Page Views to an Object Profile
- Configuring a Custom Caption and Description for a Page View
- Adding Configured Reports to a Workspace
- Deleting a Page View or Configured Report from a Workspace or Object Profile
- Sequencing the Order of the Workspaces and Configured Reports in the Object Profile

Creating a New Workspace for an Object Profile

When you configure the workspaces for an object profile in a custom object view, all of the workspaces associated with the standard custom view are typically included in the creation of the new custom object view. These workspaces can be modified, as needed, by adding and removing page views.

Once a custom workspace has been configured, you can copy that workspace and reuse it in other custom object views for the same object class. The copied workspace can then be further modified by adding and removing page views.

To create a new workspace in an object profile of a custom object view

1) Right-click the new custom object view  and select **New Workspace**. The new workspace  is displayed below the custom object view node.
2) Define the following attributes, if needed:

- **Caption**: Enter a title to change the caption of the workspace.

- **Visible** attribute. Set to `False` if the workspace should not be displayed in the object profile. This may be useful if you are reusing a custom object view and would like to configure different visibility settings for common workspaces in different custom object views.

3) In the toolbar, click the **Save** button to save your changes. You can now add or remove page views to the custom view.

The following information is available:

Adding Page Views to an Object Profile

You may decide that particular page views have data that is significant to users accessing the object profile and that you therefore want the page view to be easily and immediately accessible to the user community. In this case, you can add one or more page views to the **Attributes Section** of the object profile. Page views can also be assigned directly to the object profile rather than grouped in a workspace. For more information, see the section Adding Page Views to the Object Profile Independent of a Workspace.

To add a page view to a workspace:

1) Right-click the custom workspace and select **Add View**. The **Select View** editor opens. All relevant and permissible page views not yet added to the object view will be displayed. The **Comments** attribute provides a general description about the purpose of the view.

   As an alternative to the **Add View** functionality, you can also right-click an individual page view in a workspace and select **Copy View**. The page view can be pasted to another workspace by right-clicking the workspace and selecting **Paste**.

2) Select one or more page views that you want to display in the workspace and click **OK**. You can simultaneously select multiple page views by holding down the CTRL key while you select views. Repeat this procedure to add all necessary page views to the selected workspace.

3) The page views are added to the workspace. For each page view, you can specify whether a bookmark or express view may be created for the view. In the explorer, click the page view to activate its attribute window and define the following, as needed:

   The **Can Create Bookmark** and **Can Create Express View** attributes will be automatically set to `False` and disabled for views for which bookmarks or express views cannot be set. For example, bookmarks cannot be created for functionalities used for administrative purposes that are available via an administrative user profile or configuration functionalities that are not associated with objects governed by access permissions. Also, some views cannot be bookmarked because they are only accessible via other views. This is the case, for example, with the *Map Objects Page View* (`ITMPM_MapObjectsReport`), which can only be opened from the *Business Support Map Page View* (`ITMPM_Matrix`) or *Business Support Map Report* (`ITMPM_MatrixReport`) for a master plan map.

- **Can Create Bookmark**: Select `True` if a bookmark may be created for the view. Select `False` if a bookmark may not be created for the view. The definition you make here is applicable to all instances in which this view is implemented in Alfabet, regardless of the object class or object class stereotype for which the view is implemented.

Please note that a page view may only be added to a user's splash screen if the **Can Create Bookmark** attribute is set to `True` for the respective view. For more information about the splash screen capability, see the chapter *Creating a Splash Screen As Your Start Page* in the reference manual *Getting Started with Alfabet*.

- **Can Create Express View**: Select `True` if an express view may be created for the view. Select `False` if an express view may not be created for the view. The definition you make here is applicable to all instances in which this view is implemented in Alfabet, regardless of the object class or object class stereotype for which the view is implemented.

4) To sequence the page views and configured reports, click the object view to activate its attribute window. In the **View Sequence** attribute, click the **Browse** button to open the **Sort Entries** editor. In the **Sort Entries** editor, select a page view or configured report in the list and click the up/down arrows in the upper right corner of the dialog. Repeat this for all page views and configured reports until they are listed in the correct sequence. Select the **Sort Alphabetically** checkbox if the page views and configured reports should be sorted alphabetically. Click **OK** to close the editor.

   Please note that the **Attributes Section** is not listed in the editor. The **Attributes Section** is always the first item in the object view. Therefore, the first item listed in the **Sort Entries** editor will be the first page view/configured report/workspace to follow the **Attributes Section**.

5) In the toolbar, click the **Save** button to save your changes.

Configuring a Custom Caption and Description for a Page View

Captions and descriptions may be customized for page views assigned to the protected or custom object view. This is particularly useful for page views targeting object class stereotypes where the names deviate from the standard class names. The custom caption and custom description can be defined for either a selected object class or for all object classes in the context of a specified view scheme. The custom page view captions and descriptions can be translated via the localization capabilities available in Alfabet Expand. For more information about manually or automatically translating the custom strings in the solution configuration, see the section Modifying, Translating and Managing the Vocabularies.

To define a custom caption or description for a page view:

1) Right-click the object view that the page view is assigned to and select **Show Usage** in order to understand which view schemes the object view is assigned to. This is helpful if the object view is assigned to multiple view schemes so that you know which view schemes to define the custom caption/description for.

2) Right-click the page view and select **Specify Caption and Description**. The **Caption and Description for <Page View Name>** editor opens displaying all view schemes configured for your enterprise.

3) In the **Select Class** field, select one of the following:

- If the custom caption or description is relevant to the page view regardless of the object class for which it is configured, select <`Class Independent`>. For example, if you want to configure a custom caption for the *Attachments Page View* (`ObjectAttachments`) that should be displayed for all object

classes in the view schemes you specify, you should select <Class Independent>. In this case, the page view caption and description will be changed for all instances of the page view for the specified class.

- If the configuration should only be relevant for a specific object class in the view scheme, select the relevant object class. If the caption and description is not explicitly specified for an object class stereotype, the caption and description will be applied to all object class stereotypes based on the object class.
- If the configuration should only be relevant for a specific object class stereotype in the view scheme, select the relevant object class stereotype.

4) In the row of the view scheme that is relevant for the object view that you are configuring, enter the custom caption in the **Caption** cell and/or a description in the **Description** cell.

5) Click **OK** to save the changes or click **Cancel** to exit without saving.

6) In the toolbar, click the **Save**  button to save your changes.

Adding Configured Reports to a Workspace

You can add configured reports to a custom workspace. In this case, the report will be available in a manner similar to a page view. The caption of the configured report will be displayed in the interface and users can select the configured report in a workspace as they would a standard Alfabet page view.

Keep the following in mind when assigning configured reports to custom object views:

- Only configured reports of the type `Query` or `Custom` can be assigned to custom object views.
- A configured report with the **Report State** attribute set to `Planned` or `Retired` will be displayed in the custom object view, but an error message will be displayed if the user attempts to open the configured report.
- The text entered in the **Caption** attribute of the configured report is displayed as the title of the configured report in the configured report itself and as the default caption for the configured report in the object view. The report caption displayed in the object view can be overwritten by defining the **Caption** attribute for the respective report object assigned to the object view. This has no effect on the caption displayed in the configured report itself. Therefore, the caption can be different for the object view and the configured report itself.
- The text entered in the **Description** attribute of the configured report is displayed as the page view description in the object view of the report.
- You can provide customized context-sensitive help to your user community in order to understand the content of a configured report that you have configured. The Help file is stored externally and the URL of the help file is defined in the configured report. When the user views the configured report in the Alfabet interface, he/she can click the **Help** button that opens the Alfabet online Help. The context-sensitive help screen that provides access to the context-sensitive online Help will display an option that can be clicked to open the hyperlinked help document.

- The **Can Create Bookmark** and **Can Create Express View** attributes are configured for configured reports in the context of the report configuration in the **Reports** tab. Please note that a configured report may only be added to a user's splash screen if the **Can Create Bookmark** attribute is set to `True` for the respective report. For more information about the splash screen capability, see the chapter *Creating a Splash Screen As Your Start Page* in the reference manual *Getting Started with Alfabet*.

For more information about configuring a configured report, see the chapter Configuring Reports.

Configured reports can also be assigned directly to the object view rather than grouped in a workspace.

To add one or more configured reports to a workspace.

1) Right-click the custom workspace and select **Add Report**. The **Select Report** editor opens.
2) Select one or more configured reports you would like to add to the workspace and click **OK**. You can simultaneously select multiple configured reports by holding down the CTRL key while you select reports.
3) The configured reports are added to the workspace. Repeat this procedure to add all necessary configured reports to the selected workspace.
4) To sequence the page views and configured reports in the workspace, click the workspace to activate its attribute window. In the **View Sequence** attribute, click the **Browse** button to open the **Sort Entries** editor.
5) Select a page view or configured report and place it in the correct sequence by clicking the up/down arrows in the editor. Click **OK** to close the editor and save the definition.
6) Sequence any other relevant reports or page views
7) In the toolbar, click the **Save** button to save your changes.

Deleting a Page View or Configured Report from a Workspace or Object Profile

To delete a page view or configured report from a workspace or object view, right-click the page view or configured report and select **Delete**.

Sequencing the Order of the Workspaces and Configured Reports in the Object Profile

To define the order that the workspaces should appear in the object view, click the object view to activate the attribute window. In the attribute window, click the **Browse** button in the **Attribute Sequence** attribute to open the editor. Move the workspaces and configured reports up and down and place them in the correct sequence by clicking the up/down arrows in the editor. Click **OK** to close the editor and save the definition.

Configuring Conditional Constraints for an Object Profile

Specifying condition constraints in object profiles can provide a more dynamic user experience and ensure that users are seeing and defining the data that is pertinent to the task at hand. The display of the data can be suppressed based on the definition of a value for a property of the associated class. For example, if the **Is SOX Relevant** attribute is set to `True` for the application, then you might want to display the **Business Analysis** workspace in the object profile, but not display if in the application is not SOX-relevant. In this case, you would assign a visibility condition to the workspace.

The following steps are required to configure a visibility condition for an object profile:

- Define a configured report of the type `Query` or `NativeSQL`. The query should search the Alfabet database and return at least one row as a result to fulfill the condition if the **Check Result Type** attribute of the condition is set to `Positive`. When the **Check Result Type** attribute of the condition is set to `Negative` the query return no records. For more information about configuring reports of the type `Query` or `NativeSQL`, see the chapter Configuring Reports.
- Set the **Category** attribute of the report to the relevant category name specified for conditions in the XML object ***UseCaseCategories***. Definition of the XML object ***UseCaseCategories*** is explained in the section Assigning a Category for Specific Functional Use to a Configured Report.
- Create the condition and specify the configured report with the query as described below in the section Creating Conditions to Implement in an Object Profile.
- Specify the visibility of the interface control targeted by the condition is visible via the **Visibility Condition** attribute as described below in the section Assigning Conditions to Interface Controls in the Object Profile.

- Creating Conditions to Implement in an Object Profile
- Assigning Conditions to Interface Controls in the Object Profile

Creating Conditions to Implement in an Object Profile

Conditions can be used in the context of custom editors, object cockpits, and filters in configured reports and can be reused as needed. Conditions for the interface controls in object profiles must be based on queries defined in configured reports. The condition state must be set to `Active` in order to make the condition available in the **Visibility Condition** attribute for interface controls.

To create a condition:

1) Go to the **Presentation** tab and right-click the **Conditions** folder and select **New Condition**. The condition state will be set to `Plan` per default for the new condition.
2) Click the new condition  and define the following in the attribute window:

- **Technical Name**: Enter a name for the condition. It is recommended that the name indicates the meaning of the condition. All active conditions will be displayed in drop-down lists for the **Visibility Condition** attribute.

- **Group**: Enter the name of a new condition folder that you want to store the condition in or select an existing condition folder.

3) In the **Type** field, select `Report` to specify a configured report for the condition. The configured report should already be configured and should be of the type `Query` or `NativeSQL`. The query should search the database and return one row as a result to fulfill the condition. The report is only selectable for conditions if the **Category** attribute of the report is set to `Condition`. Define the following attributes:

- **Report**: Select the configured report that should be executed for the condition. The query should search the Alfabet database and return at least one row as a result to fulfill the condition if the **Check Result Type** attribute is set to `Positive`. When the **Check Result Type** attribute is set to `Negative` the query return no records.
- **Check Result Type**: Select either `Positive` or `Negative` to define what constitutes a fulfilled condition.
- `Positive` means that the condition is fulfilled if the query associated with the configured report delivers a result.
- `Negative` means that the condition is fulfilled if the query associated with the configured report delivers no result.

4) Once the condition is complete and ready to be implemented, the condition state must be set to `Active`. To do so, right-click the condition and select **Set Condition State to 'Active'**. The active condition will be available in the **Visibility Condition**, **Presence Condition**, and **Read-Only Condition** attributes for custom editor fields, filter fields in configured reports, object cockpits, and object profiles.

5) In the toolbar, click the **Save** button to save your changes.

Assigning Conditions to Interface Controls in the Object Profile

You can specify that a workspace or page view is visible or not visible based on whether a condition is fulfilled.

To define conditions for object profiles:

1) Go to the **Presentations** tab and navigate to the object profile that you want to define.
2) Click the workspace, page view, or configured report that you want to apply a visibility condition to.
3) Expand the **Local Settings** section of the attribute window. In the **Visibility Condition** field, click the **Browse** button. In the **Visibility Condition** editor select the configured report in the **Use** column. Click **OK** to save the definition and close the editor.
4) In the toolbar, click the **Save** button to save your changes.

Configuring Object Cockpits for a Custom Object View

An object cockpit provides users with an immediate and transparent overview of data for a selected object. It is as an abbreviated and focused presentation of object data, typically from a particular perspective such as an architecture perspective or a strategy perspective.

Multiple object cockpits can be available per object profile. For example, one object cockpit for the object class Application might display data relevant for understanding the application in the as-is architecture, another cockpit might display data relevant to master planning issues, and a third cockpit might display relevant data for cost and budget issues. The object cockpit is a configuration that is available in addition to the more comprehensive object profile.

An object cockpit can be created and configured in the context of a custom object view. Although an object view may have multiple object cockpits defined, it is possible to hide object cockpits in the context of the view scheme configuration. In this case, only the object cockpits visible for the view scheme will be available to the associated user profile.

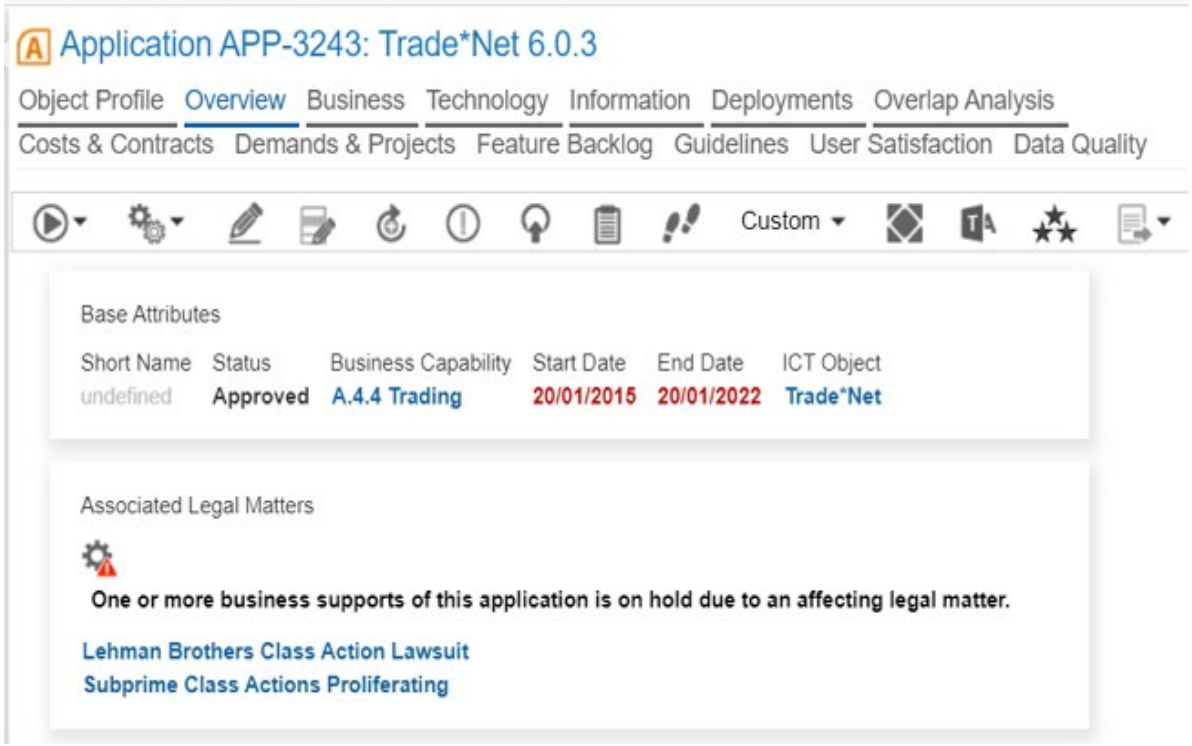


FIGURE: Object profile and multiple object cockpits

There are two possibilities when configuring object cockpits:

- You can use the configuration possibilities available that leverage the browser's layout capability in HTML 5. This configuration is based on a flow panel. This configuration is recommended, and it ensures that object cockpits are not squeezed, stretched, or truncated when the interface is resized. When resizing the interface, the content displayed in object cockpits will be redistributed in the available space and the content will not be cut-off. The flow panel ensures that a configured view or set of views will be bumped down and placed below the preceding view or set of views. Business graphics embedded in an object cockpit will be automatically scaled down in size to fit in the available screen space of the device the user is using. Pictures and icons can be implemented as placeholders that link to views or reports. This is especially useful for large reports for which only a

portion of the report is displayed in the object cockpit. This method is described in the section Configuring the Object Cockpit By Means of a Flow Panel.

- You can use the configuration possibilities that were available in releases of Alfabet prior to the introduction of HTML 5-based user interface and user interactions. This configuration is based on a table layout panel. This method is available for reasons of backward compatibility. However, the table layout method is not recommended because the data in the object cockpit may be squeezed, stretched, or truncated when the interface is resized. This method is described in the section Configuring the Object Cockpit By Means of a Table Grid.

Object cockpits can also be implemented in a configured report. In this case, a filter field is displayed in the configured report that allows the user to select an object in order to view its data in an object cockpit. Please note that the object cockpits available in a configured report will depend on the view scheme configuration associated with the user profile that the user is accessing the configured report with. For more information about implementing an object cockpit in a configured report, see the Creating a Configured Report Displaying an Object Profile or Object Cockpit in the chapter Configuring Reports.

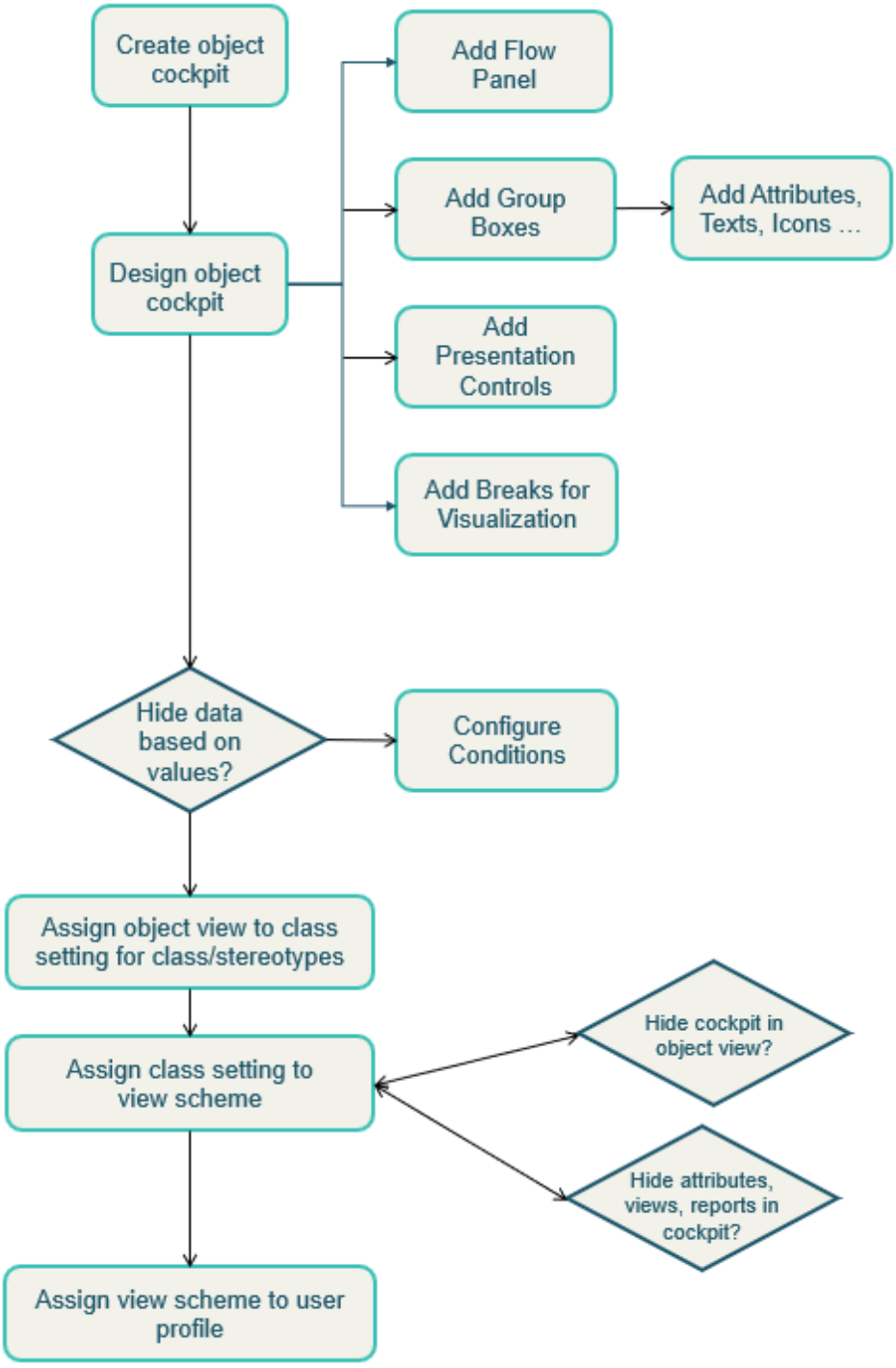
The following information is available:

- Conceptualizing the Object Cockpit
- Creating an Object Cockpit
- Configuring the Object Cockpit By Means of a Flow Panel
- Adding Group Boxes to Display Attributes
- Adding Static Text to the Object Cockpit
- Adding a Property to the Object Cockpit
- Adding a Generic Attribute to the Object Cockpit
- Adding an Indicator to the Object Cockpit
- Adding a Check Entry to the Object Cockpit
- Adding a Query to the Object Cockpit
- Adding a URL to the Object Cockpit
- Adding a Dynamic Web Link to the Object Cockpit
- Adding Presentation Objects to Display Page Views and Configured Reports
- Adding a Floating Group Box to Contain Presentation Objects
- Adding a Floating Group Box as a Filter Panel for Reports
- Adding a Presentation Object for a Different Target Object
- Adding Placeholders for Page Views and Configured Reports
- Adding Workflow, Assignment, Microsoft Teams Meeting Information and Object Validation Information to the Object Cockpit
- Adding Icons to the Object Cockpit
- Adding Embedded HTML to the Object Cockpit

Conceptualizing the Object Cockpit

Before you begin to define the object cockpit, you should conceptualize what information should be displayed in the object cockpit. You can define multiple object cockpits for a protected or custom object view. Each object cockpit can be configured to be relevant to a specific perspective (for example, a business perspective, technical perspective, financial perspective, etc.). This allows you to create object cockpits with a manageable amount of data that users can quickly and easily comprehend.

The following image illustrates the steps to configure an object cockpit:



Because the custom object view may be implemented in multiple class settings, you may have many object cockpits that are relevant for the custom object view. During the context of configuring a view scheme, you can specify which of the object cockpits should be relevant for the class setting implemented in the user profile. Furthermore, if you only want users to view the object cockpits you have configured, you can hide the standard or custom object profile from the user interface.

To conceptualize the object cockpit, you should consider the following questions in order to develop a general idea of the data that should be available to users working with the object cockpit:

- Which object class properties should be displayed in the object cockpit? Which of these properties are particularly important and require special highlighting to attract the user's attention? Options included providing captions, changing the font size or color of a cell caption, adding borders to table cells, or configuring color or picture instructions that add color or icons to a table cell based on the object class property value.
- Should check entries be included to control the value defined for a property? Check entries are based on queries configured to find and flag data with a message and/or icon. For example, a check entry could be configured to ensure that the object class property `ICT Object` for the object class `Application` has a value defined. If no ICT object is defined, a warning symbol or message can be displayed informing the user that this data is missing for the application.
- Should referenced properties be displayed that must be fetched via a query? For example, a query could be defined to display all projects for which an application has been defined in the as-is architecture.
- Should indicators be displayed in the object cockpit?
- Should static HTML be displayed in the object cockpit? Should each property displayed in the overview section have a corresponding table cell displaying a caption or explanatory text? Any static HTML code can be included in the object cockpit including, for example, images, links, and forms.
- Should the object cockpit display standard Alfabet views and configured reports? If so, which views and reports should be included in the object cockpit? Should users be able to navigate to the views or perhaps to other views that might be relevant to the user's context? Please note that in order to make the amount of data in the object cockpit manageable, it is recommended that you not include more than 10 views in an object cockpit.
- Should the data in the page views and configured reports be displayed or should an icon be displayed that allows the user to navigate to the full-view of the page view or report. Please note that a variety of icons are available that symbolize the content of the page view or configured report are available. For example, icons are available with a diagram symbol, matrix symbol, portfolio symbol, etc.

Creating an Object Cockpit

You can create multiple object cockpits for a protected object view or a custom object view. Each object cockpit can be differently designed, as needed.

Please note that new object cockpits will typically be created with a Flow Panel interface control rather than a Table Layout interface control. This is determined via the XML attribute `UseGUISchemeStyleForNewCockpits` in the XML object ***SolutionOptions***, which is set to `False` per default. When set to `False`, a new cockpit is created with a flow panel and the **Ignore GUI Scheme Common Container Skin** attribute of the flow panel is automatically set to `True`.

To create a new object cockpit:

1) Navigate to the protected object view or custom object view that you want to create an object cockpit for. Right-click the object view and select **Create Object Cockpit**. The new object cockpit s displayed below the selected object view node.

You can copy an existing object cockpit as the basis to create a new one and paste it to the object view of another object view configured for the same object class. To create a new object cockpit based on a copy of an existing object cockpit, right-click the protected or custom object view and select **Create Object Cockpit**. Right-click the existing object cockpit that you want to copy and select **Copy**. Right-click the new object cockpit and select **Paste**. You can modify the copied object cockpit as needed.

2) Click the new object cockpit below the object view node. The attribute window is displayed in the right pane. The **Class Name** attribute displays the object class that the object cockpit is based on. Define the following attributes, as needed:

- **Name**: Edit the automatically-generated technical name, if necessary. This name is not displayed in the Alfabet interface.

A validation mechanism checks for correct syntax when defining a technical name. The technical name of a configuration object may not begin with an empty space nor include any of the following special characters: **\ / * ? " > < | ':**

Furthermore, please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.

Furthermore, if the technical name of a configuration object is changed, the name will be correctly updated in other configuration objects referencing the changed object during design time. Please note that the name of a changed configuration object will not be updated in guide pages nor if the guide pages reference a configuration object listed in the **Show Usage** functionality. If you plan to change the name of a configuration object, the reference in the guide page should be changed prior to changing the configuration object name.

- **Caption**: Enter the caption to display on the object cockpit tab in the user interface.
- **Show Toolbar**: Define the visibility of the standard toolbar for the selected object cockpit. The standard toolbar will allow users to access basic functionalities available via such buttons as the **Mark as Reviewed** button and **Notify Authorized User** button. Depending on the object class that you are working with, other buttons such as the **Edit** button, **Change Object State** button, **Workflow** button, etc. may be available. Select `True` if the standard toolbar should be displayed for the object cockpit. Select `False` if the standard toolbar should not be displayed for the object cockpit.

Please note that the same toolbar is used in both the object profile and all object cockpits defined for the object class for which **Show Toolbar** is set to `True`.

3) Right-click the object cockpit and select **Design Object Cockpit**. The design editor is displayed in the center pane. The new object cockpit is automatically filled with the Flow Panel interface control.

4) Click anywhere in the Flow Panel interface control to activate the attribute window. If the flow panel is selected, its handles will be displayed.

5) Set the **Dock** attribute to `Fill` to ensure that the flow panel is docked in the object cockpit. You can now add group boxes containing attributes, presentation objects containing page views or configured reports, and breaks to structure the content:

A validation mechanism checks for inconsistencies in the layout and docking definition of interface controls in guide views, configured reports, and other configured views. Please note that an error message will be displayed when the user attempts to open an incorrectly configured view if two or more interface controls have no docking defined in the view. The error message will provide detailed information about the faulty configuration.

The **Height** and **Width** attributes are irrelevant for the configuration of the object cockpit. The object cockpit will be automatically sized to fill the available space in the user interface.

6) Click the  button to save the new object cockpit. You must first add group boxes or presentation objects to the flow panel in order to view the object cockpit in the Alfabet Web interface. You can now add interface controls to the new object cockpit.

Configuring the Object Cockpit By Means of a Flow Panel

You can use the configuration possibilities available that leverage the browser's layout capability in HTML 5. This configuration is based on a flow panel and has the advantage that object cockpits are not squeezed, stretched, or truncated when the interface is resized. Group boxes containing attributes and presentation objects (views and reports) will be dynamically positioned according to the current width of the Alfabet Web interface. When resizing the interface, the content displayed in object cockpits will be redistributed in the available space and the content will not be cut-off. The flow panel ensures that a configured view or set of views will be bumped down and placed below the preceding view or set of views. Business graphics embedded in an object cockpit will be automatically scaled down in size to fit in the available screen space of the device the user is using. Pictures and icons can be implemented as a placeholder that link to views or reports. This is especially useful for large reports for which only a portion of the report is displayed in the object cockpit.

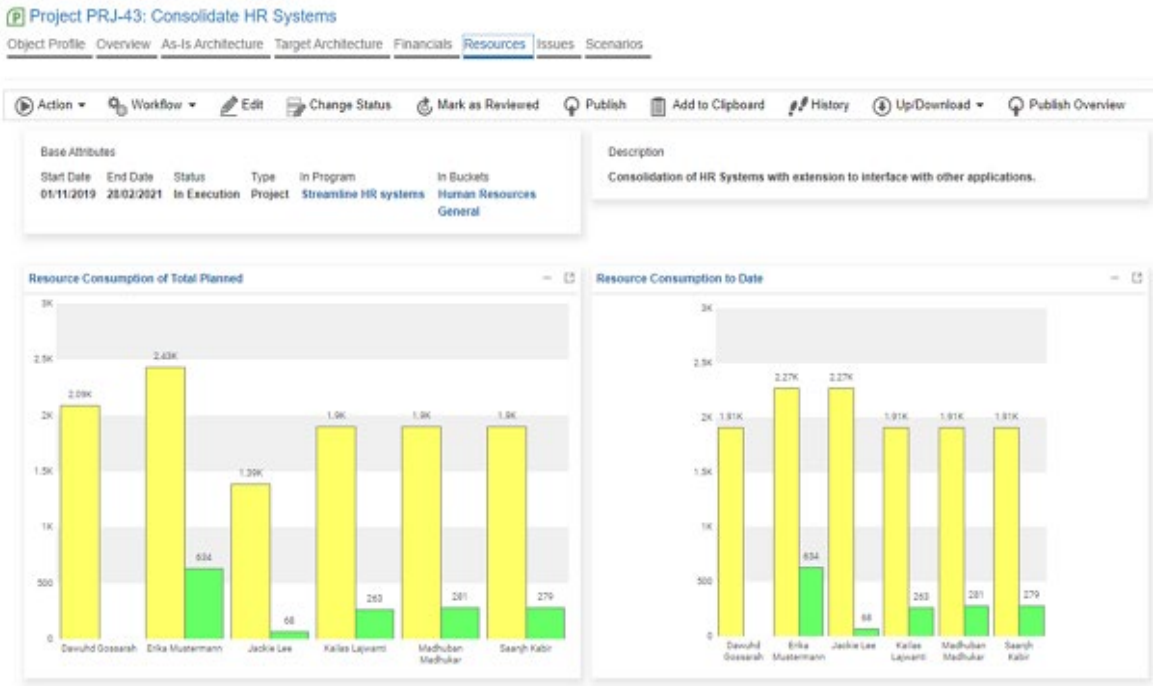


FIGURE: Object cockpit based on a flow panel

The placement of the interface controls in the object cockpit designer as well as their alignment and spacing horizontally is not relevant to the displayed output. The actual display in the Alfabet interface at runtime will be determined by the sequence and linebreaks that you define in the object cockpit configuration. However, it is recommended that you position the interface controls in the object cockpit designer in the order that you want them to be displayed in order to make the conceptualization and design of the object cockpit easier.

Please note that font and hyperlink colors displayed in the object cockpit are determined by the configuration of the GUI schemes implemented in your enterprise. For more information, see the section Configuring GUI Scheme Definitions for the Alfabet Interface.

When you design the layout of the object cockpit, you must consider the following factors:

- **The order in which you want information to be displayed on the interface.** All interface controls that you place in the object cockpit must be sequenced in order to determine their order from left to right and top to bottom on the interface. The interface controls are assigned a tab order in the last step of the design process but it is useful to have a general idea of what this order will be when you begin to configure the object cockpit.
- **The width in pixel of the individual interface controls (group boxes and presentation objects).** It is recommended that Group Box and Presentation Object interface controls have a base width of either 400, 600, or1200 pixel. The sum of the width of multiple interface controls (group boxes and presentation objects) that shall be displayed in one row should not exceed 1200 pixel. For example, if you want to display three reports in the same row, you could define each presentation object to have a width of 400 pixel and conclude the row with a break <`break`> element. If the screen size is smaller than 1200 pixel or the user manually reduces the width of the screen is less than 1200 pixel, the last report preceding the break <`break`> element that doesn't fit in the user interface will be bumped down to the next row. If an interface control (group box or presentation object) is wider than the available screen space, the content will be truncated.

- **The height in pixel of the individual group boxes.** You can either define a static height for a group box or allow the height to be automatically sized based on the content in the group box. If you define a static height, the group box will retain the defined size regardless of the amount of content in it and a scrollbar will be added to the group box if the content exceeds the height. If you want the group box to automatically adjust its height based on the amount of content, then you must set the Automatic Sizing `True`. In this case, any height definition will be ignored. Please note, however, that if you specify that the height of one group box in a row is greater than the height of the other group boxes (or you define some but not all group boxes to be automatically sized), empty space may result in the row. This is because the bottom edge of the largest group box will be used to determine the top edge of the next row of interface controls.
- **The height in pixel of the individual presentation objects.** All Presentation Object interface controls (reports or page views) require a defined height and width. The height of a presentation object may be configured to be automatically sized based on its content. A zoom functionality will be available automatically for views with business graphics.

The following provides best practice recommendations to configure object cockpits optimized for a resolution of1920 x 1080 (full HD) down to 1200 x 786 (HD ready) which is common for current monitors, projectors, and laptops.

- A Flow Panel interface control is automatically placed in a new object cockpit. The Flow Panel interface control allows the resizing mechanisms in the browser to be leveraged and is required in order to configure an object cockpit that is optimized for HTML 5. A Flow Panel interface control should be the base of the object cockpit. The **Dock** attribute should be set to `Fill`. A validation mechanism checks for inconsistencies in the layout and docking definition of interface controls in object cockpits, configured reports, and other configured views. Please note that an error message will be displayed when the user attempts to open an incorrectly configured view if two or more interface controls have no docking defined in the view. The error message will provide detailed information about the faulty configuration.

  A validation mechanism checks for inconsistencies in the layout and docking definition of interface controls in guide views, configured reports, and other configured views. Please note that an error message will be displayed when the user attempts to open an incorrectly configured view if two or more interface controls have no docking defined in the view. The error message will provide detailed information about the faulty configuration.

- Static Text, Icon, and HTML Content and Value Control interface controls (properties, indicators, check entries, queries, URLs,etc.) can be added to the Flow Panel interface control or Group Box interface control. It is recommended to group such interface controls in group boxes in order to better control the visual layout.
- Multiple Group Box interface controls can be added to the flow panel to visually structure the data. The content placed inside a group box is arranged by the browser vertically. The size of the interface controls inside the group box will be honored. If the content in the group box does not fit in the space available, scroll bars will be automatically added.
- Linebreaks should be configured to create a linefeed in the layout in order to visually separate attribute groups from presentation objects (page views and reports) or to thematically structure views in sections. The content before each linebreak should have the same height. Otherwise the content will not be horizontally aligned.
- You can define color and styles for all interface controls. Styles can be configured for all object cockpits via various GUI scheme attributes in order to create a coherent look-and-feel for your

enterprise. Please note that color definition will be ignored if the interface control represents a referenced property (hyperlink) that can be clicked in order to navigate. In this case, the color will be displayed as a blue hyperlink. For more information about configuring styles in the object cockpit, see the section Configuring Styles and GUI Scheme Settings for Object Cockpits.

- You can copy existing configurations of interface controls (such as group boxes, value controls, or presentation objects from an existing object cockpit to the one you are currently designing.

The following information is available:

- Adding Group Boxes to Display Attributes
- Adding Static Text to the Object Cockpit
- Adding a Property to the Object Cockpit
- Adding a Generic Attribute to the Object Cockpit
- Adding an Indicator to the Object Cockpit
- Adding a Check Entry to the Object Cockpit
- Adding a Query to the Object Cockpit
- Adding a URL to the Object Cockpit
- Adding a Dynamic Web Link to the Object Cockpit
- Adding Presentation Objects to Display Page Views and Configured Reports
- Adding a Floating Group Box to Contain Presentation Objects
- Adding a Floating Group Box as a Filter Panel for Reports
- Adding a Presentation Object for a Different Target Object
- Adding Placeholders for Page Views and Configured Reports
- Adding Workflow, Assignment, Microsoft Teams Meeting Information and Object Validation Information to the Object Cockpit
- Adding Icons to the Object Cockpit
- Adding Embedded HTML to the Object Cockpit
- Configuring Conditional Restraints in the Object Cockpit
- Creating Conditions to Implement in an Object Cockpit
- Assigning Conditions to Interface Controls in the Object Cockpit
- Adding Linebreaks to Visually Structure the Information in the Object Cockpit
- Configuring Styles and GUI Scheme Settings for Object Cockpits
- Sequencing the Content in the Flow Panel
- Configuring Drill-Down Navigation in the Object Cockpit

Adding Group Boxes to Display Attributes

A Group Box interface control allows you to structure and display attributes that shall be contained in the group box. You can create one Group Box interface control that contains all attributes that you want to display, or you can create multiple Group Box interface controls with different sets of attributes.

A Group Box interface control can be used in other contexts:

- As a collapsible container to group two or more Presentation Object interface controls such as a set of widget reports. For more information, see Adding a Floating Group Box to Contain Presentation Objects.
- As a collapsible container to group a set of filters targeting Presentation object interface controls that implement configured reports. In this case, the filter settings will be applied to all relevant configured reports in the object cockpit. For more information, see the section Adding a Floating Group Box as a Filter Panel for Reports.

The content placed inside a group box is arranged by the browser vertically. The attribute data is structured vertically as two columns, one containing the attribute caption and the other, the attribute value.The size of the interface controls inside the group box will be honored. If the content in the group box does not fit in the space available, scroll bars will be automatically added. The user can collapse the group box by clicking the collapse (-) symbol and expand the group box by clicking the expand (+) symbol. The collapsed group box will reduce to the width and height of the group box caption.

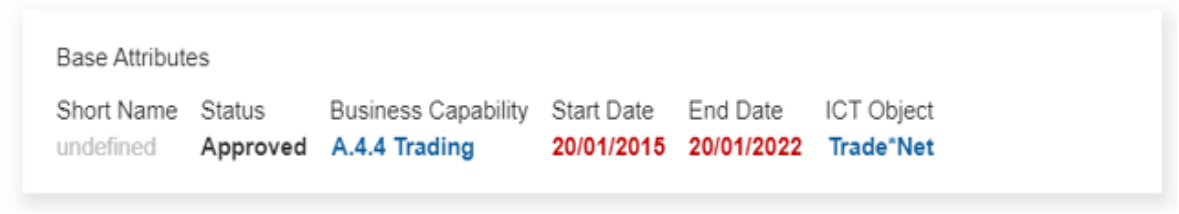


FIGURE: Group box with attributes

Please consider the following when conceptualizing the group box and the data that shall be displayed in it:

- The width that you define for a Group Box interface control will influence the number of Group Box interface controls that may be displayed horizontally in a row. Therefore, you should consider how many Group Box interface controls you would ideally like to display horizontally in a row and the width of each. It is recommended that Group Box interface controls have a base width of either 400, 600, or 1200 pixel. The sum of the width of multiple interface controls (group boxes and presentation objects) that shall be displayed in one row should not exceed 1200 pixel. For example, if you want to display three group boxes in the same row, you could define each group box to have a width of 400 pixel and conclude the row with a break <`break`> element. If the screen size is smaller than 1200 pixel or the user manually reduces the width of the screen is less than 1200 pixel, the last group box preceding the break <`break`> element that doesn't fit in the user interface will be bumped down to the next row. If an interface control (group box or presentation object) is wider than the available screen space, the content will be truncated.
- The height of a group box can be automatically sized based on the content in the group box or you can specify a static height for the group box. If the content of the group box exceeds the maximum height definition, a scrollbar will be added so that the user can scroll to see the complete content in the group box. In order to better control the visual layout and ensure an object cockpit with reduced white space, it is recommended that you configured a maximum height for group boxes.

- If multiple group boxes shall be visualized in the object cockpit:

- Break elements (<break>) may be configured to create a linefeed between multiple group boxes in order to visually structure the group boxes in sections. For example, if 4 group boxes are displayed in the object cockpit, you may configure all of them to have a width of 600 pixel and place a line break between the second and third group box. This would result in the first and second group boxes being horizontally aligned in a row and the third and fourth group box being horizontally aligned on the next row.

- The height that you define for a Group Box interface control will influence the horizontal alignment of the group boxes. If multiple group boxes shall be visualized in the object cockpit, all Group Box interface controls located before a line break should have the same height, otherwise the content will not be horizontally aligned and there will be excessive white space in the object cockpit.

- It is recommended that multiple Group Box interface controls have the same width. If Group Box interface controls have different widths (600 pixel and 1200 pixel), line wrapping may be an issue. If different widths cannot be avoided, all Group Box interface controls located before a line break should be sequenced based on their size as follows: 200 pixel, 600 pixel, 600 pixel, 1200 pixel. This provides an adequate display on both, high and low resolution cockpits.

- Static Text, Icon, and HTML Content and Value Control interface controls (properties, indicators, check entries, queries, URLs,etc.) can be added to the Group Box interface control. The attribute data is structured vertically as two columns, one containing the attribute caption and the other, the attribute value. Please consider the following if a high number of interface controls are added to a group box:

- Multiple Group Box interface controls should be configured if a high number of interface controls shall be included in the object cockpit. The content should be structured to make it easy for the user to understand how the content is organized.

- Attributes which are guaranteed to have one line maximum should not be mixed up with multi-line attributes in one Group Box interface control. The attributes that are guaranteed to consist of only one line should be grouped in one Group Box interface control and attributes that require multiple lines of text should be grouped in another Group Box interface control.

- Break elements (<break>) can be used in the group box to create a linefeed and separator line to improve the layout of data in the group box. For example, you could group a set of attributes with basic information in the group box, add a break element (<break>) after the basic data attributes, and add another set of attributes about risk assessment to follow the break. The **Tab Index** attribute must specify the correct sequence for all attributes and breaks in the group box.

When an object cockpit is published to a DOC or PDF file, the user can select which group boxes and page views to include in the DOC or PDF file as well as the orientation of the each group box and view in the **Object Document** editor. During the publication process, group boxes will be converted to tables, thus providing a structured visualization of the data in the DOC and PDF file. Linebreaks in group boxes will be removed from the file when an object cockpit is published to a DOC or PDF file. For more information about publishing an object cockpit to a PDF or DOC file, see the section *Exporting Data* in the reference manual *Getting Started with Alfabet*.

To add a Group Box interface control to the object cockpit:

1) In the toolbar of the object cockpit designer, click the **Group Box** button and click in the flow panel. You can add multiple Group Box interface controls. If you only define one Group Box

interface control and it is configured to be very wide, its display may be truncated if the width of the object cockpit is smaller than anticipated or is decreased by the user.

2) Click the Group Box interface control to ensure that its attribute window is open. Define the following attributes, as needed:

- **Sub-Type**: Select `Attributes`.
- **Caption**: Define a caption for the group box. This attribute may not be left empty.

The font color and font style of the caption of group boxes is specified via the **Attributes Group Caption Font** attribute of the GUI scheme. For more information about defining the styling of the object cockpit, see the sections Configuring Styles and GUI Scheme Settings for Object Cockpits.

3) Specify the height of the group box:

- To specify that the height of the group box automatically adjusts to the size of its contents, set the **Automatic Sizing** attribute to `True`. Please consider the following:
- The **Automatic Sizing** attribute is only visible in the attribute window if the **Use Height as Max. Height** attribute is set to `False`.
- If the Group Box interface control contains a Static Text interface control, the **Automatic Sizing** attribute should also be set to `True` for the Static Text interface control.
- To specify that the height of the group box maintains a static size, regardless of the content of the group box, specify in pixel the size of the group box in the **Height** attribute. If the content of the group box exceeds the maximum height definition, a scrollbar will be added so that the user can scroll to see the complete content in the group box.
- **Use Height as Max. Height**: Select `True` if the maximum height of the group box shall not exceed the defined height. If the content of the group box exceeds the maximum height definition, a scrollbar will be added so that the user can scroll to see the complete content in the group box. If the **Automatic Sizing** attribute is set to `True`, the **Use Height as Max. Height** attribute will be ignored and the height will be automatically sized.

4) Specify the width of the group box:

- **Width**: Define the width in pixel for the group box. Any group box with a defined width that is greater than the defined width of the object cockpit will be truncated and not all content will be visible.
- **Item Max. Width**: Enter the maximum width of the interface controls in the group box. Enter -1 if there shall be no limitation on the maximum width.
- **Item Min. Width**: Enter the minimum width of the interface controls in the group box. Enter -1 if there shall be no limitation on the minimum width.

5) Specify the placement of the group box. You can do this via drag-and-drop of the attribute box or by entering values in pixel in the **Top** and **Left** attributes.

6) Click the **Save**  button to save the new group box.

7) Add the relevant Static Text, Icon, and HTML Content and Value Control interface controls to the group box as described in the following sections.

8) Add a break (<break>) element between groups of attributes in the group box to add a linefeed and separator, as needed. For more information about configuring a linebreak, see the section Adding Linebreaks to Visually Structure the Information in the Object Cockpit.

9) Define the styling of the group box as needed. For more information, see the section Configuring Styles and GUI Scheme Settings for Object Cockpits.

10) Specify the **Tab Index** attribute for all Static Text, Icon, and HTML Content and Value Control interface controls and break elements (<break>) in the Group Box interface control in order to specify the correct sequence and layout in the group box. The sequence should begin with 1. The sequence that you specify is the order in which the interface controls and break elements are displayed as well as the order to navigate through the group box via the TAB key.

Adding Static Text to the Object Cockpit

A Static Text interface control can be add to the Flow Panel interface control or Group Box interface control to visualize text such as instruction or guidelines to provide the user community with specific information.

To add and configure a Static Text interface control:

1) In the toolbar of the object cockpit designer, click the **Static Text** button and click in the Flow Panel interface control or Group Box interface control.

2) Click the Static Text interface control to activate its attribute window. Define the following attributes as needed: The **Control Type** attribute is automatically set to StaticText.

- **Control Type**: This attribute is automatically set to StaticText.
- **Caption**: Enter the text that should be displayed for the Static Text interface control. This could be a short text such as a property caption or comprehensive information.

3) To specify styles such as the font and color for the Static Text interface control, set the **User Style** attribute to True. Expand the **Style** section of the attribute window and specify the relevant styles as needed.

4) In the toolbar, click the **Save** button to save your changes.

Adding a Property to the Object Cockpit

The Value Control interface control allows you to add standard or custom properties that have been configured for the object class that the object cockpit has been created for. If a property should be highlighted with a color or icon if it has a specified value, then you can configure color or picture instructions for the Value Control interface control.

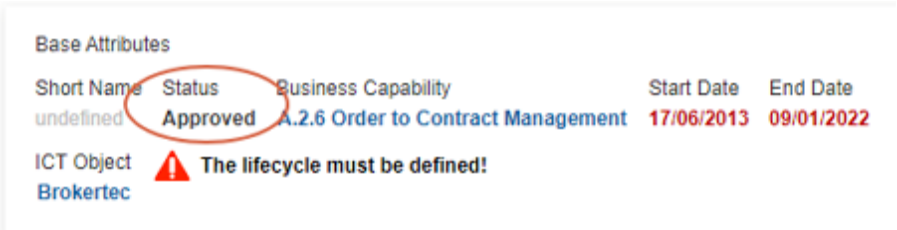


FIGURE: Status property in a group box

To add a standard or custom object class property via a Value Control interface control to the object cockpit:

1) In the toolbar of the object cockpit designer, click the **Value Control** button and click in the Group Box interface control.
2) Click the Value Control interface control to activate its attribute window. The **Control Type** attribute is automatically set to `ValueControl`. Define the following properties, as needed:

- **Sub-Type**: Select `Property`.
- **Property** attribute (in the **Value Definition** section): Select the standard or custom object class property that should be associated with the Value Control interface control.

  Please note that if the property that you are defining is of the type `Reference`, you must specify the **Show Properties** attribute in the **Value Definition** section, otherwise a defined value for the property will not be displayed in the object cockpit.

- **Caption**: Define a caption for the Value Control interface control.
- **Inline Navigation**: Select `True` if users should be able to navigate to an object referenced by the object class property. The object class property will be displayed as a blue hyperlink.
- **Filters** attribute (in the **Value Definition** section): Optional. Define picture instructions or color instructions to highlight the object class property with a specified color or icon based on the object class property value. If picture instructions or color instructions are to be defined for the object class property, enter the instructions for the filter. For picture instructions, the icon specified first must be imported to the icon gallery. The icon can be 22x22, 30x30, or an icon in the free size icon gallery. For more information about defining instructions, see the section Using Instructions to Format the Results of an Alfabet or Native SQL Query in the chapter Defining Queries.
- The picture can be defined by writing the following instruction to the **Filters** attribute: `PictureAssignment(ConditionString,"Argument", IconName);` For example, to define picture instructions for a `Boolean` value for a property, you might enter the following instructions:

  ```
  PictureAssignment(EqualTo,False,Bubble_Red);
  PictureAssignment(EqualTo,True,Bubble_Green);
  ```

- The coloring can be defined by adding the following instruction to the Alfabet query: `ColorAssignment(ConditionString,"Argument",TextColor,CellColor);` For example, to define a color filter for object state values, you might enter the following instructions:

  ```
  ColorAssignment(EqualTo,Plan,Black,Silver);
  ColorAssignment(EqualTo,Active,Yellow,Green);
  ColorAssignment(EqualTo,Retired,White,Red);
  ```

- **Hint**: Optional. Enter text for a tooltip. The tooltip will be displayed when the user points to the attribute in the object cockpit.
- **Enable HTML Content**: If the **HTML Content** attribute is set to `True` for the Memo interface control in the editor where the property is defined, the attribute will be automatically set to `True` and the HTML formatting for the property will be displayed in the object cockpit. If the **Read-Only** attribute is set to `True` for the Value Control interface control, the HTML editor will also be available to edit the property directly in the user interface of the object cockpit. The **Enable HTML Content** attribute can be set to `False` if HTML format shall not be displayed.

- **Read-Only**: Set to `True` if inline editing shall be disabled for the property in the context of the object cockpit. Set to `False` if inline editing shall be permissible for the property in the context of the object cockpit. For more information about the configuration required to allow Inline editing of scalar and reference properties, see the section Configuring Inline Editing of Attributes in the Object View.

  Please note the following regarding the inline editing functionality:

  - Only scalar and reference properties that are editable in the editor/wizard available in the object profile or object cockpit can be edited via inline editing.
  - All unique constraints defined for the class as well as any post-conditions configured for the wizard associated with the object profile/object cockpit will be applied to the data entered.

    Please note that data is saved to the Alfabet database prior to the post-condition being validated. Therefore, data entered via inline editing will be saved even if the post-condition is not fulfilled. This is because the data must be available in the Alfabet database in order to be evaluated for the post-condition. If the data entered does not fulfill the post-condition, the configured warning message will be displayed explaining that the input must be corrected in order to fulfill the post-condition.

  - Inline editing may be limited or prevented if the syntax of the post-condition is not correct. In this case, an error will be displayed if the user tries to open the wizard.

3) To specify styles such as the background color, borders, padding, margins, and font of the value displayed for the Value Control interface control, set the **Use Style** attribute to `True`. Expand the **Style** section of the attribute window and specify the relevant styles as needed.

   The font color and font style of the caption of Value Control interface control is specified via the **Value Control Caption Font** attribute of the GUI scheme. For more information about defining the styling of the object cockpit, see the sections Configuring Styles and GUI Scheme Settings for Object Cockpits.

4) In the toolbar, click the **Save**  button to save your changes.

Adding a Generic Attribute to the Object Cockpit

This section is only relevant for object cockpits configured for the object classes `Application`, `Component`, `Deployment`, `Deployment Element`, `Standard Platform`, `Standard Platform Element`, `Stack`, `Stack Element`, and `Stack Item`.

If generic attributes have been created for an object in the *Generic Attributes Page View* (`ObjectGenericAttributes`), you can configure the object cockpit to display the generic attributes and their values. For more information about working with generic attributes, see the section Implementing the Generic Attribute Concept Instead of Custom Properties in the chapter Configuring the Class Model.

To display generic attributes in the object cockpit:

1) In the toolbar of the object cockpit designer, click the **Value Control** button and click in the Group Box interface control.

2) Click the Value Control interface control to activate its attribute window. The **Control Type** attribute is automatically set to `ValueControl`. Define the following properties, as needed:

- **Sub-Type**: Select `AttributeQuery`.
- **Query**: Enter a query to find the generic attributes for the object class. The query should be specified for the class `GenericAttribute` and return the relevant `SHOW` and `SORT` properties.

The following example displays an Alfabet query specified to display generic attributes for a configured object profile/object cockpit:

```
ALFABET_QUERY_500

FIND GenericAttribute

WHERE(AND GenericAttribute.Owner CONTAINS:BASE)

QUERY_XML

   <QueryDef></QueryDef>

      <ShowProperty Type="Property"
      ClassName="GenericAttribute" Name="Name" />

      <ShowProperty Type="Property"
      ClassName="GenericAttribute" Name="Value" />

      <ShowProperty Type="Property"
      ClassName="GenericAttribute" Name="Type" />

      <SortProperty Type="Property"
      ClassName="GenericAttribute" Name="Name" />
```

- **Hint**: Optional. Enter text for a tooltip. The tooltip will be displayed when the user points to the attribute in the object cockpit.
- **Caption**: Not relevant for a Value Control interface control of the sub-type `AttributeQuery`.

3) To specify styles such as the background color, borders, padding, margins, and font of the value displayed for the Value Control interface control, set the **Use Style** attribute to `True`. Expand the **Style** section of the attribute window and specify the relevant styles as needed.

4) In the toolbar, click the **Save** button to save your changes.

Adding an Indicator to the Object Cockpit

An indicator type/evaluation type can be configured for a Value Control interface control. If the indicator type has been evaluated for the object, the indicator will be displayed in the object cockpit. You can configure the display of the numeric value and/or an icon for the indicator.

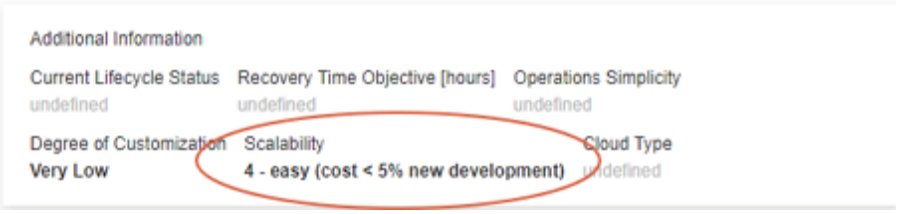


FIGURE: Numeric indicator in a group box

Evaluation types and their indicator types must first be configured in the **Evaluations and Portfolios** functionality in the **Configuration** module in order to see results in this report. The evaluation types must be assigned to the respective object class in the **Class Configuration** functionality in the **Configuration** module. For more information about the configuration of evaluation types, see the chapter *Configuring Evaluations, Prioritization Schemes, and Portfolios* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.

Please note that the indicators configured for project evaluation types are not stored in the Alfabet database and therefore cannot be displayed in object cockpits. If project indicators need to be displayed in the object cockpit, please configure a Value Control interface control with the **Sub-Type** attribute set to `ValueByQuery` to fetch the relevant data. For more information about configuring a query, see the section Adding a Query to the Object Cockpit. For more information regarding project evaluation types, see the section *Configuring Reference and Evaluation Data Required for Project Management* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.

To add an indicator via a Value Control interface control to the object cockpit:

1) In the toolbar of the object cockpit designer, click the **Value Control** button and click in the Flow Panel or Group Box interface control.
2) Click the Value Control interface control to activate its attribute window. The **Control Type** attribute is automatically set to `ValueControl`. Define the following attributes, as needed:

- **Sub-Type** attribute, select `Indicator`.
- **Indicator** attribute´(in the **Value Definition** section); Enter the name of the evaluation type that the indicator type is assigned to and the name of the indicator type. You should use the syntax: `<EvaluationType>/<IndicatorType>`. Please note that the names of the evaluation type and indicator type must be spelled correctly.
- **Display Style**: Select one of the following to determine how the indicator is displayed:
- `Default`: Displays the default visualization for the indicator.
- `Text`: Displays the semantic value of the indicator. If the indicator's semantic value is empty, the numeric value will be displayed.
- `Icon`: Displays the icon representing the indicator.
- `IconAndText`: Displays first, the icon representing the indicator and second, the semantic value of the indicator. If the indicator's semantic value is empty, the numeric value will be displayed.
- `TextAndIcon`: Displays first, the semantic value of the indicator and second, the icon representing the indicator. If the indicator's semantic value is empty, the numeric value will be displayed.
- **Caption**: Define a caption for the Value Control interface control.

- **Hint**: Optional. Enter text for a tooltip. The tooltip will be displayed when the user points to the attribute in the object cockpit.

3) To specify styles such as the background color, borders, padding, margins, and font of the value displayed for the Value Control interface control, set the **Use Style** attribute to `True`. Expand the **Style** section of the attribute window and specify the relevant styles as needed.

The font color and font style of the caption of Value Control interface control is specified via the **Value Control Caption Font** attribute of the GUI scheme. For more information about defining the styling of the object cockpit, see the sections Configuring Styles and GUI Scheme Settings for Object Cockpits.

4) In the toolbar, click the **Save**  button to save your changes.

Adding a Check Entry to the Object Cockpit

A check entry is a mechanism based on an Alfabet query or native SQL query that reviews the definition of a property for a specified standard or custom object class property for the object displayed in the object cockpit. If the query finds results, a configured icon and message can be displayed providing the user with information about the check.

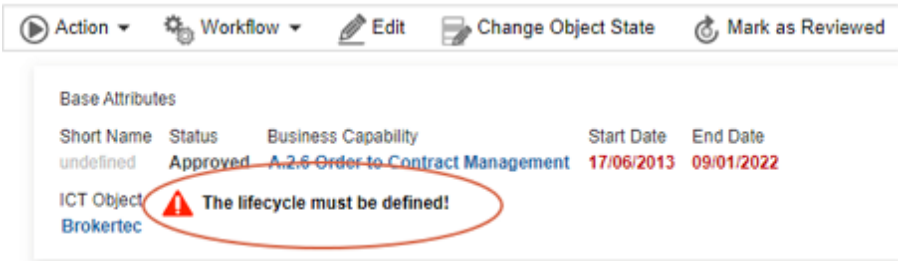


FIGURE: Check entry for lifecycle definition in a group box

To add a check entry via a Value Control interface control to the object cockpit:

1) In the toolbar of the object cockpit designer, click the **Value Control**  button and click in the group box.

2) Click the Value Control interface control to activate its attribute window. The **Control Type** attribute is automatically set to `ValueControl`. Define the following attributes, as needed:

- **Sub-Type**: Select `CheckEntry`.
- **Query** attribute (in the **Value Definition** section): Define an Alfabet query or a native SQL query to return a set of properties. For more information about the special conditions that apply to native SQL queries used in Alfabet configurations, see the section Defining Native SQL Queries in the chapter Defining Queries.

The following is an example of an Alfabet query defined to validate whether a lifecycle has been defined for the selected application:

```
ALFABET_QUERY_500

FIND
```



```
    Application
    InnerJoin TimeStatus ON TimeStatus.Owner =
    Application.RefStr
WHERE
    Application.REFSTR =:BASE
```

The following is an example of an Alfabet query defined to validate that a role has been defined for the selected application:

```
ALFABET_QUERY_500
FIND
    Application
    InnerJoin Role ON Role.Object = Application.RefStr
    InnerJoin RoleType ON Role.RoleType = RoleType.RefStr
WHERE
    (AND
    RoleType.Name = 'Architect'
    Application.RefStr =:BASE
    )
```

- **Check Result Type**: Select one of the following to define what constitutes a validated check:
- `Positive` means that the condition is fulfilled if the selected property has a value or the query has delivered a result (at least one row.)
- `Negative` means that the condition is fulfilled if the selected property has no value or the query doesn't deliver any results.

For example, to validate whether a property is set, an Alfabet query is specified that returns the value for the object class property. The condition is that the Alfabet query returns a result. In this case, the **Check Result Type** attribute must be set to `Positive`.

On the other hand, to validate that a property is not set to a specific value because that value is not allowed in the current environment, an Alfabet query must be specified that searches for the object class property value. The condition is only met if the Alfabet query returns no result. In this case, the **Check Result Type** attribute must be set to `Negative`.

- **Message** attribute. Enter the text to display if the check entry is fulfilled. This text should provide the user with information about what is required to resolve the check entry.
- **Icon**: Select the icon to display if the check entry is fulfilled. Any icon that you want to add to the object cockpit first must be imported to the icon gallery. For more information, see the section Adding and Maintaining Icons for the Alfabet Interface.
- **Display Style**: Select one of the following to determine how the icon and message are displayed:
- `Default`: Displays the icon defined in the **Icon** attribute and message defined in the **Message** attribute.
- `Text`: Displays the message defined in the **Message** attribute.

- `Icon`: Displays the icon defined in the **Icon** attribute.
- `IconAndText`: Displays first the icon and second the message.
- `TextAndIcon`: Displays first the text and second the icon.
- **Caption**: Not relevant for a Value Control interface control of the sub-type `CheckEntry`.
- **Hint**: Optional. Enter text for a tooltip. The tooltip will be displayed when the user points to the attribute in the object cockpit.

3) To specify styles such as the background color, borders, padding, margins, and font of the value displayed for the Value Control interface control, set the **Use Style** attribute to `True`. Expand the **Style** section of the attribute window and specify the relevant styles as needed.

The font color and font style of the caption of Value Control interface control is specified via the **Value Control Caption Font** attribute of the GUI scheme. For more information about defining the styling of the object cockpit, see the sections Configuring Styles and GUI Scheme Settings for Object Cockpits.

4) In the toolbar, click the **Save**  button to save your changes.

Adding a Query to the Object Cockpit

A query can be specified in order to find specific objects to display in the object cockpit. Be defining `Query` as the **Sub-Type** attribute for the Value Control interface control, you can configure a query to find a set of results that should be displayed. By defining `ValueByQuery` as the **Sub-Type** attribute for the Value Control interface control, you can configure a query whereby only the first row of the result is displayed. The Value Control interface control will be displayed in the Alfabet interface in the size that you define, even though during design time it may be distorted to show the entire query.

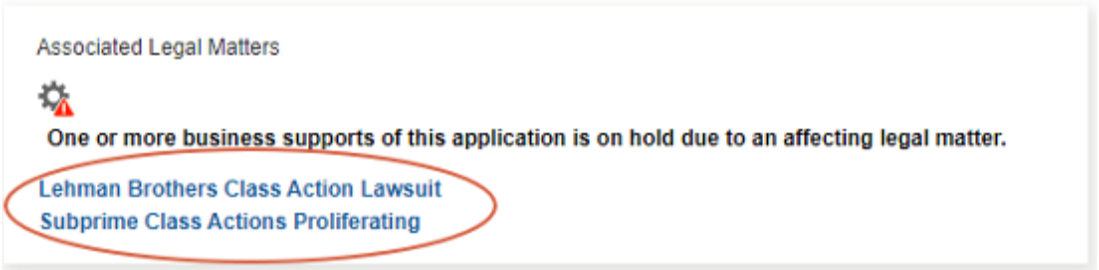


FIGURE: Query returning policy stereotypes

To add an Alfabet query or a native SQL query via a Value Control interface control to the object cockpit:

1) In the toolbar of the object cockpit designer, click the **Value Control**  button and click in the Group Box interface control.
2) Click the Value Control interface control to activate its attribute window. The **Control Type** attribute is automatically set to `ValueControl`. Define the following attributes, as needed:

- **Sub-Type**: Select one of the following:

- `Query` to configure an Alfabet or native SQL query to find a set of results that will be displayed (The subtype `Query` returns a scalar value), or
- `ValueByQuery` to configure a query to display only the first row of the result set the subtype `ValueByQuery` returns an Alfabet object). Ideally, the result set should return only one result. However, if a set of results is found, only the first row will be displayed and all other rows will be ignored.

If the **Sub-Type** attribute for the Value Control interface control is set to `ValueByQuery` and the query is a native SQL query, the instruction `SETCOLUMNFORMAT` must be included in the query, otherwise the formatting of dates displayed in the object cockpit may be incorrect.

It may be that the length of the data in the returned dataset is longer than the width of the cell. In this case, if the returned dataset contains one result, the value will be line-wrapped if the name is long. However, if multiple entries are displayed, the lines will not be wrapped in order to ensure that the individual results are recognizable and immediately usable for navigation.

- **Query**: (in the **Value Definition** section) Define an Alfabet query or a native SQL query to return a set of properties. For more information about the special conditions that apply to native SQL queries used in Alfabet configurations, see the section Defining Native SQL Queries in the chapter Defining Queries.

The following is an example of an Alfabet query of the type `Query` defined to find all projects whereby the selected application is defined in the project's as-is architecture:

```
ALFABET_QUERY_500

FIND

    Project

    InnerJoin ProjectArch ON ProjectArch.Project =
    Project.REFSTR

    InnerJoin Application ON ProjectArch.Object =
    Application.REFSTR

WHERE

    Application.REFSTR CONTAINS:BASE

    AUTODSINFO

    QUERY_XML

    <QueryDef>

    <ShowProperty Type="Property" ClassName="Project"
    Name="Name" />

    </QueryDef>
```

For value controls of the sub-type `ValueByQuery` that do not have a string defined via the attribute **Format String**, coloring and picture assignment can be assigned via Alfabet query language instructions. Only a subset of the available formatting options is available for this context:

- The `RowColorAssignment` and `ColorAssignment` instructions can both be used to change text and background color based on a condition. Please note

however that the value control returns the content of all cells of the dataset in a single string and therefore the whole text will be colored, even if a `ColorAssignment` instruction to color a single cell in a row is used.

- The `PictureAssignment` instruction can be used to return a value as an icon instead of a text or icon plus text. Only one of the values in the returned dataset can be converted to an icon. If the dataset returns cells containing text and a cell returning an icon, the icon will be displayed in front of the text string that is concatenated from the texts in all cells returning text, independent of the position of the icon in the dataset. If the picture assignment defines that both text and picture shall be displayed in the column, the text will be displayed within the concatenated text string in the correct position of the column it is defined for. If the `PictureAssignment` instruction specifies that the icon shall be displayed first, the icon will be displayed in front of the complete text string. If the `PictureAssignment` instruction specifies that the icon shall be displayed last, the icon will be displayed behind the complete text string. If an `ALT` text is defined in the picture assignment, it will be displayed as tooltip on the icon. Legend text specifications will be ignored in this context.

- **Format String** attribute (in the **Value Definition** section): If properties other than the `Name` property of the referenced object should be displayed in the object cockpit, specify which properties of the referenced object(s) to display in the table. The properties are specified as variables and must always be enclosed in curly brackets. For example, to display the name and status of a reference object, you would enter `{Name} Status={Status}`. Please note that any variable used in the **Format String** attribute must first be defined in the **Show Properties** attribute.

- **Inline Navigation**: Select `True` if users should be able to navigate to an object referenced by the object class property. The object class property will be displayed as a blue hyperlink. Please note that drill-down navigation is only possible if the **Sub-Type** attribute is set to `Query`. It is not possible if the **Sub-Type** attribute is set to `ValueByQuery`.

- **Hint**: Optional. Enter text for a tooltip. The tooltip will be displayed when the user points to the attribute in the object cockpit.

3) To specify styles such as the background color, borders, padding, margins, and font of the value displayed for the Value Control interface control, set the **Use Style** attribute to `True`. Expand the **Style** section of the attribute window and specify the relevant styles as needed.

The font color and font style of the caption of Value Control interface control is specified via the **Value Control Caption Font** attribute of the GUI scheme. For more information about defining the styling of the object cockpit, see the sections Configuring Styles and GUI Scheme Settings for Object Cockpits.

4) In the toolbar, click the **Save**  button to save your changes.

Adding a URL to the Object Cockpit

You can define a URL (a static Web link) in the object cockpit.

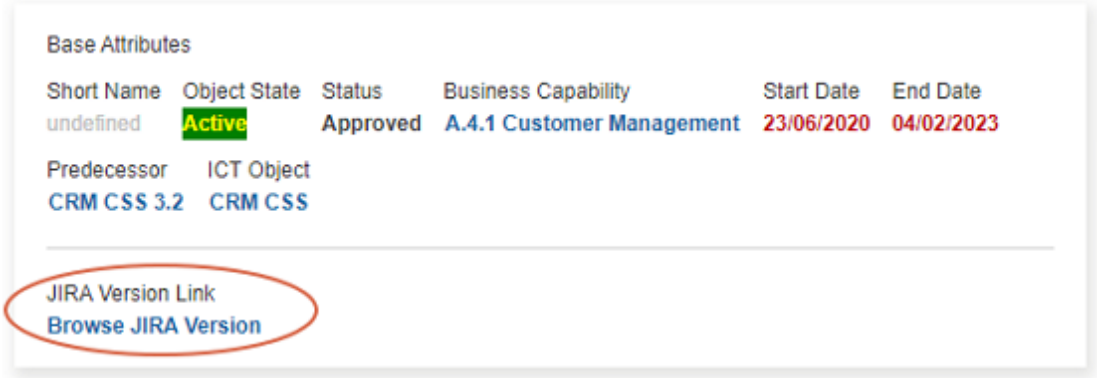


FIGURE: URL link in a group box

There are two different ways to make links accessible to users in the object cockpit:

- Display one link that requires a single click in the object cockpit. The link may either be an ALFI_URI object defined as a web link in the **Attachments** page view for the object or the link may be specified as a static link. When clicked, the target link opens in a new tab in the user's standard browser.

  If multiple links are defined in the **Attachments** page view, the most recently created link will be displayed.

- Display multiple links. The links must be ALFI_URI objects defined as web links in the **Attachments** page view. When the user clicks the link, a preview will open with an **Open Link** button. When clicked, the target link opens in a new tab in the user's standard browser.

An URL can alternatively be displayed in the object cockpit by means of a custom object class property of the type URL. To add a custom object class property of the type URL, see the section Adding a Property to the Object Cockpit.

The static web links will be automatically displayed in dark blue with an underline to indicate that it is a hyperlink.

To add a URL via a Value Control interface control to the object cockpit:

1) In the toolbar of the object cockpit designer, click the **Value Control** button and click in the Group Box interface control.
2) Click the Value Control interface control to activate its attribute window. The **Control Type** attribute is automatically set to `ValueControl`. Define the following attributes, as needed:

- To display one link that requires a single click in the object cockpit, set the **Sub-Type** attribute to `AlfaUriLink`.
- To display multiple links, set the **Sub-Type** attribute to `Query`. The query returns a scalar value.

3) In the **Query** attribute (in the **Value Definition** section), define an Alfabet query or a native SQL query to return the URL defined for the `ALFA_URI.OBJECT`. For more information about defining queries, see the chapter Defining Queries.

The following is an example of an Alfabet query defined to find the URL specified for the object displayed in a selected object cockpit.


```
ALFABET_QUERY_500
    FIND ALFA_URI
    WHERE ALFA_URI.OBJECT CONTAINS:BASE
    QUERY_XML
        <QueryDef>
            <ShowProperty Type="Property" ClassName="ALFA_URI"
            Name="NAME" />
        </QueryDef>
```

For a value control with the **Sub-Type** attribute set to `AlfaUriLink`, you can alternatively specify a static link. For example:

```
SELECT 'https://softwareag.com' AS LINK
```

4) In the toolbar, click the **Save** button to save your changes.

Adding a Dynamic Web Link to the Object Cockpit

Dynamic web links can be specified in value controls in object cockpits. Dynamic web links reference applications that are accessible via a URL. The dynamic web link references a specific location in the application, allowing you to access information in the application that is relevant to the object that the user is currently working with in Alfabet. The dynamic web links will be automatically displayed in dark blue with an underline to indicate that it is a hyperlink.

FIGURE: Dynamic web link in a group box

The relevant dynamic web links must first be configured for the relevant object class in the XML object ***WebViewManager***. At runtime, the dynamic web link will only be displayed if the object class and object class stereotype match the definition in the XML object ***WebViewManager***. For more information about configuring the XML object ***WebViewManager***, see the section Configuring Dynamic Web Links That Users Can Access.

The following is an example of an XML definition for dynamic web links in the XML object ***WebViewManager***:

```
<WebViewDef
    Caption = "Cumulocity Links"
    ClassNames = "Device:Cumulocity Device">
        <WebLinkDef Caption = "Cumulocity Device Link"
        HRef="https://alfabet.cumulocity.com/apps/devicemanagement/ind
```



```
                ex.html#/device/ {SC_CumulocityID} /info"
                PropNames="SC_CumulocityID" Target="Search"/>
        </WebViewDef>
```

To add a dynamic web link via a Value Control interface control to the object cockpit:

1) In the toolbar of the object cockpit designer, click the **Value Control** button and click in the Group Box interface control.
2) Click the Value Control interface control to activate its attribute window. The **Control Type** attribute is automatically set to `ValueControl`. Define the following attributes, as needed:

- **Sub-Type**: Select `DynamicWebLink`.
- **Web Link** attribute (in the **Value Definition** section): The selector displays all configured dynamic web links that have been defined for the relevant object class in the XML object ***WebViewManager***. Select the dynamic web link to display in the object cockpit.
- **Caption**: Define a caption for the dynamic web link. If no caption is defined for the value control, the caption defined in the XML object ***WebViewManager*** will be displayed.

3) In the toolbar, click the **Save** button to save your changes.

Adding Presentation Objects to Display Page Views and Configured Reports

You can add multiple Presentation Object interface controls in order to display page views and configured reports in the object cockpit. All views embedded in the object cockpit are ReadOnly and users will not be able to edit data in the context of the object cockpit. However, if users may need to edit data, you can provide a means for the user to navigate to the view from the object cockpit. For each view or configured report that you display in the object cockpit, you can specify a drill-down navigation that opens the page view/configured report in the object cockpit. The navigation can also be specified to open a page view/configured report other than that displayed in the object cockpit that would be relevant for the user context. If a configured report is embedded in the object cockpit and that report does not have the **Report State** attribute set to `Active` (perhaps because the report designer has changed the state in order to modify the configured report), a message will be displayed in the object cockpit where the report is embedded explaining that the report is currently under construction.

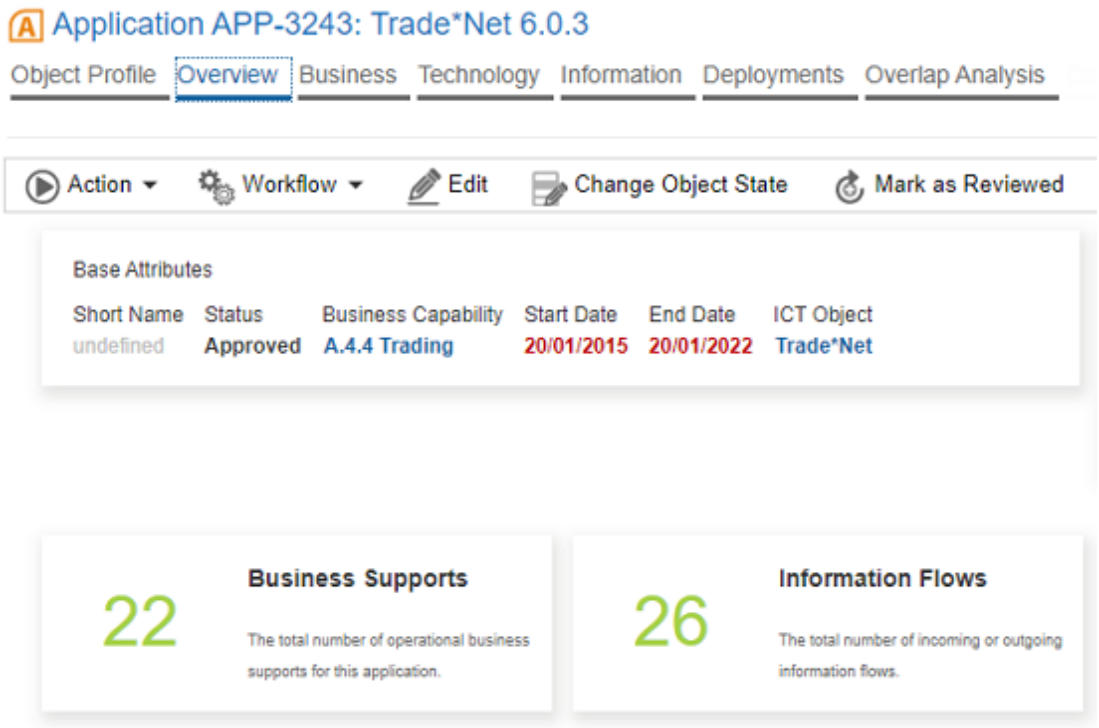


FIGURE: Two presentation objects based on widget reports

In some cases, you may want the object cockpit to consist only of one or more configured reports.

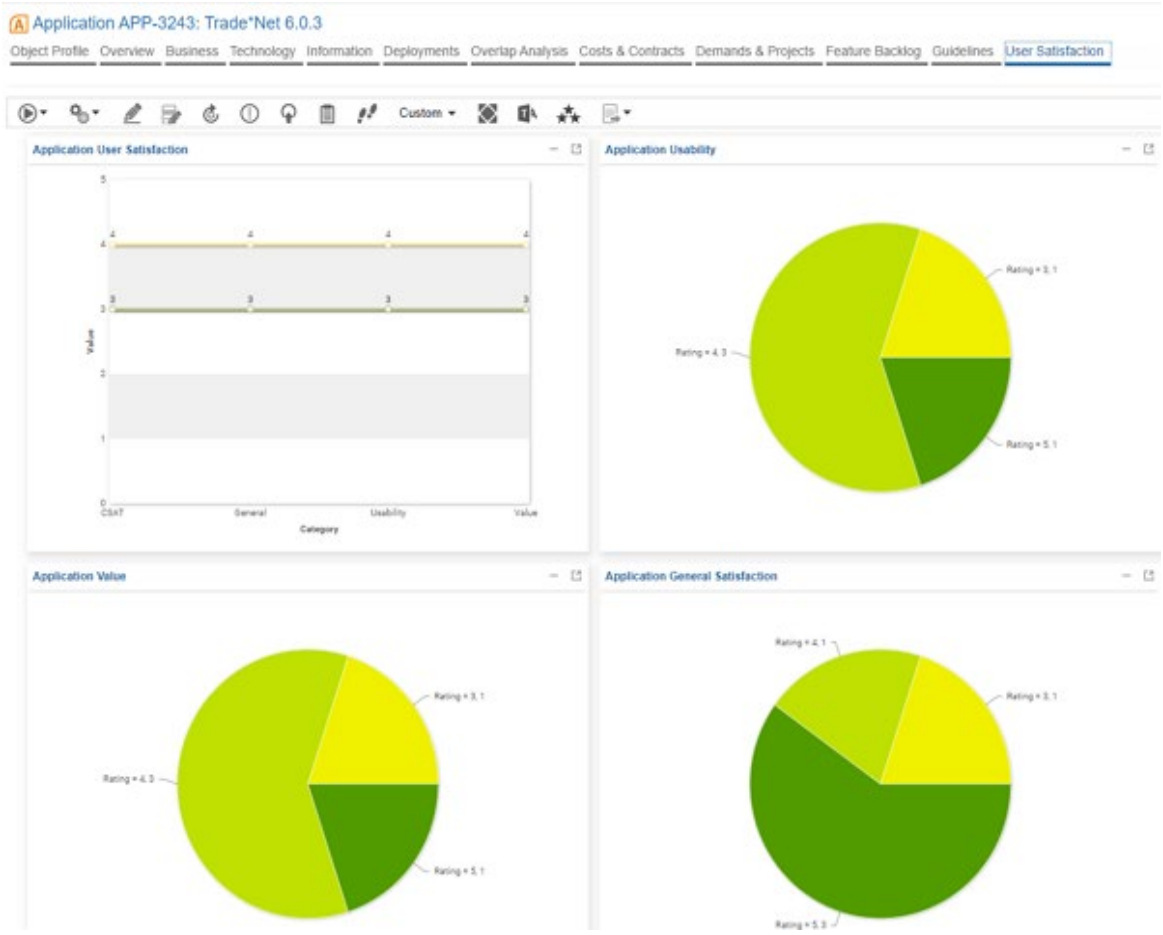


FIGURE: Object cockpit with four configured reports

The width that you define for a Presentation Object interface control will influence the number of Presentation Object interface controls that may be displayed horizontally in a row. Therefore, you should consider how many Presentation Object interface controls you would ideally like to display horizontally in a row and the width of each. It is recommended that Presentation Object interface controls have a base width of either 400, 600, or 1200 pixel. The sum of the width of multiple interface controls (group boxes and presentation objects) that shall be displayed in one row should not exceed 1200 pixel. For example, if you want to display three reports in the same row, you could define each presentation object to have a width of 400 pixel and conclude the row with a break <break> element. If the screen size is smaller than 1200 pixel or the user manually reduces the width of the screen is less than 1200 pixel, the last report preceding the break <break> element that doesn't fit in the user interface will be bumped down to the next row. If the data displayed in the Presentation Object interface control is dense (as in a diagram or business support map, for example), you might want the width of the Presentation Object interface control to be 1200 pixel. If an presentation object is wider than the available screen space, the content will be truncated.

For more information about adding break <break> elements to structure presentation objects and attribute boxes in the flow panel, see the section Adding Linebreaks to Visually Structure the Information in the Object Cockpit.

A height must be defined for the presentation objects, although it is possible to configure that the height is automatically reduced based on the content in the presentation object in order to avoid excessive white space in the object cockpit. Presentation objects with no available data will display the text **No data provided...**. Users can also collapse and expand the presentation objects in order to optimize the use of space and reduce the amount of scrolling required. A collapse button (−) will be displayed in the right corner header of the embedded view/report. When clicked, the view/report will collapse to show only the header bar with the caption of the view/report. The contents will be automatically rearranged in the object cockpit when the view/report is collapsed. The user can click the expand button (+) in the view/report header to expand the view. This setting will be stored in the user context settings. A zoom functionality will also be available automatically for views with business graphics.

Furthermore, a navigation button has been added to the header bar of embedded views and reports. When clicked, the view or report will open to fit the full screen. The user can also navigate to the view or report by clicking the hyperlinked caption of the view or report in the header bar.

As an alternative to displaying the data for a page view or configured report directly in the object cockpit, you can implement placeholders which act like links to open the relevant page view or configured report. For more information, see the section Adding Placeholders for Page Views and Configured Reports.

The **Add to Clipboard** button available in the toolbar of page views and relevant configured reports can be used in the context of the object cockpit without explicitly opening the embedded view/report. The functionality will also be available in the **Preview** window when opened in the context of the embedded view or report.

To add a standard page view or configured report via a Presentation Object interface control to the object cockpit

1) In the toolbar of the object cockpit designer, click the **Presentation** button and click in the Flow Panel interface control. The Presentation Object interface control is displayed in the Flow

Panel interface control and should be selected. The **Control Type** attribute is automatically set to `Presentation`. Define the following attributes, as needed:

- **Sub-Type**: Select `View`.
- **Source**: Click the **Browse** button and in the **Select Source** editor that opens, select either `Configured Reports` or `Page Views`. If you select `Page Views`, all relevant and permissible page views that have not yet been added to the object view will be displayed in the **Select Source** editor. If you select `configured Reports`, all configured reports will be displayed in the **Select Source** editor. Select the relevant view/configured report in the drop-down field and click **OK**.

  Offline executed reports cannot be embedded in object cockpits and therefore will not be displayed in the drop-down list in the **Source** attribute.

- **Caption**: Enter a caption for the view/report. This attribute must be filled in if you want to configure the page view/configured report so that the user may navigate to it. The caption may differ from the actual name of the view/report. Please note that the caption specified for configured reports displaying business graphics will not be displayed in the context of the object cockpit. Instead, the caption specified in the **Caption** attribute of the presentation object embedded in the object cockpit will be displayed.
- **Inline Navigation**: Select `True` to configure the page view/configured report so that users can navigate to it. The caption will be displayed as a blue hyperlink that the user can click to open the view/report.
- **Width**: Define the width in pixel for the presentation object. Any presentation object with a defined width that is greater than the defined width of the object cockpit will be truncated and not all content will be visible. Because the presentation object's typically contain lots of information, it is recommended that they not be smaller than 400 pixel in width.
- **Height**: Define the height in pixel for the presentation object. Please note that if you specify that the height of one presentation object in a row is greater than the height of the other presentation objects, empty space may result in the row. This is because the bottom edge of the largest presentation object will be used to determine the top edge of the next row. It is recommended that all presentation objects in a row preceding a break <`break`> element have the same height.

  You can configure the height of a Presentation Object interface control to be automatically reduced based on the content in the view. This ensures that the data is dense and excessive white space is not displayed. In order to specify that the height of presentation objects shall be automatically reduced based on their content, see the section Configuring Styles and GUI Scheme Settings for Object Cockpits.

- **Hint**: Optional. Enter text for a tooltip. The hint will be displayed when the user points to the embedded view in the object cockpit.
- **Can Print**: Set to `False` if the **Print** button displayed in the floating toolbar of the embedded page view/configured report should be disabled. The attribute is set to `True` per default.

2) To add a caption for the view or report, click the **Static Text** button in the design editor and place the Static Text interface control directly above the presentation object. In the **Caption** attribute for the static text, enter a caption for the view. For more information about how to create static text, see the section Adding Static Text to the Object Cockpit.

3) To add padding to the view or report, define the **Style** > **Margin** attributes for each presentation object in the object cockpit and set the **Use Style** attribute to `True`. For example, enter 2 in the **All** attribute in the **Margin** section of the attribute window

4) In the toolbar, click the **Save**  button to save your changes.

Adding a Floating Group Box to Contain Presentation Objects

Two or more Presentation Object interface controls can be placed inside a collapsible group box, thus allowing the set of views to be hidden or expanded in the object cockpit as needed. For example, a set of widget reports might be placed inside such a group box in order use the space in the object cockpit more efficiently. The user can collapse the group box by clicking the collapse (-) symbol and expand the group box by clicking the expand (+) symbol. The collapsed group box will reduce to the width and height of the group box caption. For general information about positioning group boxes in the object cockpit, see the section Adding Group Boxes to Display Attributes.

To add a Group Box interface control containing Presentation Object interface controls to the object cockpit:

1) In the toolbar of the object cockpit designer, click the **Group Box**  button and click in the flow panel.

2) Click the Group Box interface control to ensure that its attribute window is open. Define the following attributes, as needed:

- **Sub-Type**: Select `Floating`.
- **Caption**: Define a caption for the group box. This attribute may not be left empty.

The font color and font style of the caption of group boxes is specified via the **Attributes Group Caption Font** attribute of the GUI scheme. For more information about defining the styling of the object cockpit, see the sections Configuring Styles and GUI Scheme Settings for Object Cockpits.

3) Specify the height of the group box in pixel in the **Height** attribute. Please note that the height you define will depend on the height of the presentation objects you place inside the group box. The height of the group box should be larger than the height of the presentation objects so that they can be adequately displayed.

4) Specify the width of the group box:

- **Width**: Define the width in pixel for the group box. Any group box with a defined width that is greater than the defined width of the object cockpit will be truncated and not all content will be visible.
- **Item Max. Width**: Enter the maximum width of the interface controls in the group box. Enter -1 if there shall be no limitation on the maximum width.
- **Item Min. Width**: Enter the minimum width of the interface controls in the group box. Enter -1 if there shall be no limitation on the minimum width.

5) Specify the placement of the group box. You can do this via drag-and-drop of the attribute box or by entering values in pixel in the **Top** and **Left** attributes.

6) Click the **Save**  button to save the new group box.

7) Add the relevant Presentation Object interface controls to the group box as described in the following section Adding Presentation Objects to Display Page Views and Configured Reports.

8) Add a break (`<break>`) element between groups of attributes in the group box to add a linefeed and separator, as needed. For more information about configuring a linebreak, see the section Adding Linebreaks to Visually Structure the Information in the Object Cockpit.

9) Specify the **Tab Index** attribute for all interface controls and break elements (`<break>`) in the Group Box interface control in order to specify the correct sequence and layout in the group box. The sequence should begin with 1. The sequence that you specify is the order in which the interface controls and break elements are displayed as well as the order to navigate through the group box via the TAB key.

Adding a Floating Group Box as a Filter Panel for Reports

Report filters available in the configured reports that are embedded in the object cockpit can be placed inside the group box of type `Floating`. This includes filters based on Checked List Box and Combo Box interface controls. The user can set the report filters, which are applied to all relevant reports displayed in the object cockpit. This supports a "what-if analysis" whereby the end user could change the filter settings to immediately display different results across multiple dimensions.

To add a Group Box interface control containing report filters to the object cockpit:

1) In the toolbar of the object cockpit designer, click the **Group Box**  button and click in the flow panel.

2) Click the Group Box interface control to ensure that its attribute window is open. Define the following attributes, as needed:

- **Sub-Type**: Select `Floating`.
- **Caption**: Define a caption for the group box. This attribute may not be left empty.

The font color and font style of the caption of group boxes is specified via the **Attributes Group Caption Font** attribute of the GUI scheme. For more information about defining the styling of the object cockpit, see the sections Configuring Styles and GUI Scheme Settings for Object Cockpits.

3) Specify the height of the group box in pixel in the **Height** attribute. Please note that the height you define will depend on the height of the interface controls you place inside the group box. The height of the group box should be larger than the height of the interface controls so that they can be adequately displayed.

4) Specify the width of the group box:

- **Width**: Define the width in pixel for the group box. Any group box with a defined width that is greater than the defined width of the object cockpit will be truncated and not all content will be visible.
- **Item Max. Width**: Enter the maximum width of the interface controls in the group box. Enter -1 if there shall be no limitation on the maximum width.

- **Item Min. Width**: Enter the minimum width of the interface controls in the group box. Enter -1 if there shall be no limitation on the minimum width.

5) Specify the placement of the group box. You can do this via drag-and-drop of the attribute box or by entering values in pixel in the **Top** and **Left** attributes.

6) Click the **Save** button to save the new group box.

7) Add the relevant report filters that are configured for the reports embedded in the object cockpit directly to the Group Box interface control. It is easiest to simply copy the report filter in the report and paste it to the Group Box interface control. For more information about configuring report filters, see the section Defining Filters for Configured Reports and Selectors in the chapter Configuring Reports in the chapter Defining Queries.

8) For each report embedded in the object cockpit that the filters shall apply to, set the **Refresh on Submit** attribute to `True`.

9) Create a button to execute the filters and place it in the Group Box interface control. The **Submit** attribute must be set to `True` for the button. For more information about creating a submit button, see the section Configuring the Submit Button in the chapter Defining Queries.

10) The new **Save Context** attribute has been added to the attribute grid for the object cockpit. This attribute should be set to `True` so that the user context settings for the filters can be saved.

11) Specify the **Tab Index** attribute for all interface controls and break elements (`<break>`) in the Group Box interface control in order to specify the correct sequence and layout in the group box. The sequence should begin with 1. The sequence that you specify is the order in which the interface controls and break elements are displayed as well as the order to navigate through the group box via the TAB key.

Adding a Presentation Object for a Different Target Object

A view can be displayed in an object cockpit for an object that is not the target object of the object cockpit. For example, a workflow diagram could be displayed in the context of the object that it is associated with or the **LIfecycle** page view could be displayed for the ICT object that owns the application that is the target object of the object cockpit.

You can define either an Alfabet query or native SQL query. The query should be defined to find only a single object. The object returned by the query will be the base object of the view displayed in the object cockpit.

You must ensure that the object being searched for by the query exists and can be found by the query. If an object cannot be found by the query, the message **No data provided...** will be displayed for the view.

To configure the object cockpit to display a view for a different target object:

1) In the toolbar of the object cockpit designer, click the **Presentation** button and click in the Flow Panel interface control. The Presentation Object interface control is displayed in the Flow Panel interface control and should be selected. The **Control Type** attribute is automatically set to `Presentation`. Define one of the following attributes, as needed:

- **Base Object via Alfabet Query**: Specify an Alfabet query via the **Alfabet Query Builder** to find the relevant object. The result of the query should have the `REFSTR` of the object which should be used as

base object in the first column. For Alfabet queries, `SHOW` properties do not need to be defined. The `REFSTR` of the object found by the query is selected automatically as the query result.

The following example illustrates an Alfabet query that searches for the ICT object that the application is assigned to.

```
ALFABET_QUERY_500
FIND
ICTObject
  InnerJoin Application
        ON Application.ICTObject = ICTObject.REFSTR

WHERE
Application.REFSTR CONTAINS  :BASE
```

- **Base Object via SQL Query**: Enter a native SQL query to find the relevant object. The query must be of the type `SELECT`. The result of the query should have the `REFSTR` of the object which should be used as base object in the first column. Therefore, the `REFSTR` of the object must be defined as the first `SELECT` property in the native SQL query.
- **Base Object via Query as Text**: Enter a native SQL query or Alfabet query to find the relevant object.

2) Once you have defined the base object query for the Presentation Object interface control, specify the view to open for the object found by the base object query. To do so, click the **Browse** button available for the **Source** attribute. In the **Select Source** editor that opens, select `Configured Reports`. All configured reports will be displayed in the **Select Source** editor. Select the relevant configured report in the drop-down field and click **OK**.

3) In the toolbar, click the **Save**  button to save your changes.

Adding Placeholders for Page Views and Configured Reports

As an alternative to displaying the data for a page view or configured report directly in the object cockpit, you can implement placeholders via an Icon interface control. The placeholders are icons that are displayed instead of the page view or configured report. This is especially useful for complex views or large reports for which only a portion of the report is displayed in the object cockpit. Users can click the icon and open the relevant page view or configured report. Alfabet provides a number of icons that represent various view and report types including, for example, diagrams, Gantt charts, gauge reports, matrix reports, portfolio reports, etc. However, you can use any of your enterprise's icons that have been added to the icon gallery. For more information, see the section Adding and Maintaining Icons for the Alfabet Interface.

FIGURE: Placeholder for report in object cockpit

To add a placeholder to a page view or configured report via an Icon interface control:

1) In the toolbar of the object cockpit designer, click the `Icon` button and click in the flow panel. An empty box displays the undefined icon in the object cockpit. The **Control Type** attribute is automatically set to `Icon`. Define the following attributes, as needed:

- **Source**: Select either `Reports` or `Views` and select the relevant page view/configured report in the drop-down field and click **OK**.
- **Caption**: Enter the text to display below the placeholder. For example, this could be the name of the page view/configured report or other text that may be relevant.
- **Sub-Type**: Select `IconFree` to add one of the standard icons provided by Alfabet to represent the specified page view/configured report. You can also select `Icon` to add an icon sized 22x22 pixel or `IconLarge` to add an icon sized 30x30 that is available in the icon gallery.
- **Icon**: Select the icon that you want to display for the specified page view/configured report.

2) Click the **Save** button to save the new break.

Adding Workflow, Assignment, Microsoft Teams Meeting Information and Object Validation Information to the Object Cockpit

The number of open assignments and open workflow activities as well as the number of Microsoft Teams meetings scheduled today for the current user about the current object can be displayed in object cockpits. The relevant number will be automatically generated at runtime and the respective link will open when the user clicks the text. Additionally, a wizard validations section will display all wizard post-conditions that have been violated for the object displayed in the object cockpit.

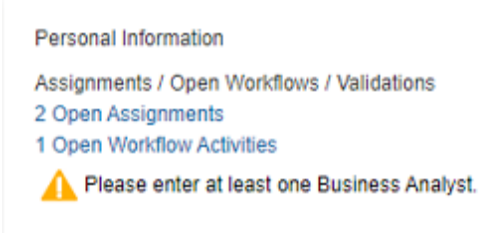


FIGURE: Personal Info section shows open assignments, workflows, and violated post-conditions

To include this information, you must add a Value Interface control of sub-type `PersonalInfo` to the flow panel. You can specify individually whether assignment, workflow, and wizard validation information shall be displayed in the object cockpit. In the user interface, a personal information box will be added to the object profile displaying the caption **Assignments** / **Open Workflows** / **Validations**. The following will be displayed:

- **Assignments**: A link showing the number of open assignments will be displayed. When the user clicks the link, the **My Assignments** page view will open displaying the pending assignments for the object.
- **Open Workflows**: A link showing the number of open workflows will be displayed. When the user clicks the link, the **Workflow Activities Explorer** will open displaying the active workflow steps that the user is responsible for.

  Instead of displaying the default **Workflow Activities Explorer**, a custom explorer can be configured to open instead. The custom explorer must be specified in the **Custom Workflow Activities Explorer** attribute of the relevant user profile. For more information about configuring custom explorers, see the chapter Configuring a Custom Selector for Search Functionalities, and for more information about specify the custom explorer for a user profile, see the chapter Creating User Profiles for the User Community.

- **Validations**: Each wizard post-condition that has been violated will be displayed. The message configured for the wizard post-condition will be displayed as information about why the validation failed. For more information about the configuration of wizard post-conditions, see the section Defining a Post-Condition for a Wizard Step.

To add the personal information box to the object cockpit:

1) In the toolbar of the object cockpit designer, click the **Value Control** button and click in the Flow Panel interface control.
2) Click the Value Control interface control to activate its attribute window. The **Control Type** attribute is automatically set to `ValueControl`. Define the following attributes, as needed:

- **Sub-Type**: Select `PersonalInfo`.
- **Caption**: Define a caption for the personal information box.
- **Value Definition**: Expand this section and specify the following as needed:
- **Assignments**: Set to `True` to include the number of assignments and the respective link to access the **My Assignments** page view.
- **Workflow**: Set to `True` to include the number of open workflow activities and the respective link to access the **My Workflow Activities Explorer**.
- **Wizard Validations**: Set to `True` to include violated wizard post-conditions and the message configured for the wizard post-condition

3) To specify styles such as the background color, borders, padding, margins, and font of the value displayed for the Value Control interface control, set the **Use Style** attribute to `True`. Expand the **Style** section of the attribute window and specify the relevant styles as needed.

   The font color and font style of the caption of Value Control interface control is specified via the **Value Control Caption Font** attribute of the GUI scheme. For more

information about defining the styling of the object cockpit, see the sections Configuring Styles and GUI Scheme Settings for Object Cockpits.

4) In the toolbar, click the **Save** button to save your changes.

Adding Icons to the Object Cockpit

You can add graphic images to a Group Box interface control or Flow Panel interface control that are available in the icon gallery in Alfabet Expand. Any icon that you want to add to the object cockpit first must be imported to the icon gallery. For more information, see the section Adding and Maintaining Icons for the Alfabet Interface.

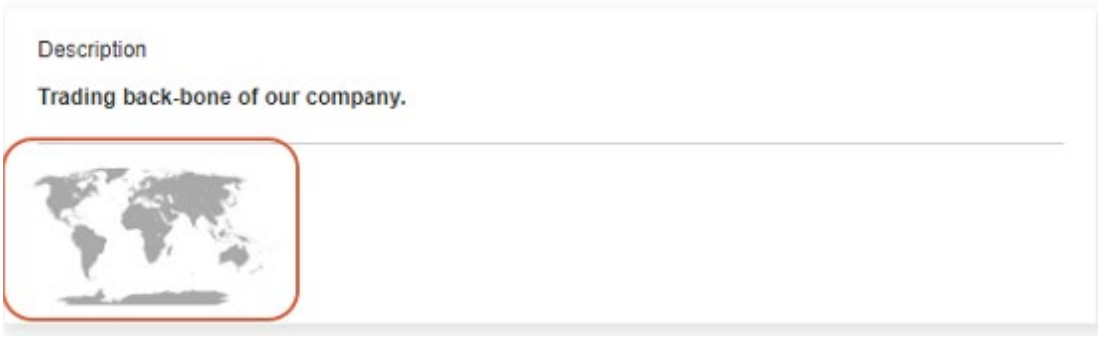


FIGURE: Icon displaying global map

To add an Icon interface control to the object cockpit:

1) In the toolbar of the object cockpit designer, click the **Icon** button and click in the Flow Panel interface control or Group Box interface control. An empty box displays the undefined icon in the object cockpit. The **Control Type** attribute is automatically set to `Icon`. Define the following attributes, as needed:

- **Sub-Type**: Select the relevant option based on the size of the icon that you want to add:
- `Icon`: To add an icon that is 22x22 pixel in size.
- `IconLarge`: To add an icon that is 30x30 pixel in size.
- `IconFree`: To add an icon that has an arbitrary size.
- **Icon**: Select the relevant icon that you want to display in the object cockpit. The icons that you can choose from are based on the value you selected in the **Sub-Type** attribute.

2) To change the size of the icon, adjust the values in the **Height** and **Width** attributes as needed.

3) In the toolbar, click the **Save** button to save your changes.

Adding Embedded HTML to the Object Cockpit

The HTML Content interface control allows you to display text with HTML formatting (font and color) as well as images, links to external URLs, and HTML forms. Variables such as @BASE may be used in the URL link to specify content for static links or server variables.

Please note that the HTML Content interface control should not be added to a Group Box interface control. The HTML Content interface control should only be placed in a Flow Panel interface control or a Table Layout Panel interface control

Please note that if the object cockpit is published via the **Publish** button, the HTML content will not be included in the DOC or PDF file.

To add an HTML Content interface control to an object cockpit:

1) In the toolbar of the object cockpit designer, click the **HTML Content** button and click in the Flow Panel interface control.
2) Click the HTML element to activate its attribute window. The **Control Type** attribute is automatically set to `Html`. Define the following attributes, as needed:

- **Sub-Type**: Select `EmbeddedHTML`.
- **HTML Source**: Enter the HTML code. Please keep the following in mind:
- The HTML must be compliant with XHTML, XML-conform HTML, compliant with HTML 5, and use standard HTML tags. The HTML header implements standard HTML elements.
- The code must start with an `<xhtml>` tag and end with `</xhtml>`. Standard HTML elements must be written in lower-case letters in order to be correctly parsed: `<xhtml>`, `<html>`, `<head>`, `<body>`, and `<culture_>`. The definition of `<head>`, `<body>`, and `<culture_>` is optional.
- The formatting of the HTML can either be written explicitly in the `<body>` element or can be specified via a stylesheet that is stored in the **Internal Document Selector**. In this case, the HTML must refer to the target CSS file in a `<link>` element. The CSS file should contain all necessary styles to display the content of the HTML. Please note that the CSS file may not be stored in the root folder of the **IDOC** explorer. The CSS file stored in the **Internal Document Selector** must be located in a document folder that is subordinate to the root folder of the **IDOC** explorer. To upload a file to the **Internal Document Selector**, see the section *Uploading Documents and Managing User Permissions to Document Folders in the Internal Document Selector* in the reference manual *User and Solution Administration*. For example:

```
<xhtml>
    <culture_1033>
        <html>
        <body
        style="margin:4px;background:#3d4b60;overflow:hidden;">
            <link type="text/css" rel="stylesheet"
            href="IDOC:\CSS\my_style_file.css"></link>
            <p>This is the content for the header of the wizard
            step.</p>
            ...
```



```
        </body>
        </html>
    </culture_1033>
</xhtml>
```

- All CSS formatting instructions must end with `!important` to ensure that they are processed by the Alfabet application. For example:

```
<xhtml>
    <culture_1033>
        <html>
        <body style="margin:4px !important;background:#3d4b60
        !important;overflow:hidden !important;">
            <style type="text/css">
                p.header
                {
                    font-family:verdana !important;
                    font-size:18px !important;
                    color:#ff0000 !important;
                    text-align: Left !important;
                }
            </style>
            <p class="header">This is the content for the header of
            the wizard step.</p>
            ...
        </body>
        </html>
    </culture_1033>
</xhtml>
```

- If translation to a secondary language is required, you must provide the translation in the HTML specification. Please note that HTML texts cannot be translated via the **Translation Editor** available in Alfabet Expand. Please consider the following:
    - If the HTML description is required in additional languages, each language text must be defined within language elements (For example, `<culture_1031>` for English, `<culture_1033>` for German, etc.)
    - The element `<culture_xxx>` must be specified as a child of the root element `<xhtml>`. The element `<html>` must be specified as a child of the element `<culture_xxx>`. The element `<html>` contains the HTML specification in the relevant language. For example, to provide information in English and German:

```
<xhtml>
    <culture_1033>
        <html>
            <body>
```



```
                    <h3>Glossary: Application</h3>
                    <p>An application is a fully-functional integrated
                    IT product that provides functionality to end
                    users and/or to other applications. As such, an
                    application supports the business to accomplish
                    its mission. Applications operate on a platform
                    made up of hardware and software components
                    necessary to run the application.</p>
                </body>
            </html>
        </culture_1033>
        </culture_1031>
            <html>
                <body>
                    <h3>Glossar: Applikation</h3>
                    <p>Eine Applikation ist ein voll funktionsfähiges,
                    integriertes IT-Produkt, das Funktionalitäten für
                    Endanwender und/oder für andere Applikationen
                    bietet. Eine Applikation unterstützt das
                    Unternehmen bei der Zielerreichung. Applikationen
                    werden auf einer Plattform betrieben, die aus den
                    für die Ausführung der Applikation erforderlichen
                    Hardware- und Software-Komponenten besteht.</p>
                </body>
            </html>
        </culture_1031>
    </xhtml>
```

3) In the toolbar, click the **Save** button to save your changes.

Configuring Conditional Restraints in the Object Cockpit

Specifying conditional constraints in object cockpits can provide a more dynamic user experience and ensure that users are seeing and defining the data that is pertinent to the task at hand. The display of data in the object cockpit that isn't relevant to the context at hand can be suppressed based on the definition of a value for a property of the associated class. You can specify that interface controls such as Value Query or Presentation Object are visible/not visible based on whether one or more conditions are fulfilled. If all visibility conditions fail for an interface control, the visibility will be revoked for that interface control.

For example, if the **Is SOX Relevant** attribute is set to `True` for the application, then you would want to display the **Business Support Map Report** embedded in the object cockpit. In this case, you would assign a visibility condition to the Presentation Object interface control specifying the **Business Support Map Report** and map the variables in the condition to the **Is SOX Relevant** attribute that the variable references.

The following steps are required to configure a visibility condition for a Presentation Object or constraints for interface controls in object cockpits:

- Define a configured report of the type `Query` or `NativeSQL` and set the **Category** attribute of the report to a category name defined in the XML object ***UseCaseCategories*** for the use case `Conditions`. The query should search the Alfabet database and return at least one row as a result to fulfill the condition if the **Check Result Type** attribute is set to `Positive`. When the **Check Result Type** attribute is set to `Negative` the query return no records. For more information about configuring reports of the type `Query` or `NativeSQL`, see the chapter Configuring Reports.
- Create the condition and specify the configured report with the query.
- To specify that the interface control targeted by the condition is visible, specify the **Visibility Condition** attribute.

The following information is available:

- Creating Conditions to Implement in an Object Cockpit
- Assigning Conditions to Interface Controls in the Object Cockpit

Creating Conditions to Implement in an Object Cockpit

Conditions can be used in the context of custom editors, object cockpits, and filters in configured reports and can be reused as needed. Conditions for the interface controls in object cockpits can be based on either expressions or queries defined in configured reports. The condition state must be set to `Active` in order to make the condition available in the **Visibility Condition** attribute for interface controls.

To create a condition:

1) Go to the **Presentation** tab and right-click the **Conditions** folder and select **New Condition**. The condition state will be set to `Plan` per default for the new condition.
2) Click the new condition  and define the following in the attribute window:

- **Technical Name**: Enter a name for the condition. It is recommended that the name indicates the meaning of the condition. All active conditions will be displayed in drop-down lists for the **Visibility Condition** attribute.
- **Group**: Enter the name of a new condition folder that you want to store the condition in or select an existing condition folder.

3) Conditions for the interface controls in object cockpits can be based on either expressions or queries defined in configured reports. Please note the following:

- In the **Type** field, select `Report` to specify a configured report for the condition. The configured report should already be configured and should be of the type `Query` or `NativeSQL`. The query should search the Alfabet database and return at least one row as a result to fulfill the condition if the **Check Result Type** attribute is set to `Positive`. When the **Check Result Type** attribute is set to `Negative` the query return no records. The report is only selectable for conditions if the **Category** attribute of the report is set to `AEMF`. Define the following attributes:
- **Report**: Select the configured report that should be executed for the condition.
- **Check Result Type**: Select either `Positive` or `Negative` to define what constitutes a fulfilled condition.

- `Positive` means that the condition is fulfilled if the query associated with the configured report delivers a result.
- `Negative` means that the condition is fulfilled if the query associated with the configured report delivers no result.

- In the **Type** field, select `Expression` to specify an expression for the condition. Define the following attributes:
- **Expression**: Click the **Browse**  button to open the editor and enter the expression that shall be evaluated for the condition. Multiple expressions may be specified, whereby individual expressions must be placed in parenthesis. Values must be placed in single quotes. For example: `(@InterfaceControl_1 == 'True') I ((@InterfaceControl_2 == 'True') & (@InterfaceControl_3 == 'False'))`. The following operators and conditions may be used in the expression:

equals	@Param == 'STRING' (or 'True' and 'False' for Boolean values)
does not equal	@Param != 'STRING'
in	@Param IN ['STRING', 'STRING', 'STRING'] (used when the value is a list)
not in	!(IN ['STRING', 'STRING', 'STRING'] (used when the value is a list)
begins with	@Param BEGINS_WITH 'STRING'
ends with	@Param ENDS_WITH 'STRING'
contains	@Param CONTAINS 'STRING'
does not contain	!(CONTAINS 'STRING')
is between	(@Param > NUMERIC1) & (@Param < NUMERIC2)
is empty	@Param EMPTY
is not empty	!(@Param EMPTY)
is greater than	@Param > NUMERIC
is greater than or equal to	@Param >= NUMERIC
is less than	@Param < NUMERIC

is less than or equal to	@Param <= NUMERIC
AND	&
OR	|
NOT	&

- **Check Result Type**: Select either `Positive` or `Negative` to define what constitutes a fulfilled condition.
    - `Positive` means that the condition is fulfilled if the variable targeted by the expression has a value.
    - `Negative` means that the condition is fulfilled if the variable targeted by the expression has no value.
- Once the condition, has been written, you can test the condition. To do so, right-click the condition and select **Test Condition**. In the editor, enter a valid value in the **Parameter Value** field and click **OK**. The message box will indicate if the condition is valid based on the value entered or if an error has occurred.

4) Once the condition is complete and ready to be implemented, the condition state must be set to `Active`. To do so, right-click the condition and select **Set Condition State to 'Active'**. The condition will be available in the **Visibility Condition**, **Presence Condition**, and **Read-Only Condition** attributes for custom editor fields, filter fields in configured reports, object cockpits, and object profiles.

5) In the toolbar, click the **Save**  button to save your changes.

Assigning Conditions to Interface Controls in the Object Cockpit

You can specify that an interface control is visible/not visible based on whether a condition is fulfilled.

To define conditions for object cockpits:

1) Go to the **Presentations** tab and navigate to the object cockpit that you want to define.
2) Right-click the object cockpit  that you want to define and select **Design Object Cockpit**. The object cockpit is displayed in the editor in the center pane.
3) Click the interface control that is the target of the condition and define the following attributes:

- **Visibility Condition**: Click the **Browse**  button and in the **Visibility Condition** editor select the configured report or expression in the **Select Condition** field. If the condition is based on an expression, define the following:
- **Parameter Name**: Displays the variables used in the condition.
- **Parameter Value**: This column is not relevant for object cockpits.
- **Parameter Value Reference**: Select the interface control that shall be visible if the variable in the **Parameter Name** column to the source interface control (Interface Control 1) of the condition.

For example, in the case of example about the **SOX Relevance** attribute and the **Business Support Map Report** embedded in the object cockpit in the **Application** object cockpit: the `SimpleBooleanOnTrue` condition has been created with the expression `@VALUE == 'True'`. You would define the attributes for the **Visibility Condition** as follows:

- **Select Condition**: Select `SimpleBooleanOnTrue`.
- **Parameter Name**: Displays `VALUE`.
- **Parameter Value Reference**: Select `SOXRelevant`. This parameter represents the variable to be evaluated for the parameter in the **Parameter Name** column.

Therefore, if the **SOX Relevance** interface control is set to `True`, then the `SimpleBooleanOnTrue` condition is fulfilled and the **Business Support Map Report** interface control will be visible.

- Click **OK** to save the definition and close the editor.

4) In the toolbar, click the **Save**  button to save your changes.

Adding Linebreaks to Visually Structure the Information in the Object Cockpit

You can add multiple linebreaks to create separators in the layout. Adding a break `<break>` element to the object cockpit forces the interface control following the break to bump to the next row. In this way, you can add visual space and structure the layout of interface controls. This might be useful, for example, to visually separate attribute boxes from page views or to thematically structure page views in sections.

A break `<break>` element can be added to the flow panel to structure Group Box interface controls and Presentation Object interface controls. For example, if you want to display three reports in the same row, you could define each presentation object to have a width of 400 pixel and conclude the row with a break `<break>` element. If the screen size is smaller than 1200 pixel or the user manually reduces the width of the screen is less than 1200 pixel, the last report preceding the break `<break>` element that doesn't fit in the user interface will be bumped down to the next row. If an interface control (group box or presentation object) is wider than the available screen space, the content will be truncated.

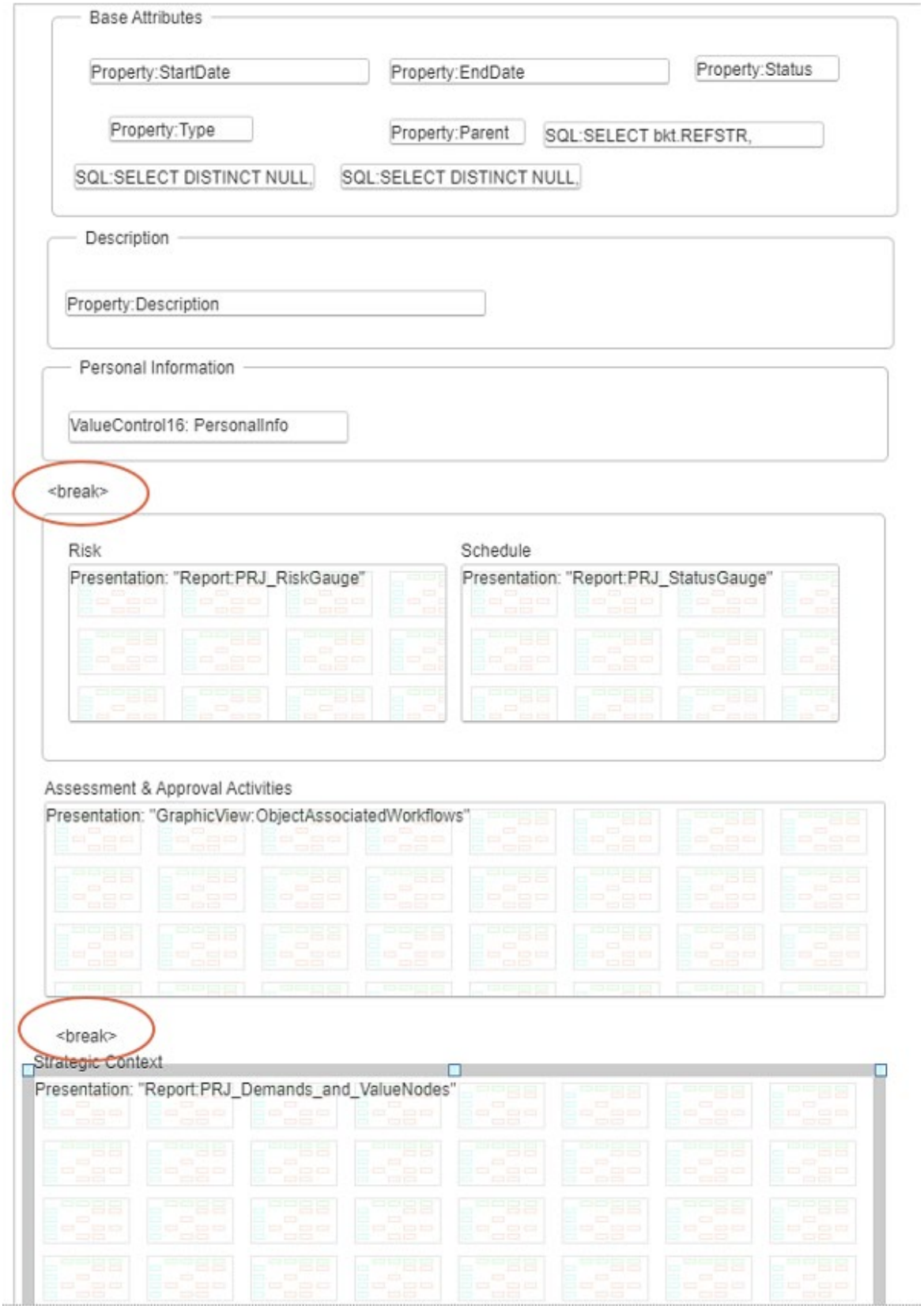


FIGURE: Configuration of line breaks

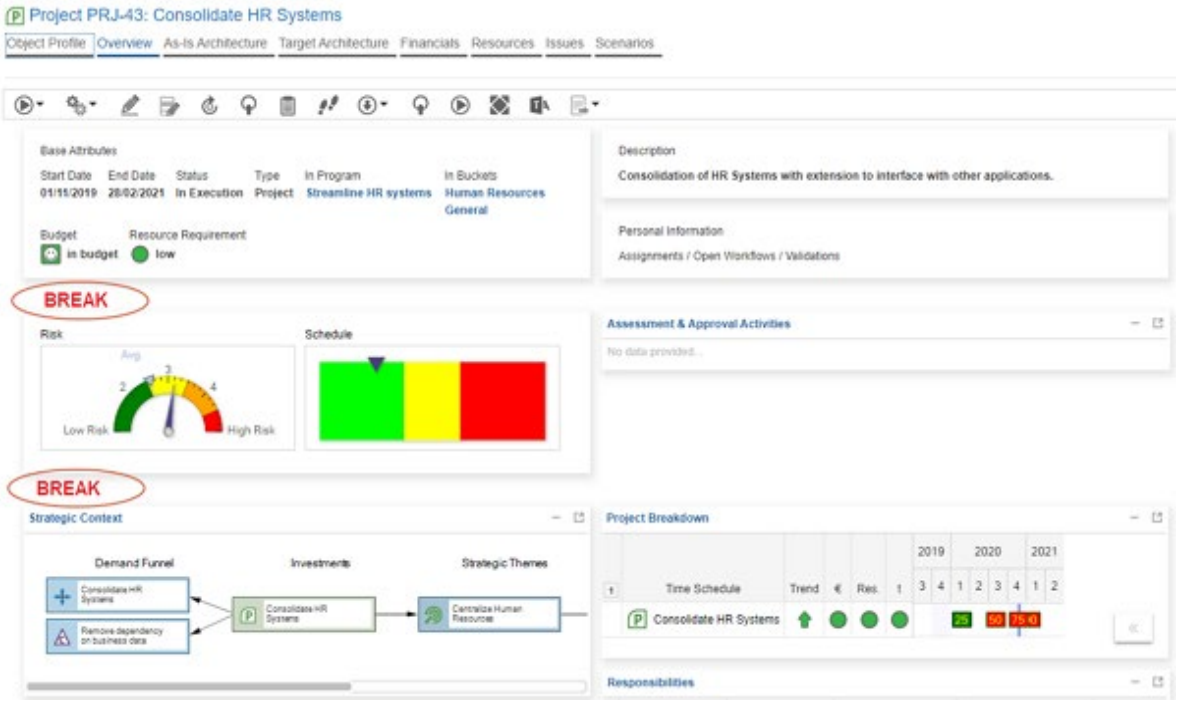


FIGURE: Result of configuration of line breaks

A break `<break>` element can be added in the context of a Group Box interface control to structure the various interface controls in the group box. In this case, the break `<break>` element causes a linebreak and adds a separator line in the group box. The linebreak with separator line is only displayed in a Group Box interface control for which the **Sub-Type** attribute is set to `Attributes`. The functionality of the break `<break>` element remains unchanged otherwise.

To add a break `<break>` element to the object cockpit:

1) In the toolbar of the object cockpit designer, click the **Static Text** button and click in the Flow Panel interface control. The Static Text interface control is displayed in the Flow Panel interface control.
2) In the **Caption** attribute, enter `<break>`. Be sure to include the angle brackets.
3) Repeat this throughout the object cockpit to structure its layout.
4) Specify the **Tab Index** attribute to specify the break element (`<break>`) in the sequence of interface controls. The sequence that you specify is the order in which the interface controls and break elements are displayed.
5) Click the **Save** button to save the new linebreak.

Configuring Styles and GUI Scheme Settings for Object Cockpits

You can specify the styling of Flow Panel, Group Box, Presentation Object, and Value Control interface controls embedded in object cockpits:

- Flow Panel interface controls: To specify the skin style of the flow panels (margins, padding, foreground color, and font color) embedded in all object cockpits, specify the **Common View Control Skins** attribute for the relevant GUI scheme. To do so, do the following:

- Expand the **Global Control Skins** section of the attribute window of the relevant GUI scheme, go to the **Common Control Skins** attribute and specify the margins, padding, foreground color, and font color for flow panels as needed. The styling specified in the **Common Control Skins** attribute specified for a GUI scheme will be applied to flow panels in splash screens, guide views, and console reports. Please note that styles cannot be defined for individual Flow Panel interface controls.

- Apply the skin style defined in the GUI scheme to all flow panels in all object cockpits as well as all splash screens, guide views, and console reports. To do so, right-click the **Object Views** node in the **Presentation** tab and select **Turn GUI Scheme Common Container Skin On for All Flow Panel Cockpits**. The **Ignore GUI Scheme Container Skin** will be set to `False` for the Flow Panel interface control for all object cockpits and the GUI scheme settings will apply to the object cockpit.

- Attribute Group interface controls: To specify the font color and font style of the captions of attribute groups displayed in object cockpits, define the **Attributes Group Caption Font** attribute for the relevant GUI scheme. To do so, expand the **Captions Styling** section of the attribute window of the relevant GUI scheme and specify the **Font Color** and **Font Style** attributes as needed.

- Presentation Object interface controls:

- To specify the font color and font style for the captions of the Presentation Object interface controls embedded in flow panels in the object cockpit and guide views, specify the **Flow Panel Presentation Object Caption Font** attribute for the relevant GUI scheme. Please note that this configuration will only be applied to the captions of presentation objects if the **Inline Navigation** attribute has been set to `False` for the Presentation Object interface control. To do so, expand the **Captions Styling** section of the attribute window of the relevant GUI scheme and specify the **Font Color** and **Font Style** attributes as needed.

- To specify the automatic sizing of the heights of Presentation Object interface controls based on their content displayed, specify the following attributes in the relevant GUI scheme. Go to the **Cockpits** section in the attribute window of the GUI scheme, expand the **Cockpit Contained Objects Behavior** section, and specify the following:

- **Automatically Reduce Height**: Set to `True` to ensure that views with no data are shrunk and that minimal white space is displayed in the object cockpit.

- **Height Difference Threshold**: Enter a percentage value to specify the minimal height difference if the content in page views and configured reports embedded in object cockpits has different values. This improves the visualization in that views with marginally different values are stretched so that they are consistent in their layout. A minimum height difference of 20% is recommended.

- To improve the layout of multiple presentation objects embedded in an object cockpit, adjust the padding of the presentation objects. To do so, define the relevant attributes in the **Margin** section in the attribute window of the Presentation Object interface control. For example, set the **All** attribute to 2 to add 2 pixels to the bottom, top, left, and right margins of the presentation object. Set the **Use Style** attribute to `True` for the presentation object.

- Value Control interface controls:

- To specify the font color and font style for the captions of the Value Control interface controls embedded in flow panels of the object cockpits associated with the GUI scheme, specify the **Value Control Caption Font** attribute for the relevant GUI scheme. To do so, expand the **Captions Styling**

section of the attribute window of the relevant GUI scheme and specify the **Font Color** and **Font Style** attributes as needed.

- To specify styles such as the background color, borders, padding, margins, and font of the value displayed for the Value Control interface control, set the **User Style** attribute to `True` in the attribute window of the value control. Expand the **Style** section of the attribute window and specify the relevant styles as needed. Please note that it may not make sense to specify styles for all sub-types available for Value Control interface controls.

For more information about configuring the GUI scheme for a user profile, see the section Configuring GUI Scheme Definitions for the Alfabet Interface in the chapter Configuring the Visualization of the Alfabet Interface as well as a detailed documentation of all GUI scheme attributes in the chapter *Overview of GUI Scheme Attributes* in reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

Sequencing the Content in the Flow Panel

Once you have added all interface controls to the object cockpit, you can specify their sequence. The definition of an explicit tab order informs the browser to position the group boxes in a specified sequence and layout. This is also relevant for users working in the Alfabet interface using keyboard shortcuts. The tab order also determines the order of the content when exported via the **Publish** capability.

The content can be sequenced in two ways.

1) Define the sequence via the **Define Tab Order** functionality: Click in the object cockpit and click the **Format** menu that appears in the top toolbar. Select **Define Tab Order** in the drop-down menu. Blue boxes with numbers will appear on all interface controls starting with the flow panel. The flow panel will have the number 00. The following interface control shall be assigned the number 00,01 then next one 00,02, etc. If a group box has the number 00,02, for example, then the interface controls in the group box shall be assigned the numbers 00,02,01, 00,02,02, 00,02,03, etc. To define the tab order, click in each blue box for each interface control sequentially to set its tab order. Click the **Save** button to save the tab order. You must hide the tab order in order to make any further changes to the configuration of the object cockpit. To do this, reselect **Format** > **Define Tab Order**.

FIGURE: Sequencing interface controls in flow panel

2) Define the sequence of the interface controls via the **Tab Index** attribute on each interface control: Click each interface control in the object cockpit to activate its attribute window. In the **Tab Index** attribute, entering the sequence number to define the interface control's place in the tab sequence. Click the **Save** button to save the number for the interface control. Continue this procedure for all interface controls.

FIGURE: Sequencing interface controls in group box

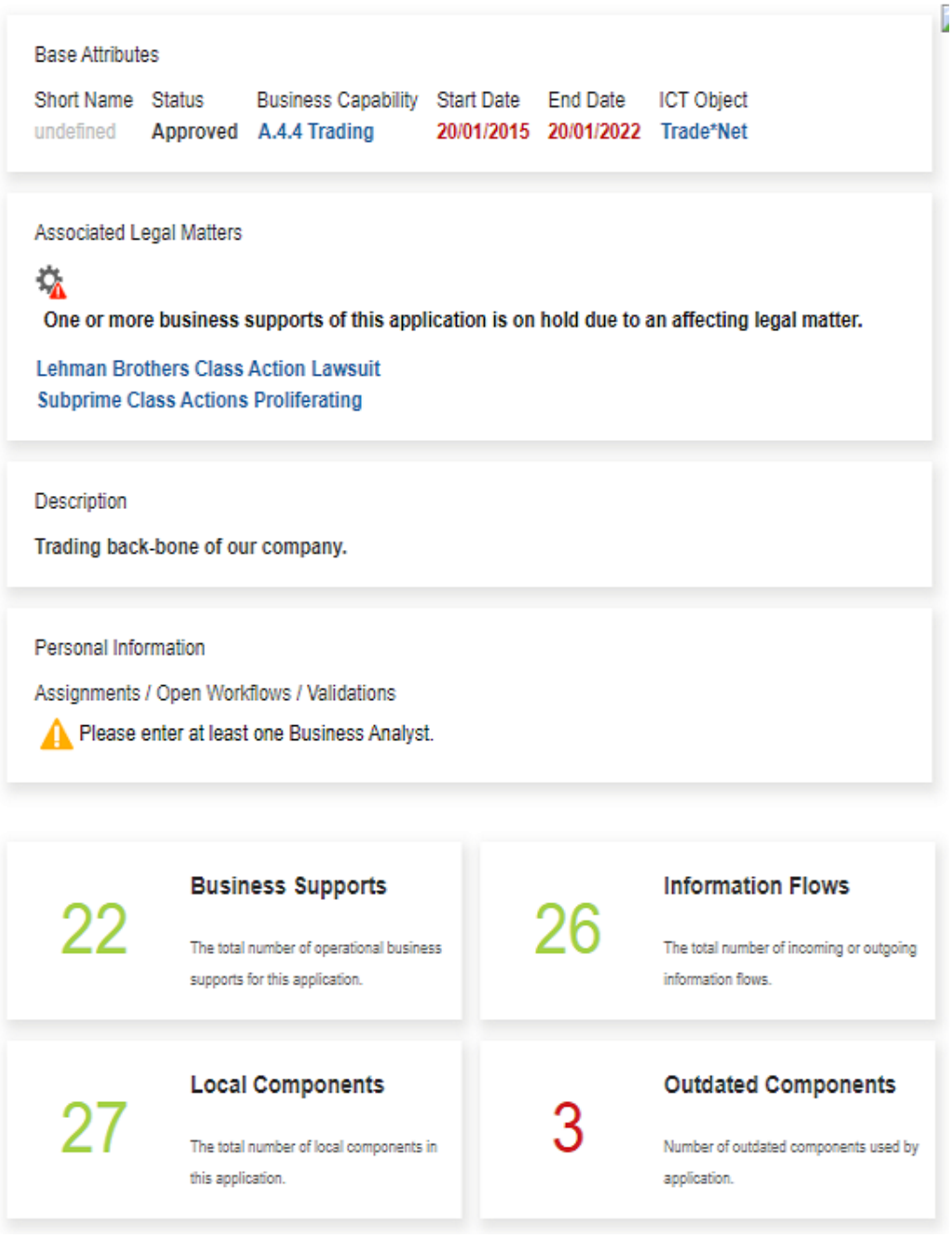


FIGURE: Result of sequenced group box and flow panel

Configuring Drill-Down Navigation in the Object Cockpit

Drill-down navigation can be defined for any of the following:

- An object referenced via a Value Control interface control of the type `Property` or `Query`.
- A page view or configured report specified in the **Source** attribute for a Presentation Object interface control.
- A page view or configured report specified in the **Source** attribute for a Static Text interface control.

If users should be able to navigate via one of the above options, select the interface control in the group box or flow panel and set the **Inline Navigation** attribute to `True` in the attribute window. Click the **Save** button to save your changes. The icon indicating that drill-down navigation is possible will be automatically added to the upper-right corner of the interface control embedded in the object cockpit.

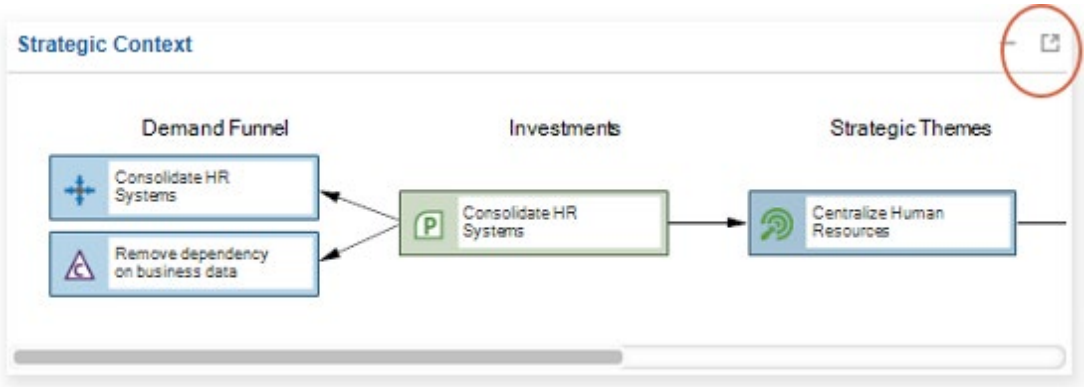


FIGURE: Icon for drill-down navigation

Configuring the Object Cockpit By Means of a Table Grid

The following information describes the configuration possibilities that were available in releases of Alfabet prior to the introduction of HTML 5-based user interface and user interactions. This configuration is based on a Table Layout Panel interface control. Object cockpits rendered in HTML 4 are configured in a static Table Layout Panel interface control that does not allow for dynamic positioning of the interface controls based on the width of the software interface. This method is available for reasons of backward compatibility however it is not recommended to use the Table Layout Panel interface control because the data in the object cockpit may be squeezed, stretched, or truncated when the interface is resized.

Instead, it is recommended that you base the object cockpit on a Flow Panel interface control in order to leverage the browser's layout capability in HTML5. This configuration is recommended and it ensures that object cockpits are not squeezed, stretched, or truncated when the interface is resized. The configuration for object cockpits rendered in HTML 5 ensures that group boxes containing attributes and presentation objects (views and reports) are dynamically positioned according to the current width of the Alfabet Web interface. When resizing the interface, the content displayed in the Flow Panel interface control will be redistributed in the available space and the content will not be cut-off. The Flow Panel interface control ensures that a configured view or set of views will be bumped down and placed below the preceding view or set of views. Business graphics embedded in an object cockpit will be automatically scaled down in size to fit in the available screen space of the device the user is using. Pictures and icons can be implemented as a placeholder that link to views or reports. This is especially useful for large reports for which only a portion

of the report is displayed in the object cockpit. This method is described in the section Configuring the Object Cockpit By Means of a Flow Panel.

Please note that font and hyperlink colors displayed in the object cockpit are determined by the configuration of the GUI schemes implemented in your enterprise. For more information, see the section Configuring GUI Scheme Definitions for the Alfabet Interface.

For more information about specifying the content of the table layout panel, see the relevant information in the section Configuring the Object Cockpit By Means of a Flow Panel:

The following information is available:

- Adding the Table Layout Panel to the Object Cockpit
- Configuring the Header Panel of the Table

Adding the Table Layout Panel to the Object Cockpit

The default size of the Table Layout Panel interface control is 2000 pixel in height and 800 pixel in width. At run time, the object cockpit will be automatically sized to the needs of the content of the object cockpit. However, it may be necessary to resize the object cockpit for the design process if an excessive number of controls must be configured and added to the object cockpit. In order to ensure that business graphics can be adequately viewed on a variety of devices, it is recommended that object cockpits are designed for a lower screen resolution (such as 1280 x 800 or 1280 x 1024). Furthermore, the graphic views embedded in the Table Layout Panel interface control should be vertically positioned rather than horizontally for small display sizes. The object cockpit will be automatically sized to fill the available space in the user interface.

Each table cell in the Table Layout Panel interface control serves as a frame in which you can add content to the object cockpit. The content can either be a static text, property, check entry, indicator, query, icon, page view, configured report, HTML text, URL, or link to a document in the **Internal Document Selector**. The content is added to the table cell via the respective toolbar buttons in the object cockpit designer. Once content has been placed in the table cell, the size of the table cell can be modified, as needed.

Please note that the tab sequence for the cells in the Table Layout Panel interface control is from left-to-right and top-to-bottom. The **Tab Index** attribute defined for the interface controls in each table cell will be ignored. This ensures a consistent navigation in the case of user profiles configured for barrier-free accessibility.

1) Remove the Flow Panel interface control that was automatically included in the object cockpit when it was first created. To do so, click anywhere in the object cockpit and click the **Delete** button in the toolbar.
2) To add a Table Layout Panel interface control to the object cockpit, click the **Table Layout Panel** button and click in the object cockpit designer pane.
3) The default Table Layout Panel interface control is made up of 3 rows and 3 columns. The first row is a header panel that straddles the 3 columns. Click in the Table Layout Panel interface control to activate the attribute window. The **Table Layout** attribute displays the value 3x3 (3 rows x 3 columns). You can edit the **Table Layout** attribute, as needed, and click the **Save** button to save your changes.

The **Height** and **Width** attributes are irrelevant for the configuration of the object cockpit. The object cockpit will be automatically sized to fill the available space in the user interface.

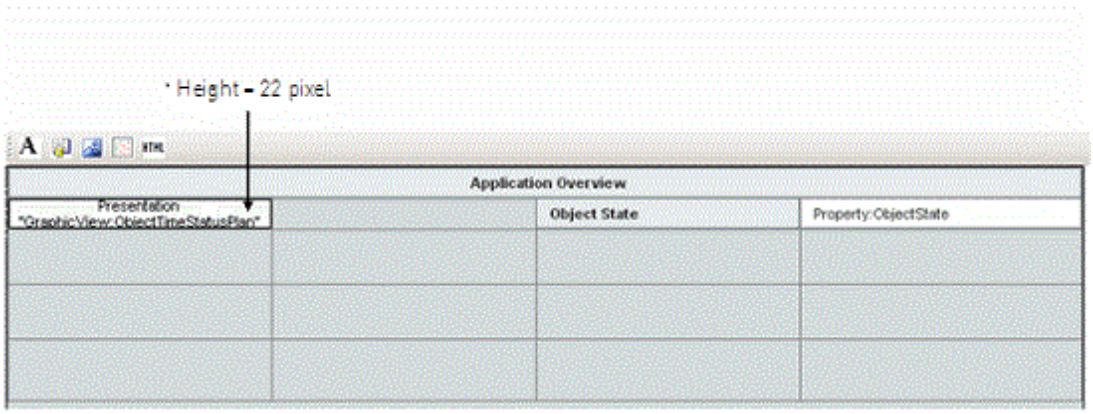
4) You can do any of the following:

- Add content to the table cell in the Table Layout Panel interface control and copy the table cell or adjust its size. To add content to the table cell, click the relevant button in the toolbar and click in the table cell that you want to place it. The following can be added by means of the toolbar in the object cockpit designer pane:
- **Static Text** : Use to add a caption or other instructional text to the table cell. All text in a Static Text interface control in a Table Layout Panel interface control will be displayed in upper-case letters in the Alfabet interface, regardless of its definition in Alfabet Expand.
- **Value Control** : Use to add a property, check entry, indicator, or query to the table cell.
- **Icon** : Use to add an icon to the table cell.
- **Presentation Object** : Use to add a standard page view or configured report to the table cell.
- **HTML Content** : Use to display any static HTML code including, for example, images, links, and forms. The HTML must be XML-conform HTML, compliant with HTML 5, and use standard HTML tags.
- Edit the column span or row span of a table cell containing content. To do so, click the element to open its attribute window. Adjust the values in the **Column Span** and **Row Span** attributes, as needed. You may not enter a number less than 1 nor greater than the corresponding column or row definition in the table's **Table Layout** attribute.
- Copy a table cell containing content and paste it to an empty table cell. This allows you to copy elements that have already been adjusted to have the correct size. To do so, right-click the element you want to copy and select **Copy** in the context menu and right-click the empty table cell and select **Paste**. The copied element can be modified, as needed. The key operations CTRL + C and CTRL + V can be used to copy and paste cockpit elements.
- Change the table layout value to add an estimated total number of rows and columns. To do so, click the table to view the attribute window in the right pane. The **Table Layout** attribute will display the maximum number of defined rows and columns in the syntax <rows>x<columns>. The default value is 3x4. Edit this value, as needed. Please note that additional columns and rows can later be added ad-hoc. The **Table Layout** attribute will be automatically updated as the table is built.

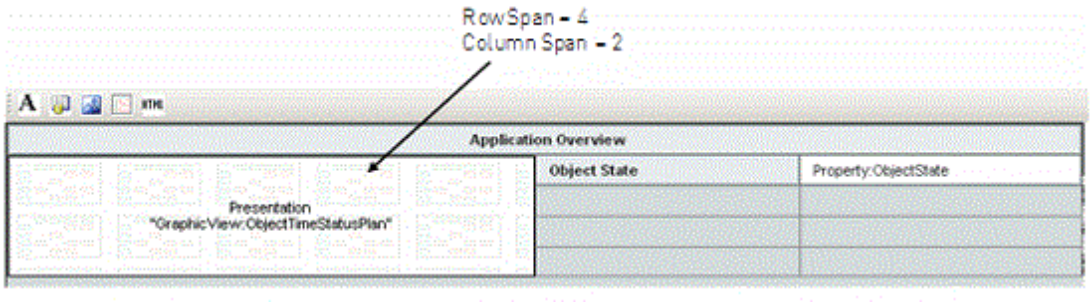
5) Once you have modified the table, click the **Save** button to save your changes.

To create an object cockpit with table cells that span over several rows or columns of the table like that displayed above, you would proceed as follows:

1) Add the relevant content (for example, a presentation object) to the relevant cell. In this case, the cell in the first row and column. Set the **Height** attribute to a small value such as 22 pixel.

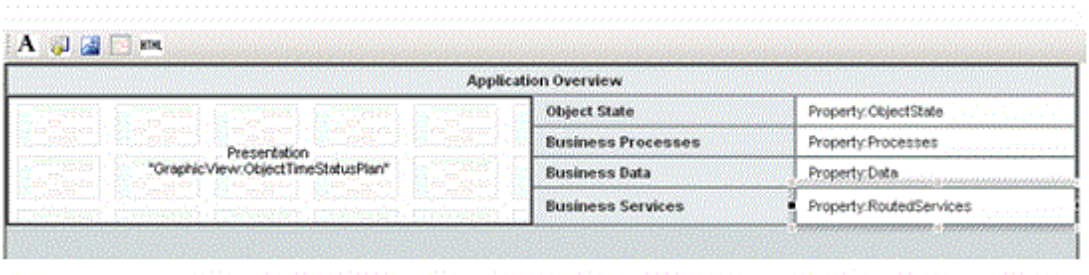


2) Add additional content to the remaining columns in the first row. The **Height** attributes will automatically be set to 22 pixel.



3) Next, define the **Row Span** attribute of the table cell that should span one or more rows. In the example, the **Row Span** attribute has been defined as 4 and the **Column Span** attribute has been set to 2. Remember that the total value of all **Row Span** attributes of all rows may not exceed the number of rows in the **Table Layout** attribute of the object cockpit. In this case, the table layout is 5 x 4, whereby the first row displays a caption for the object cockpit.

Please note that the configured row span and column span is ignored for a page view displaying a diagram when the object cockpit is published to a DOC or PDF file. In this case, the size of the diagram will be decreased in the publication.



4) Configure the remaining cells in the object cockpit accordingly. All cells must be filled with content and no cell may remain empty.

5) In the toolbar, click the **Save**  button to save your changes.

Configuring the Header Panel of the Table

The default table that is automatically generated includes a header panel for the first row. This row contains a Static Text interface control, per default.

If the number of columns of the default table has been modified, you can adjust the size of the header panel. Furthermore, you can adjust the height of the row and specify a caption to display in the header row of the object cockpit.

1) Click the Static Text interface control in the table to activate its attribute window. Define the following attributes, as needed.

- **Caption**: Enter the text that should be displayed as the title of the object cockpit: The caption will be displayed in upper-case letters in the user interface, regardless of their definition in Alfabet Expand.
- **Automatic Sizing**: Select `False` if you want to adjust the height of the row. This allows you to define a distinct size for the table cell containing the static text element.
- **Height**: Enter a pixel value for the height of the header panel.
- **Column Span**: Enter the total number of columns if the number of columns has changed and you want the header panel to straddle all columns.
- Expand the **Style** section to define the style and formatting of the text, as needed.

2) In the toolbar, click the **Save** button to save your changes.

Defining the Default Object Cockpit for a Custom Object View

If multiple object cockpits have been defined for a custom object view, you can define a default object for a specified view scheme.

To define a default object cockpit:

1) Right-click the custom object view and select **Configure Object View**. The **View Configuration** editor opens. The selected object view is displayed in the **Select Object** field.
2) In the **Select View Scheme** field, select the relevant view scheme that you are configuring the object view for.
3) If you are working with an object class for which object class stereotypes have been defined, select the relevant object class stereotype in the **Select Base Class** field.
4) In the table, scroll down to the table section labeled **Cockpits** and in the drop-down menu in the **Default Cockpit** column, select the object cockpit that should be displayed as the default for the selected view scheme. Click the **OK** button to save your changes and close the editor.
5) In the toolbar, click the **Save** button to save your changes.

For additional information about configuring the object view, see the sections Defining the Visibility of Object Class Properties, Workspaces, Page Views/Configured Reports, and Object Cockpits in an Object View and Hiding Functionalities in an Object View in the chapter Configuring User Profiles for the User Community.

Adding Data Quality Widgets to Object Cockpits

Additional information about data in a object cockpit can be provided via a quality widget in a pop-up window. The quality widget is either a configured report or a standard Alfabet view that can be assigned to multiple object cockpits as well as configured reports. When the user opens or loads the object cockpit that the quality widget is assigned to, the quality widget will be displayed in a separate pop-up window that opens for a short time. The quality widget pop-up window opens by default in the upper right corner of the Alfabet user interface below the main menu. This position is configurable. Depending on the configuration of the primary configured report, either the complete quality widget will be displayed or only the caption will be displayed in a title bar and the user can click the title to open the quality widget.

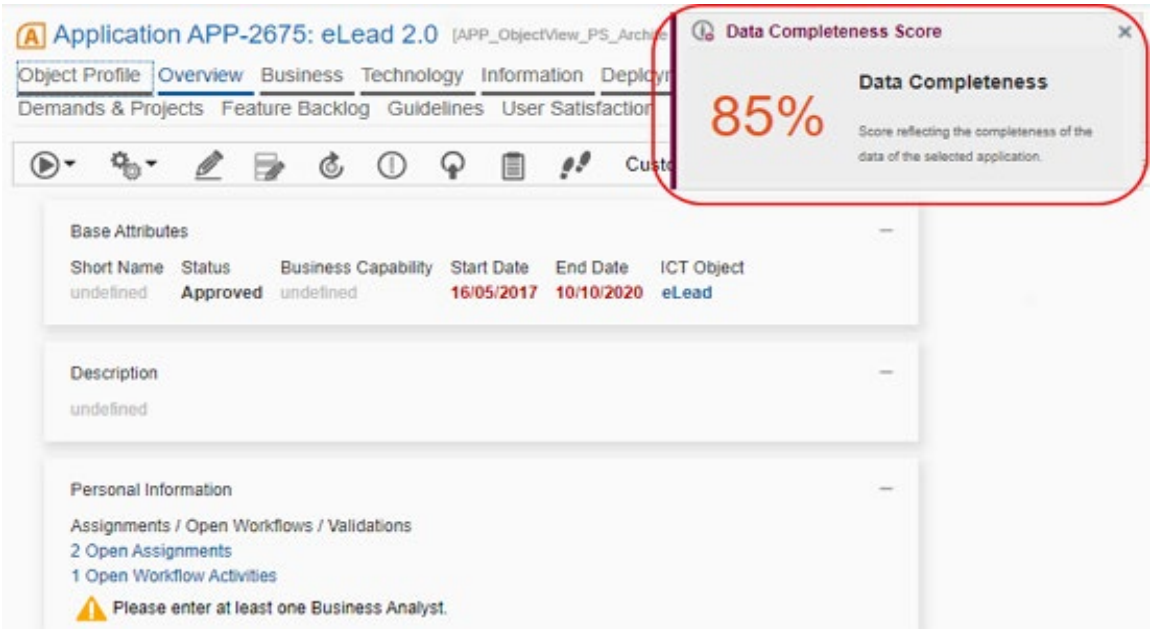


FIGURE: Secondary view displaying quality widget based on a configured report

A typical use case for quality widgets is a widget report that opens to provide information to the user about the quality of the data in a object cockpit. For example, the user could be provided with information about the completeness of the indicators specified for the object in the object cockpit. A configured link available in the quality widget could then open a view that allows the user to provide the missing data and thus correct the issue. When the user clicks the link available in the quality widget report, the view will open in a new tab. The user can then define the relevant indicators, return to the original tab, refresh the view and view the configured report with the updated data.

The following additional configuration is required for secondary views:

- The configured reports that are implemented as quality widgets must first be configured before they can be assigned to an object cockpit, wizard, or wizard step. For more information about the configuration of the configured reports, see the section Integration of Quality Widgets in Configured Reports, Object Cockpits and Wizards.
- The visualization of secondary window skins is configured via the **Quality Widget** attributes for a GUI scheme. For more information, see the section Configuring GUI Scheme Definitions for the Alfabet Interface as well as the section *Attributes Displayed in the Grid Section: Secondary Window Skins* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

To assign a quality widget to an existing object cockpit:

1) Right-click the object cockpit and select **Add Secondary View**. A secondary view is added to the explorer and the attributes of the secondary view are displayed in the attribute window.
2) Define the following attributes of the secondary view:

- **Name**: Enter a unique name for the secondary view. The name is a technical name and shall not contain special characters or whitespaces. The technical name is not displayed in the user interface.
- **Caption**: Enter a caption for the secondary view that shall be displayed in the title bar of the secondary view in the Alfabet user interface and in the tooltip displayed for the icon of the secondary view in the slide-in toolbar. Please note that the caption should be short. It will not be displayed in the title bar if it is too long.
- **Role**: Select `QualityWidget`.
- **View Type**: Select `Report`.
- **View Name**: Select the configured report that shall be displayed as secondary view. The drop-down list displays the names of all available configured reports of the allowed report types widget report, gantt chart report, and business chart report.
- **Show Automatically**: Select `True` if you would like the secondary view to be fully displayed for a few seconds when the user accesses the object cockpit or configured report that the secondary view is defined for. Select `False` if only the caption of the secondary view shall be displayed in a title bar for a few seconds when the user accesses the object cockpit or configured report that the secondary view is defined for. The secondary view will then open if the user clicks the title bar.
- **Position**: Select the position of the open quality widget on the Alfabet user interface. The quality widget window can be placed in the center of the Alfabet user interface or in one of the corners of the user interface beneath the main toolbar. The quality widget pop-up window opens by default in the upper-right corner of the Alfabet user interface below the main menu.
- **Height**: Define the inner height of the quality widget window in pixel. This is the height available for the display of the configured report opened in the window.
- **Width**: Define the inner width of the quality widget window in pixel. This is the width available for the display of the configured report opened in the window.

3) In the toolbar, click the **Save** button.

Configuring Inline Editing of Attributes in the Object View

Inline editing of scalar and reference properties may be configured so that users can edit data directly in the object profile and object cockpit. If inline editing is supported, users can edit scalar attributes for an existing object directly in the **Attributes** section of an object profile or in an object cockpit without needing to open the object's editor or wizard.

Please note the following regarding the inline editing functionality:

- Only scalar and reference properties that are editable in the editor/wizard available in the object profile or object cockpit can be edited via inline editing.
- All unique constraints defined for the class as well as any post-conditions configured for the wizard associated with the object profile/object cockpit will be applied to the data entered.

  Please note that data is saved to the Alfabet database prior to the post-condition being validated. Therefore, data entered via inline editing will be saved even if the post-condition is not fulfilled. This is because the data must be available in the Alfabet database in order to be evaluated for the post-condition. If the data entered does not fulfill the post-condition, the configured warning message will be displayed explaining that the input must be corrected in order to fulfill the post-condition.

- Inline editing may be limited or prevented if the syntax of the post-condition is not correct. In this case, an error will be displayed if the user tries to open the wizard.

For more information about configuring post-conditions for wizard steps, see the section Defining a Post-Condition for a Wizard Step.

When the user points to the relevant interface control, the **Edit**  icon will be displayed next to the attribute to indicate the object can be edited inline. Users can click in the field and depending on the type of field, define text, select a value in a drop-down list, or specify a date. Upon completion, the user can then click the **Save**  button to save the definition to the Alfabet database or click the **Cancel**  button to remove the definition.

Permissibility of inline editing can be specified by your solution designer on the level of an object view, class setting, or GUI scheme. If the **Prevent Inline Editing in Object Views** is not explicitly defined, the default value `False` will be applied. Please note the following:

- If the GUI scheme attribute **Prevent Inline Editing in Object Views** is set to `True`, then inline editing is not permissible for all object profiles and object cockpits in all object views relevant for the user profile that the GUI scheme is assigned to.
- If the GUI scheme attribute **Prevent Inline Editing in Object Views** is set to `False`, then inline editing is permissible for all object profiles and object cockpits in all object views relevant for the user profile that the GUI scheme is assigned to. This setting can be overruled for an individual class or object view. Please consider the following if the GUI scheme attribute **Prevent Inline Editing in Object Views** is set to `False`:

- If you set the class setting attribute **Prevent Inline Editing in Object Views** to `True`, then inline editing will not be permissible for the relevant object class.
- If you set the class setting attribute **Prevent Inline Editing in Object Views** to `False` and set the object view attribute **Prevent Inline Editing in Object Views** attribute to `True` for an object view of the relevant class, then inline editing will not be permissible in the object view.
- If the **Prevent Inline Editing in Object Views** attribute is set to `False` for the GUI scheme, class setting, and object view, you can specify that an individual Value Control interface control is not permissible by setting the **Read-Only** attribute of the interface control to `True`.

- If **Prevent Inline Editing in Object Views** is set to `False`, inline editing may be disabled for an interface control in an object cockpit if the **Read-Only** attribute is set to `True` for the interface control.

Inline editing is not possible in object views in which the **Edit** button has been excluded for the view scheme. For more information about hiding buttons, see the section Hiding Functionalities in an Object View.

For more information about configuring GUI schemes, see the section Configuring GUI Scheme Definitions for the Alfabet Interface in the chapter Configuring the Visualization of the Alfabet Interface as well as a detailed documentation of all GUI scheme attributes in the chapter *Overview of GUI Scheme Attributes* in reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

Configuring the Toolbar of the Object View

Various aspects of the object view toolbar can be configured. Standard toolbar buttons can be hidden in the context of a user profile as described in the section Hiding Functionalities in an Object View in the chapter *Configuring User Profiles for the User Community*.

Furthermore, styles can be configured for the toolbar in all object views relevant to a user profile. Additionally, custom buttons that will open a specified editor, wizard, standard Alfabet view, or configured report can be configured to be displayed in a toolbar of a custom object view.

The following information is available:

- Configuring Styles for the Toolbar in an Object View
- Configuring Custom Buttons for the Toolbar of a Custom Object View

Configuring Styles for the Toolbar in an Object View

You can specify the borders, background color, and grey scaling of images for the toolbar displayed in object views via the **Object Profile Toolbar Style** attribute in the GUI scheme.

To do so, expand the **GUI Schemes** node in the **Presentation** tab and select the relevant GUI scheme you want to define. Expand the Toolbars section and then expand the **Object Profile Toolbar Style** attribute to specify the **Background Color**, **Border Color**, **Border Width**, and **Render Images in Grey Scale** attribute attributes, as needed. Please note that the **Render Images in Grey Scale** attribute will not apply to Microsoft Internet Explorer.

For more information, see the section Configuring GUI Scheme Definitions for the Alfabet Interface in the chapter Configuring the Visualization of the Alfabet Interface as well as a detailed documentation of all GUI scheme attributes in the chapter *Overview of GUI Scheme Attributes* in reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

Configuring Custom Buttons for the Toolbar of a Custom Object View

You can create a button to display in the toolbar of the custom object view that will open a specified editor, wizard, standard Alfabet view, or configured report.

One of the use cases for the custom button is to provide configured audit information to users accessing the custom object view. Because the **Object Audit** view displayed for a selected object displays all information about the object independent of the access permission available to the user viewing the audit information, you may want to consider configuring a configured report to display audit information in accordance with the visibility rules of the associated user profile(s). Please note the following configuration requirements.

- Create a native SQL-based configured reports to extract specific information from the audit history such as the changes made to the object by a specific user or specific types of changes made to the object (for example, new information flows added to an application). For information about the structure of the database table for auditing and how to configure such a report, see the section Defining Audit Management Related Configured Reports in the chapter Configuring Reports. Once the report is configured, you must make it available in the relevant custom object view by configuring a custom button for the relevant custom object view, as described below.
- The standard **Audit Trail**  button can be hidden from the relevant custom object view in the context of the user profile configuration. For more information, see the section Hiding Functionalities in an Object View in the chapter Configuring User Profiles for the User Community.
- For general information about how to make the auditing capability available for an object class, see the section Specifying History Tracking for an Object Class in the chapter Configuring the Class Model.

To create a new button for a custom object view

1) Right-click the new custom object view  and select **Create Button**. The new button is displayed below the custom object view node.
2) Click the new button to activate the attribute window and define the following attributes, as needed:

- **Button Type**: Select `Action`. When a user clicks the button displayed in the object view, the specified editor, wizard, view or configured report will open.
- **Name**: Enter a unique name for the button. Special characters are not allowed. This value is displayed in the solution interface if the **Caption** attribute has not been defined.
- **Caption**: Enter the caption that should be displayed on the button. If no caption is defined, the value in the **Name** attribute will be displayed on the button.
- **Extract Caption for Translation**: Set to `True` if the button caption shall be extracted to the Meta-Model vocabulary for translation. Set to `False` if the button caption shall not be extracted to the Meta-Model vocabulary for translation.
- To specify the editor, wizard, standard Alfabet view, or configured report that will open when the button is clicked:
- *For editors*:

- **Operation**: Select `Edit` to specify that a standard editor should be linked to the custom button.
- **View**: Select the standard editor that should open when the button is clicked by a user:

- *For wizards*:
  - **Operation**: Select `WizardEdit` to specify that a standard or custom wizard should be linked to the custom button.
  - **View**: Enter `Wizard`:<NameOfConfiguredWizard>. You must spell the name of the wizard exactly as it is spelled in the **Name** attribute of the custom wizard.
- *For standard Alfabet views*:
  - **Operation**: Select `Navigate` to specify that a standard Alfabet view should be linked to the custom button.
  - **View**: Select the relevant view in the drop-down list.
- *For configured reports:*
  - **Operation**: Select `Navigate` to specify that a configured report should be linked to the custom button.
  - **View**: Enter `Report`:<NameOfConfiguredReport>. You must spell the name of the report exactly as it is spelled in the **Name** attribute of the configured report.
- **Apply Operation To**: Select `Base Instance` to specify that the data in the element specified in the **View** attribute is for the object that the user is currently working with.
- **Disable In Show Mode**: Select `True` if the button should only be available to the users with ReadWrite access permissions to the object. Select `False` if the button should be available to any user accessing the object's object profile.
- **View Mode**: Select `Edit` if the user opening the view or configured report should have Read/Write access permissions to the view or select `ReadOnly` if the user should have Read/Only access permissions to the view.
- **Icon**: Select an icon to display on the button. The icon should not be larger than 22 x 22 pixel.
- **Hint**: Enter text for a tooltip. The tooltip will be displayed when the user points to the button.

3) To sort the order of the buttons created for the object profile, click the object view  that the buttons have been created for to activate the attribute window. In the **Buttons** attribute, click the **Browse**  button. An editor opens that allows the sequence of the buttons in the toolbar to be defined via the **Up/Down**  button. Sort the buttons and click O **K** to close the editor.

4) In the toolbar, click the **Save**  button to save your changes. You can now add or remove page views to the custom view.

Making the Custom Object View Available to the User Community

Once the custom object view and, if relevant, its object cockpits, have been configured, it can be implemented in a user profile. To do so, the following must be carried out:

- The object view must be assigned to a class setting that will be implemented in the user profile. This is described in the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.
- The class setting must then be assigned to a view scheme. This is described in the section Configuring a View Scheme for a User Profile in the chapter Configuring User Profiles for the User Community.
- The object view can then be further configured in the context of the view scheme it is assigned to. At this point, you can choose to hide the visibility of object class properties, page views, configured reports and entire workspaces. This is described in the section Modifying the Object View for Implementation in Multiple User Profiles. You an also hide functionalities available in the toolbar of the object view. This is described in the section Hiding Functionalities in an Object View in the chapter Configuring User Profiles for the User Community.
- Finally, the view scheme that the object view is associated with must be assigned to a user profile to make it available to the Alfabet user community. This is described in the section Assigning the View Scheme to a User Profile in the chapter Configuring User Profiles for the User Community.

Modifying the Object View for Implementation in Multiple User Profiles

Once the object profile and object cockpits have been configured for the custom object view, you can modify the custom object view so that it can be reused in various view schemes that are assigned to different user profiles. The object view can be individually modified for each view scheme in which it will be implemented. For each view scheme, you can specify which workspaces, page views, configured reports, attributes, etc. should be displayed in the custom object view. In this way, a custom object view can be very broadly configured and refined for the user profile context in which it will be implemented.

The following is possible:

- Refine the visibility of various elements of the custom object view for multiple view schemes. The configuration possibilities include:
- Hide standard or custom object class properties assigned to the **Attributes** section of the object profile
- Hide individual page views and configured reports.
- Hide an entire workspace. When you hide an entire workspace, all page views assigned to the workspace will also be hidden.
- Hide individual object cockpits
- Specify an object cockpit as the default object cockpit that is displayed when the user navigates to the object view. The user can switch to another cockpit, as needed.

- Specify which functionalities are available in the toolbar of the object view/object cockpit. This is described in the section Hiding Functionalities in an Object View in the chapter Configuring User Profiles for the User Community.
- Hide the object profile in a custom object view. This is configured on the level of the class setting.
- Specify which workspaces, page views, configured reports, or object cockpits should be hidden in a custom object view when the current object is based on the value defined for a specified standard or custom property. For example, if a custom object view is configured for the object class `Device`, you could specify that different object cockpits are hidden when accessing the object view for the object class stereotype Physical Device vs. Virtual Device.

Visibility for interface controls in object profiles and object cockpits can also be configured via conditional constraints. This is described in detail in the sections Configuring Conditional Constraints for an Object Profile and Configuring Conditional Restraints in the Object Cockpit.

The following information is available:

- Configuring Visibility Issues of the Custom Object View for a View Scheme
- Excluding the Object Profile from a Custom Object View
- Hiding Aspects of a Custom Object View Based on Standard or Custom Properties

Configuring Visibility Issues of the Custom Object View for a View Scheme

The visibility of object class properties, workspaces, page views, and configured reports can be specified for a standard object view or custom object view. Additionally, you can specify which object cockpits are visible for the user profile as well as which object cockpit is the default object cockpit that the user will navigate to the first time in the user session.

Please note that the first time that a user navigates to an object view for a specific object class, the object cockpit configured as the default cockpit will be displayed. If no object cockpit is specified as the default for the user profile, the user will automatically navigate to the first object cockpit in the list of object cockpits in the **Default** field in the **View Configuration** editor available via a user profile. If no default object cockpit has been configured, the object profile will be displayed per default.

Please note that this configuration is done at the level of the individual object view. Each relevant object view must be explicitly configured. For each view scheme assigned to a user profile, the following should be configured:

- Hide standard or custom object class properties assigned to the **Attributes** section of the object profile
- Hide individual page views and configured reports.
- Hide an entire workspace. When you hide an entire workspace, all page views assigned to the workspace will also be hidden.
- Hide individual object cockpits

- Specify an object cockpit as the default object cockpit that is displayed when the user navigates to the object view. The user can switch to another cockpit, as needed.
- Specify which functionalities are available in the toolbar of the object view/object cockpit. This is described in the section Hiding Functionalities in an Object View.

To configure the visibility of object class properties, page views, configured reports, workspaces, and object cockpits in an object view:

1) Open the Alfabet interface by right-clicking the relevant user profile and selecting **Configure User Profile**.
2) Navigate to the object view that you want to configure and click the **Configure View**  button in the upper right corner of the view. The **Customization Editor** opens.
3) Define the following as needed:

- **Select Configuration Object**: Displays the current configuration object that is being configured.
- **Select View Scheme**: Select the view scheme that the configuration is relevant for.
- **Select Hierarchy Level**. Select the hierarchy level for which the configuration is being defined. The levels are listed in the order of the hierarchy, starting with **All Classes**, followed by the object class and then the object class stereotypes.
- **Currently Applied Settings**: Displays the inherited configuration that is applied to the selected configuration object if the option **Inherited Settings from Parent Configuration Object** is selected in the **Choose One Settings Mode** menu. This field will be empty if settings have not been inherited.
- Select the relevant option in the **Choose One Settings Mode** menu in the toolbar of the dataset. The options available will depend on the configuration object that you are working with:
- **Explicit Settings from Selected Configuration Object**: Select to display a dataset that allows visibility to be explicitly specified for the relevant aspects of the selected configuration object displayed in the **Select Configuration Object** field.
- **Inherited Settings from Parent Configuration Object**: Select if the selected configuration object shall inherit the exclusion settings configured for the relevant parent configuration object. The **Currently Applied Settings** field will display the inherited configuration that is applied to the selected configuration object if the option **Inherited Settings from Parent Configuration Object** is selected in the **Choose One Settings Mode** menu.
- **No Explicit Settings from Configuration Object**: Select to display the current settings for the selected configuration object. This dataset allows no explicit settings to be configured. The **No Explicit Settings from Configuration Object** option is displayed instead of the **Inherited Settings from Parent Configuration Object** option if no settings can be inherited.

4) If **Explicit Settings from Selected Configuration Object** has been selected in the **Choose One Settings Mode** menu, you can configure the visibility of the **Attribute Section**, a section for each workspace, and a **Cockpits** section. Expand the section of the table in order to display the respective attributes, views, or cockpits that you want to configure by clicking the + symbol. If necessary, navigate to the next page in the **View Configuration** editor. The **View Configuration** editor displays the following columns:

- **Name**: Displays the name of the interface element in the object view (**Attribute** section, attributes, reports, workspaces, views, etc.) in the object view.

- **Caption**: Displays the caption defined for the interface element in the object view.
- **Type**: Displays the type of interface element in the object view (`Attribute`, `Report`, `Workspace`, `Cockpit`, etc.)
- **Excluded**: Click in the cell to set an X to specify that the respective interface element in the object view should be hidden. For example, to hide an object cockpit, scroll to the **Cockpits** section of the table and set an X in the **Excluded** column for the object cockpit that you want to hide for the selected view scheme.
- **Default Cockpit**: Click in the cell to set an X to specify that the object cockpit is the default view. The first time that a user navigates to an object view for a specific object class, the default object cockpit configured will be displayed. The user can switch to another object cockpit as needed. If no object cockpit is specified as the default for the user profile, the user will automatically navigate to the first object cockpit in the list of object cockpits in the **Default** field in the **View Configuration** editor of the user profile. If no object cockpits have been configured, the object profile will be displayed per default.
- **Usage Type**: Displays the specification of the user type that the view is intended for (`All`, `Viewer`, `Data Entry`, `Functional`, `Expert`).

5) Click the **OK** button to save the changes in the **Customization Editor** editor or click **Cancel** to exit without saving the changes.

To configure the visibility of object class properties, page views, configured reports, workspaces, and object cockpits in an object view:

1) Right-click the new custom object view  and select **Configure Object View**. The **View Configuration** editor opens.
2) In the **Select View Scheme** field, select the view scheme that the configuration is relevant for.
3) In the **Choose One Settings Mode** menu, select the **Explicit Settings from Selected Configuration Object** option to configure the visibility of the **Attribute Section**, a section for each workspace, and a **Cockpits** section.
4) Expand the section of the table in order to display the respective attributes, views, or cockpits that you want to configure by clicking the + symbol. If necessary, navigate to the next page in the **View Configuration** editor. The **View Configuration** editor displays the following columns:

- **Name**: Displays the name of the interface element in the object view (**Attribute** section, attributes, reports, workspaces, views, etc.) in the object view.
- **Caption**: Displays the caption defined for the interface element in the object view.
- **Type**: Displays the type of interface element in the object view (`Attribute`, `Report`, `Workspace`, `Cockpit`, etc.)
- **Excluded**: Click in the cell to set an X to specify that the respective interface element in the object view should be hidden. For example, to hide an object cockpit, scroll to the **Cockpits** section of the table and set an X in the **Excluded** column for the object cockpit that you want to hide for the selected view scheme.
- **Default Cockpit**: Click in the cell to set an X to specify that the object cockpit is the default view. The first time that a user navigates to an object view for a specific object class, the default object cockpit configured will be displayed. The user can switch to another object cockpit as needed. If no object cockpit is specified as the default for the user profile, the user will automatically navigate to the first object cockpit in the list of object cockpits in the **Default** field in the **View Configuration** editor of the user profile. If no object cockpits have been configured, the object profile will be displayed per default.

- **Usage Type**: Displays the specification of the user type that the view is intended for (`All`, `Viewer`, `Data Entry`, `Functional`, `Expert`).

5) Click the **OK** button to save the changes in the **View Configuration** editor or click **Cancel** to exit without saving the changes.

Excluding the Object Profile from a Custom Object View

You can exclude the object profile for a relevant class setting so that it is not displayed in the Alfabet user interface when a user with the user profile accesses the custom object view: To hide the object profile, navigate to the relevant class setting assigned to the view scheme of the relevant user profile. In the attribute window of the class setting, set the **Suppress Object Profile** attribute to `True`. In this case, the object profile will be hidden and only the object cockpits will be displayed. For more information about defining class settings, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes.

Hiding Aspects of a Custom Object View Based on Standard or Custom Properties

The XML object ***ObjectViewCustomFilters*** allows you to explicitly specify which workspaces, page views, configured reports, or object cockpits should be hidden in a custom object view when the current object is based on the value defined for a specified standard or custom property. For example, the following is possible:

- If a custom object view is configured for the object class `Device`, you could specify that different object cockpits are hidden when accessing the object view for the object class stereotype Physical Device vs. Virtual Device.
- If a custom object view is configured for an object class stereotype Business Application, you could specify that different workspaces are hidden when accessing the business applications with the object state set to Active vs. business applications with the object state set to Retired.

Please note that the XML object ***ObjectViewCustomFilters*** is a blacklist that allows you to specify the content that shall **not** be available when users access custom object views for the specified object class stereotype.

To edit the XML object ***ObjectViewCustomFilters***:

1) Go to the **Presentations** tab, expand the **XML Objects** folder, and expand the **SolutionManagers** folder.
2) Right-click the XML object ***ObjectViewCustomFilters*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see the section Working with XML Objects.
3) In the XML attribute ***ObjectViewFilter***, specify the following XML attributes for each object class stereotype relevant for the project scenario configuration:

For example, to hide an object cockpit displaying information about virtual devices:

```
<ObjectViewFilter ClassName="Device" Property="Stereotype"
Value="PhysicalDevice" >
```



```
<Item Type="Cockpit"
Name="DVC_ObjectView_VitualDevice_Overview" />

</ObjectViewFilter>
```

- `ClassName`: Enter the name of the object class.
- `Stereotype`: If you specifying the custom object view configured for an object class stereotype, enter the name of the object class stereotype.
- `Property`: Enter the name of the object class property upon which the filtering should be based. For example, you might specify the `Stereotype` property for an object class, or you might specify another standard or custom property such as `State` or `Status` that might be relevant for an object class or object class stereotype.
- `Value`: Enter the value of the property for which the content specified in the XML attribute `Type` should be hidden.

4) Add a child XML element ***Item*** for each workspace, page view, configured report, or object cockpit that should be hidden for the specified object class/object class stereotype:

- `Type`: Enter any of the following:
- "`WS`" to specify a workspace that should be hidden.
- "`View`" to specify a page view or configured report that should be hidden.
- "`Cockpit`" to specify an object cockpit that should be hidden.
- `Name`: Enter the technical name of the relevant workspace, page view, configured report, or object cockpit that should be hidden.

5) In the toolbar, click the **Save**  button to save the XML definition.

Providing Custom Online Help for an Object View

You can provide custom context-sensitive help for any custom object view or object cockpit configured by your solution designer. You can specify only one custom context-sensitive help link per object view and only one custom context-sensitive help link per object cockpit. Users will be able to access the context-sensitive help for the object profile or object cockpit that is currently displayed in the Alfabet user interface.

The custom context-sensitive help that you provide for your user community must be available via a URL that can be viewed in a browser. For each custom object view, you can decide whether custom context-sensitive help should be displayed and whether the standard context-sensitive help should or should not be displayed. If you specify that both the standard context-sensitive help and the custom context-sensitive help are to be displayed, the link for the standard context-sensitive help will be listed first in the **Help Selector**, followed by the custom context-sensitive help link. The links will be displayed in the **Help Selector** with the syntax **Custom Help on<Object View Caption>** or **Custom Help on<Object Cockpit Caption>**. For general information about the standard and custom context-sensitive help, see the section Understanding the Context-Sensitive Help.

Furthermore, you can specify content for the automated help assistant and make it available to a selected object view and/or object cockpit. The automated help assistant will be displayed in a fly-in element in the

upper-right corner of the user interface when the user accesses the object view or object cockpit in the Alfabet user interface. The automated help assistant can be closed, whereby it will drop into the slide-in toolbar and can be opened for the current view at another time. For general information about the automated help assistant capability, see the section Understanding the Automated Help Assistant.

Server variables can be used in the **Custom Context-Sensitive Help URL** attribute and the **Automated Assistant URL** attribute to specify the custom help links. Server variables allow you to define all or part of the URL definition in the alias server setting instead of directly defining it in the configuration in Alfabet Expand. Use of server variables is, for example, useful in order to change a Web server name defined in multiple URL definitions in Alfabet Expand. Instead of changing all URL definitions, only the variable value in the alias server setting must be updated. For more information about defining server variables for custom online help, see the section Configuring Server Variables for the Custom Help.

For more information about defining tooltips for the custom and protected properties that are displayed in the **Attributes** section of an object view, see the section Providing Custom Tooltips for Custom and Protected Object Class Properties.

To specify custom help for a custom object view or object cockpit:

1) Go to the **Presentation** tab, expand the **Object Views** explorer node and navigate either to the standard object vie w or object cockpit that you want to define a custom help for.
2) Define the following attributes, as needed.

- **Automated Assistant URL**: Enter the URL or server variable that targets the content to display in the automated help assistant assigned to the object view or object cockpit.

  The size of the automated help assistant, its header color and position in the Alfabet user interface, and the notch in the slide-in toolbar are configured for each configuration object type in the GUI scheme. For more information about defining GUI schemes, see the section Configuring GUI Scheme Definitions for the Alfabet Interface. For details about the individual GUI scheme attributes that are relevant for the automated help assistant, see the section *Overview of GUI Scheme Attributes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

- **Custom Context-Sensitive Help URL**: Enter the URL or server variable that targets the custom context-sensitive help.
- **Standard Context-Sensitive Help Index**: Displays the HTML file that is available as the standard context-sensitive help for the object view.

  The **Standard Context-Sensitive Help Index** attribute can be edited for a custom object view for purposes of backward compatibility. Prior to release 10.1, users specified the location of the external help file using the following syntax: `CUSTOM_HELP:<URL>` (For example: `CUSTOM_HELP:http://helphost/objectview1.html`). The link to the external help file will be displayed in the **Help Selector**. Server variables can be used in the help links. It is recommended that you specify your custom help by means of the **Custom Context-Sensitive Help URL** attribute.

- **Standard Context-Sensitive Help Visible**: Select `True` to display both the standard and custom context-sensitive help file in the **Help Selector** Select `False` if the link to the standard context-sensitive help should not be available and should not be displayed in the **Help Selector**.

3) In the toolbar, click the **Save** button to save your changes.

Managing Custom Object Views in Object View Groups

Custom object views can be assigned to and managed in object view groups. This allows you to structure and organized your enterprise's custom object views according to your enterprise's need. For example, you could organize custom object views according to the user profiles they are assigned to. You can create multiple object view groups, as needed.

To create a new object view group:

1) Go to the **Object Views** folder, click a custom object view that you want to assign to a new object view group. The attribute window is displayed in the right pane.
2) In the **Group** attribute, enter the name of the new object view group and click the Return key. The new object view group is displayed in the below the **Object Views** node. The custom object view has been removed from its position in the object view explorer and is now displayed below the new object view group.
3) In the toolbar, click the **Save** button to save your changes.

Deleting an Object View

Before an object view is deleted, you should check to see if the object view is implemented in other configurations. The custom object view will be irrevocably deleted from the Alfabet database.

To delete a custom object view:

1) Right-click the custom object view and click **Show Usage** and ensure that the object view is not implemented in other critical configuration objects. Click the **Close** button to close the **Objects Usage** editor.
2) Right-click the custom object view that you want to delete and select **Delete**.
3) Confirm the warning by clicking **Yes**. The object view is deleted from the Alfabet database.

Chapter 11: Configuring User Profiles for the User Community

Alfabet Expand enables you to create and configure user profiles that allow the appropriate users in the user community to access the set of functionalities that are relevant to their tasks and responsibilities. For example, a data entry user profile might provide access to such functionalities as **Capture Applications**, **Capture Peripherals**, **Capture Components**, etc., or a strategic planning user profile might provide access to such functionalities as **View As-Is Architecture**, **Plan Target Architecture**, **Manage Master Plans**, etc.

User profiles are the basis of user administration in Alfabet and serve as the entry point when accessing Alfabet. Every user must log in with a user profile that has been assigned to him/her by a user administrator. Therefore, all users accessing Alfabet must be assigned at least one user profile. However, users may possess multiple user profiles in accordance with their responsibilities in the user community and in the enterprise as a whole. A user can switch to another permissible user profile at any point during a user session.

A user profile specifies the Alfabet functionalities available to a user, the visibility and editability of object classes and object class attributes, as well as the availability of associated capabilities including, for example, wizards and workflows.

Each user profile must have one view scheme assigned to it that bundles various class settings. The user profile may have only one class setting defined per object class.

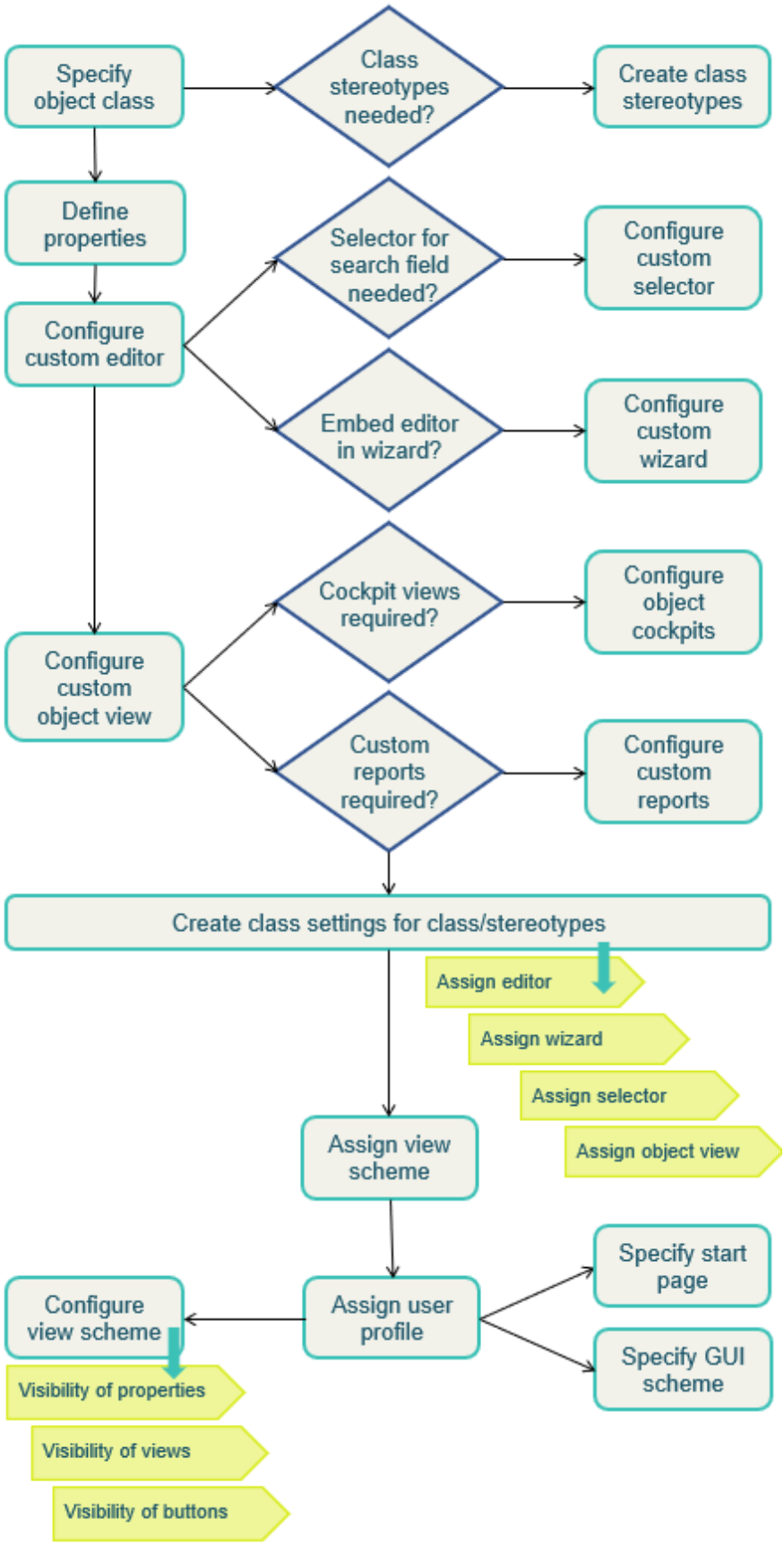


FIGURE: Overview of user profile configuration

Typically, the creation of custom object class properties, custom editors, custom wizards, and custom object views should be complete before beginning the task of configuring class settings for the view scheme that will be assigned to the user profile. The configuration of class settings and their assignment to view schemes are an essential part of the configuration of the user profile. Either a standard class setting or a custom class setting must be available for each object class that users will work with via the user profile. Once custom object class properties, custom editors, custom wizards, and custom object views have been configured, you can begin the following steps which constitute the actual task of user profile configuration.

- Configure class settings for all object classes relevant to the user profile.

A class setting is a specification about an object class or object class stereotype. A class setting is associated with a view scheme which is assigned to a user profile. The class setting therefore specifies what the user can see and do with an object class in the context of the user profile that he/she uses to access Alfabet. The class setting includes such information as which standard or custom object view is displayed, whether users will enter data in a standard or custom editor vs. a standard or custom wizard, whether a standard or custom selector will be used to find objects in the object class, and which standard and custom object class properties will be hidden from view.

Multiple settings can be defined for an object class. However, only one class setting per object class can be assigned to a view scheme so that only one class setting will be available for an object class in a given user profile. Please keep the following in mind when conceptualizing the object views, class settings, and view schemes that you plan to implement in the user profiles you configure for your user community:

- Multiple custom object views can be defined for an object class, but only one object view can be assigned to a class setting.
- Multiple class settings can be defined for an object class, but only one class setting can be assigned per object class to a view scheme.
- Only one view scheme can be assigned to a user profile.

- Assign the relevant class settings to a view scheme and specify the visibility of data and functionality for the view scheme.

A view scheme is associated with a user profile and groups a set of class settings that describe the visibility of objects in the associated object classes. Only one class setting per object class may be assigned to a view scheme.

The view scheme also contains the definition about the functionalities available in the object views accessible via the view scheme. For each standard and configured object view available to the view scheme, workspaces, page views, and toolbar buttons that allow object data to be created, edited, and deleted, for example, can be hidden.

The view scheme thus determines the object classes that are visible to the user. Once a standard class setting or custom class setting has been specified for each relevant object class, the class settings must be assigned to a view scheme.

- Assign the view scheme to a user profile and define access to functionalities via a Alfabet menu bar or navigation pages.

In addition to the configuration of the class settings and the assignment of a view scheme to a user profile, the solution designer must also specify whether the user profile allows Read/Write or ReadOnly permissions. Finally, the means that users use to access the Alfabet functionalities once they log in with the user

profile must be configured. The solution designer can choose to define access to the functionalities via guide pages with hyperlinks or via drop-down menu items displayed in the top toolbar.

User profiles are typically created by a user administrator in the Alfabet interface via the **User Profiles Administration** functionality that can be accessed via the `Admin` user profile or by a solution designer in Alfabet Expand. Users can then be assigned to a user profile in the Alfabet interface via either the **Users Administration** functionality or **User Profiles Administration** functionality that can be accessed via the `Admin` user profile. For more information, see the sections *Assigning User Profiles to a Selected User* or *Assigning Users to a User Profile* in the reference manual *User and Solution Administration*.

- Refine visibility issues for the user profile

For each object profile available in the user profile, the solution designer can further refine the visibility of object class properties and availability of functionality in various parts of the interface. Thus, data and functionality that would be inappropriate for the users assigned to the user profile can be hidden from view. The following can be specified for a view scheme assigned to a user profile:

- Visibility of standard and custom object class properties in standard editors, custom editors, and custom wizards
- Visibility of object class properties (in the **Attributes** section), page views and configured reports in standard and custom object views
- Visibility of standard and custom object class properties in page views and configured reports
- Functionality available via toolbar buttons in object views, object cockpits, page views, and configured reports (for example, create, copy, edit, navigate, etc).

The following information is available:

- Creating User Profiles for the User Community
- Making Functionalities Accessible to a User Profile
- Defining Access to the Functionalities via a Guide Page
- Defining Access to the Functionalities via the Menu Bar
- Specifying the Type of Device That the User Profile Will Access
- Configuring Class Settings for Object Classes and Object Class Stereotypes
- Conceptualizing a Class Setting
- About the Templates Available for Class Settings
- Creating a Custom Class Setting for a Protected Object Class or Object Class Stereotype
- Creating a Class Setting for a Custom Object Class
- Configuring a View Scheme for a User Profile
- Regulating Access Permissions via Class Settings
- Creating a View Scheme and Assigning Class Settings
- Assigning the View Scheme to a User Profile

Creating User Profiles for the User Community

User profiles are the basis of user administration in Alfabet and serve as the entry point when accessing Alfabet. Every user must log in with a user profile that has been assigned to him/her by a user administrator. Therefore, all users accessing Alfabet must be assigned at least one user profile. However, users may possess multiple user profiles in accordance with their responsibilities in the user community and in the enterprise as a whole. A user can switch to another permissible user profile at any point during a user session.

A user profile specifies the Alfabet functionalities available to a user, the visibility and editability of object classes and object class attributes, as well as the availability of associated capabilities including, for example, wizards and workflows.

Each user profile must have one view scheme assigned to it that bundles various class settings. The user profile may have only one class setting defined per object class.

User profiles are typically created by a user administrator in your enterprise. A user profile can have multiple functionalities assigned to it and a functionality can be assigned to multiple user profiles.

The **User Profiles** node in the **Admin** tab in Alfabet Expand allows you to view and edit all existing user profiles as well as create new user profiles. For more information about configuring the attributes relevant for the self-administration capability for a user profile, see the section Configuring User Profile Request or Assignment for the User Community

Please note that user profiles can also be created in the **User Profiles Administration** functionality in Alfabet that can be accessed via the `Admin` user profile. Consider the following:

- If your enterprise is working with an external repository (for example, LDAP), the external ID can be captured for user profiles via the alternative editor `UserProfileWithExternalID_Editor`. This editor should be specified in the **Edit View** attribute for a custom class setting for the class `ALFA_USERPROFILE`.
- Users are assigned to a user profile in the **User Profiles Administration** functionality accessible via the `Admin` user profile. For more information, see the section *Assigning Users to a User Profile* in the reference manual *User and Solution Administration*. Users are created in the **Users Administration** functionality accessible via the `Admin` user profile. For more information, see the section *Defining and Managing Users* in the reference manual *User and Solution Administration*.

To create a user profile:

1) Go to the **Admin** tab, right-click the **User Profiles** node and select **New User Profile**. A new user profile  is displayed in the explorer.
2) In the attribute window, define the following attributes:

- **Name**: Enter a caption for the user profile. This is the name of the user profile that users will see when logging in to Alfabet.

  The names of user profiles will be available in the `METAMODEL` vocabulary and can be translated in the context of the vocabularies. The translated user profile name will be displayed in the masthead of the Alfabet user interface as well as in the **Change User Profile** menu in the <Alfabet User Name> menu in the main toolbar. Please note however that the translation of the user profile name will not be displayed in the **User Profiles Administration** and **User Administration** functionalities in the Alfabet user interface as well as the **User Profiles** node in Alfabet Expand. For more information about translating strings in the vocabularies, see the section Modifying, Translating and Managing the Vocabularies in the reference manual *Configuring Alfabet with Alfabet Expand*.

- **Type**: If the user profile should have editing permissions, select `Read/Write`. If the user profile should only have viewing permissions, select `ReadOnly`.
- **Device Type**: Specify which type of device may be accessed by the user profile. Select `Browser` to indicate that Alfabet may only be accessed by the user profile in a Web-based browser, `App` to indicate that Alfabet may only be accessed by the user profile in the Alfabet Mobile Portfolio Manager which is available for mobile devices or `Unspecified` to indicate that Alfabet may be accessed by the user profile in both a Web-based browser an the Alfabet Mobile Portfolio Manager. Please note that if the user profile will be accessed Alfabet on mobile devices, it is recommended that the visual layout of the object cockpits, guide pages/guide views, etc. is optimized for mobile devices.
- **GUI Scheme**: The drop-down menu displays all GUI schemes configured by your enterprise. Select the GUI scheme that should be used to visualize the Alfabet interface. The configuration of GUI schemes is described in the section Configuring GUI Scheme Definitions for the Alfabet Interface in the chapter Configuring the Visualization of the Alfabet Interface. For more information about configuring user profiles for barrier-free accessibility, see the section Configuring Barrier-Free User Profiles.
- **Guide Page**: Select an existing guide page or guide view from the drop-down list to assign it to the user profile. A guide page/guide view is the start page that is displayed when Alfabet is accessed with the user profile. The guide page/guide view contains links to Alfabet functionalities, thus defining the scope of access to functionalities for the user logged in with the user profile. Guide pages and guide views are configured with the tool Guide Pages Designer and must be uploaded to the Alfabet database

in order to be available in the drop-down list in the **User Profile** editor. For more information about how to create, edit, and upload guide pages and guide views, see the reference manual *Designing Guide Pages for Alfabet*. For more information about whether to assign a guide page/guide view to the user profile or whether to configure access via menus, see the section Making Functionalities Accessible to a User Profile.

- **Enable Feedback Bot**: Select `True` if the user profile may access the Feedback Bot configured for your enterprise. The Feedback Bot allows users to provide feedback for a view, configured report, object cockpit, guide view, etc. For more information about the configuration of the Feedback Bot, see the section Configuring the Feedback Bot.
- **Enable Feedback for View**: Select `True` to activate the **Feedback for Current View** capability. This is relevant for user profiles responsible for reviewing and responding to feedback provided via the Feedback Bot.

Feedback that has been provided for a view or report via the Feedback Bot can be displayed in the Alfabet user interface in a seconday view for those users responsible for reviewing and responding to the feedback. This allows the responsible users to navigate the Alfabet user interface and see the feedback for the relevant view where they currently are. A secondary view with the caption **Feedback for Current View** will be displayed with a link if feedback has been provided for the view, configured report, object cockpit, guide view, etc. Clicking the link will open the *Feedback Review Functionality* in a new browser tab which displays all feedback in detail for the view. To implement the **Feedback for Current View** capability, the **Enable Feedback for View** checkbox must be selected in the **User Profile** editor for the relevant user profile and the **Enable Check Feedback for View** checkbox must be selected for the relevant user in the **User Settings** editor.

- **Enable User Self-Administration**: Select `True` if all named users can assign a permissible user profile to themselves via the **Assign User Profile** option in the < **Alfabet User Name** > menu available in the Alfabet interface. For more information about configuration the user self-administration, see the section Configuring User Profile Request or Assignment for the User Community. Please note that the **Workflow Template** attribute is also only relevant for the self-administration capability.
- **Is Administrative User Profile**: Select `True` if the user profile is an administrative user profile. An administrative user profile has access to all objects, document folders, configured reports and standard views including those specified as administrative. If the checkmark is not set, users will only have access to object and document folders for which they have access permissions and will not have access to any standard views or configured reports that are specified to be administrative. The **Is Administrative User Profile** attribute will be set to `False` by default for all user profiles except the administrative user profiles.

Business functions that are relevant for an administrative user profile must be explicitly added to the user profile. For an overview of all views that are specified as administrative views, see the section *Standard Views Accessible via Administrative User Profiles* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

Please note that for any standard or custom business functions assigned to an administrative user profile, the **Restrict to Administrative User Profiles** must be set to `True` for a standard business function or custom explorer if only users with an administrative user profile may access the business function. For more information about configuring standard business functions and custom explorers, see the chapter Configuring Standard Business Functions and Custom Explorers.

It is recommended that express views are not sent via administrative user profiles. Per default, the XML attribute `EnableExpressViewForAdminProfiles` in the XML object ***SolutionOptions*** is set to "false" and therefore express views will not be sent via user profiles for which the **Is Administrative User Profile** attribute is set to `True`.

If your enterprise wants express views to be sent via administrative user profiles, the XML attribute `EnableExpressViewForAdminProfiles` must be set to "true".

- **Use for Anonymous Users**: If the user profile should be used as the default user profile that persons defined as anonymous users should use to access Alfabet with, select `True`. You can only select `True` for this attribute if you have defined the value `ReadOnly` for the **Type** attribute.

  If you attempt to define a user profile of the type `Read/Write` as the default user profile for anonymous users, the definition will be ignored. If you define more than one user profile as the default user profile for anonymous users, the first user profile in the alphabetical order of user profiles will be used as the default user profile for anonymous users. All other definitions will be ignored.

  Please consider that a user of the type `Anonymous` typically accesses Alfabet infrequently and therefore, will only have `ReadOnly` access. Only one user profile may be defined as the default user profile that anonymous users use to access Alfabet.

  A user of the type `NamedUser` may be assigned to a user profile configured with either Read/Write or ReadOnly access. A named user may be assigned multiple user profiles (including the user profile that is the default for anonymous users). For information about how to define a Alfabet user as a user of the type `NamedUser` or `Anonymous`, see the section *Defining and Managing Users* in the reference manual *User and Solution Administration*.

- **View Scheme**: Click the **Drop-Down**  button to select the relevant view scheme that applies to the user profile when accessed by an external application or via a hyperlink in an email notification. For more information about configuring a view scheme, see the section Configuring a View Scheme for a User Profile.

- **Use WAI-ARIA**: Set to `True` if the WAI ARIA specification should be implemented for barrier-free accessibility. For more information about the configuration of barrier-free user profiles, see the section Configuring Barrier-Free User Profiles.

- **Automated Assistant Enabled**: Set to `True` if the automated assistant capability should be enabled for the user profile. Set to `False` if all assistants available for the user profile shall be disabled. It is recommended that the **Enable Automated Assistant** attribute is set to `False` for user profiles requiring barrier-free accessibility as well as administrative user profiles or super users providing help desk services, for example. For more information about configuring the automated assistant capability, see the chapter Providing Custom Online Help to the User Community.

- **Automated Assistant URL**: Enter the URL or server variable that targets the content to display in the assistant assigned to the user profile.

- **Custom Context-Sensitive Help URL**: Enter the URL or server variable that targets the custom context-sensitive help for the user profile. For more information about configuring custom context-sensitive help, see the chapter Providing Custom Online Help to the User Community.

- **Standard Context-Sensitive Help Index**: Displays the HTML file that is available as the standard context-sensitive help for this user profile. A standard context-sensitive help is only available for administrative user profiles. The standard online help provides information about the user

administration and solution administration functionalities available that can be accessed in the Alfabet user interface via administrative user profiles.

- **Standard Context-Sensitive Help Visible**: Select `True` to display both the standard and custom context-sensitive help file in the **Help Selector** Select `False` if the link to the standard context-sensitive help should not be available to the user profile and should not be displayed in the **Help Selector**.
- **Workflow Activities Explorer**: The configuration of guide views and guide pages may contain a hyperlinked text or button displaying the number of workflow steps that the user is responsible for and that when clicked opens the **Workflow Activities Explorer**. Instead of displaying the default **Workflow Activities Explorer**, a custom explorer can be configured to open instead. Select the custom explorer that should be opened in the context of the guide view/guide page configuration for the selected user profile. For more information about configuring the hyperlinked text or button displaying the number of workflow steps that the user is responsible for in guide views/guide pages, see the section *Adding Workflow, Assignment, Collaboration, and Microsoft Teams Meeting Links to the Guide View* in the reference manual *Designing Guide Pages for Alfabet*. For more information about configuring custom explorers, see the chapter Configuring Standard Business Functions and Custom Explorers.

3) In the toolbar, click the **Save** button to save your changes.

Making Functionalities Accessible to a User Profile

Once a user profile has been created, you must configure a means for users to open the Alfabet functionalities with the user profile that they are logging on with. You can either configure a navigation page providing hyperlinks to the functionalities with optional text describing the functionalities, or you can configure a top menu bar containing drop-down menus that allow users to select the functionality they want to open.

Please be aware that the `Admin` user profile as well as any other user profile specified to be an administrative user profile grant Read/Write permissions to all objects accessed via the user profile. Therefore, users with an administrative user profile should typically be able to access those functionalities that provide them with access to the objects that they may need to edit. Typically, this includes the user administration functionalities, solution configuration functionalities, and search functionalities. Business functions that are relevant for an administrative user profile must be explicitly added to the user profile.

Please note that for any standard or custom business functions assigned to an administrative user profile, the **Restrict to Administrative User Profiles** must be set to `True` for a standard business function or custom explorer if only users with an administrative user profile may access the business function. For more information about configuring standard business functions and custom explorers, see the chapter Configuring Standard Business Functions and Custom Explorers.

There are a number of functionalities that must be configured before they can be implemented in the Alfabet software. The following functionalities require configuration before implementation:

- For more information regarding the business function `ComplianceConfiguration` and `ComplianceInstances`, and `Home_ComplianceObjects`, see Configuring the Compliance Management Capability

- For more information regarding the business function `PRJ_Management`, see the section Configuring the Project Management Capability.
- For more information regarding the business function `VM_StrategyDeduction`, see the section Configuring the Strategy Deduction Capability.
- For more information regarding the business functions `InitiateWorkflow`, `WF_Activites`, `WF_Administration`, and `Workflows`, see the section Configuring Workflows.
- For information regarding the configuration of the following data capture functionalities, see the section Configuring Standard Data Capture Environments.

  - `APP_CaptureApplication`
  - `APP_UserApplications`
  - `COM_CaptureComponents`
  - `COM_UserComponents`
  - `DEM_CaptureDemands`
  - `DVC_CaptureDevices`
  - `DVC_UserDevices`
  - `ICTO_CaptureICTObjects`
  - `ICTO_UserICTObjects`
  - `ITMPM_CaptureMaps`
  - `PRF_CapturePeripherals`
  - `PRF_UserPeripherals`
  - `PRJ_CaptureProjects`
  - `PRJ_SubProjects`

The following information is available:

- Defining Access to the Functionalities via a Guide Page
- Defining Access to the Functionalities via the Menu Bar
- Specifying the Type of Device That the User Profile Will Access

Defining Access to the Functionalities via a Guide Page

A guide page serves as the start page for a user profile. A guide page is a customized HTML file with hyperlinks that provide users access to the software and support them in their work. A guide page could include, for example, informational text or images about enterprise-specific workflows in the Alfabet solution, links to specific functionalities in Alfabet, URL links to internal documents or the Web, or hyperlinks to other guide pages.

Software AG provides a standard set of HTML-based guide pages with the standard Alfabet product that serve as an example of how guide pages can be configured. You can access the sample guide pages using the `FullAccess` user profile.

The configuration of guide pages is carried out in the tool Guide Pages Designer. For detailed information about how to configure guide pages for the user profiles in your user community, see the reference manual *Designing Guide Pages for Alfabet*.

You must assign a guide page to a user profile to display it in the Alfabet interface.

To assign the guide pages to a user profile:

1) Go to the **Admin** tab, expand the **User Profiles** explorer node and click the user profile that you want to assign a guide page to.

2) In the **Guide Page** attribute, click the arrow to select the guide page in your guide page project. The guide page names will be listed in the drop-down list without the file extension.

Please note that if you specify that the user profile is to be used in mobile devices, (**Device Type** = `App`), it is recommended that the visual layout of the selected guide page is associated with a user profile should be optimized for such mobile devices.

3) In the toolbar, click the **Save** button to save your changes.

Defining Access to the Functionalities via the Menu Bar

To configure access to the functionalities via the menu bar at the top of the Alfabet interface, you can create drop-down menus containing menu items that allow users to select the functionality that they want to work with. Please keep in mind that if a high number of drop-down menus are configured for a user profile and users are working with Alfabet on a small display, it is possible that the captions of the menus may be truncated.

Custom explorers can also be configured for your Alfabet solution. A custom explorer is considered a functionality and can be made accessible to the Alfabet interface via the menu bar or a guide page. The following information assumes that any necessary custom explorers have already been configured. For more information, see the section Configuring Standard Business Functions and Custom Explorers.

Please note that if you create menu items to make functionalities available via the menu bar, the **Guide Page** attribute may not be defined for the user profile. If a guide page is selected in the **Guide Page** attribute, the guide page definition will take precedence and the menu items configured for the user profile will not be applied. For more information about how to assign a guide page to the user profile, see Defining Access to the Functionalities via a Guide Page.

For an overview of the technical names of the business functions, their caption displayed in the solution interface, and any addressable sub-functions, see the chapter *Business Functions and Their Sub-Functionalities* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

To configure the menu bar to display drop-down menus listing one or more functionalities:

1) Go to the **Admin** tab, expand the **User Profiles** node, right-click the user profile that you want to define and select **Create Menu Item**. A menu item labeled **New Menu Item** will be added below the user profile node. This menu item is the top-level menu item.

The top-level menu item will be displayed as a button in the toolbar. No functionality should be assigned to this menu item. Please note that if you want to provide access to only one functionality, you must still create a top-level menu item with a subordinate menu item to which the functionality is assigned. A top-level menu item cannot provide access to a functionality.

2) Click the new menu item in the explorer to open the attribute window. Define the following attributes, as needed.

3) In the **Caption** attribute, enter the text that should be displayed on the button in the Alfabet menu bar.

The ampersand (&) is not allowed in the caption of a menu item.

4) To add a functionality to the drop-down the menu, right-click the top-level menu item and select **Add Business Function**. In the **Select Business Functions** editor, select one or more functionalities that you want to add to the menu. If your enterprise has configured custom explorers, these will also be displayed in the drop-down menu. Click **OK** to add the functionalities to the selected menu item. The menu items (functionalities) are displayed below the selected top-level menu item .

Keep the following in mind when selecting functionalities in the editor:

- **CTRL + mouse-click** allows you to select one or more functionalities.
- **CTRL + Shift + mouse-click** to select a block of objects

5) Click a menu item (functionality) to further define its attributes:

- **Caption**: Enter the text that should be displayed for the functionality in the Alfabet menu bar. If you do not explicitly define this attribute, the default caption will be displayed for the functionality.
- **Arguments**: If you have selected any of the functionalities listed below, you must specify arguments to define the link target. The syntax for this entry is `ContextArgs=` *<Type of Argument>:<Value of Argument>* (for example: `ContextArgs=Explorer:COMG_Explorer` or `ContextArgs=Class:Application`):
- **Simple Search** functionalities (`GenericSearch`, `Simple_Search`, `Browse`):
  - If the search functionality should be limited to a set of object classes, specify the object class names in a comma-separated format with the argument `ContextArgs=` *<ObjectClassName>*. For example, to search for an object in the object classes `Application`, `Component`, and `ICTObject`, enter the following: `ContextArgs=Class:Application,Component,ICTObject`.
  - If the search functionality should be limited to a set of object classes and object class stereotypes, specify the object class stereotype in the comma-separated list as: *<ObjectClassName>:<ObjectClassStereotypeName>*. For example, to include the application stereotypes `BusinessApplication` and `TechnicalApplication` in the list, enter: `ContextArgs=Class:Application:BusinessApplication,Application:TechnicalApplication,Component,ICTObject`.
- **Capture Projects** functionalities (`PRJ_CaptureProjects` and `PRJ_CaptureProjects_Ex`): If the functionalities should open with the filter preset to a defined project stereotype. You must specify the project stereotype with the entry `ContextArgs=` *<ProjectStereotypeName>*. For example, `ContextArgs=ProjectStep`.
- **Generic Object Viewer** functionality (`GenericObjectViewer`): You must provide information about which object class can be searched via the Edit Search interface control. You must specify the object class name with the entry `ContextArgs=Class:` *<ObjectClassName>*. For example, `ContextArgs=Class:Application`.

- If you set a link to a functionality that includes sub-functionalities (for example, the functionalities **Define Business Standards** (`Business_Standards`) and **Define IT Standards** (`IT_Standards`), you must specify which sub-functionality should be accessed. If no sub-functionality is assessed, the default sub-functionality will be automatically displayed. In the case of the functionalities **Define Business Standards** (`Business_Standards`), this would be `BO_Explorer`. In the case of **Define IT Standards** (`IT_Standards`), this would be `COM_Explorer`. The syntax to specify the sub-functionality is *<Type of Argument>=<Value of Argument>*. For example, for the functionality `CostManagement`, you could specify either `ContextArgs=Explorer:COSTCG_Explorer` or `ContextArgs=GraphicView:COSTC_Overview`: To determine which functionalities have sub-functionalities, see the chapter *Business Functions and Their Sub-Functionalities* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

6) In the toolbar, click the **Save** button to save your changes to the menu item.

7) Click the top-level menu item to define the sequence of the functionalities in the drop-down menu. In the **Sub-Items** attribute, click the **Browse** button. In the editor, click the **Up/Down** arrows to sequence the functionalities in the drop-down menu. The top of the list represents at the top of the drop-down menu. Click **OK** to close the editor and save the sequence of the menu items.

8) In the toolbar, click the **Save** button to save your changes.

9) Repeat this procedure to add all top-level menu items and their respective functionalities that should be accessible to the selected user profile.

10) Click the user profile to define the sequence of the top-level menu items in the drop-down menu. In the **Sub-Items** attribute, click the **Browse** button. In the editor, click the **Up/Down** arrows to sequence the top-level menu items. The top of the list represents the menu items furthest to the left on the menu bar. Click **OK** to close the editor and save the sequence of the menu items.

11) In the toolbar, click the **Save** button to save your changes.

Specifying the Type of Device That the User Profile Will Access

When defining the user profile, you can specify whether the user profile should be used to access Alfabet in a Web-based browser or in the Alfabet Mobile Portfolio Manager, which is available for mobile devices. Please note that if you the user profile will be used to view Alfabet on mobile devices, it is recommended that the visual layout of the object cockpits, guide pages, etc. associated with a user profile should be optimized for such mobile devices.

To specify which type of device may be accessed by the user profile, define the **Device Type** attribute available in the attribute window for the user profile. Select `Browser` to indicate that Alfabet may only be accessed by the user profile in a Web-based browser, `App` to indicate that Alfabet may only be accessed by the user profile in the Alfabet Mobile Portfolio Manager, or `Unspecified` to indicate that Alfabet may be accessed by the user profile in both a Web-based browser and the Alfabet Mobile Portfolio Manager.

Configuring Class Settings for Object Classes and Object Class Stereotypes

A class setting is a specification about an object class or object class stereotype. A class setting is associated with a view scheme which is assigned to a user profile. The class setting therefore specifies what the user can see and do with an object class in the context of the user profile that he/she uses to access Alfabet. The class setting includes such information as which standard or custom object view is displayed, whether users will enter data in a standard or custom editor vs. a standard or custom wizard, whether a standard or custom selector will be used to find objects in the object class, and which standard and custom object class properties will be hidden from view.

Multiple settings can be defined for an object class. However, only one class setting per object class can be assigned to a view scheme so that only one class setting will be available for an object class in a given user profile.

You must configure a custom class setting for a custom object class, protected object class, or object class stereotype if any of the following is required for the respective user profile:

- custom object view
- custom editor
- custom wizard
- exclusion of standard object class properties
- object class stereotypes
- custom icon for object class

Please note that the object class stereotypes, custom editor, custom wizard, and custom object view must be configured before you can define the class setting. The following information is available:

- Conceptualizing a Class Setting
- About the Templates Available for Class Settings
- Creating a Custom Class Setting for a Protected Object Class or Object Class Stereotype
- Creating a Class Setting for a Custom Object Class

Conceptualizing a Class Setting

Class settings determine technical information about an object class or object class stereotype (such as the object view, wizard, editor, and preview properties to display) as well as your specific class setting configuration. A class setting must be created for a set of configuration objects available to a user profile.

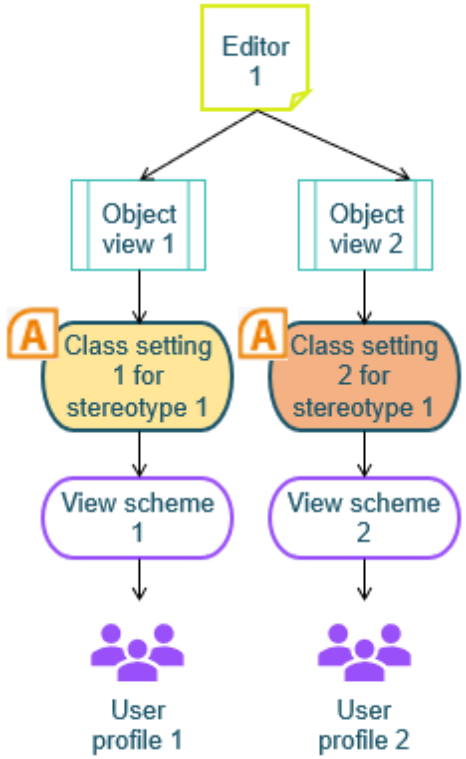


FIGURE: Example configuration for a stereotype for the class Application

Please keep the following in mind when conceptualizing the object views, class settings, and view schemes that you plan to implement in the user profiles you configure for your user community:

- Multiple custom object views can be defined for an object class, but only one object view can be assigned to a class setting.
- Multiple class settings can be defined for an object class, but only one class setting can be assigned per object class to a view scheme.
- Only one view scheme can be assigned to a user profile.

Typically, a class setting should be defined for each wizard available for an object class, since each wizard is usually accessed in a different user context (via a view scheme assigned to a user profile).

This does not apply to wizards that have been defined for the workflow capability. Wizards defined for workflows are not assigned to a class setting but rather are assigned to a workflow template. For details about defining a workflow, see Configuring Workflows.

The issues that you can specify in a class setting for an object class are:

- Which object profile should be displayed for the object class. Should the standard object view be used or has a custom object view been configured? (Standard object views are typically implemented for object classes such as `Person` or `UserGroup`, for example.)
- Should configured object cockpits should be displayed for the object class? If object cockpits have been configured, should the object profile be hidden?
- Which wizard, if any, should be associated with the class setting?

- Which custom editor, if any, should be associated with the class setting?
- Which properties should be displayed in the preview available for objects in the object class?
- Should this class setting be the default class setting for the object class? Alfabet Expand provides a default class setting for every protected object class in the meta-model so you are not required to define a class setting for every object class. If you do not explicitly define this class setting as the default (**Default** = `True`), the default class setting template provided by Software AG will be used.
- Which protected object class properties and custom object class properties should be hidden from the Alfabet interface for this class setting?
- Should this object class be searchable?
- Should a custom icon be displayed in explorers for this object class? Should custom foreground and background colors be displayed?
- Should objects of this object class be displayed in the Alfabet views available to display new and changed objects or recently used objects? (These views are: `New_Objects`, `Recent_Objects`, and `New_Recent_Objects` which displays a view with a table for new objects and a table for recent objects. These views must be explicitly assigned to the relevant user profile.)
- Should an **Add Misssing <ObjectClass>** button be added to the standard or custom object selectors specified in the **Selector Definition** attribute allowing an object of the object class/object class stereotype to be created if it is not in the inventory?

You can define multiple class settings for each class. Each class setting should have a unique name. Only one class setting can be assigned per object class to a view scheme.

Before a class setting is deleted, you should check to see if the class setting is implemented in other configuration objects such as view schemes or workflow templates. To do so, right-click the class setting and click **Show Usage**. A window will be displayed showing the configuration objects using the selected class setting. For more information regarding the **Show Usage** functionality, see the section Reviewing the Usage of a User Profile.

Alfabet provides an **Archive Manager** functionality that allows objects to be archived. The **Archive Object** symbol displayed in the table below indicates that the page view may potentially be archived for the object class. The set of information that is archived for an object class is determined by the class setting configured for the user profile of the user archiving the object. The **Archive Manager** functionality is only accessible to users accessing Alfabet with an administrative user profile. For more information about the **Archive Manager** functionality, see the section *Deleting and Archiving Alfabet Objects* in the reference manual *User and Solution Administration*.

About the Templates Available for Class Settings

Alfabet Expand has a number of class setting templates that should be used when configuring the class settings for your enterprise. The **Class Settings** folder is available in the **Presentation** tab and contains sub-folders for all Alfabet object classes that are potentially available to users in the Alfabet interface. For each object class, one or more class settings may be available that can be copied in order to define class settings specific to the needs of your enterprise. In each object class folder, you may see one or all of the following class setting templates.

- <ObjectClass>_Standard: This standard template will be used for most class setting definitions. This template is available for all object classes available in the **Class Settings** node. It contains the basic information for the respective class setting including, for example, the assigned object view, search properties, and selector definition. This template also allows you to configure visibility in the custom wizards that you have created for your enterprise. Standard class settings cannot be deleted.

  If a class setting is not explicitly assigned to a view scheme, then the standard template <ObjectClass>_Standard will be assigned per default tot he view scheme.

- <ObjectClass>_Example: This template is available for object classes for which standard custom editors and standard wizards have been configured by Software AG. These classes include: Application, Component, Demand, Device, ICTObject, ITMasterPlanMap, Peripheral, StandardPlatform. You should use this template when you want to define the visibility for the standard Alfabet wizards provided for the above classes.

  Software AG provides a number of data maintenance functionalities that are driven by wizards. The wizards implemented in the **Capture** <ObjectClass> functionalities are determined by the wizard definition for the respective class setting assigned to a view scheme. These data maintenance functionalities include **Capture ICT Objects**, **Capture Applications**, **Capture Components**, **Capture Devices**, **Capture Peripherals**, **Capture Demands**, and **Capture Master Plan Maps**. For more information about configuring these functionalities, see the section Configuring Standard Data Capture Environments.

- <ObjectClass>_RM: This template is available for object classes relevant to the **Release Management** functionality. These object classes include Application, Component, and StandardPlatform. You should use this template if you want to define class settings for the above classes in the context of the **Release Management** functionalities.

- <ObjectClass>_ReadOnly: This template is available for a specific set of object classes for which a ReadOnly view is required to control accessibility. These classes include, for example, CostCenterType, CostType, EvaluationType, IncomeType, IndicatorType, IndicatorTypeComputationRule, INDLUT_ReadOnly, Portfolio, Person, RISKTMPL_ReadOnly, RISKSET_ReadOnly, RISKCLS_ReadOnly, RoleType, Skill, UserGroup. These classes are typically used to configure or administrate your Alfabet solution. Therefore, it is recommended that the <ObjectClass>_ReadOnly class setting should be used for user profiles that have no configuration or administration responsibilities.

  In addition to the class setting templates USER_ReadOnly and USER_Standard*, the class Person also has a class setting template USER_LimitedEditability which allows users to edit custom object class properties only.

  If the editability of user information should be limited, then you must explicitly define either the USER_ReadOnly class setting or USER_LimitedEditability class setting. Otherwise, the default template USER_Standard* will apply. The USER_Standard* class setting grants all users with accessibility to an object in the relevant object class to edit all user information.

  If your enterprise is working with an external repository (for example, LDAP), the external ID can be captured for user profiles via the alternative editor UserProfileWithExternalID_Editor. This editor should be specified in the **Edit View** attribute for a custom class setting for the class ALFA_USERPROFILE.

Creating a Custom Class Setting for a Protected Object Class or Object Class Stereotype

You can create multiple class settings for an object class or object class stereotype.

The following information describes the configuration of class settings for general use in Alfabet. The attributes **Allow Read via Rest API**, **Allow Write via Rest API**, and **Allow Update from External Reference Data Service** are for special import functionalities and are not explained in this section. They are described in the context of the import functionalities in the section Importing Objects of Configuration Relevant Object Classes from a Master Database in this reference manual and in the reference manual *Alfabet RESTful API*.

To create a custom class setting:

1) Go to the **Presentation** tab and expand the **Class Settings** node.
2) In the **Class Settings** folder, navigate to the folder for the object class that you want to define class settings for and right-click the public or private class setting template  that you want to copy. Select either:

- **New Class Settings as Copy** to create a new class setting for the selected object class. A copy of the class settings template is added to the class setting folder.
- **New Class Settings for Stereotype as Copy** to create a class setting for an object class stereotype. In the dialog that opens, select the object class stereotype and click **OK**. A new class setting folder will be created for the object class stereotype which will contain the new class setting for the object class stereotype. The naming convention of the class setting folder and class setting is: <ObjectClass:ObjectClassStereotype>. The class setting for the object class stereotype can be configured in the same manner as class settings for conventional object classes. Additional class settings can be created for the object class stereotype via the **New Class Settings as Copy** functionality.

For additional details about defining a class setting for an object class stereotype based on the class `Project`, see the section Configuring the Project Management Capability.

3) All attributes specified for the class setting that you copied will also be copied to the new class setting. These attributes can be edited, as needed. Click the new class setting template to open the attribute window and enter a technical name for the class setting in the **Name** attribute.

It is recommended that the name of the class setting indicates the view scheme that it is being defined for and the name of the view scheme indicates the user profile it is assigned to. In other words, the name of the class setting should thus indicate the user profile that it is associated with.

A validation mechanism checks for correct syntax when defining a technical name. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.

4) Define the following attributes, as needed:

- **Default**: Set to `True` if this class setting should be the default class setting for this object class. Set to `False` if this class setting should not be the default class setting for this object class. If no value is defined, the class setting template `<ObjectClass>_Standard*` provided by Software AG will automatically be used as the default.

- **Searchable**: If the object class should be searchable in the Alfabet search functionalities, select `True`. Select `False` if the object class should not be searchable in the Alfabet search functionalities.

  Please note the following regarding the implementation of searchability in class settings:

  - Not all object classes are searchable. For an overview of the object classes that are searchable per default in the standard Alfabet search functionality, see the chapter *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.
  - If object class stereotypes have been defined for the object class, you can specify searchability on the level of the object class and/or object class stereotype:
    - If an object class stereotype is to be searchable, set the **Searchable** attribute to `True` for each relevant class setting for the object class stereotype. The object class stereotype will be displayed in the **Search for** filter in the search functionality.
    - If the entire class is to be searchable irrespective of the object class stereotype, set the **Searchable** attribute to `True`. The name of the object class will be displayed in the **Search for** filter in the search functionality.
  - If you set the **Searchable** attribute to `True`, you can implement a custom selector for the selected object class in the *Simple Search Functionality* (`GenericSearch`, `Simple_Search`). To implement a custom selector, select a custom selector that is relevant for the object class. All custom selectors that have been configured for your solution will be displayed. You must ensure that you select a custom selector that is meaningful for the view and defined for the relevant object class. For more information about configuring custom selectors, see the section Configuring a Custom Selector for Search Functionalities.
  - If you set the **Searchable** attribute to `True`, you can define and sequence the object class properties that should be used to sort the objects in this object class when found via search functionalities. In the **Sort Properties** attribute, select the object class properties in the list to use to order the object class properties and determine their sequence by using the **Up/Down** buttons.

- **Selector Definition**: Specify a custom selector to replace the standard selector for the object class. Custom selectors assigned to a class setting will globally replace the standard selector in all cases where the selector is implemented except in cases where a hard-coded selector is required or a custom selector is explicitly specified in the **Customization Editor** used when configuring the associated user profile. For more information, see the section Hiding Functionalities in an Object View.

  Please note that new objects may be created in the context of class-based standard and custom object selectors with a minimal amount of navigation required. When ad-hoc adding of new objects is enabled, the **Add Missing <Class.Caption>** button will be available in the selector and uses can create objects needed to define a reference in the context of the selector. When the user clicks the **Add Missing <Class.Caption>** button, the configured functionality/explorer, configured report, or standard view in which the user can create the new object will open in a separate browser tab. Once the missing object has been created, the user can return to the original browser tab with the object selector and search in the selector for the new

object. If the **Add Missing <Class.Caption>** button shall be available in the standard object selector or the custom selector specified in the **Selector Definition** attribute, you must enable the button by specifing the view that shall open to add the missing objects. This must be specfied in the **View for Missing Object Button** available in the **Views** section of the attribute window as described below.

- **Object View**: Select the standard or custom object view that should be displayed for this class setting. Please keep the following in mind regarding the object view setting:
- An object view can be reused for multiple class settings.
- If you select a custom object view, all object cockpits configured for a custom object view will also be included. However, you can hide the object cockpits that you consider to not be relevant for the users that will be working with this class setting. Further visibility issues can also be refined for a custom object view in the context of the view scheme it is assigned to. For example, you can hide functionalities and aspects of the object profile. For more information, see the section Hiding Functionalities in an Object View.
- Set the **Suppress Object Profile** attribute to `True` if you want only the configured object cockpits to be displayed and the standard or custom object profile to be hidden from the user interface.
- The **Graphic View** attribute allows a single page view to be displayed rather than an object view in the Alfabet interface. This attribute is typically used for diagram page views that can only be accessed via another page view (for example, `ApplicationDiagramStatusLink`). This attribute is configured in some private class settings available in the standard Alfabet product. It is highly recommended that the **Graphic View** attribute is not configured for custom class settings. Please note that if the **Object View** attribute is defined for the class setting, it will override the **Graphic View** definition and the selected object view will be displayed.
- **Prevent In-Line Editing in Object Views**: Specify whether in-line editing is permissible in the object view specified in the **Object View** field. Specify `True` if in-line editing shall be prevented in the object view assigned to the class setting. Select `False` if in-line editing shall be allowed in the object view assigned to the class setting If no value is selected, in-line editing shall be allowed per default (= `False`).

  Please note that the definition of the **Prevent In-Line Editing in Object Views** attribute for a class setting will override the setting of the **Prevent In-Line Editing in Object Views** attribute for the associated GUI scheme. However, the definition of the **Prevent In-Line Editing in Object Views** attribute for an object view will override the setting of the **Prevent In-Line Editing in Object Views** attribute for the associated class setting.

- **Default Editor Type**: Define whether the standard/custom editor or a configured wizard should open when the user clicks the **Edit** button in a relevant page view or object view. If no value is set for the **Default Editor Type** attribute, the specification of the **Default Editor Type** attribute in the default class setting for the object class will be applied. Specify one of the following:
- To implement an editor: Select `SimpleEditor` in the **Default Editor Type** attribute if an editor should be implemented for views in which an object of this object class can be created. Define the following attributes, as needed if you have specified `SimpleEditor`:
  - **Edit View**: If you do not want to display the conventional standard Alfabet editor when users click the **Edit** button, select a different Alfabet editor that should be implemented.

- **Custom Editor**: If you want to implement a custom editor that has been configured for the object class in addition to the Alfabet editor, select the custom editor. If no custom editor is specified, only the standard Alfabet editor will be implemented. If you specify a custom editor in the **Custom Editor** attribute, the class settings will determine the visibility of the custom object class properties that may potentially be displayed in the custom editor. You may only select one custom editor per class setting. For more information about defining a custom editor, see Configuring Custom Editors.

- To implement a configured wizard: Select `Wizard` in the **Default Editor Type** attribute if a configured wizard should be implemented for views in which an object of this object class can be created. Specify which wizard to implement in the **Wizard** attribute. If you specify a wizard in the **Wizard** attribute, the class settings will determine the visibility of the standard and custom object class properties that may potentially be displayed in the wizard. You may only select one wizard per class setting. For more information about defining a custom editor, see Configuring Wizards.

  A wizard cannot be implemented for the following views:

  - **Assignments** page view (`ObjectAssignments`)
  - **Check In Inventory** page view (`PRJ_CheckInReport`
  - **To-Be Architecture** page view (`PRJ_NewArchitecture`)
  - **Service Levels** page view (`BMSVC_ServiceLevels`)
  - **Evaluations by Objects** page view (`CMPL_UserEvaluationsByObjects`)
  - **Evaluations by Controls** page view (`CMPL_UserEvaluationsByObjects`)

  The configuration of a custom wizard will apply to all instances of the custom wizard in Alfabet. For example, if you hide object class properties or functionalities in a custom wizard, the object class properties or functionalities will be hidden for all instances of the custom wizard. In other words, a selected wizard cannot be differently configured in the context of different object class stereotypes. The wizard configuration will apply to each object class stereotype that the custom wizard is assigned to. If different configurations are required for a custom wizard, it is recommended that you create a new wizard via the copy functionality and modify the new wizard as needed. Once the wizard is configured, it can be selected in the **Wizard** attribute for the class setting that you create for the object class stereotype.

- **Editor Rendering Options**. You must specify the rendering style of the interface controls in the custom editor. Two options are possible and the method you choose will determine how you go about designing the custom editor. You can choose the traditional rendering style, which reflects the explicit layout of all interface controls in the custom editor as designed by the solution designer, or the stack rendering style which automatically positions the visible interface controls in a one- or two-column linear list. Editor rendering may also be specified on the level of a class setting, thus allowing the editors to be differentiated on a class-by-class basis for a user profile. For an overview of configuring the rendering styles of standard and custom editors, see the section Specifying the Rendering Definition of the Custom Editor. Expand the **Editor Rendering Options** section and define the following:

- **Rendering Style**: Select either `Traditional` to display standard and custom editors as the have been explicitly designed or select `Stack` to display standard and custom editors as a one- or two-columnar linear list. If you select `Stack`, define the attributes listed below.
- **Stack Layout Type**: If **Stack** has been selected for the **Rendering Style** attribute: Select `OneColumn` if the captions and interface controls shall be displayed as a single left-aligned column in the editor. Select `TwoColumns` if the captions and interface controls shall be displayed as two left-aligned columns in the editor. Please note that the sequence of the interface controls is determined by the **Tab Index** attribute defined for each interface control.
- **Preserve Layout in Group Boxes**: Select `True` if the position of interface controls in a Group Box interface control should follow a traditional rendering style and not be changed to have a stack rendering style. Select `False` if the interface controls in a Group Box interface control should have a stack rendering style.
- **Show Hint In-Line**: Select `True` if the help texts defined via the **Hint** attribute of the interface controls shall be displayed in the editor below the editor field. The hint icon specified in the **Interface Control Hint Icon** attribute will not be displayed if the **Show Hint In-Line** attribute is set to `True`. Select `False` if the help texts should not be displayed below the editor field.
- **Override GUI Scheme Settings**: Select `True` if the class setting shall override the GUI scheme setting of the editor rendering options. Select `False` if the class setting shall not override the GUI scheme definition of the editor rendering options.
- **View for Missing Object Button**: This attribute allows the **Add Missing <Class.Caption>** button to be available in the standard object selector or the custom selector specified in the **Selector Definition** attribute. The **Add Missing <Class.Caption>** button enables users to create objects needed to define a reference in the context of the selector that has been opened in a page view, search field in an editor/wizard, or a filter field in a view. The caption on the button will specify the caption of the relevant class or object class stereotype that the class setting is specified for. By clicking the **Add Missing <Class.Caption>** button, the specified functionality/explorer, configured report or standard view in which the user can create the new object will open in a separate browser window. Once the missing object has been created, the user can return to the original browser tab displaying the object selector and search in the selector for the new object.

  Please note that the **Add Missing <Class.Caption>** button capability is not supported in the Search functionalities (`SRCH_GenericSearchWraper`).

To add the **Add Missing <Class.Caption>** button to the standard selector or custom object selector specified in the **Selector Definition** field, click the **Browse**  button to open the **Select View** editor. Define the following fields as needed:

- In the **View Type** pane, select the relevant radio button to specify whether a functionality/explorer, configured report, or standard view should open in which the new object should be created.
- In the **View** field, select the respective functionality/explorer, configured report, or standard view that should open so that the user can create the new object.
- **Properties in Preview**: Define which standard and custom properties are displayed in the preview window that opens via a click-and-hold action on an object in a tabular dataset or business graphic. To define the preview properties for the class setting, click the **Browse**  button to open the editor. Set a checkmark for all properties that should be displayed in the preview. You can define the order of the object class properties displayed in the preview by using the **Up/Down**  buttons. Please note the following:

- The object class properties are displayed in two columns sequenced from left to right and top to bottom.
- If the **Properties in Preview** attribute is not defined, the `Name`, `Description`, and, if applicable, `Stereotype` properties will be displayed per default.
- Any standard and custom properties of the type `Reference` that are defined in the **Properties in Preview** attribute can be visualized as a link that allows navigation to the object profile/object cockpit of the referenced object. The properties displayed for an object referenced by the standard or custom property will be visualized as defined in the **Image Properties** attribute of the class setting defined for the referenced object class/object class stereotype.

  For example, if you include the property `Domain` in the **Properties in Preview** attribute for the class setting for the class `Application`, then the **Image Property** attributes defined for the class setting for the class `Domain` will be displayed for the **Primary Domain** attribute in the preview window for applications.

- **Image Properties**: The image properties are typically used to represent objects in a condensed form in the Alfabet user interface. Image properties may be used, for example, for objects displayed in the preview window, auto-fill function in editors and filter fields, full-text searches, log files, etc. To define the image properties for the class setting, click the **Browse**  button to open the editor. Set a checkmark for all properties that should be displayed for objects in the object class/object class stereotype associated with the class setting. You can define the order of the object class properties by using the **Up/Down**  buttons.

  For most classes, this may simply be the property `Name`. However, for the class `Application` you might want `Name` and `Version` to be displayed or for the class `Domain` you might want `LevelID` and `Name` to be displayed.

- **Consider in New Objects**: Define whether objects in this object class that have been added by the user with the last seven days should be displayed in the **New Objects** functionality (`New_Objects`) and **New or Recent Objects** functionality (`New_Recent_Objects`). These functionalities display a list of objects in Alfabet that have been added or modified during the last seven days. This list is dynamically generated via the `LAST_UPDATE` field in the database table of the relevant artifact class. Select `True` if objects of this object class should be displayed in the **New Objects** functionality (`New_Objects`) and **New or Recent Objects** functionality (`New_Recent_Objects`) available in the relevant user profile. Select `False` if objects should not be displayed. If no value is defined, the default value `False` will apply and no objects will be displayed for the object class.

  Please note the following additional configuration requirements:

  - The functionalities `New_Objects` or `New_Recent_Objects` must be explicitly assigned to the relevant user profile. For more information, see the section Making Functionalities Accessible to a User Profile.
  - This functionality must be explicitly activated for Alfabet as a whole via the `SaveRecentObjects` parameter in the server alias configuration. For more information, see the section *Configuration Attributes for the Alfabet Components* in the reference manual *System Administration* or contact your system administrator.

- **Consider in Recent Objects**: Define whether objects in this object class that have been accessed or edited by the user with the last seven days should be displayed in the **Recent Objects** functionality (`Recent_Objects`) and **New or Recent Objects** functionality (`New_Recent_Objects`). These functionalities display a list of objects in Alfabet that have been accessed during the last seven days. This is a user-specific list. Data is stored in the database table `ALFA_RECENTOBJECTS`. Select `True` if objects of this object class should be displayed in the **Recently Used Objects** functionalities available in the relevant user profile. Select `False` if objects should not be displayed. If no value is defined, the default value `False` will apply and no objects will be displayed for the object class.

- **Format String**: Configure the standard and custom object class properties to display in the header of the page views configured for the object views that are assigned to the class setting. The **Format String** value for the class setting will also be displayed in the header of the associated object view if no **Format String** attribute is explicitly defined for that object view. The object class properties must be listed comma-separated on the first line. The formatting of the object class properties is specified as variables enclosed in curly brackets.

  If the **Format String** attribute is not defined for the object view, the **Format String** attribute for the class setting that the object view is assigned to will be displayed. The **Format String** attribute for an object view will have precedence over the **Format String** attribute for the class setting. If a **Format String** attribute is not specified for neither the object view nor the class setting, the following will be displayed: `<Object.Class.Caption|Object.Stereotype.Caption + Object.ID + ":" + Object.ClassSetting.ImageProps>`. If no image properties are defined in the class setting, the **Name** property will be used. If neither image properties nor the **Name** property are defined in the class setting, the header will be defined as follows: `<Object.Class.Caption|Object.Stereotype.Caption + Object.ID + ":">`.

  The **Format String** attribute can be translated. For more information about defining custom translations, see the section Modifying, Translating and Managing the Vocabularies in the chapter Localization and Multi-Language Support for the Alfabet Interface.

- **Icon**: You can optionally replace the default icon displayed for the object class in the Alfabet user interface. To do so, select the icon in the drop-down menu.

- **Foreground Color** and **Background Color**: (Optional) Select a color in the color picker to define the foreground and background colors when displaying an object in standard Alfabet views and configured reports with business graphics (for example, portfolios and bar charts).

  Please note the following:

  - If colors have been explicitly defined for configured reports, the specification in the configured report will have precedence over the class setting definition. For more information about the definition of configured reports, see Creating a Graphic Report.
  - If color rules are specified for the display of objects in map views and diagram views, the color rules will have precedence over the class setting definition. For more information about color rules, see the section *Configuring Color Rules for Map Views and Diagram Views* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.
  - If a color selector is configured in a custom editor, the custom color will have precedence over the class setting definition. For more information about

configuring the color selector for a custom editor, see the section Adding a Color Selector to the Custom Editor in the chapter Configuring Custom Editors in the reference manual *Configuring Alfabet with Alfabet Expand*.

- Users may also explicitly specify a color for objects in object classes that have a ***Color*** property and that the user has access permissions to. This is carried out via the **Colors**  button which is available in standard Alfabet views and configured reports with business graphics (for example, portfolios and bar charts) in the Alfabet user interface. Please note that if the user changes the color of an object, the selected color is used for the object in all views and reports for all Alfabet users. This capability is typically used for demonstration purposes and it is highly recommended that the **Colors**  button is hidden from Alfabet views and configured reports in order to prevent users from changing object colors across the user community. The **Colors**  button may be globally hidden via the **Allow Edit Object Colors** attribute in the server alias or may be hidden on a per-view basis. For more information about globally hiding the **Colors**  button via the server alias, see the section *Allowing Users to Change the Color Assigned to Objects in Business Graphics* in the reference manual *System Administration*.

- **Report Collection**: Configured reports selected in this attribute will be accessible via tabs in tabular configured reports and automatically generated reports returning objects of the object class or object class stereotype. This feature requires pre-configuration. Configured reports must explicitly be configured to provide meaningful information about the objects listed in the tabular dataset they are accessible from. The attribute should only be used as part of the complete configuration described in the section Integration of Configured Reports as Report Collection Into Tabular Configured Reports.

5) In the toolbar, click the **Save**  button to save your changes.

Creating a Class Setting for a Custom Object Class

If you have configured a custom object class, you must explicitly create a class setting for that object class. To do so, right-click the **Class Settings** folder and select **New Class Settings** in the context menu. In the **Select Class** editor, select the custom object class that you want to create the class setting for and click **OK**. A new class setting folder with the name of the selected custom object class will be added to the **Class Settings** folder and below the folder node the new class setting will be displayed. Define the class setting as described in the section Creating a Custom Class Setting for a Protected Object Class or Object Class Stereotype.

Configuring a View Scheme for a User Profile

A view scheme captures the information about which class settings are applied to the object classes that are implemented in the associated user profile. When you define the view scheme, you must clarify which class setting should be implemented for the relevant object classes that will be accessed via the view scheme. Only one class setting can be defined per object class for a view scheme. The class setting determines the object view to use as well as which custom editor or wizard should open when the **Edit** button is clicked in the toolbar of a page view or object view. If no class setting is explicitly defined for a view scheme, the default class setting (<ObjectClass>_Standard*) will be automatically applied to the view scheme definition. For more information about defining a class setting, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes.

Once the relevant class settings have been assigned to the view scheme, you can access the view scheme in the Alfabet interface and further refine the view scheme by hiding various elements from the Alfabet interface. In this phase of configuration, you can determine which standard and custom object class properties to exclude from editors, which functionalities (toolbar buttons) to exclude from the toolbars of object views and page views, and which page views to exclude from the explorer root node and object view.

Although it is possible to reuse a view scheme and assign it to multiple user profiles, it is recommended that only one view scheme be configured per user profile.

It is recommended that all views associated with administrative functions such as those typically accessible via the `Admin` user profile are explicitly excluded from the view schemes of all user profiles that are not authorized to perform administrative tasks. For on overview of these functionalities and the tasks that can be performed with them, see the reference manual *User and Solution Administration*.

It is advised that you contact Software AG Support before excluding an object class from a view scheme in order to avoid inadvertently hiding dependent views. Please note that if you exclude an object class, you will hide the object class from the **Search** functionality, relevant explorer trees, and any views in the Alfabet solution that are dependent on that object class. As an alternative to excluding an object class, you can hide entire workspaces or individual page views available for an object class and thus avoid interfering with dependent object classes. The following examples illustrate the impact of excluding an object class:

- If the object class `MarketProduct` is excluded from a view scheme, all page views relevant for the object class `BusinessSupport` will also be inaccessible in the context of other object classes. This is because the object class is one possible class that can be used in the definition of a business support. Therefore, any users accessing an organization, business process, or master plan map with a user profile that the view scheme is assigned to would not see the **Business Support Map** page views (for example, `ITMPM_Matrix`, `ITSM_Matrix`, or `ITMPM_MatrixReport` or any other page views displaying business supports.
- If the object class `BusinessFunction` is excluded from a view scheme, all page views relevant for the object class `BusinessService` will also be inaccessible in the context of other object classes. For example, any users accessing an application with a user profile that the view scheme is assigned to would not see the **Business Services** page view (`APP_ProvidedServices`) and **Services Matrix** page view (`APP_ProvidedServicesMatrix`).

The following information is available:

- Regulating Access Permissions via Class Settings

- Creating a View Scheme and Assigning Class Settings
- Assigning the View Scheme to a User Profile

Regulating Access Permissions via Class Settings

In some cases, you may want users to see data but have only limited or no permissions to edit particular information such as, for example, data about other users, reference data for costs or indicators, or the enterprise's master plans and IT strategies. The following table provides important information about which class settings should be assigned to the view schemes that are configured for specific types of user profiles. The recommendations should be implemented so that access permissions may be correctly executed in your Alfabet solution. Once you have defined which class settings are relevant for the selected view scheme, you will be able to refine the visibility issues for the functionalities, object views, editors, and page views assigned to the view scheme. This is explained in the section Refining Visibility Issues in the View Scheme.

Type of User Profile to Configure:	Recommendation:
For the `Admin` user profile. For on overview of the administrative functionalities that can be accessed via the `Admin` user profile, see the reference manual *User and Solution Administration*.	**A user logged on with the Admin user profile has edit permissions to all objects at all times, regardless of the user authorization definition or the release status assigned to an object. Therefore, it is critical that only users who may change such data are assigned the Admin user profile!** You should select the following standard version of the class settings provided by Software AG • `USER_Standard` for the class setting `Person` • `USERG_Standard` for the class setting `UserGroup` These class settings can be assigned to other user profiles that are responsible for editing and creating user and user group data. In this case, the function `ADMIN_UsersOverview` should be assigned to the relevant user profile.
User profiles that provide access to the object class `Person`:	If the users accessing a user profile should have only limited or no edit permissions to user information, then you must explicitly define either the `USER_ReadOnly` class setting or the `USER_LimitedEditability` class setting. Otherwise, the default class setting `USER_Standard*` will apply. The following standard class settings are available for the object class `Person`: • `USER_Standard*`: This is the default class setting and should **only** be assigned to users with administrative user profiles. The `USER_Standard*` class setting grants users the ability to edit all user data associated with objects that they have access permissions to.

Type of User Profile to Configure:	Recommendation:
	• `USER_LimitedEditability`: This class setting allows users to edit only custom object class properties for the object class `Person` associated with objects that they have access permissions to. This class setting is to be used for user profiles with conventional Read/Write access permissions. This is the class setting that is typically used for the majority of view schemes. • `USER_ReadOnly`: This class setting is to be used for user profiles that have ReadOnly access permissions to the user data associated with objects that they have access permissions to.
User profiles that provide access to the object class `UserGroup`:	If the users accessing a user profile should have only limited or no edit permissions to user group information, then you must explicitly define the `USERG_ReadOnly` class setting. Otherwise, the default class setting `USERG_Standard*` will apply. The following standard class settings are available for the object class `UserGroup`: • `USERG_Standard*`: This is the default class setting and should **only** be assigned to user groups with administrative user profiles. The `USERG_Standard*` class setting grants users in the user group the ability to edit all user group data associated with objects that they have access permissions to. • `USERG_ReadOnly`: This class setting is to be used for user profiles that should have ReadOnly access permissions to the user group data associated with objects that they have access permissions to.
User profiles accessing the functionalities in the **Configuration** module. For on overview of these functionalities and the tasks that can be performed with them, see the reference manual *Configuring Evaluation and Reference Data in Alfabet*.	You should select the following standard versions of the class settings provided by Software AG. These class settings provide Read/Write access permissions to the reference data configured in the **Configuration** module. • `COSTCT_Standard` for the class setting `CostCentreType` • `COSTTYP_Standard` for the class setting `CostType` • `EVALTYP_Standard` for the class setting `EvaluationType` • `INCTYP_Standard` for the class setting `IncomeType` • `INDTYP_Standard` for the class setting `IndicatorType` • `IDCT_CR_Standard` for the class setting `IndicatorTypeComputationRule`

Type of User Profile to Configure:	Recommendation:
	• `ROLETYP_Standard` for the class setting `RoleType` • `SKL_Standard` for the class setting `Skill` Only a user administrator should assign Read/Write access permissions to user profiles accessing the **Configuration** module. For more information, see the section *Defining and Managing User Profiles* in the reference manual *User and Solution Administration*.
User profiles with **NO** administrative or configuration responsibilities	You should select the following ReadOnly versions of the class settings provided by Software AG. • `COSTCT_ReadOnly` for the class setting `CostCentreType` • `COSTTYP_ReadOnly` for the class setting `CostType` • `EVALTYP_ReadOnly` for the class setting `EvaluationType` • `INCTYP_ReadOnly` for the class setting `IncomeType` • `INDTYP_ReadOnly` for the class setting `IndicatorType` • `IDCT_CR_ReadOnly` for the class setting `IndicatorTypeComputationRule` • `ROLETYP_ReadOnly` for the class setting `RoleType` • `SKL_ReadOnly` for the class setting `Skill` • `USER_ReadOnly` for the class setting `Person` • `USRG_ReadOnly` for the class setting `UserGroup`
For user profiles accessing the **View As-Is and To-Be** (`View_AsIs_ToBe`) functionality.	You should select the following ReadOnly versions of the class settings provided by Software AG. • `ITMP_ReadOnly` for the class setting `MasterPlan` • `ITMPFLD_ReadOnly` for the class setting `MasterPlanPlan` • `ITMPM_ReadOnly` for the class setting `MasterPlanMap` • `ITS_ReadOnly` for the class setting `Strategy` • `ITSFLD_ReadOnly` for the class setting `ITStrategyFolder` • `ITSM_ReadOnly` for the class setting `ITStrategyMap`
User profiles containing **Search** functionalities (for example,	You must ensure that all object classes that are to be searchable in the **Search** functionalities have the attribute **Searchable** set to `True` for the class setting assigned to the view scheme. The

Type of User Profile to Configure:	Recommendation:
Simple Search, **Full-Text Search**, etc.)	**Searchable** attribute is set to `True` per default for the class setting <ObjectClass> `_Standard`. For an overview of all object classes that are searchable per default in the standard class settings, see the chapter *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

Creating a View Scheme and Assigning Class Settings

A view scheme is associated with a user profile and groups a set of class settings that describe the visibility of objects in the associated object classes. Only one class setting per object class may be assigned to a view scheme.

The view scheme also contains the definition about the functionalities available in the object views accessible via the view scheme. For each standard and configured object view available to the view scheme, workspaces, page views, and toolbar buttons that allow object data to be created, edited, and deleted, for example, can be hidden.

To create a view scheme and assign class settings to it:

1) Go to the **Presentation** tab and expand the **View Schemes** node.
2) To create the view scheme, do one of the following:

- Right-click the **View Schemes** folder and select **New View Scheme** to create a new view scheme from scratch, or
- Click a view scheme you want to copy and select **New View Scheme as Copy** to create a new view scheme based on an existing view scheme.

3) You will see the new view scheme  added to the **View Schemes** folder. In the attribute window, enter a name in the **Name** attribute.

A validation mechanism checks for correct syntax when defining a technical name. The technical name of a configuration object may not begin with an empty space nor include any of the following special characters: **\ / * ? " > < |':**

Furthermore, please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.

Furthermore, if the technical name of a configuration object is changed, the name will be correctly updated in other configuration objects referencing the changed object during design time. Please note that the name of a changed configuration object will not be updated in guide pages nor if the guide pages reference a configuration object listed in the **Show Usage** functionality. If you plan to change the name of a configuration object, the reference in the guide page should be changed prior to changing the configuration object name.

4) You should now systematically determine whether each object class/object class stereotype requires a custom class setting or whether the standard class setting suffices for your needs. To open the **View Scheme** editor for the selected view scheme, double-click the view scheme you are configuring. The **View Scheme** editor opens in the center pane.

5) The table labelled **Object Class Configuration** displays object classes/object class stereotypes on the first level. Click the + to expand the table below the object class or object class stereotype that you want to configure. When you expand the object class node, you will see all existing standard class settings and custom class settings configured by your enterprise for the selected object class. You can do the following in the **Object Class Configuration** table:

- To specify which class setting should be implemented in the selected view scheme, click in the **Use in View Scheme** column, set an X for the class setting to be implemented for the selected view scheme. Carry out this procedure for all object classes relevant to the view scheme.
- To create a new class setting for a view scheme, select the object class/object class stereotype in the **Object Class Configuration** table and click the **Create Class Settings** button in the toolbar above the table section. A new class setting will be added below the existing class settings with a default name. Click the class setting to open the property grid and define the attributes as described in the section Configuring Class Settings for Object Classes and Object Class Stereotypes.
- To create an XLS file displaying all class settings used in the selected view scheme, click the **Export to Excel** button in the toolbar above the **Object Class Configuration** table.

6) The table labelled **View Configuration** displays the configured object views, graphic views, and presentation objects implemented in the selected view scheme. Select an object view or presentation object that you want to remove from the view scheme and click the **Delete Selected Configuration** button. The selected configuration will be removed from the view scheme but will not be deleted from other view schemes that it might be assigned to. To create an XLS file displaying all object view and presentation object configurations used in the selected view scheme, click the **Export to Excel** button in the toolbar above the **View Configuration** table.

7) In the toolbar, click the **Save** button to save your changes. Once you have defined which class settings are relevant for the selected view scheme, you will be able to refine the visibility issues for the object views, editors, and page view functionalities for the view scheme. This is explained in the section Refining Visibility Issues in the View Scheme.

Assigning the View Scheme to a User Profile

In order to further refine visibility issues for the view scheme, you must access the view scheme in the Alfabet interface. Therefore, the view scheme must be assigned to a user profile in order to access Alfabet. Once Alfabet opens, you can access all object views associated with the class settings defined for the view scheme and configure the object views, editors, functionality, etc. available in the selected view scheme.

User profiles are typically created by a user administrator in your enterprise. User profiles can be created in Alfabet Expand as well as in the tool Alfabet Administrator or in the **User Profiles Administration** functionality that can be accessed via the `Admin` user profile.

Please note that changes made to user profiles via the **User Profiles Administration** functionality that can be accessed via the `Admin` user profile are not automatically updated to the user profile configuration in the configuration tool Alfabet Expand that is concurrently in use. The

solution designer working in Alfabet Expand must either use the **Rescan Tree** functionality or close and reopen the database to view the updated user profiles.

1) Go to the **Admin** tab, expand the **User Profiles** node, and select the user profile that you want to assign the view scheme to.
2) In the attribute window of the selected user profile, click the drop-down button in the **View Scheme** attribute to select the relevant view scheme.
3) In the toolbar, click the **Save** button to save your changes.

Once a view scheme has been assigned to a user profile, you can further refine visibility in the view scheme as well as review and test your configuration in the Alfabet interface. For more information, see Refining Visibility Issues in the View Scheme.

Refining Visibility Issues in the View Scheme

Two functionalities are available in the context menu of a user profile that allow you to view the solution configuration in a fully-functional Alfabet interface. The option **Run Application with Selected User Profile** opens Alfabet as it would be run for the end user and configuration is not possible. The option **Configure User Profile** opens Alfabet in configuration mode so that the solution designer can configure user profiles and refine visibility issues.

To access the Alfabet interface via the **Run Application with Selected User Profile** functionality or the **Configure User Profile** functionality, the solution designer must be assigned as a user to the user profile that is being used to access the Alfabet interface. The access permissions defined for the user profile (ReadOnly or Read/Write) are also applied when accessing Alfabet via the **Run Application with Selected User Profile** option.

Before you can access the Alfabet interface or further configure visibility issues for a view scheme, you must assign the relevant view scheme to a user profile. This is necessary to specify visibility issues in a view scheme as well as review/test your configuration of the Alfabet solution. For more information about assigning a view scheme to a user profile, see the section Assigning the View Scheme to a User Profile.

The configuration of class settings, view schemes, and user profiles play a role in what aspects of the configuration will be available to a user with a specified user profile. The following overview provides a guidline of which configuration object is relevant to specify visibility.

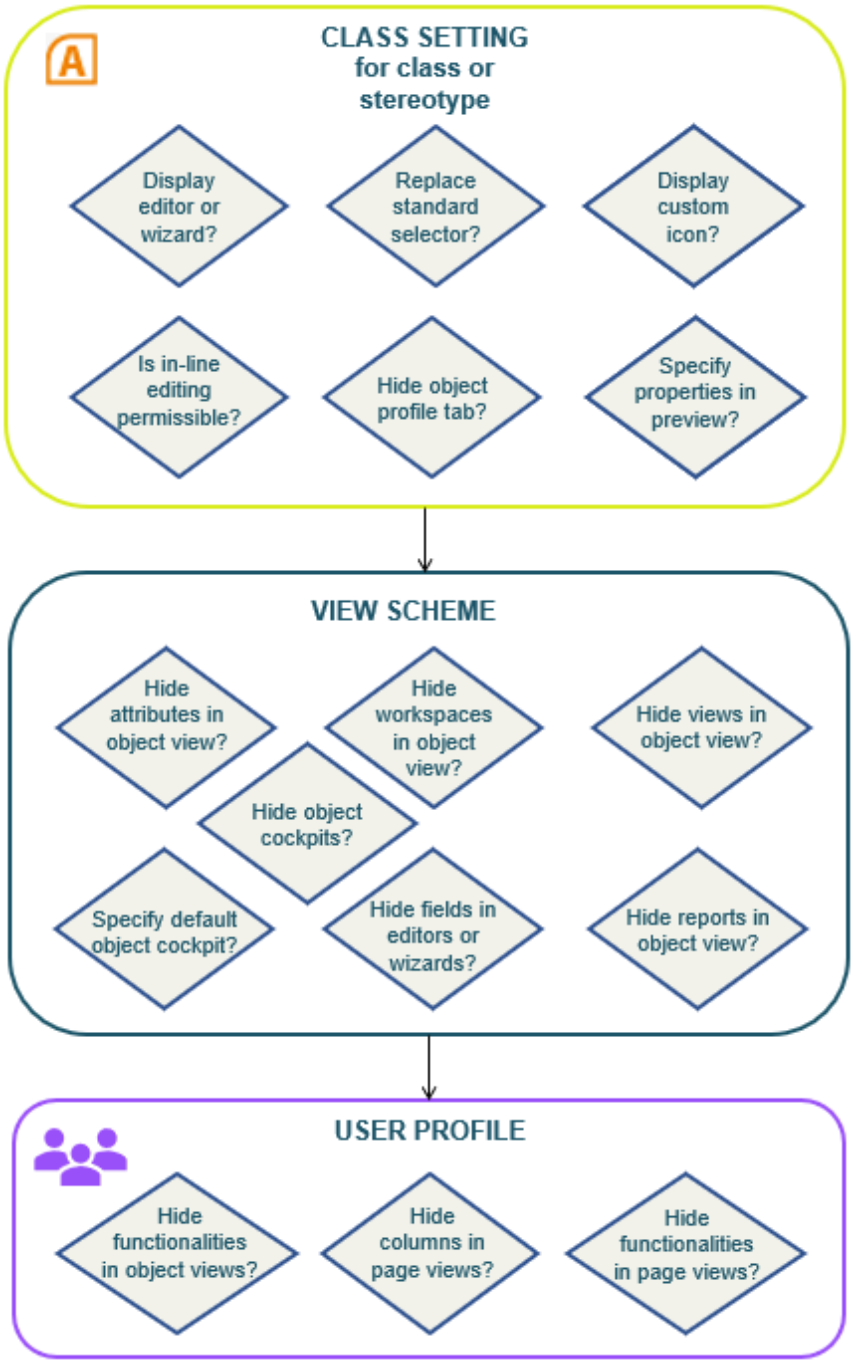


FIGURE: Configuration objects controlling visibility

Once a view scheme has been assigned to a user profile, you can access the Alfabet interface and refine the following for the view scheme:

- Visibility of standard and custom object class properties in standard editors, custom editors, and custom wizards
- Visibility of object class properties (in the **Attributes** section of the object profile), workspaces, page views and configured reports in standard and custom object views
- Visibility of standard and custom object class properties in page views and configured reports

- Functionality available via toolbar buttons in object views, object cockpits, page views, and configured reports (for example, **Create**..., **Copy**..., **Edit**..., **Navigate**..., etc.).

You can access the Alfabet interface from Alfabet Expand by right-clicking the user profile  that has the relevant view scheme assigned and select **Configure User Profile**. The Alfabet interface will open in a browser window.

After you have customized the view scheme (for example, by customizing a wizard or object view) in the Alfabet Web interface, you must reread the meta-model in order to update the changes to Alfabet Expand. Therefore, once the **Customization Editor** has been closed and you have returned to the Alfabet Expand interface, you must select **Meta-Model** > **Reread Meta-Model** to ensure that the customization is not overwritten.

The following information is available:

- Hiding Object Class Properties in Editors and Wizards
- Defining the Visibility of Object Class Properties, Workspaces, Page Views/Configured Reports, and Object Cockpits in an Object View
- Defining the Visibility of Page Views/Configured Reports Available at the Root Node of an Explorer
- Specifying the Visibility of Object Class Properties in Page Views
- Hiding Functionalities in an Object View
- Hiding Functionalities in a Page View or Configured Report

Hiding Object Class Properties in Editors and Wizards

You can hide standard or custom object class properties in standard or custom editors as well as the editors implemented in wizards. Please note the following about hiding object class properties in editors and wizards:

- Any object class properties that you hide in an editor cannot be defined or edited by users accessing Alfabet with the associated user profile.
- A red star in an editor indicates that the object class property is a mandatory property. It is possible to exclude a mandatory property in an editor or wizard as long as the mandatory property is not part of a class key definition. A class key definition specifies one or a combination of properties that must have unique values when the object is created. For more information about the configuration of class keys, see the section Configuring Class Keys for Object Classes in the chapter Configuring the Class Model.
- In conjunction with the task of hiding object class properties in editors, you can also specify a custom selector that is to be used in place of the standard object selector in editor fields that require a reference to an object to be defined. For more information about the configuration of a custom selector, see the section Configuring a Custom Selector for Search Functionalities.
- Properties that are hidden in an editor are ONLY excluded in the editor for the selected view scheme. If the object class properties should not be displayed at all in the Alfabet interface, then you must ensure that they are completely excluded from the view scheme and therefore not displayed in the **Attributes** section of a custom object view, page views, and configured reports.

- Properties that are hidden in an editor embedded in a wizard are excluded in that wizard for all user profiles that the wizard is assigned to.

  The configuration of a custom wizard will apply to all instances of the custom wizard in Alfabet. For example, if you hide object class properties or functionalities in a custom wizard, the object class properties or functionalities will be hidden for all instances of the custom wizard. In other words, a selected wizard cannot be differently configured in the context of different object class stereotypes. The wizard configuration will apply to each object class stereotype that the custom wizard is assigned to. If different configurations are required for a custom wizard, it is recommended that you create a new wizard via the copy functionality and modify the new wizard as needed. Once the wizard is configured, it can be selected in the **Wizard** attribute for the class setting that you create for the object class stereotype.

- If you are configuring a custom wizard, you will typically have wizard steps in which an editor, page view, or configured report is embedded. Follow the procedure described below to configure the embedded editor. You can also configure the page views and configured reports in the custom wizard, as described in the sections Specifying the Visibility of Object Class Properties in Page Views and Hiding Functionalities in a Page View or Configured Report. For a complete description of configuring visibility issues in a custom wizard, see the section Hiding Object Class Properties and Functionalities in the Wizard for a Specific User Profile in the chapter Configuring Wizards.
- If you are configuring the `DPL_Editor`, the `Stack` object class property may not be hidden otherwise an error may occur when deployments are created.
- System editors such as the **User Settings** (`ALFA_USER_SETTINGS_Editor`) editor or the **Personal Info** (`User_PersonalInfo_Editor`) editor are not configurable.

To hide object class properties in an editor:

1) Open the Alfabet interface by right-clicking the relevant user profile and selecting **Configure User Profile**.
2) Navigate to the editor or wizard that you want to configure. This could be the editor available via the **Edit**  button in an object view or via the relevant menu or button option in the toolbar of a page view.
3) In the upper right corner of the editor, click the **Configure View**  button. The **Customization Editor** opens and displays all protected and custom object class properties available for the object class.
4) Define the following as needed:

- **Select Configuration Object**: Displays the current configuration object that is being configured.
- **Select View Scheme**: Select the view scheme that the configuration is relevant for.
- **Select Hierarchy Level**. Select the hierarchy level for which the configuration is being defined. The levels are listed in the order of the hierarchy, starting with **All Classes**, followed by the object class and then the object class stereotypes.
- **Currently Applied Settings**: Displays the inherited configuration that is applied to the selected configuration object if the option **Inherited Settings from Parent Configuration Object** is selected in the **Choose One Settings Mode** menu. This field will be empty if settings have not been inherited.

- Select the relevant option in the **Choose One Settings Mode** menu in the toolbar of the dataset. The options available will depend on the configuration object that you are working with:
- **Explicit Settings from Selected Configuration Object**: Select to display a dataset that allows visibility to be explicitly specified for the relevant aspects of the selected configuration object displayed in the **Select Configuration Object** field.
- **Inherited Settings from Parent Configuration Object**: Select if the selected configuration object shall inherit the exclusion settings configured for the relevant parent configuration object. The **Currently Applied Settings** field will display the inherited configuration that is applied to the selected configuration object if the option **Inherited Settings from Parent Configuration Object** is selected in the **Choose One Settings Mode** menu.
- **No Explicit Settings from Configuration Object**: Select to display the current settings for the selected configuration object. This dataset allows no explicit settings to be configured. The **No Explicit Settings from Configuration Object** option is displayed instead of the **Inherited Settings from Parent Configuration Object** option if no settings can be inherited.

5) If **Explicit Settings from Selected Configuration Object** has been selected in the **Choose One Settings Mode** menu, you can configure the visibility of the tabs and object class properties in the editor as well as whether to use a custom selector instead of a standard object selector for relevant fields. The **View Configuration** editor displays the following columns for an editor:

- **Name**: Displays the technical name of a tab or object class property in a tab.
- **Caption**: Displays the caption defined for the tab or object class property in a tab.
- **Default Selector**: Displays the name of the default selector used to find objects if an object selector is required to define the object class property. This field will only by filled in if an object selector is actually implemented.
- **Excluded**: Click in the cell to set an X to specify that the respective element shall be hidden in the wizard. Click the X to clear the exclusion setting and make the element visible in the editor.
- **Custom Selector**: Select a custom selector to replace the default selector displayed in the **Default Selector** column. A drop-down menu will display all custom selectors configured for your solution. You must ensure that you select a custom selector relevant to the context for which it will be used. For more information about the configuration of a custom selector, see the section Configuring a Custom Selector for Search Functionalities.

6) Click the **OK** button to save the changes in the **Customization Editor** editor or click **Cancel** to exit without saving the changes.

Defining the Visibility of Object Class Properties, Workspaces, Page Views/Configured Reports, and Object Cockpits in an Object View

The visibility of object class properties, workspaces, page views, and configured reports can be specified for a standard object view or custom object view. Additionally, you can specify which object cockpits are visible for the user profile as well as which object cockpit is the default object cockpit that the user will navigate to the first time in the user session.

Please note that the first time that a user navigates to an object view for a specific object class, the object cockpit configured as the default cockpit will be displayed. If no object cockpit is specified as the default for the user profile, the user will automatically navigate to the first object

cockpit in the list of object cockpits in the **Default** field in the **View Configuration** editor available via a user profile. If no default object cockpit has been configured, the object profile will be displayed per default.

Please note that this configuration is done at the level of the individual object view. Each relevant object view must be explicitly configured. For each view scheme assigned to a user profile, the following should be configured:

- Hide standard or custom object class properties assigned to the **Attributes** section of the object profile
- Hide individual page views and configured reports.
- Hide an entire workspace. When you hide an entire workspace, all page views assigned to the workspace will also be hidden.
- Hide individual object cockpits
- Specify an object cockpit as the default object cockpit that is displayed when the user navigates to the object view. The user can switch to another cockpit, as needed.
- Specify which functionalities are available in the toolbar of the object view/object cockpit. This is described in the section Hiding Functionalities in an Object View.

To configure the visibility of object class properties, page views, configured reports, workspaces, and object cockpits in an object view:

1) Open the Alfabet interface by right-clicking the relevant user profile and selecting **Configure User Profile**.
2) Navigate to the object view that you want to configure and click the **Configure View**  button in the upper right corner of the view. The **Customization Editor** opens.
3) Define the following as needed:

- **Select Configuration Object**: Displays the current configuration object that is being configured.
- **Select View Scheme**: Select the view scheme that the configuration is relevant for.
- **Select Hierarchy Level**. Select the hierarchy level for which the configuration is being defined. The levels are listed in the order of the hierarchy, starting with **All Classes**, followed by the object class and then the object class stereotypes.
- **Currently Applied Settings**: Displays the inherited configuration that is applied to the selected configuration object if the option **Inherited Settings from Parent Configuration Object** is selected in the **Choose One Settings Mode** menu. This field will be empty if settings have not been inherited.
- Select the relevant option in the **Choose One Settings Mode** menu in the toolbar of the dataset. The options available will depend on the configuration object that you are working with:
- **Explicit Settings from Selected Configuration Object**: Select to display a dataset that allows visibility to be explicitly specified for the relevant aspects of the selected configuration object displayed in the **Select Configuration Object** field.
- **Inherited Settings from Parent Configuration Object**: Select if the selected configuration object shall inherit the exclusion settings configured for the relevant parent configuration object. The **Currently Applied Settings** field will display the inherited configuration that is applied to the selected

configuration object if the option **Inherited Settings from Parent Configuration Object** is selected in the **Choose One Settings Mode** menu.

- **No Explicit Settings from Configuration Object**: Select to display the current settings for the selected configuration object. This dataset allows no explicit settings to be configured. The **No Explicit Settings from Configuration Object** option is displayed instead of the **Inherited Settings from Parent Configuration Object** option if no settings can be inherited.

4) If **Explicit Settings from Selected Configuration Object** has been selected in the **Choose One Settings Mode** menu, you can configure the visibility of the **Attribute Section**, a section for each workspace, and a **Cockpits** section. Expand the section of the table in order to display the respective attributes, views, or cockpits that you want to configure by clicking the + symbol. If necessary, navigate to the next page in the **View Configuration** editor. The **View Configuration** editor displays the following columns:

- **Name**: Displays the name of the interface element in the object view (**Attribute** section, attributes, reports, workspaces, views, etc.) in the object view.
- **Caption**: Displays the caption defined for the interface element in the object view.
- **Type**: Displays the type of interface element in the object view (`Attribute`, `Report`, `Workspace`, `Cockpit`, etc.)
- **Excluded**: Click in the cell to set an X to specify that the respective interface element in the object view should be hidden. For example, to hide an object cockpit, scroll to the **Cockpits** section of the table and set an X in the **Excluded** column for the object cockpit that you want to hide for the selected view scheme.
- **Default Cockpit**: Click in the cell to set an X to specify that the object cockpit is the default view. The first time that a user navigates to an object view for a specific object class, the default object cockpit configured will be displayed. The user can switch to another object cockpit as needed. If no object cockpit is specified as the default for the user profile, the user will automatically navigate to the first object cockpit in the list of object cockpits in the **Default** field in the **View Configuration** editor of the user profile. If no object cockpits have been configured, the object profile will be displayed per default.
- **Usage Type**: Displays the specification of the user type that the view is intended for (`All`, `Viewer`, `Data Entry`, `Functional`, `Expert`).

5) Click the **OK** button to save the changes in the **Customization Editor** editor or click **Cancel** to exit without saving the changes.

To configure the visibility of object class properties, page views, configured reports, workspaces, and object cockpits in an object view:

1) Open the Alfabet interface by right-clicking the relevant user profile and selecting **Configure User Profile**.

2) Navigate to the object view that you want to configure and click the **Configure View**  button in the upper right corner of the view. The **Customization Editor** opens.

3) Define the following as needed:

- **Select Configuration Object**: Displays the current configuration object that is being configured.
- **Select View Scheme**: Select the view scheme that the configuration is relevant for.

- **Select Hierarchy Level**. Select the hierarchy level for which the configuration is being defined. The levels are listed in the order of the hierarchy, starting with **All Classes**, followed by the object class and then the object class stereotypes.
- **Currently Applied Settings**: Displays the inherited configuration that is applied to the selected configuration object if the option **Inherited Settings from Parent Configuration Object** is selected in the **Choose One Settings Mode** menu. This field will be empty if settings have not been inherited.
- Select the relevant option in the **Choose One Settings Mode** menu in the toolbar of the dataset. The options available will depend on the configuration object that you are working with:
- **Explicit Settings from Selected Configuration Object**: Select to display a dataset that allows visibility to be explicitly specified for the relevant aspects of the selected configuration object displayed in the **Select Configuration Object** field.
- **Inherited Settings from Parent Configuration Object**: Select if the selected configuration object shall inherit the exclusion settings configured for the relevant parent configuration object. The **Currently Applied Settings** field will display the inherited configuration that is applied to the selected configuration object if the option **Inherited Settings from Parent Configuration Object** is selected in the **Choose One Settings Mode** menu.
- **No Explicit Settings from Configuration Object**: Select to display the current settings for the selected configuration object. This dataset allows no explicit settings to be configured. The **No Explicit Settings from Configuration Object** option is displayed instead of the **Inherited Settings from Parent Configuration Object** option if no settings can be inherited.

4) If **Explicit Settings from Selected Configuration Object** has been selected in the **Choose One Settings Mode** menu, you can configure the visibility of the **Attribute Section**, a section for each workspace, and a **Cockpits** section. Expand the section of the table in order to display the respective attributes, views, or cockpits that you want to configure by clicking the + symbol. If necessary, navigate to the next page in the **View Configuration** editor. The **View Configuration** editor displays the following columns:

- **Name**: Displays the name of the interface element in the object view (**Attribute** section, attributes, reports, workspaces, views, etc.) in the object view.
- **Caption**: Displays the caption defined for the interface element in the object view.
- **Type**: Displays the type of interface element in the object view (`Attribute`, `Report`, `Workspace`, `Cockpit`, etc.)
- **Excluded**: Click in the cell to set an X to specify that the respective interface element in the object view should be hidden. For example, to hide an object cockpit, scroll to the **Cockpits** section of the table and set an X in the **Excluded** column for the object cockpit that you want to hide for the selected view scheme.
- **Default Cockpit**: Click in the cell to set an X to specify that the object cockpit is the default view. The first time that a user navigates to an object view for a specific object class, the default object cockpit configured will be displayed. The user can switch to another object cockpit as needed. If no object cockpit is specified as the default for the user profile, the user will automatically navigate to the first object cockpit in the list of object cockpits in the **Default** field in the **View Configuration** editor of the user profile. If no object cockpits have been configured, the object profile will be displayed per default.
- **Usage Type**: Displays the specification of the user type that the view is intended for (`All`, `Viewer`, `Data Entry`, `Functional`, `Expert`).

5) Click the **OK** button to save the changes in the **Customization Editor** editor or click **Cancel** to exit without saving the changes.

Defining the Visibility of Page Views/Configured Reports Available at the Root Node of an Explorer

The visibility of page views and configured reports available for the root node of an explorer can be specified. To configure the visibility of page views configured reports for the workspace associated with an explorer root node:

1) Open the Alfabet interface by right-clicking the relevant user profile and selecting **Configure User Profile**.
2) Navigate to the explorer that you want to configure and in the upper right corner of the workspace, click the **Configure View**  button. The **Customization Editor** opens.
3) Define the following as needed:

- **Select Configuration Object**: Displays the current configuration object that is being configured.
- **Select View Scheme**: Select the view scheme that the configuration is relevant for.
- **Select Hierarchy Level**. Select the hierarchy level for which the configuration is being defined. The levels are listed in the order of the hierarchy, starting with **All Classes**, followed by the object class and then the object class stereotypes.
- **Currently Applied Settings**: Displays the inherited configuration that is applied to the selected configuration object if the option **Inherited Settings from Parent Configuration Object** is selected in the **Choose One Settings Mode** menu. This field will be empty if settings have not been inherited.
- Select the relevant option in the **Choose One Settings Mode** menu in the toolbar of the dataset. The options available will depend on the configuration object that you are working with:
- **Explicit Settings from Selected Configuration Object**: Select to display a dataset that allows visibility to be explicitly specified for the relevant aspects of the selected configuration object displayed in the **Select Configuration Object** field.
- **Inherited Settings from Parent Configuration Object**: Select if the selected configuration object shall inherit the exclusion settings configured for the relevant parent configuration object. The **Currently Applied Settings** field will display the inherited configuration that is applied to the selected configuration object if the option **Inherited Settings from Parent Configuration Object** is selected in the **Choose One Settings Mode** menu.
- **No Explicit Settings from Configuration Object**: Select to display the current settings for the selected configuration object. This dataset allows no explicit settings to be configured. The **No Explicit Settings from Configuration Object** option is displayed instead of the **Inherited Settings from Parent Configuration Object** option if no settings can be inherited.

4) If **Explicit Settings from Selected Configuration Object** has been selected in the **Choose One Settings Mode** menu, you can configure the visibility of the page views available on the root node of the explorer. The **Customization Editor** editor displays the following columns:

- **Name**: Displays the name of the page views and configured reports.
- **Caption**: Displays the caption defined for the page views and configured reports.

- **Type**: Displays either `GraphicView` or `Report`.
- **Usage Type**: Displays the specification of the user type that the view is intended for (`All`, `Viewer`, `Data Entry`, `Functional`, `Expert`).
- **Excluded**: Click in the cell to set an X to specify that the respective interface element in the object view should be hidden.

5) Click the **OK** button to save the changes in the **Customization Editor** editor or click **Cancel** to exit without saving the changes.

Specifying the Visibility of Object Class Properties in Page Views

You can define a default layout for many of the tabular reports available in standard page views. Tabular reports are views that display a table with columns. Each column represents a standard or custom object class property. The values for the standard or custom object class property are displayed in the column cells.

Please note the following:

- If the same page view is reused in another object view, the configuration will be applied to all instances of the page view associated with the selected view scheme.
- Each Alfabet user may configure the layout of standard Alfabet views displaying a tabular datasets if the **Allow User Specific Configuration** parameter in the server alias is set to `True`. When set to `False`, configuration of the layout of tabular reports is restricted to a solution designer accessing the Alfabet solution via Alfabet Expand and t configuration is then implemented for all users.

Tabular configured reports are always configurable by the Alfabet user independent of the setting defined in the **Allow User Specific Configuration**. The user-specific configuration is activated per report. The layout can then be changed by the user and is stored independently of the user profile in the user context settings. The solution designer cannot define a layout valid for all users logged in with the same user profile, even if the **Allow User Specific Configuration** parameter in the server alias is set to `FALSE`.

For more information about the **Allow User Specific Configuration** attribute, see the section *Configuration Attributes for the Alfabet Components* in the reference manual *System Administration*.

- The following cannot be hidden in page views as it represents core functionality for the view:

- **Comments** column in the **Affected Architecture** page view (`PRJ_AffectedArchitectureElements`)
- **CRUDP** and **In/Out** columns in the **Business Data** page view (`COM_BusinessData`)
- **CRUDP** and **In/Out** columns in the **Business Objects** page view (`BFN_BusinessObjects`)
- Please note that custom object class properties that display URLs (**Property Type** = `URL`) cannot be displayed via columns in a page view/configured report and are not available in the **Presentation Object Configuration** editor.

To configure the visibility of object class properties (columns) in standard page views:

1) Open the Alfabet interface by right-clicking the relevant user profile and selecting **Configure User Profile**.

2) Navigate to the page view or configured report in the relevant object view.

3) In the toolbar, click the **Configure View**  button to open the **Presentation Object Configuration** editor. Set a checkmark in the **Visible** column for each custom object class property that should be visible as a column in the view. Clear a checkmark to hide a custom property in the view. You can use the **Select All** and **Clear All** buttons to select or clear the checkboxes in the **Visible** column.

   Please note that if object class properties have been defined as excluded for the associated class setting, they will not be displayed on the interface of the relevant editor. However, for technical reasons, they will continue to be listed as visible in the **Visible** column of the **Presentation Object Configuration** editor even though the object class property is not visible:

4) In the **Presentation Object Configuration** editor, specify the sequence of the columns in the view by selecting an object class property and clicking the **Up** and **Down** buttons. Repeat this step until all object class properties specified as visible are in the correct sequence.

5) To freeze columns so that they cannot be resized or resequenced in the view, select the number of columns that shall be frozen in the dataset in the **Number of Fixed Columns** field. The columns will be frozen starting with the first column on the left. The fixed columns cannot be manually resized or resequenced.

   If the **Number of Fixed Columns** s attribute is defined in the **Presentation Object Configuration** editor and a view is exported to PDF, the specification of the **Number of Fixed Columns** attribute might be ignored and the dataset will be exported as a normal table without fixed columns. Please note that the fixed column definition will only be dropped if the number of fixed columns makes paging of the table across multiple PDF pages impossible with a row header repetition logic. If this is undesired, the user is advised to select a different paper format and rerun the export.

6) Click the **OK** button to save the changes in the **Presentation Object Configuration** editor or click **Cancel** to exit without saving the changes.

Hiding Functionalities in an Object View

Object views typically feature a toolbar with menus and buttons that allow you to carry out tasks relevant to the object. For example, toolbar buttons are usually available that allows users to edit an object, trigger a workflow, access an audit history about the object, and publish the object's data to an Microsoft Word or a PDF file, etc. You can hide many toolbar buttons that provide functionality for a selected object view. In this case, the buttons will be greyed out and cannot be used. Please note the following about the configuration of the toolbar:

- Most functionalities can be disabled so that they are not displayed in the toolbar, thus preventing users from performing certain actions in the selected object view. These functionalities are displayed in the **Customization Editor**.

- Some functionalities are part of the standard configuration of Alfabet, and their visibility cannot be configured. Please note the following
- Standard functionalities that cannot be disabled will not be displayed in the **Customization Editor**.
- The **Workflow** button that is available for ALL object views, but it will be hidden automatically if the current object class is not referred to for a workflow template via the **Base Class** or **Source Class** attributes. The **Workflow** button may be hidden if needed.
- The **Add to Clipboard** button is available in the toolbar of object profiles for the classes **Application** (`App_Profile`), **Component** (`COM_Profile`), **Demand** (`DEM_Profile`), **Device** (`DEV_Profile`), **ICT Object** (`ICTO_Profile`), **Peripheral** (`PER_Profile`), **Project** (`PRJ_Profile`), **Organization** (`ORG_BSP_Profile`), **Standard Platform** (`SPL_Profile`), **Service Product** (`SRVPRD_Profile`), **Technology** (`TLG_Profile`), **Vendor** (`VDR_Profile`), **Vendor Product** (`VP_Profile`), and `Generic_Profile`).
- The **Audit Trail** button in the object view toolbar provides access to the **Object Audit** view (`Object_Audit`). Please note that the **Object Audit** view (`Object_Audit`) displays ALL information about the object regardless of the access permissions assigned to the user accessing the view. If this information should not be available to users accessing the object view, then it is recommended that you hide the standard **Audit Trail** button from the object view toolbar and create a configured report based on native SQL that the provides a customized display of audit information for users. For information about the structure of the database table for auditing and how to configure such a report, see the section Defining Audit Management Related Configured Reports in the chapter Configuring Reports. Once the report is configured, you must make it available in the relevant custom object view by configuring a custom button for the relevant custom object view. If the standard **Audit Trail** button should not be implemented in the custom object view, you must ensure that it is hidden in the relevant view scheme. For more information, see the section Configuring Custom Buttons for the Toolbar of a Custom Object View in the chapter Configuring Object Views.
- If a drop-down menu provides one or multiple options, you can hide one or more of these functionalities. Please note that if you hide all functionalities available for a menu, you must explicitly exclude the parent menu in order to ensure that it is hidden from the toolbar.
- You can also specify a custom selector to open in place of the standard selector for relevant toolbar options. The configuration of custom selectors and implementation in the Alfabet solution is described in detail in the section Configuring a Custom Selector for Search Functionalities.
- Please note that this configuration is done at the level of the individual object view. Each relevant object view must be explicitly configured.

To configure the toolbar for a selected object view:

1) Open the Alfabet interface by right-clicking the relevant user profile and selecting **Configure User Profile**.
2) Navigate to the object view of any object in the object class for which you want to define the visibility of object view functionalities and in the upper right corner of the workspace, click the **Configure View** button. The **Customization Editor** opens.
3) To specify the visibility of functionalities, select `Toolbar` in the **Select Configuration Object** field.

4) In the **Choose One Settings Mode** menu in the toolbar of the dataset, select **Explicit Settings from Selected Configuration Object** to display a dataset that allows visibility to be explicitly specified for the selected object view.

5) In the **Excluded** column, click in the cell to set an X to specify that the respective menu or action button is hidden in the view.

> If a button includes a drop-down menu providing one or multiple options, you can hide one or more of these functionalities. Please note that if you hide all functionalities available for a button, you must explicitly exclude the parent button in order to ensure that it is hidden from the toolbar.

6) Click the **OK** button to save the changes in the **Customization Editor** or click **Cancel** to exit without saving the changes.

Hiding Functionalities in a Page View or Configured Report

Alfabet standard page views feature a variety of functionalities in the toolbar. For example, toolbar buttons are typically available that allows users to create a new object, add an existing object, edit, delete or detach an object, or navigate to an object view. Most functionalities can be hidden so that they are not displayed in the toolbar, thus preventing users from performing certain actions in the selected page view. Please note the following about the configuration of the toolbar:

- **Please note that this configuration is done at the level of the page view. If the same page view is reused in another object view, the configuration will be applied to all instances of the page view associated with the selected view scheme.**
- Most functionalities can be hidden so that they are not displayed in the toolbar, thus preventing users from performing certain actions in the selected page view. These functionalities are displayed in the **Customization Editor**.

  > The **Export** button cannot be excluded from configured console reports although it is displayed in the **Customization Editor**.

- Some functionalities are part of the standard configuration of Alfabet. This is the case, for example, with the **Add to Clipboard** button or **Configure View** button. Standard functionalities cannot be hidden and are therefore NOT displayed in the **Customization Editor**.
- Functionalities in page views cannot be hidden in the context of a wizard. Wizards must be configured in the context of the **Configure Wizard** option available for the wizard.
- If a drop-down menu provides one or multiple options, you can hide one or more of these functionalities. Please note that if you hide all functionalities available for a menu, you must explicitly exclude the parent menu in order to ensure that it is hidden from the toolbar.
- If the class settings defined in the view scheme of the user profile allow context creation via the AlfaBot for an object class, at least one button marked as available for the AlfaBot should be included in the user profile configuration. Set a checkmark in the **Available for AlfaBot** column of the **Customization Editor** if a button shall be used by the AlfaBot to identify this page as relevant for context dependent creation of an object. For more information about the configuration of the AlfaBot, see the section Implementing the AlfaBot for Navigation via a Full-Text Command.

- You can also specify a custom selector to open in place of the standard selector for relevant toolbar options. The configuration of custom selectors and implementation in the Alfabet solution is described in detail in the section Configuring a Custom Selector for Search Functionalities.
- If you are configuring the **Responsibilities** page view, it is possible to exclude role types from the **Person** menu and **Organization** menu. For each relevant role type, click in the **Excluded** column for the corresponding row.
- If users should not have access to the Alfabet Diagram Designer, then the **Open Diagram** button available in diagram page views should be hidden in those page views. For an overview of all views in which the **Open Diagram** button is available, see the section *Page Views Providing Access to the Alfabet Diagram Designer* in the reference manual *Designing IT Landscape Diagrams in Alfabet*.
- Users may also explicitly specify a color for objects in object classes that have a ***Color*** property and that the user has access permissions to. This is carried out via the **Colors** button which is available in standard Alfabet views and configured reports with business graphics (for example, portfolios and bar charts) in the Alfabet user interface. Please note that if the user changes the color of an object, the selected color is used for the object in all views and reports for all Alfabet users. This capability is typically used for demonstration purposes, and it is highly recommended that the **Colors** button is hidden from Alfabet views and configured reports in order to prevent users from changing object colors across the user community. The **Colors** button may be globally hidden via the **Allow Edit Object Colors** attribute in the server alias or may be hidden on a per-view basis. For more information about globally hiding the **Colors** button via the server alias, see the section *Allowing Users to Change the Color Assigned to Objects in Business Graphics* in the reference manual *System Administration*.
- Several page views have two buttons or menu options with the same caption. These buttons/menu options typically open similar editors with some differences in functionality. Each button/menu option has a different tooltip describing the functionality of the editor that will open. If these views are implemented in the selected user profile, you should navigate to the following views and hide one of the buttons:

Technical Name of Page View	Caption of Page View	Duplicate Button/Menu Option
APP_CaptureApplications	**Capture Applications**	**Mass Update**
APP_CaptureApplications_Ex	**Capture Applications**	**Mass Update**
APP_Lifecycle	**Lifecycle**	**Edit Lifecycle**
APP_UserApplications	**Document Applications**	**Mass Update**
BusinessData	**Business Data**	**Edit Attributes**

Technical Name of Page View	Caption of Page View	Duplicate Button/Menu Option
CNTR_CaptureContracts	**Capture Contracts**	**Mass Update**
CNTR_CaptureContracts	**Capture Contracts**	**Mass Update**
COM_CaptureComponents	**Capture Components**	**Mass Update**
COM_CaptureComponents_Ex	**Capture Components**	**Mass Update**
COM_Lifecycle	**Lifecycle**	**Edit Lifecycle**
COM_UserComponents	**Document Components**	**Mass Update**
COM_InformationFlows	**Information Flows**	**Create New Incoming Information Flow.**
COM_InformationFlows	**Information Flows**	**Create New Outgoing Information Flow.**
COMCT_Lifecycle	**Lifecycle**	**Edit Lifecycle**
DEV_CaptureDevices	**Capture Devices**	**Mass Update**
DEV_CaptureDevices_Ex	**Capture Devices**	**Mass Update**
DEV_UserDevices	**Document Devices**	**Mass Update**
ENTRLS_TimeSchedule	**Enterprise Release Time Schedule**	**Show Enterprise Milestones**
ICTO_CaptureICTObjects	**Capture ICT Objects**	**Mass Update**
ICTO_CaptureICTObjects_Ex	**Capture ICT Objects**	**Mass Update**

Technical Name of Page View	Caption of Page View	Duplicate Button/Menu Option
ICTO_Lifecycle	**Lifecycle**	**Edit Lifecycle**
ICTO_UserICTObjects	**Document ICT Objects**	**Mass Update**
ITSM_StrategyCompliance-Report	**Strategic Migration Alignment Report**	**Edit**
ObjectAttachments	**Attachments**	**Add Document**
ObjectDeputies	**Deputies**	**Create Deputy**
Lifecycle	**Lifecycle**	**Edit Lifecycle**
PRF_CapturePeripherals	**Capture Peripherals**	**Mass Update**
PRF_CapturePeripherals_Ex	**Capture Peripherals**	**Mass Update**
PRF_UserPeripherals	**Document Peripherals**	**Mass Update**
PRJ_InputDemands	**Input Demands**	**Create**
SPL_CaptureStandardPlatforms	**Capture Standard Platforms**	**Mass Update**
SPL_CaptureStandardPlatforms_Ex	**Capture Standard Platforms**	**Mass Update**
SPL_Lifecycle	**Lifecycle**	**Edit Lifecycle**
SVRPD_Lifecycle	**Lifecycle**	**Edit Lifecycle**
VP_CaptureVendorProducts	**Capture Vendor Products**	**Mass Update**

Technical Name of Page View	Caption of Page View	Duplicate Button/Menu Option
VP_CaptureVendorProducts_Ex	**Capture Vendor Products**	**Mass Update**
VP_UserVendorProducts	**Document Vendor Products**	**Mass Update**

To hide functionalities in a selected page view:

1) Open the Alfabet interface by right-clicking the relevant user profile and selecting **Configure User Profile**.
2) Navigate to the page view of any object in the object class for which you want to define the visibility of page view functionalities and in the upper right corner of the workspace, click the **Configure View**  button. The **Customization Editor** opens.
3) Define the following as needed:

- **Select Configuration Object**: Displays the current configuration object that is being configured.
- **Select View Scheme**: Select the view scheme that the configuration is relevant for.
- **Select Hierarchy Level**. Select the hierarchy level for which the configuration is being defined. The levels are listed in the order of the hierarchy, starting with **All Classes**, followed by the object class and then the object class stereotypes.
- **Currently Applied Settings**: Displays the inherited configuration that is applied to the selected configuration object if the option **Inherited Settings from Parent Configuration Object** is selected in the **Choose One Settings Mode** menu. This field will be empty if settings have not been inherited.
- Select the relevant option in the **Choose One Settings Mode** menu in the toolbar of the dataset. The options available will depend on the configuration object that you are working with:
- **Explicit Settings from Selected Configuration Object**: Select to display a dataset that allows visibility to be explicitly specified for the relevant aspects of the selected configuration object displayed in the **Select Configuration Object** field.
- **Inherited Settings from Parent Configuration Object**: Select if the selected configuration object shall inherit the exclusion settings configured for the relevant parent configuration object. The **Currently Applied Settings** field will display the inherited configuration that is applied to the selected configuration object if the option **Inherited Settings from Parent Configuration Object** is selected in the **Choose One Settings Mode** menu.
- **No Explicit Settings from Configuration Object**: Select to display the current settings for the selected configuration object. This dataset allows no explicit settings to be configured. The **No Explicit Settings from Configuration Object** option is displayed instead of the **Inherited Settings from Parent Configuration Object** option if no settings can be inherited.

4) If **Explicit Settings from Selected Configuration Object** has been selected in the **Choose One Settings Mode** menu, you can configure the visibility of the functionalities in the page view or configured report. The **Customization Editor** displays the following columns:

- **Name**: Displays the name of the menu, menu option, or action button.
- **Caption**: Displays the caption defined for the menu or action button.
- **Type**: Displays the type of element.
- **Excluded**: Click in the cell to set an X to specify that the respective menu or action button is hidden in the view.

> If a button includes a drop-down menu providing one or multiple options, you can hide one or more of these functionalities. Please note that if you hide all functionalities available for a button, you must explicitly exclude the parent button in order to ensure that it is hidden from the toolbar.

- **Custom Selector**: Select a custom selector to replace the default selector, if necessary. A drop-down menu will display all custom selectors configured for your solution. You must ensure that you select a custom selector relevant to the context for which it will be used.
- **Default Selector**: Displays the name of the default selector used to find objects. This field will only by filled in if an object selector is implemented.

5) Click the **OK** button to save the changes in the **Customization Editor** or click **Cancel** to exit without saving the changes.

Reviewing the Solution Configuration in the Alfabet Interface

Two functionalities are available in the context menu of a user profile that allow you to view the solution configuration in a fully-functional Alfabet interface. The option **Run Application with Selected User Profile** opens Alfabet as it would be run for the end user and configuration is not possible. The option **Configure User Profile** opens Alfabet in configuration mode so that the solution designer can configure user profiles and refine visibility issues.

To access the Alfabet interface via the **Run Application with Selected User Profile** functionality or the **Configure User Profile** functionality, the solution designer must be assigned as a user to the user profile that is being used to access the Alfabet interface. The access permissions defined for the user profile (ReadOnly or Read/Write) are also applied when accessing Alfabet via the **Run Application with Selected User Profile** option.

Once you have accessed the Alfabet interface, you can further define or modify the visibility of various interface elements for the selected view scheme in the context of the user profile. For more information, see Refining Visibility Issues in the View Scheme.

To review and edit the solution configuration:

1) Go to the **Admin** tab, expand the **User Profiles** node, and right-click the user profile  that you want to validate and select **Run Application with Selected User Profile**. The Alfabet solution opens.
2) Navigate through Alfabet to review the configuration, as needed.

Reviewing the Usage of a User Profile

The **Show Usage** functionality allows you to understand the configuration objects in which a specific user profile is implemented. Additionally, you can review the number of users assigned to a specific user profile.

To access the **Show Usage** dialog, right-click the user profile  and select **Show Usage**. A matrix will show the number of users assigned to the selected user profile. Click **OK** to close the dialog. For more information, see the section Using the Show Usage Functionality in the chapter Getting Started with Alfabet Expand.

Configuring Barrier-Free User Profiles

Software AG provides a number of options to support accessibility to Alfabet functionality and content in conformance with established accessibility guidelines and requirements.

- **JAWS® for Windows® screen reader software**: Alfabet can be used in conjunction with JAWS® for Windows® software (v. 18) in order to support visually impaired users. Please note the following information about working with JAWS:

- The screen reader will read out the login information displayed in the Alfabet user interface about the current user, user profile, and mandates when the start page is in focus if Alfabet is rendered in Google Chrome® and Microsoft Internet Explorer.

- The screen reader will read out the text of the interface element that currently has the focus. This includes, for example, menus and menu options, data-entry fields, static texts, and buttons.

- The screen reader will read out the ARIA label, control type, and help-related content such as page view descriptions or editor and filter field descriptions. Additionally, relevant keyboard shortcuts will also be read out loud by the screen reader. For example, if the keyboard shortcut SHIFT + F7 is used to select a block of text to ease navigation via tabbing, the screen reader will read aloud the keyboard shortcut SHIFT + F7.

- The screen reader will read out if a field is a mandatory field.

- The texts for the interface elements will be read out in the language the user has selected in the Alfabet user interface if the same language has been selected in the JAWS® for Windows® software.

- **Tabbing navigation and keyboard shortcuts**: Users can use tabbing navigation to navigate through the interface. Keyboard shortcuts and key sequences allow the user to move the focus in the Alfabet interface and carry out tasks without using the mouse. Keyboard shortcuts are available in all functionalities in Alfabet and can be used to navigate via the menu bar of the Start page, explorers, object profiles, page views, editors, wizards, filters, and selectors.

For an overview of barrier-free accessibility in Alfabet as well as detailed information about the use of tabbing navigation and keyboard shortcuts in the user interface, see the section *Barrier-Free Accessibility in Alfabet* in the reference manual *Getting Started with Alfabet*.

Please note the following configuration requirements and recommendations to implement barrier-free accessibility:

- The implementation of JAWS® for Windows® software with Alfabet must be explicitly enabled for the relevant user profile. The **Use WAI-ARIA** attribute should be set to `True` for the relevant user profile in order to implement the WAI ARIA (Web Accessibility Initiative – Accessible Rich Internet

Applications) specification for barrier-free accessibility. If the **Use WAI-ARIA** attribute is set to `True` for a user profile, the following changes will be automatically implemented to support barrier-free accessibility:

- Conventional tabular datasets will be replaced by flat datasets with no hierarchical grouping in the table, no legend, and no cell coloring or icons in the cells. If a dataset is empty, the text " **No data provided** " will be read by the screen reader software. The **Export** > **MS File** and **MS PowerPoint** options will be available and the **Export** > **HTML** options removed. Please note that a special set of keyboard combinations are available for this dataset and are described in the section *Using Keyboard Shortcuts in Page Views* in the reference manual *Getting Started with Alfabet*.

  Please note that editable cells with JAWS® for Windows® software will only be read by the JAWS® for Windows® software if Alfabet is rendered in Google Chrome®. This functionality is not available if Alfabet is rendered in Microsoft Internet Explorer.

- The auto-complete functionality that is typically available for edit search fields or combo-boxes in editors and filter fields will be automatically disabled.
- The pop-up calendar that is typically available to select dates will be replaced by a simple data entry field for date fields.
- The slide-in toolbar, secondary windows, and AlfaBot will not be available in the Alfabet user interface.
- Placeholder texts displayed in editor fields and filter fields will be automatically disabled.
- It is recommended that high contrast mode is specified for the browsers used by visually impaired users. Please note that the default high-contrast mode used in Microsoft Windows may display the background as pure black and white without support for inverse colors, thus causing any specified background colors to be not visible. Therefore, it is highly recommended that users render the Alfabet interface with Google Chrome® or Mozilla® Firefox® with their add-on extensions for high-contrast mode.
- It is recommended that access to Alfabet functionalities is available via menus in the main toolbar rather than via guide pages. This will ease navigation through the interface considerably for users requiring barrier-free accessibility. For more information about considering accessibility via the Alfabet menu bar, see the section Defining Access to the Functionalities via the Menu Bar. If you do provide accessibility only through guide pages and guide views, please note the following:
- For all elements created in a guide page project including the search field, top menus, application links, internal links, external links, help links, and document links, the **Caption** attribute should be configured so that a caption can be read for the element and the **Hint** attribute should be configured so that information relevant to navigation can be read. In the case of guide views, the tab sequence should be explicitly defined via the **Tab Order** button in the toolbar of the guide view designer or the **Tab Index** attribute available for each interface control in the guide view.
- The configuration of the welcome panel should allow texts to be read aloud by the screen reader capability provided in conjunction with JAWS® for Windows® software. Please note that the attributes **Welcome Name Text**, **Welcome Name Description**, **Welcome Name Tooltip**, **Logo Image Alt**, and **Welcome Name Picture Alt** are available for this purpose.
- In order to support accessibility to the Alfabet interface for users that are required to use keyboard shortcuts, it is recommended that the visualization of the focus is configured according to the individual needs of the users via the ***AlfaGuiScheme*** definition. This will ease navigation through the interface considerably for users requiring barrier-free accessibility.

The ***AlfaGuiScheme*** definition allows you to specify the font type to display as well as the colors used for various aspects of the interface. Multiple GUI schemes can be created so that one or more ***AlfaGuiScheme*** objects can be created explicitly for the purposes of configuring barrier-free accessibility as well as styling requirements for different entities in a federated enterprise. The GUI scheme must be assigned to the relevant object profile in the **GUI Scheme** attribute available on the object profile. The ***AlfaGuiScheme*** definition will be applied to all aspects of the Alfabet interface including any configured guide pages. For more information about configuring ***AlfaGuiScheme*** objects, see the section Configuring GUI Scheme Definitions for the Alfabet Interface in the chapter Configuring the Visualization of the Alfabet Interface as well as examples of the definition of each attribute in the section *Overview of GUI Scheme Attributes* of the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

Please note that the following attributes either specify the ways that focus is visualized, or the font is colored in the Alfabet interface. Please consider especially which attributes in the **Application** and **Data Rows Highlighting** grid sections should be specified for the user profile. The attributes you need to define will depend on the needs of the users working with the user profile. The following attributes in the **Application** grid section are particularly relevant to support users working with keyboard shortcuts.

- **Focus Background Color**
- **Focus Foreground Color**
- **Focus Outline Color**
- **Focus Outline Style**
- **Focus Outline Width**

- It is recommended that object profiles are available in the user profiles requiring accessibility to Alfabet functionality via keyboard shortcuts rather than via object cockpits. This will ease navigation through the interface considerably for users requiring barrier-free accessibility. For more information, see the chapter Configuring Object Views. Please note that if object cockpits based on a Table Layout Panel interface control are implemented in a barrier-free user profile, the tab sequence for the cells in the table will automatically be from left-to-right and top-to-bottom. The **Tab Index** attribute defined for the interface controls in each table cell will be ignored. This ensures a consistent navigation for user profiles configured for barrier-free accessibility.
- It is recommended that the **Enable Automated Assistant** attribute for the use profile definition is set to `False` for user profiles requiring barrier-free accessibility.
- Keyboard shortcuts are not supported in the context of the **Workflow Activities Explorer** view (`WFS_Explorer`). Therefore, one of the **Workflow Activities** views (`WF_Activities`,`WF_ActivitiesFiltered`, `WF_ActivitiesWithInitiation`,) must be implemented for user profiles requiring accessibility to workflow activities via keyboard shortcuts. For more information about the configuration of views for workflows, see the section Making the Workflow Capability Available to the User Community in the chapter Configuring Workflows.
- Keyboard shortcuts are not supported in the Alfabet Diagram Designer. Therefore, the **Open Diagram** button should be hidden in all views providing access to the Alfabet Diagram Designer for user profiles requiring the use of keyboard shortcuts. For an overview of all views that provide access to the Alfabet Diagram Designer, see the section *Getting Started with the Alfabet Diagram Designer* in the reference manual *Designing IT Landscape Diagrams in Alfabet*. For more information about hiding the **Open Diagram** button, see the section Hiding Functionalities in a Page View or Configured Report.
- Keyboard shortcuts are not supported in the standard **Lifecycle** editor (`ObjectLifecycleEditor`), which requires the dragging of interface elements to specify lifecycle phases. The standard

Lifecycle editor should be replaced with the alternative **Lifecycle** editor (`ObjectLifecycleEditor_TAB`), which displays a tabular dataset and allows the lifecycle phases to be defined by tabbing to rows representing lifecycles phases and defining individual date fields or the duration of the lifecycle phase as expressed in number of years, months, and days. The **Align** menu provides options to define the end dates based on the defined start dates or to define the start and end dates based on the defined durations.

If you implement `ObjectLifecycleEditor_TAB`, you should hide the **Edit Lifecycle** button that opens `ObjectLifecycleEditor` in all **Lifecycle** page views that the user profile will access. Likewise, you should ensure that the **Edit Lifecycle** button that opens `ObjectLifecycleEditor_TAB` should be visible. The following pages are relevant:

- `APP_Lifecycle` to define application lifecycles
- `COM_Lifecycle` to define component lifecycles
- `COMCT_Lifecycle` to define component catalog lifecycles
- `ICTO_Lifecycle` to define ICT object lifecycles
- `SPL_Lifecycle` to define standard platform lifecycles
- `SRVPRD_Lifecycle` to define service product lifecycles
- `ObjectTimeStatusPlan` to define lifecycles for all other object classes supporting lifecycle definition. For an overview of the object classes for which lifecycles can be defined, see the chapter *Overview of Configurable Features for Object Classes* in the *Configuring Alfabet with Alfabet Expand - Appendix*.
- For more information about how to hide/show the **Edit Lifecycle** buttons, see the section Hiding Functionalities in a Page View or Configured Report.

- The screen reader software reads all **Caption** attributes and **Hint** attributes available for the standard interface controls in the Alfabet user interface. If you have configured custom editors, custom selectors, configured reports, custom object profiles, etc., you must ensure that the **Caption** attribute should is specified. Further information about the content expected for the custom interface control as well as other user assistance can be specified via the **Hint** attribute.
- For configured reports, the `PictureAssignment` instruction has been amended with the option to define a text for the icon that can be read as the description for the icon. For more information, see the section Displaying Graphics for Cells Containing Specific Data.

Configuring User Profile Request or Assignment for the User Community

In an enterprise with a significant number of users, the task of fulfilling user requests to assign a new user profile to a user may be difficult to manage. Software AG provides a Self Administration capability that can be configured to allows users to request user profiles or to automatically assign themselves new user profiles. If the capability is implemented, a logged on user will see the option **Assign User Profile** in the < **AlfabetUserName** > menu available in the Alfabet interface. The drop-down menu will list the permissible user profiles that a user may assign to him/herself.

The solution designer should thus clarify the following for the user profiles implemented in the enterprise:

- Shall users be able to request a specified user profile through a managed approval process? If this capability is implemented, users requesting the permissible user profile via the **Assign User Profile** in the **User** menu will trigger a configured workflow that begins and manages a request and approval process.
- Shall users be able to automatically assign a specified user profile to themselves? If this capability is implemented, ALL named user will be able to automatically assign him/herself the permissible user profile via the **Assign User Profile** in the < **AlfabetUserName** > menu available in the Alfabet interface. This situation typically applies to ReadOnly user profiles or, for example, user profiles that allow a set of users to capture demands or requests for technology usage.

You must ensure that the XML attribute `ShowProfileSelfAdminMenu` in the XML object ***PlatformConfiguration*** is set to "`true`" in order to ensure that the **Assign User Profile** option is available in the < **AlfabetUserName** > menu available in the Alfabet interface. If this XML attribute is not set to "`true`", users will have no mechanism to trigger the self-administration or user profile request processes. For more information about configuring the XML object ***PlatformConfiguration***, see the section Configuring the Visibility of Tabs in Alfabet Expand.

The **Enable User Self-Administration** and **Workflow Template** attributes can also be defined for a user profile via the **User Profile Administration** functionality that can be accessed via an administrative user profile. For more information, see the reference manual *User and Solution Administration*.

Configuring a Managed User Profile Request Process

You can configure the user profile so that users can request a permissible user profile via the **Assign User Profile** in the < **AlfabetUserName** > menu available in the Alfabet interface. When a user selects the user profile in the menu, they will trigger a configured workflow that begins a managed request and approval process.

To implement the capability so that users may request a specified user profile through a managed approval process, you must carry out the following steps:

- First, via the **Workflows** tab in Alfabet Expand, configure the workflow template that you want to implement for the user profile request. Because a different workflow template can be configured for each user profile, the workflow can be tailored to the specific user profile initiating the request. The entire set of workflow functionality is available to the solution designer. For more information about the configuration of workflow templates, see the chapter Configuring Workflows.

- Next, select the configured workflow template in the **Workflow Template** attribute for the relevant user profile. The workflow template will be triggered when the user selects the user profile via the **Assign User Profile** in the < **AlfabetUserName** > menu available in the Alfabet interface.
- In the toolbar, click the **Save** button to save the user profile configuration.

Configuring the Automatic Assignment of a User Profile

You can configure the user profile so that users will be able to automatically assign a specified user profile to themselves via the **Assign User Profile** in the < **AlfabetUserName** > menu available in the Alfabet interface. This situation would typically apply to ReadOnly user profiles or user profiles that, for example, allow a set of users to capture demands or requests for technology usage.

Please note that if the **Enable User Self-Administration** attribute is set to `True`, all named users can assign a permissible user profile to themselves via the **Assign User Profile** option in the < **AlfabetUserName** > menu available in the Alfabet interface.

To implement the capability so that users may assign themselves a permissible user profile, you must carry out the following steps:

1) Go to the **Admin** tab, expand the **User Profiles** node, and click the user profile that you want to configure.
2) In the attribute window, set the **Enable User Self-Administration** attribute to `True`.

   Please note that if you are implementing the self-administration capability, any user may assign this user profile to him/herself. This may include users who access Alfabet as anonymous users. Therefore, you may decide that the self-administration function should only be implemented for user profiles of the type `ReadOnly`. It is recommended that you carefully consider which users may be assigning themselves a user profile via the self-administration capability. This is especially critical when configuring the `Admin` user profile, which grants `ReadWrite` access permissions to ALL objects in Alfabet!

3) In the toolbar, click the **Save** button to save the user profile configuration.

Chapter 12: Providing Custom Online Help to the User Community

A standard context-sensitive help is available in Alfabet that provides the user community with information to understand the standard views in the Alfabet user interface that they are currently working with. The standard online help available for your Alfabet product contains documentation for all functionalities in the software, including those that may not be part of the configuration of your Alfabet solution.

Because Alfabet is highly configurable, it may be that you require additional customized help for your solution configuration in order to provide relevant information to your user community.

Various mechanisms are available in Alfabet that allow you to provide information to your user community to help them understand how to perform their tasks using the standard functionalities provided by Alfabet as well as your enterprise's solution configuration:

- A standard context-sensitive help is available in Alfabet that provides the user community with information to understand the standard views in the Alfabet user interface that they are currently working with. The standard online help available for your Alfabet product contains documentation for all functionalities in the software, including those that may not be part of the configuration of your Alfabet solution
- A custom context-sensitive help can be made available as a URL for a standard business function/custom explorer, custom object profile and object cockpit, standard page view, or configured report included in the Alfabet user interface.
- A custom help link may be made available for a user profile, thus allowing information to be available to relevant users regardless of the view that they are currently looking at.
- An automated help assistant can be defined that opens a pop-up displaying information provided by your enterprise to help occasional users with their tasks in Alfabet. The automated help assistant can be made available for a specific user profile, guide view/guide page, standard business function/custom explorer, custom object profile and object cockpit, standard page view, or configured report.
- Custom tooltips can be defined for all custom properties as well as for protected properties.

The content for the custom context-sensitive help as well as the automated help assistant capability must be located in the same domain as the Alfabet Web Application. The installation of the standard context-sensitive help is part of the installation procedure described in the section *Installation* in the reference manual *User and Solution Administration*.

The following information is available:

- Specifying Custom Help for Editors and Wizards
- Specifying Custom Help for Guide Views and Guide Pages
- Providing Custom Tooltips for Custom and Protected Object Class Properties
- Configuring Server Variables for the Custom Help

Understanding the Context-Sensitive Help

A standard context-sensitive help is available in Alfabet that provides the user community with information to understand the standard views in the Alfabet user interface that they are currently working with. The standard context-sensitive help available for your Alfabet product contains documentation for all functionalities in the software, including those that may not be part of the configuration of your Alfabet solution.

In addition to the standard help, you can also provide a custom context-sensitive help for any standard business function/custom explorer, custom object profile and object cockpit, standard page view, or configured report included in the Alfabet user interface. Alfabet provides you with the possibility to decide per view whether the custom context-sensitive help should be displayed and whether the standard context-sensitive help should or should not be displayed. The custom context-sensitive help that you provide for your user community must be available via a URL or server variable(s) that targets a URL that can be viewed in a browser. Users can access the standard context-sensitive help and/or custom context-sensitive help for the current view that they are working with.

Additionally, a custom context-sensitive help link may be made available for a user profile, thus allowing information to be available to relevant users regardless of the view that they are currently looking at.

The standard context-sensitive help available for Alfabet is hierarchically structured and mapped to the technical structure of the interface. The display of the context-sensitive help links in the **Help Selector** mirrors this structure. Standard and custom context-sensitive help links that are specified to be visible will be displayed in the following sequence in the **Help Selector**:

- Solution Help
- Solution Help Search
- Custom Help on User Profile
- Standard Help on Business Function
- Custom Help on Business Function
- Standard Help on Explorer
- Custom Help on Explorer
- Standard Help on Object View
- Custom Help on Object View
- Custom Help on Object Cockpit
- Standard Help on Page View
- Custom Help on Page View/Configured Report

Users will be able to access the standard online help and/or custom online help for the current view that they are working with by clicking the **Help** button in the upper-right corner of the interface. The **Help Selector** will open providing the available help links listed according to the hierarchical structure of the interface. The user can click a link in the **Help Selector** to open the relevant help. The online help will open in a separate browser tab.

The custom context-sensitive online help cannot be defined for more than one culture. The URL defined in the **Custom Context-Sensitive Help URL** attribute will be displayed for the specified view, regardless of the current language of the user interface. The text **Custom Help on...** displayed in the **Help Selector** will be translated as well as the name of the linked view, providing that a translation is available for that view.

The content for the custom context-sensitive help must be located in the same domain as the Alfabet Web Application. The installation of the standard context-sensitive help is part of the installation procedure described in the section *Installation* in the reference manual *User and Solution Administration*.

Understanding the Automated Help Assistant

Training users how to process a specific use case or to carry out their tasks in Alfabet can be challenging. The automated help assistant capability supports the enterprise in training efforts as well as disseminating relevant information to occasional users about the Alfabet functionalities that they use. The automated help assistant capability allows the enterprise to provide a URL targeting an HTML document, video, animation, PowerPoint file, etc. with explanations about methodology, use cases, procedural instructions, and complex processes when working with wizards with many wizard steps, for example. The automated help assistant can be configured for any user profile and displayed on the Home page whether that is a guide view/guide page, splash screen, or storyboard. Automated help assistants can also be configured for a custom wizard, custom object profile and object cockpit, standard business function/custom explorer, standard page view, or configured report. If the standard page view or configured report is embedded in a wizard step, the automated help assistant will also be available for that wizard step.

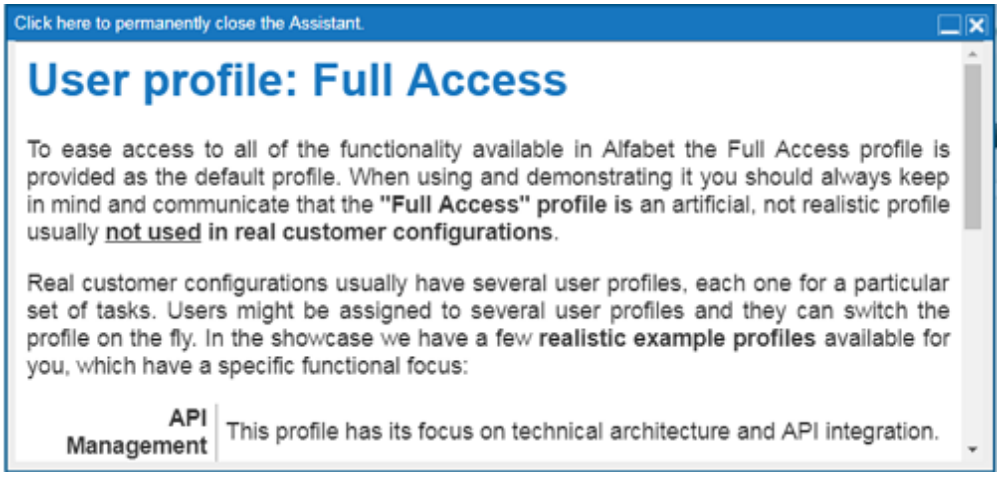


FIGURE: Automated hep assistant for a user profile

When a relevant guide view, wizard, etc. is opened for which an automated help assistant has been configured, a fly-in element will be displayed for a few seconds in the upper-right corner of the user interface below the main menu. Users can click the fly-in element to open the automated help assistant which will open

in a separate window. If the automated help assistant is not opened by the user, it will drop into the slide-in toolbar on the right of the screen in order to provide an unobstructed view of the current view. The automated help assistant for the current view can be opened at any time, as needed, by clicking the colored notch in the slide-in toolbar.

Please note the following:

- The automated help assistant will automatically be minimized if the user clicks in the Alfabet user interface.
- To close the automated help assistant, the user must click the close (**X**) button in the automated help assistant window.
- To reopen the automated help assistant for the current view, click the colored notch in the slide-in toolbar and click the automated help assistant icon to open the assistant.

Please note the following regarding the configuration of the automated help assistant capability:

- The automated help assistant capability must be activated for the user profile(s) that shall have access to the assistants. To enable the automated help assistant capability for a user profile, set the **Enable Automated Assistant** attribute for the user profile to `True`. It is recommended that the **Enable Automated Assistant** attribute is set to `False` for user profiles requiring barrier-free accessibility. Please note that an individual user may disable the automated help assistant capability via the **Enable Automated Help Assistant** attribute in the **User Settings** editor.

  In order to simplify the specification and management of automated assistants, the URLs for automated assistants may be imported or maintained via an Excel file. To export an XLSX file with all standard and custom configuration objects for which an automated help assistant may be specified, click **Help Manager** > **Export Automated Assistants**. An XLSX file will be generated with a column displaying the configuration object type (user profile, business function, editor, etc.), the name of the configuration object, a subordinate configuration object if relevant (for example, an object view may have a subordinate object cockpit), and a column in which the URL may be entered. To import the XLSX file, click **Help Manager** > **Import Automated Assistants** and select the XLSX file with the specified URLs to import.

- To make the automated help assistant available for a configuration object, the URL targeting the content must be specified in the **Automated Assistant URL** attribute available for the configuration object. To do so, go to the relevant user profile, custom wizard, standard business function/custom explorer, custom object profile, object cockpit, standard page view, or configured report assigned to a custom object view and enter the URL in the **Automated Assistant URL** attribute.
- The size of the automated help assistant, its position in the Alfabet user interface as well as the color of the header of the automated help assistant and its corresponding notch in the slide-in toolbar are configured for each configuration object type in the GUI scheme. For more information about defining GUI schemes, see the section Configuring GUI Scheme Definitions for the Alfabet Interface. For details about the individual GUI scheme attributes that are relevant for the automated help assistant, see the section *Overview of GUI Scheme Attributes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.
- If your enterprise provides translations of the content of the automated help assistants, the translated content may be displayed when the user interface is rendered in a secondary language. Please note the following:

- The URL defined for the automated assistant should include a {CURRENT_LANGUAGE} parameter whereby {CURRENT_LANGUAGE} is replaced with the language of the user interface at the time of the

call. For example:
`http://autohelp.alfabet.com/\{CURRENT_LANGUAGE}/showcase/user_profile/Full_Access.html`

- The replacement value for the {CURRENT_LANGUAGE} parameter must be specified in the **Automated Assistant Parameter** attribute available for the culture. This could be, for example, DE or Deutsch or 1031 or DE-de. If no parameter is specified, the four-digit ISO code will be used (for example 1031). For more information about configuring cultures, see the section Specifying the Cultures Relevant to Your Enterprise.
- If the Alfabet Web Application is accessed via HTTPS protocol, then the URL targeting the automated help must also begin with HTTPS. Alternatively, if the Alfabet Web Application is accessed via HTTP protocol, then the URL targeting the automated help must also begin with HTTP.
- The content for the automated help assistant capability must be located in the same domain as the Alfabet Web Application. The installation of the help is part of the installation procedure described in the section *Installation* in the reference manual *User and Solution Administration*.

Assigning Custom Help and Automated Help Assistants to a User Profile

Except for the `Admin` user profile, all user profiles will typically be configured by your enterprise's user administrator and thus will have no standard help available. The `Admin` user profile, which is required to access the `Administration` module, is an exception and does provide a standard context-sensitive help that provides access to the documentation required to understand the functionalities available to the `Admin` user profile.

You can provide a link to custom context-sensitive help for any user profile configured by your solution designer as well as for the `Admin` user profile. You can specify only one custom context-sensitive help per user profile. The context-sensitive help entry will be displayed in the **Help Selector** with the syntax **Custom Help on <User Profile Name>**. The custom context-sensitive help link will be displayed in the **Help Selector** as long as the user is logged in with the user profile. For general information about the standard and custom context-sensitive help, see the section Understanding the Context-Sensitive Help.

Furthermore, you can specify content for the automated help assistant and make it available to the user profile. The automated help assistant will be displayed the first time each user with the user profile accesses Alfabet with the user profile. The automated help assistant can be minimized, closed for the user session, or disabled entirely as needed by each user. For general information about the automated help assistant capability, see the section Understanding the Automated Help Assistant.

Server variables can be used in the **Custom Context-Sensitive Help URL** attribute and the **Automated Assistant URL** attribute to specify the custom help links. Server variables allow you to define all or part of the URL definition in the alias server setting instead of directly defining it in the configuration in Alfabet Expand. Use of server variables is, for example, useful in order to change a Web server name defined in multiple URL definitions in Alfabet Expand. Instead of changing all URL definitions, only the variable value in the alias server setting must be updated. For more information about defining server variables for custom online help, see the section Configuring Server Variables for the Custom Help.

Please note that if the automated help assistant configured for a user profile is open or minimized and the page is reloaded via the F5 key, the user profile will not be reloaded but rather the

associated guide view/guide page will be reloaded. As a result, the automated help assistant content configured for the guide view/guide page will also be reloaded.

To specify custom help for a user profile:

1) Go to the **Admin** tab, expand the **User Profiles** explorer node and click the user profile  that you want to define a custom context-sensitive help link for.
2) Define the following attributes, as needed:

- **Automated Assistant Enabled**: Set to `True` if the automated help assistant capability should be enabled for the user profile. Set to `False` if all assistants available for the user profile shall be disabled. It is recommended that the **Enable Automated Assistant** attribute is set to `False` for users requiring barrier-free accessibility.
- **Automated Assistant URL**: Enter the URL or server variable that targets the content to display in the automated help assistant assigned to the user profile.

  The size of the automated help assistant, its header color and position in the Alfabet user interface, and the notch in the slide-in toolbar are configured for each configuration object type in the GUI scheme. For more information about defining GUI schemes, see the section Configuring GUI Scheme Definitions for the Alfabet Interface. For details about the individual GUI scheme attributes that are relevant for the automated help assistant, see the section *Overview of GUI Scheme Attributes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

- **Custom Context-Sensitive Help URL**: Enter the URL or server variable that targets the custom context-sensitive help for the user profile.
- **Standard Context-Sensitive Help Index**: Displays the HTML file that is available as the standard context-sensitive help for this user profile. A standard context-sensitive help is only available for the `Admin` user profile. The standard online help provides information about the user administration and solution administration functionalities available that can be accessed in the Alfabet user interface via the `Admin` user profile.
- **Standard Context-Sensitive Help Visible**: Select `True` to display both the standard and custom context-sensitive help file in the **Help Selector** Select `False` if the link to the standard context-sensitive help should not be available to the user profile and should not be displayed in the **Help Selector**.

3) In the toolbar, click the **Save**  button to save your changes.

Assigning Custom Help and Automated Help Assistants to Standard Business Functions and Custom Explorers

You can provide a link to custom help for any standard business function provided by Alfabet as well as any custom explorer configured by your solution designer. You can specify only one custom online help per standard business function or per custom explorer. Users will be able to access the online help for the standard business function or custom explorer that is currently displayed in the Alfabet user interface.

The custom help that you provide for your user community must be available via a URL that can be viewed in a browser. For each standard business function, you can decide whether custom online help should be

displayed and whether the standard online help should or should not be displayed. If you specify that both the standard online help and the custom online help are to be displayed, the link for the standard online help will be listed first in the **Help Selector**, followed by the custom help link. The online help entry will be displayed in the **Help Selector** with the syntax **Custom Help on <Business Function Caption>** or **Custom Help on <Custom Explorer Caption>**. For general information about the standard and custom context-sensitive help, see the section Understanding the Context-Sensitive Help.

Furthermore, you can specify content for the automated help assistant and make it available to a selected standard business function or custom explorer. The automated help assistant will be displayed in a fly-in element in the upper-right corner of the user interface when the user accesses the standard business function or custom explorer in the Alfabet user interface. The automated help assistant can be closed, whereby it will drop into the slide-in toolbar and can be opened for the current view at another time. For general information about the automated help assistant capability, see the section Understanding the Automated Help Assistant.

Server variables can be used in the **Custom Context-Sensitive Help URL** attribute and the **Automated Assistant URL** attribute to specify the custom help links. Server variables allow you to define all or part of the URL definition in the alias server setting instead of directly defining it in the configuration in Alfabet Expand. Use of server variables is, for example, useful in order to change a Web server name defined in multiple URL definitions in Alfabet Expand. Instead of changing all URL definitions, only the variable value in the alias server setting must be updated. For more information about defining server variables for custom online help, see the section Configuring Server Variables for the Custom Help.

To specify custom help for a standard business function or custom explorer:

1) Go to the **Functions** tab, expand the **Business Functions** explorer node and navigate either to the standard business function or the custom explorer that you want to define a custom help for.
2) Define the following attributes, as needed:

- **Automated Assistant URL**: Enter the URL or server variable that targets the content to display in the automated help assistant assigned to the business function or custom explorer.

  The size of the automated help assistant, its header color and position in the Alfabet user interface, and the notch in the slide-in toolbar are configured for each configuration object type in the GUI scheme. For more information about defining GUI schemes, see the section Configuring GUI Scheme Definitions for the Alfabet Interface. For details about the individual GUI scheme attributes that are relevant for the automated help assistant, see the section *Overview of GUI Scheme Attributes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

- **Custom Context-Sensitive Help URL**: Enter the URL or server variable that targets the custom context-sensitive help.
- **Standard Context-Sensitive Help Index**: Displays the HTML file that is available as the standard context-sensitive help for a standard business function. This field cannot be edited.
- **Standard Context-Sensitive Help Visible**: Select `True` to display both the standard and custom context-sensitive help file in the **Help Selector** Select `False` if the link to the standard context-sensitive help should not be available and should not be displayed in the **Help Selector**.

3) In the toolbar, click the **Save** button to save your changes.

Assigning Custom Help and Automated Help Assistants to Custom Object Views and Object Cockpits

You can provide custom context-sensitive help for any custom object view or object cockpit configured by your solution designer. You can specify only one custom context-sensitive help link per object view and only one custom context-sensitive help link per object cockpit. Users will be able to access the context-sensitive help for the object profile or object cockpit that is currently displayed in the Alfabet user interface.

The custom context-sensitive help that you provide for your user community must be available via a URL that can be viewed in a browser. For each custom object view, you can decide whether custom context-sensitive help should be displayed and whether the standard context-sensitive help should or should not be displayed. If you specify that both the standard context-sensitive help and the custom context-sensitive help are to be displayed, the link for the standard context-sensitive help will be listed first in the **Help Selector**, followed by the custom context-sensitive help link. The links will be displayed in the **Help Selector** with the syntax **Custom Help on<Object View Caption>** or **Custom Help on<Object Cockpit Caption>**. For general information about the standard and custom context-sensitive help, see the section Understanding the Context-Sensitive Help.

Furthermore, you can specify content for the automated help assistant and make it available to a selected object view and/or object cockpit. The automated help assistant will be displayed in a fly-in element in the upper-right corner of the user interface when the user accesses the object view or object cockpit in the Alfabet user interface. The automated help assistant can be closed, whereby it will drop into the slide-in toolbar and can be opened for the current view at another time. For general information about the automated help assistant capability, see the section Understanding the Automated Help Assistant.

Server variables can be used in the **Custom Context-Sensitive Help URL** attribute and the **Automated Assistant URL** attribute to specify the custom help links. Server variables allow you to define all or part of the URL definition in the alias server setting instead of directly defining it in the configuration in Alfabet Expand. Use of server variables is, for example, useful in order to change a Web server name defined in multiple URL definitions in Alfabet Expand. Instead of changing all URL definitions, only the variable value in the alias server setting must be updated. For more information about defining server variables for custom online help, see the section Configuring Server Variables for the Custom Help.

For more information about defining tooltips for the custom and protected properties that are displayed in the **Attributes** section of an object view, see the section Providing Custom Tooltips for Custom and Protected Object Class Properties.

To specify custom help for a custom object view or object cockpit:

1) Go to the **Presentation** tab, expand the **Object Views** explorer node and navigate either to the standard object vie w  or object cockpit  that you want to define a custom help for.
2) Define the following attributes, as needed.

- **Automated Assistant URL**: Enter the URL or server variable that targets the content to display in the automated help assistant assigned to the object view or object cockpit.

The size of the automated help assistant, its header color and position in the Alfabet user interface, and the notch in the slide-in toolbar are configured for each configuration object type in the GUI scheme. For more information about defining GUI schemes, see the section Configuring GUI Scheme Definitions for the Alfabet Interface. For details about the individual GUI scheme attributes that are relevant for the

automated help assistant, see the section *Overview of GUI Scheme Attributes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

- **Custom Context-Sensitive Help URL**: Enter the URL or server variable that targets the custom context-sensitive help.
- **Standard Context-Sensitive Help Index**: Displays the HTML file that is available as the standard context-sensitive help for the object view.

  The **Standard Context-Sensitive Help Index** attribute can be edited for a custom object view for purposes of backward compatibility. Prior to release 10.1, users specified the location of the external help file using the following syntax: `CUSTOM_HELP:<URL>` (For example: `CUSTOM_HELP:http://helphost/objectview1.html`). The link to the external help file will be displayed in the **Help Selector**. Server variables can be used in the help links. It is recommended that you specify your custom help by means of the **Custom Context-Sensitive Help URL** attribute.

- **Standard Context-Sensitive Help Visible**: Select `True` to display both the standard and custom context-sensitive help file in the **Help Selector** Select `False` if the link to the standard context-sensitive help should not be available and should not be displayed in the **Help Selector**.

3) In the toolbar, click the **Save** button to save your changes.

Assigning Custom Help and Automated Help Assistants to Standard Page Views

You can provide a link to custom help for any standard page view configured by your solution designer. You can specify only one custom online help per standard page view. Users will be able to access the online help for the standard page view that is currently displayed in the Alfabet user interface.

The custom help that you provide for your user community must be available via a URL that can be viewed in a browser. For each standard page view, you can decide whether custom online help should be displayed and whether the standard online help should or should not be displayed. If you specify that both the standard online help and the custom online help are to be displayed, the link for the standard online help will be listed first in the **Help Selector**, followed by the custom help link. The online help entry will be displayed in the **Help Selector** with the syntax **Custom Help on <Page View Caption>**. For general information about the standard and custom context-sensitive help, see the section Understanding the Context-Sensitive Help.

Furthermore, you can specify content for the automated help assistant and make it available to a selected standard page view. The automated help assistant will be displayed in a fly-in element in the upper-right corner of the user interface when the user accesses the standard page view in the Alfabet user interface. The automated help assistant can be closed, whereby it will drop into the slide-in toolbar and can be opened for the current view at another time. If an automated help assistant is specified for a standard page view and that page view is embedded in a wizard step, the automated help assistant will also be available for the wizard step. For general information about the automated help assistant capability, see the section Understanding the Automated Help Assistant.

The custom help that you assign to a page view will be available for every instance of that page view, regardless of the object view context that you define. For example, a custom help link

assigned to the **Assignments** page view in the context of a custom object view configured for the class `Application` will also be available in the **Assignments** page view in the context the object view for the class `Component`.

Server variables can be used in the **Custom Context-Sensitive Help URL** attribute and the **Automated Assistant URL** attribute to specify the custom help links. Server variables allow you to define all or part of the URL definition in the alias server setting instead of directly defining it in the configuration in Alfabet Expand. Use of server variables is, for example, useful in order to change a Web server name defined in multiple URL definitions in Alfabet Expand. Instead of changing all URL definitions, only the variable value in the alias server setting must be updated. For more information about defining server variables for custom online help, see the section Configuring Server Variables for the Custom Help.

To specify custom help for a standard page view:

1) Go to the **Presentation** tab, expand the **Object Views** explorer node, and navigate to the page view that you want to define a custom help for.
2) Define the following attributes, as needed:

- **Automated Assistant URL**: Enter the URL or server variable that targets the content to display in the automated help assistant assigned to the standard page view.

  The size of the automated help assistant, its header color and position in the Alfabet user interface, and the notch in the slide-in toolbar are configured for each configuration object type in the GUI scheme. For more information about defining GUI schemes, see the section Configuring GUI Scheme Definitions for the Alfabet Interface. For details about the individual GUI scheme attributes that are relevant for the automated help assistant, see the section *Overview of GUI Scheme Attributes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

- **Custom Context-Sensitive Help URL**: Enter the URL or server variable that targets the custom context-sensitive help.
- **Standard Context-Sensitive Help Index**: Displays the HTML file that is available as the standard context-sensitive help for the page view. This field cannot be edited.
- **Standard Context-Sensitive Help Visible**: Select `True` to display both the standard and custom context-sensitive help file in the **Help Selector** Select `False` if the link to the standard context-sensitive help should not be available and should not be displayed in the **Help Selector**.

3) In the toolbar, click the **Save** button to save your changes.

Assigning Custom Help and Automated Help Assistants to Configured Reports

You can provide a link to custom help for any configured report created by your solution designer. You can specify only one custom online help per configured report. Users will be able to access the online help for the configured report that is currently displayed in the Alfabet user interface.

The custom help that you provide for your user community must be available via a URL that can be viewed in a browser. For each configured report, you can decide whether custom online help should be displayed and whether the standard online help should or should not be displayed. If you specify that both the

standard online help and the custom online help are to be displayed, the link for the standard online help will be listed first in the **Help Selector**, followed by the custom help link. The online help entry will be displayed in the **Help Selector** with the syntax **Custom Help on<Configured Report Caption>**. For general information about the standard and custom context-sensitive help, see the section Understanding the Context-Sensitive Help.

Furthermore, you can specify content for the automated help assistant and make it available to a selected configured report. The automated help assistant will be displayed in a fly-in element in the upper-right corner of the user interface when the user accesses the configured report in the Alfabet user interface. The automated help assistant can be closed, whereby it will drop into the slide-in toolbar and can be opened for the current view at another time. For general information about the automated help assistant capability, see the section Understanding the Automated Help Assistant.

The custom help that you assign to a configured report will be available for every instance of that configured report. For example, a custom help link assigned to the configured report **Data Quality Checklist** in the context of a custom object view configured for the class `Application` will also be available in the configured report **Data Quality Checklist** in the context the object view for the class `Component`.

Server variables can be used in the **Custom Context-Sensitive Help URL** attribute and the **Automated Assistant URL** attribute to specify the custom help links. Server variables allow you to define all or part of the URL definition in the alias server setting instead of directly defining it in the configuration in Alfabet Expand. Use of server variables is, for example, useful in order to change a Web server name defined in multiple URL definitions in Alfabet Expand. Instead of changing all URL definitions, only the variable value in the alias server setting must be updated. For more information about defining server variables for custom online help, see the section Configuring Server Variables for the Custom Help.

To specify custom help for a configured report:

1) Go to the **Reports** tab and navigate to the configured report that you want to define a custom help for.
2) Ensure that the configured report has **Report State** set to **Plan**. To change the report state, right-click the configured report and select **Set Report State to 'Plan'**.
3) Click the report view node below the configured report to open the attribute grid. Define the following attributes, as needed:

- **Automated Assistant URL**: Enter the URL or server variable that targets the content to display in the automated help assistant assigned to the configured report.

  The size of the automated help assistant, its header color and position in the Alfabet user interface, and the notch in the slide-in toolbar are configured for each configuration object type in the GUI scheme. For more information about defining GUI schemes, see the section Configuring GUI Scheme Definitions for the Alfabet Interface. For details about the individual GUI scheme attributes that are relevant for the automated help assistant, see the section *Overview of GUI Scheme Attributes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

- **Custom Context-Sensitive Help URL**: Enter the URL or server variable that targets the custom context-sensitive help.
- **Standard Context-Sensitive Help Index**: This field will be empty for a configured report.

The **Standard Context-Sensitive Help Index** attribute on the configured report node (not the configured report view node) can be edited for purposes of backward compatibility. Prior to release 10.1, users specified the location of the external help file using the following syntax: `CUSTOM_HELP:<URL>` (For example: `CUSTOM_HELP:http://helphost/report1.html`). The link to the external help file will be displayed in the **Help Selector**. Server variables can be used in the help links. It is recommended that you specify your custom help by means of the **Custom Context-Sensitive Help URL** attribute on the report view node.

- **Standard Context-Sensitive Help Visible**: Select `True` to display both the standard and custom context-sensitive help file in the **Help Selector** Select `False` if the link to the standard context-sensitive help should not be available and should not be displayed in the **Help Selector**.

4) In the toolbar, click the **Save** button to save your changes.

Specifying Custom Help for Editors and Wizards

You can specify content for the automated help assistant and make it available to a selected custom editor or custom wizard. The automated help assistant will be displayed in a fly-in element in the upper-right corner of the user interface when the user accesses the custom editor or custom wizard in the Alfabet user interface. The automated help assistant can be closed, whereby it will drop into the slide-in toolbar and can be opened for the current view at another time. For general information about the automated help assistant capability, see the section Understanding the Automated Help Assistant.

To define a custom tooltip in a custom editor, you must explicitly define the **Hint** attribute for the interface control associated with the custom object class property in the custom editor. For more information about the specification of tooltips in custom editors, see the section Adding Help Text to Interface Controls in the Custom Editor in the chapter Configuring Custom Editors.

If an automated help assistant is specified for a standard page view and that page view is embedded in a wizard step, the automated help assistant will be displayed for the wizard step. For more information about defining an automated help assistant for a standard page view, see the section Assigning Custom Help and Automated Help Assistants to Standard Page Views.

1) Go to the **Presentation** tab, expand the **Custom Editors** explorer node and navigate to the custom editor that you want to define the automated help assistant capability for. In the case of a custom wizard, expand the **Wizards** explorer node and navigate to the custom wizard that you want to define the automated help assistant capability for.
2) In the **Automated Assistant URL** attribute, enter the URL or server variable that targets the content to display in the automated help assistant assigned to the custom editor.

   The size of the automated help assistant, its header color and position in the Alfabet user interface, and the notch in the slide-in toolbar are configured for each configuration object type in the GUI scheme. For more information about defining GUI schemes, see the section Configuring GUI Scheme Definitions for the Alfabet Interface. For details about the individual GUI scheme attributes that are relevant for the automated help assistant, see the section *Overview of GUI Scheme Attributes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

3) In the toolbar, click the **Save** button to save your changes.

Specifying Custom Help for Guide Views and Guide Pages

You can specify content for the automated help assistant and make it available to a selected guide view or guide page. The automated help assistant will be displayed in a fly-in element in the upper-right corner of the user interface when the user accesses the guide view or guide page in the Alfabet user interface. The automated help assistant can be closed, whereby it will drop into the slide-in toolbar and can be opened for the current view at another time. For general information about the automated help assistant capability, see the section Understanding the Automated Help Assistant.

1) In the toolbar, click **Managers** > **Guide Page Designer**. The Guide Pages Designer opens in a browser window.
2) Select the relevant guide page project in the **Select Guide Page Project** field.
3) Click the relevant guide view or guide page node that you want to define the automated help assistant capability for.
4) In the **Automated Assistant URL** attribute, enter the URL or server variable that targets the content to display in the automated help assistant assigned to the custom editor.

   The size of the automated help assistant, its header color and position in the Alfabet user interface, and the notch in the slide-in toolbar are configured for each configuration object type in the GUI scheme. For more information about defining GUI schemes, see the section Configuring GUI Scheme Definitions for the Alfabet Interface. For details about the individual GUI scheme attributes that are relevant for the automated help assistant, see the section *Overview of GUI Scheme Attributes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

5) In the toolbar, click the **Save** button to save your changes.

Providing Custom Tooltips for Custom and Protected Object Class Properties

Custom tooltips can be defined for all custom properties as well as for protected properties. Users can view the tooltip for the custom object class property in the **Attributes** section of the respective object view. This is defined via the **Hint** attribute defined for the custom/protected object class property. Please note that the **Hint** attribute defined for the custom object class property will not be displayed in a custom editor. For more information about defining the **Hint** attribute for the custom object class property, see the section Configuring Custom Properties for Protected or Public Object Classes in the chapter Configuring the Class Model.

To define a custom tooltip in a custom editor, you must explicitly define the **Hint** attribute for the interface control associated with the custom object class property in the custom editor. For more information about the specification of custom editors, see the section Adding Help Text to Interface Controls in the Custom Editor in the chapter Configuring Custom Editors.

Configuring Server Variables for the Custom Help

The definition of server variables allows information about the targeted links to be defined in the server alias configuration. Defining the information about the URL in the server alias configuration instead of directly defining it in the configured report definitions eases the propagation of changes. You can user server variables in the definition of **Custom Context-Sensitive Help URL** attribute as well as the **Standard Context-Sensitive Help Index** attribute available for custom object views and configured reports.

A variable is referenced as `$<server variable name>`. For example, a server variable called `MYHELPDIRECTORY` is referenced as `$MYHELPDIRECTORY`. In the definition of the **Custom Context-Sensitive Help URL** attribute, the variable definition can either substitute the whole connection string or only part of the connection string. It is also possible to build the URL from a number of concatenated server variables or server variables appended with a URL. For example: `$MYHELPDIRECTORY/CustomHelpPage.html`

Server variables are defined in the Alfabet Administrator:

1) In the Alfabet Administrator, click the **Alfabet Aliases** node in the **Administrator** explorer.
2) In the table on the right, select the server alias that you want to define a server variable for and click the **Edit**  button. The alias editor opens.
3) Go to the **Variables** tab and click the **New** button. A dialog box opens.
4) In the **Variable Name** field, enter a unique name for the server variable.

   The server variable name may only contain letters (English alphabet), numbers, and the underscore symbol.

5) In the **Variable Value** field, enter all or part of the URL that should be the target of the Web link.

   If the Web link contains a special character according to XML standards (for example: &, %, ;,<,>), these characters must be replaced by their XML conformant variant (for example: `&` for &)

6) Click **OK** to save your changes. The server variable definition appears in the list of server variables.

   To edit or delete the server variable, select the server variable in the table and click the **Edit** or **Delete** button below the table.

7) Click **OK** to save your changes and close the editor. The server variable definition is now available in the server alias configuration and can be used for the URL definition specified in Alfabet Expand.

Chapter 13: Configuring Workflows

Software AG supports your enterprise in defining and maintaining workflows in which you can track and coordinate the activities that should be performed by various persons in a particular sequence. For a general overview of the workflow capability, see the section *Executing Workflows and Participating in Workflow Steps* in the reference manual *Getting Started with Alfabet*.

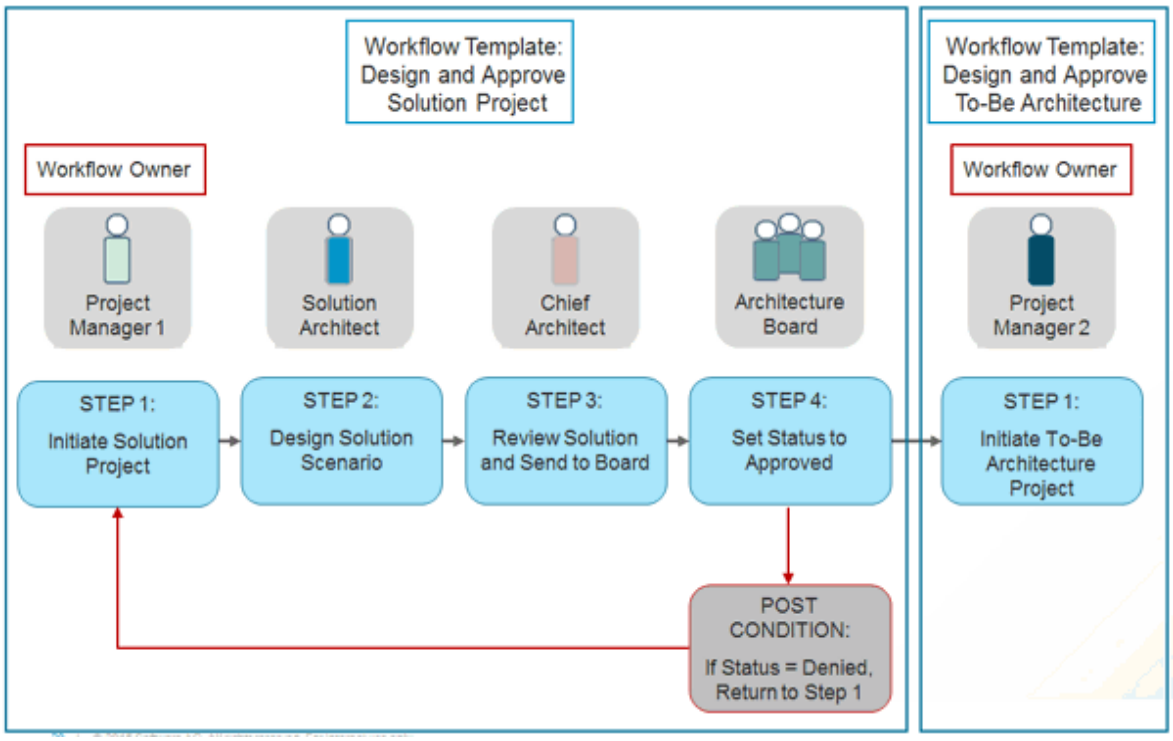


FIGURE: Example of an Alfabet workflow for project proposal and design

A workflow is a collaborative process made up of workflow steps that are typically carried out by one or more users. A workflow is based on a configured workflow template that determines the sequence of workflow steps that are to be performed on a specific object and its references by specified user(s). Workflow steps may have specific pre- and post-conditions that determine different paths to take in the workflow depending on whether the conditions are or are not met.

Typically, the workflow owner is the user who initiates and is responsible for maintaining the workflow. When a workflow is initiated by the workflow owner and when a workflow advances to the next workflow step, relevant users may be informed via automatically-generated emails of their impending responsibility for the workflow step. This functionality ensures that all relevant users are informed and reminded of their responsibilities in the collaborative workflow. The option to refuse, delegate, and pause workflow steps, remind users of an impending target date for a workflow step as well as redirect a workflow that has encountered an error enables workflow owners and workflow administrators to keep track, coordinate, and manage the completion of each workflow step and the workflow as a whole.

To implement workflows in your enterprise, you must first create a workflow template that serves as the basis for the workflow.

For an overview of the object classes for which workflows can be configured, see *Overview of Configurable Features for Object Classes*.

A workflow template is a customer-defined blueprint for one or more workflows. The template specifies what object class is the point-of-departure in the workflow, which user groups and/or user profiles may initiate and administrate the workflow, which workflow steps comprise the workflow as well as their sequence, any possible per- and post-conditions or update actions associated with a workflow step, and what kinds of workflow notifications should be sent to collaborating users in which contexts. The user who creates the workflow template is the workflow template owner.

A workflow template must have the attribute **Workflow State** attribute set to `Plan` to be configured and validated. Once the workflow template is completed and approved, the attribute **Workflow State** must be switched to `Active` in order to make it available to the user community. Once the workflow template is available in Alfabet, a permitted user may initiate a workflow based on the workflow template. Multiple workflows may be simultaneously initiated and running for a workflow template.

Once the workflow template is created, you must create the workflow steps that allow the tasks to be carried out in the workflow.

A workflow step is an activity or task in a workflow that must be performed. The workflow template may have as many workflow steps as needed to complete the relevant task. Standard and custom editors, wizards, page views, configured reports, and object profiles may be implemented in the workflow step so that users can complete the task. A workflow step may entail entering, modifying or reviewing data, or a workflow step may be configured to be automatically executed by the Alfabet system. If the workflow step is not automatically executed by the system, one or more users may be defined as responsible to perform the workflow step. A user may delegate a workflow step to another user or refuse a workflow step. Depending on the configuration of the workflow step, a user may be required to confirm a workflow step before the workflow can advance to the next workflow steps, or the confirmation of the workflow step may occur automatically. A workflow step may also have one or more associated workflow step actions which allow various operations to be performed when the workflow step is entered, exited, refused, or expired. Additionally, each workflow step may have one or more pre-conditions or post-conditions that must be fulfilled in order to enter or exit the workflow step.

If the name of a custom editor, class setting, or user profile is changed and that configuration object is implemented in a workflow template, the changed name will be correctly updated in the workflow templates where that configuration object is implemented. A warning message will be generated during the migration process if a configured report, wizard or text template that is used in a workflow template has been renamed. **The modified names of user profiles, configured reports, wizards, or text templates that are referenced in workflow templates must be explicitly retrofitted for running workflow instances by configuring an appropriate workflow migration definition.**

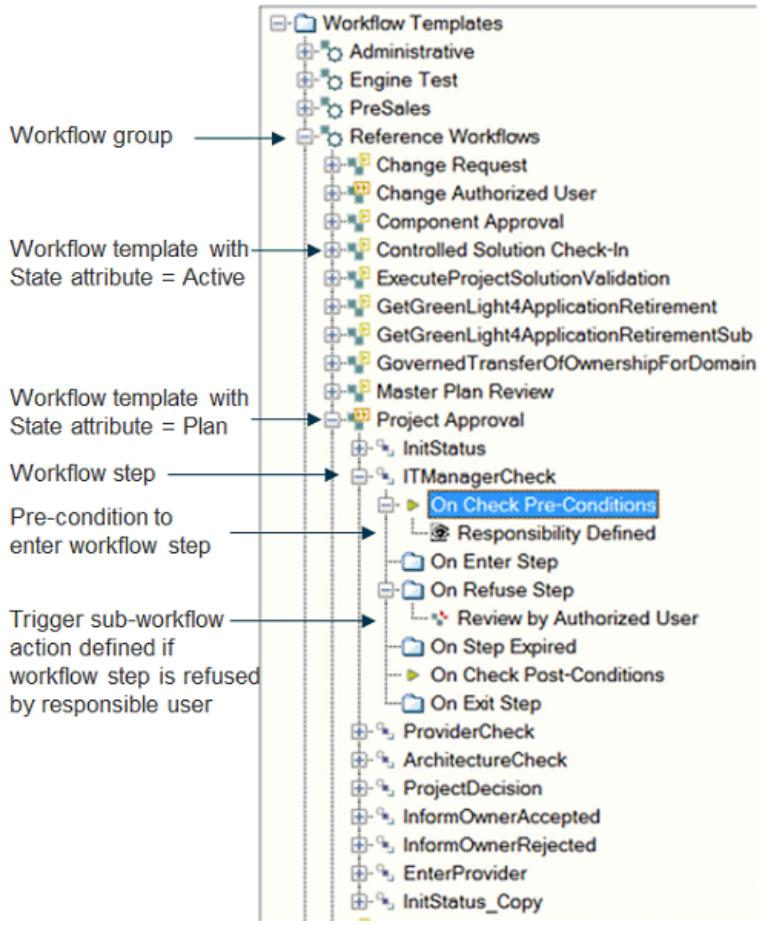


FIGURE: Workflow Templates explorer in the Workflows tab

Workflows are configured in the **Workflow Templates** explorer in the **Workflows** tab of Alfabet Expand. The **Workflow Templates** explorer displays workflow groups, their workflow templates which are listed alphabetically, the workflow steps defined for a workflow template as well as the workflow step actions that allow emails to be automatically sent, object class properties to be automatically updated, sub-workflows to be triggered, pre-conditions /post-conditions to be executed, etc when a workflow step is entered, exited, refused, or expired. Only workflow templates with the **Workflow State** attribute set to `Plan` can be configured and edited.

New workflow objects can be created in the explorer by right-clicking the relevant node and selected an option in the context menu. Click a node in the explorer to open the attribute window in the right pane of the explorer to further specify the nodes attributes.

Workflows can be started via a RESTful service call to the new workflow endpoint of the Alfabet RESTful services. The **Applicable for REST API** attribute is available for workflow templates. The attribute must be set to `True` to start a workflow based on the workflow template via the new endpoint workflow. Only workflow templates with the attribute **Automatic Start** set to `True` can be executed via RESTful service calls. Furthermore, a workflow step action can be configured to trigger an event when a workflow step is entered, refused, expired, or exited. For more information about the configuration required to implement the event capability, see the chapter Configuring Events.

In addition to the workflow capability, Software AG also provides a wizard capability that structures the data capture process for a user. For more information about configuring a wizard for structured data capture, see the chapter Configuring Wizards.

The following information is available:

- Overview of Individuals Involved in the Workflow Capability
- Best-Practice Procedure for Configuring Workflow Templates in the Test Environment
- Conceptualizing the Workflow
- Creating a Workflow Template
- Defining the Start of Workflows Based on the Workflow Template
- Configuring the Manual Start of Workflows
- Configuring the Automatic Start of Workflows
- Specifying the Objects Targeted by the Workflows
- Configuring Manually Started Workflows to Start with Existing Objects
- Configuring Automatically Started Workflows to Start with Existing Objects
- Configuring Manually-Started Workflows to Start with New Objects
- Configuring Workflows to Create New Target Objects via the Source Base Class Attribute
- Changing to a Different Base Object for a Workflow Step
- Creating a Workflow Step
- Defining the View Implemented for a Workflow Step
- Specifying a Wizard for the Workflow Step
- Specifying an Editor for the Workflow Step
- Specifying a Standard Alfabet Page View for the Workflow Step
- Specifying a Configured Report for the Workflow Step
- Specifying an Object View for the Workflow Step
- Specifying a Dynamically-Generated View for the Workflow Step
- Specifying a System-Generated Action for the Workflow Step
- Defining the Users Responsible for a Workflow Step
- Defining a Deadline and Reminders for a Workflow Step
- Configuring the Action Buttons Available for a Workflow Step
- Specifying the Confirmation of a Workflow Step of the Type Wizard or Editor
- Specifying the Explicit Confirmation of a Workflow Step of the Types Navigate, GraphicView, and Report
- Specifying the Users Required to Confirm a Workflow Step

- Making the Workflow Template Available to the AlfaBot Capability
- Making the Workflow Capability Available to the User Community
- Structuring Workflow Templates in Folders

Overview of Individuals Involved in the Workflow Capability

The following individuals are typically involved in the configuration, execution, and management of a workflow:

- **Workflow Designer**: A workflow designer configures the workflow template in Alfabet Expand. It is his/her responsibility to set the workflow template's **Workflow State** attribute to `Active` once the workflow template is ready for implementation in the production environment. The workflow designer is the initial owner of the workflow template. The workflow template owner can be changed in the **Workflow Administration** functionality via the **Change Owner** button. For more information about changing the workflow template owner, see chapter *Tracking and Managing Workflows* in the reference manual *User and Solution Administration*.
- **Workflow Owner**: A workflow owner initiates a workflow in Alfabet based on the workflow template. This can be carried out manually in the **My Workflow Activities** functionalities (`WF_UserWorkflowActivities`, `WF_UserWorkflowActivitiesExt`, and `WF_UserWorkflowActivitiesCommon`) and/or the **Workflow Activities Explorer** (`WFS_Explorer`), in the **Workflow Administration** functionality, or via a batch process. The workflow owner also tracks the progress of any workflow that he/she is the owner of in the **Workflows** section of the **My Workflows** functionality. For more information, see the section *Executing Workflows and Participating in Workflow Steps* in the reference manual *Getting Started with Alfabet*. Further, the workflow owner may interrupt and reactivate a workflow that he/she owns. Additionally, he/she can delegate a workflow step to a different user, withdraw a current workflow step thus removing it from the **My Workflow Activities** functionalities/ **Workflow Activities Explorer** of all responsible users, or, if an error has occurred, redirect the workflow to a different workflow step. These tasks can be carried out in the object profile of the relevant workflow.

All users who may potentially be identified as someone who should initiate a workflow should have a user profile that provides access to the **My Workflows** or **Initiate Workflow** functionalities. Further, the workflow administrator may change the owner of a workflow or workflow template to a user that is not necessarily associated with the permitted user profiles or permitted user groups defined for the workflow template. These users must also be able to access Alfabet with a user profile that provides access to the **My Workflows** or **Initiate Workflow** functionalities. For more information about how access is made available to the workflow capability, see the section Making Functionalities Accessible to a User Profile in the chapter Configuring User Profiles for the User Community.

The owner of a workflow is determined as follows:

- If a workflow is started manually in the **My Workflows** functionality, the current user starting the workflow is the workflow owner.
- If a workflow is started automatically via the batch tool `AlfaWorkflowCommandPrompt.exe`, then the workflow owner is based on the owner of the workflow template. If no workflow template owner is defined, then the current user executing the batch job is the workflow owner.

- If a workflow is started in the **Workflow Administration** functionality via the **Start Automatically** button, the workflow owner is based on the owner of the workflow template. If no workflow template owner is defined, then the current user triggering the workflow in the **Workflow Administration** functionality is the workflow owner.

- **Workflow Step Responsibles**: One or more users that have been identified as responsible for a workflow step execute the task required for the workflow step and confirm the completion of the workflow step in the **My Workflow Activities** functionalities. Responsible users may refuse or delegate a workflow step, if the configuration of their workflow step includes the **Refuse** and/or **Delegate** button. All users who may potentially be identified as responsible users for a workflow step should have a user profile that provides access to the relevant **My Workflow Activities** functionalities (`WF_UserWorkflowActivities`, `WF_UserWorkflowActivitiesExt`, and `WF_UserWorkflowActivitiesCommon`) and/or the **Workflow Activities Explorer** (`WFS_Explorer`). For more information about how access is made available to the workflow capability, see the section Making Functionalities Accessible to a User Profile in the chapter Configuring User Profiles for the User Community.

- **Workflow Administrator**: A workflow administrator maintains all workflow templates configured for the enterprise and tracks the progress of all workflows. The workflow administrator can change the workflow state of any workflow template, assign a new workflow owner to any workflow or workflow template, check workflow deadlines, delete old workflows, and resolve any errors that have occurred in a workflow by, for example, redirecting a workflow to a different workflow step, closing out unconfirmed workflow steps, or releasing a lock on a workflow. These tasks are carried out in the **Workflow Administration** functionality. For more information, see the section *Tracking and Managing Workflows* in the reference manual *User and Solution Administration*.

- **System Administrator**: A system administrator regularly initiates a batch process to send the email notifications generated in a workflow and initiate any automated processes that have been configured for the workflow (for example, the automatic start of workflows and automatic delete of completed workflows). This task is carried out with the `AlfaWorkflowCommandPrompt.exe`. For more information, see the section *Batch Processes for Workflows with AlfaWorkflowCommandPrompt.exe* in the reference manual *System Administration*.

Best-Practice Procedure for Configuring Workflow Templates in the Test Environment

For security reasons, workflow templates cannot be altered during runtime of the Alfabet Server. Instead, workflow templates should be created, configured and tested in a test environment.

You should ensure that the Alfabet Server is configured to send all emails to an explicit test email account instead of using the email account defined for the user that is the recipient of the email. This allows testing to be performed without changing the email account configuration of the users. For more information, see the section *Re-Routing Emails to a Defined Address for Testing* in the reference manual. *System Administration*

The general configuration of workflow templates and workflow steps is described in detail in other sections of the workflow documentation. This section provides general information about the import of the configuration to the production environment. After completion of the workflow template configuration in the test

environment, you can use one of the following methods to import the report configuration to the production environment:

- The configuration of workflow templates can be saved to an AMM updater file either separately or in combination with other configurations specified in Alfabet Expand. The AMM updater file can then be used to replace or merge the configuration in the updater file with the configuration of a target database. Workflow migration can optionally be performed in the target database as part of the AMM configuration import mechanism. This method requires that you are using Alfabet Expand in stand-alone mode. It is part of the overall database update methods and described in the chapter Applying Configuration Changes to Other Databases.
- The configuration of all workflow templates or of a single workflow template can be saved to an XML file. This file can be merged with the workflow template configuration in a target database or replace the workflow template configuration in the target database. The method is available in stand-alone and remote mode. The procedure is described below.

  1) In the test environment, open Alfabet Expand with a server alias or remote alias.
  2) In the **Workflow** tab, right-click the folder **Workflow Templates** at the top of the explorer hierarchy and select **Save As**.
  3) In the explorer window that opens, select a location for the XML file and click **Save** to save the file.
  4) In the production environment, shut down the Alfabet Server.

     For information about the shutdown of Alfabet components, see the section *Planned Outages of the Alfabet Components* in the reference manual *System Administration*.

  5) In the production environment, open Alfabet Expand.
  6) In the **Workflow** tab, right-click the folder **Workflow Templates** and select either:

- **Replace from File** to overwrite the current configuration with the configuration in the XML file, or
- **Merge from File** to merge the current configuration with the configuration in the XML file.

  Please note the following:

  - **Replace from File** functionality: All objects in the configuration will overwrite corresponding objects in the database. Database objects that have no corresponding object in the configuration file will be deleted.
  - **Merge from File** functionality: All objects in the configuration will overwrite corresponding objects in the database. Database objects that have no corresponding object in the configuration file will remain unchanged. Objects that are only available in the configuration file will be added.

  7) In the explorer window that opens, select the XML file with your new workflow configuration.
  8) Confirm the warning message. The configuration in the production environment is changed.

After replacing or merging a configuration with the configuration in an XML file or AMM file and after upgrading to another Alfabet release, it is recommended to test whether changes in the meta-model require queries to be adapted to any changed workflow configurations. A test mechanism is provided that controls whether Alfabet queries in the customer configuration match the current meta-model. The test procedure is described in the section Testing Queries for Compliance with the Current Release in the chapter Defining Queries.

Conceptualizing the Workflow

For an overview of the object classes for which workflows can be configured, see *Overview of Configurable Features for Object Classes*.

The workflow designer should consider the following issues when defining a workflow template:

- Which object class will the workflow target? Will only one object be targeted throughout the workflow or will another object also be processed at some point during the workflow? For more information, see Specifying the Objects Targeted by the Workflows.
- Will the user need to define data for objects in other object classes as well? If object classes other than the base class are required, a query must be defined to determine the base object for each workflow step that deviates from the base object class. For more information, see Changing to a Different Base Object for a Workflow Step.
- Should an existing object be processed by the workflow or should a new object be created by initiating the workflow? For more information, see Defining the Start of Workflows Based on the Workflow Template.
- Should workflows be automatically generated via batch process for the workflow template or should users manually initiate the workflows? Who may initiate workflows for a specific workflow template?
- What tasks are to be completed for each workflow step in the workflow? Which view is required by the user to complete the task? A workflow step may consist of any of the following:

- a wizard
- a standard editor
- a custom editor
- a standard Alfabet page view
- an object view
- a tabular query-based configured report
- a query-based custom report displaying graphics (such as a tree-map or layered diagram report)
- triggered sub-workflows
- a system-generated action

For more information, see Defining the View Implemented for a Workflow Step.

- Which users should be responsible for each workflow step? Must all responsible users complete the workflow step or can the workflow step be completed if only one responsible user completes the workflow step? For more information, see Defining the Users Responsible for a Workflow Step.
- Should some workflow functionalities (for example, delegate, refuse, etc.) be disabled for a particular workflow step? If this is the case, you can customize the workflow capability to show only toolbar buttons relevant for the current workflow step. For more information, see Configuring the Action Buttons Available for a Workflow Step.

- Are pre-conditions necessary for a workflow step? By defining an Alfabet query or SQL query as a pre-condition, you can stipulate the criteria that must be fulfilled in order to enter a workflow step. For more information, see Configuring Pre- and Post-Conditions for a Workflow Step.
- Are post-conditions necessary to ensure that the data is complete before a workflow step can be exited?
- Should object class properties be automatically updated in the context of a workflow step? For example, should the object class property `Status` be automatically changed to another value once a workflow step is finished? For more information, see Configuring Automatic Property Updates for a Workflow Step.
- Should email notifications be automatically sent to relevant users when a workflow step is entered, refused, delegated, or exited? For more information, see Configuring Email Notifications for Workflows.
- Should sub-workflows be triggered as the consequence of specifying a specific object class property value or entering, completing, refusing, or delegating a workflow step? What should happen in the base workflow if triggered sub-workflows are cancelled? For more information, see Configuring a Workflow Step to Trigger a Subordinate Workflow.
- Should the workflow step have a deadline? If so, should email notifications be sent reminding users of the deadline? Should the workflow owner be informed if the workflow step surpasses its deadline? For more information, see Defining a Deadline and Reminders for a Workflow Step.
- Which workflow functionalities will users require for the workflow?
  - Should users work in one of the **My Workflow Activities** functionalities (`WF_UserWorkflowActivities`, `WF_UserWorkflowActivitiesExt`, and `WF_UserWorkflowActivitiesCommon`)? These are Alfabet's standard tabular views that allow only a minimal amount of customization:
  - Should users process their workflow steps in the **Workflow Activities Explorer** functionality (`WFS_Explorer`), which is a customized view based on a configured HTML template? The **Workflow Activities Explorer** functionality features a design and layout of data that is similar to established email management systems so that users can easily and efficiently accomplish their workflow tasks. Because the view is highly configurable, users can be provided with precisely the data they need in order to process the task-at-hand with a minimum number of clicks and navigation to other views. For more information, see Specifying the Customized Workflow Activities Explorer (WFS_Explorer) or a Custom Explorer.
  - Should any object class properties be hidden in the view implemented for a wizard step?

For more information, see Making the Workflow Capability Available to the User Community.

Creating a Workflow Template

Workflow templates cannot be created and edited during run-time of the Alfabet Server. Therefore, any modifications to workflow templates or testing of workflow templates must occur in a test environment. Once configuration and testing of the workflow template has been completed, you can export the workflow template as part of the Alfabet configuration to an XML file. This XML file can later be imported to the production environment in order to make the changed or new workflow template available to the users.

The mechanism to save the workflow configuration as an XML file as well as to read or merge the configuration are available in both Alfabet Expand and the Alfabet Administrator. For more information, see the section Best-Practice Procedure for Configuring Workflow Templates in the Test Environment.

A workflow template is a customer-defined blueprint for one or more workflows. The template specifies what object class is the point-of-departure in the workflow, which user groups and/or user profiles may initiate and administrate the workflow, which workflow steps comprise the workflow as well as their sequence, any possible per- and post-conditions or update actions associated with a workflow step, and what kinds of workflow notifications should be sent to collaborating users in which contexts. The user who creates the workflow template is the workflow template owner.

A workflow template must have the attribute **Workflow State** attribute set to `Plan` to be configured and validated. Once the workflow template is completed and approved, the attribute **Workflow State** must be switched to `Active` in order to make it available to the user community. Once the workflow template is available in Alfabet, a permitted user may initiate a workflow based on the workflow template. Multiple workflows may be simultaneously initiated and running for a workflow template.

The initial creation of a workflow template is done in two steps:

- First you must create the workflow template and define general attributes including the name.
- Second you must create the initial workflow step. Once the workflow step is created, you must specify the workflow step that the workflow is to begin with. If the workflow is to begin with an existing object, you must define the **Start Step for Existing Object** attribute. If the workflow is to begin by creating a new object, you must define the **Start Step for New Object** attribute. (A workflow template may be configured to begin with both new and existing objects.)

Once the first step has been created, you can then continue defining all other workflow steps including their pre- or post-conditions or updating actions as well as more general information about the workflow template such as email notifications, target date reminders, batch processing of workflows, etc.

If the workflow template that you are configuring should be triggered in the context of another workflow, please review the information in the section Configuring a Workflow Step to Trigger a Subordinate Workflow before configuring the workflow template.

Workflows can be started via a RESTful service call to the new workflow endpoint of the Alfabet RESTful services. The **Applicable for REST API** attribute is available for workflow templates. The attribute must be set to `True` to start a workflow based on the workflow template via the new endpoint workflow. Only workflow templates with the attribute **Automatic Start** set to `True` can be executed via RESTful service calls. For more information about the configuration required to implement the event capability, see the chapter Configuring Events.

To create a workflow template and define the first initial attributes:

1) In the **Workflow** tab, right-click the **Workflow Templates** icon at the top of the explorer and select **Create New Workflow Template**. The new workflow template  is displayed in the explorer.
2) Click the workflow template to open the attribute window and begin defining some of the attributes in the section of the attribute window labeled **Common**.
3) Ensure that the **Workflow State** attribute is set to `Plan`. The workflow template may only be configured in the `Plan` state. The **Workflow State** attribute of the workflow template is edited by right-clicking the workflow template and selecting the relevant value (for example, **Set Workflow**

State to Plan) The workflow template state is updated in the **State** attribute in the attribute window.

4) In the **Caption** attribute, enter a name to display in the Alfabet interface for the workflow template.

5) In the **Comment** attribute, enter an explanation about the workflow template. Users will see the comment in the **Description** attribute in the preview pane in the **My Workflows** and **Initiate Workflows** functionalities. Therefore, the comment should be specific enough so that users understand the intention and purpose of the workflow template.

6) To change the technical name of the workflow template, enter a name for the workflow template in the **Name** attribute. This name is not visible in the Alfabet software.

A validation mechanism checks for correct syntax when defining a technical name. The technical name of a configuration object may not begin with an empty space nor include any of the following special characters: **\ / * ? " > < |':**

Furthermore, please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.

Furthermore, if the technical name of a configuration object is changed, the name will be correctly updated in other configuration objects referencing the changed object during design time. Please note that the name of a changed configuration object will not be updated in guide pages nor if the guide pages reference a configuration object listed in the **Show Usage** functionality. If you plan to change the name of a configuration object, the reference in the guide page should be changed prior to changing the configuration object name.

The **Name** attribute of the workflow template should NOT be changed if it is already referenced as a sub-workflow (once it is selected in the **Workflow Template** attribute of a workflow step action of the **Type** `TriggerWorkflow`). If the workflow template name is changed, the sub-workflows may not be correctly triggered.

7) If you would like the workflows to be automatically deleted once they are completed, select `True` in the **Auto Delete** attribute. The automatic deletion of finished workflows requires a batch process that is executed by your system administrator. For more information about initiating a batch process for workflows, see the section *Batch Processes for Workflows with AlfaWorkflowCommandPrompt.exe* in the reference manual *System Administration*.

8) If you would like to allow multiple workflow steps that fulfill their pre-conditions to be executed at the same time in parallel to one another, select `True` in the **Allow Simultaneous Execution of Steps** attribute. If workflow steps may not be simultaneously executed, select `False` in the **Allow Simultaneous Execution of Steps** attribute.

If multiple workflow steps are defined in the **Next Steps** attribute for a workflow step and the pre-conditions are fulfilled for more than one of these workflow steps, the following scenarios are possible:

- If the **Allow Simultaneous Execution of Steps** attribute is set to `False`, an error message will occur. In other words, you must ensure that only one workflow step listed in the **Next Steps** attribute of each workflow step can have its pre-conditions fulfilled and be entered.

- If the **Allow Simultaneous Execution of Steps** attribute is set to `True`, then all workflow steps in the **Next Steps** attribute that fulfill their pre-conditions will be executed in parallel. The subsequent behavior of the workflow will depend on how the **Next Steps** attributes are defined for the parallel workflow steps. There are two alternatives for the configuration of the subsequent workflow steps:

  - All parallel running workflow steps will merge directly or indirectly into the same subsequent workflow step (defined in the **Next Steps** attribute of the merging workflow steps). In this case, the subsequent workflow step will only be entered when all preceding parallel-running steps have been completed.

  - Each parallel running workflow step forms its own independent branch with subsequent workflow steps. The branches do not merge into a common workflow step. The workflow will be completed only when ALL branches reach their last workflow step. **Please note that this workflow design is not recommended. Best practice recommendation for designing workflows with parallel branches is to merge all those branches back into a single lane of execution prior to ending the workflow.**

  Please note that if a workflow includes parallel workflow steps and one of the branches forms a cycle that returns to the workflow step where the branching occurred, the availability of that workflow step will be checked prior to recreating the workflow step. If the workflow step is still active, it will not be recreated.

9) If you would like to structure the workflow template in a workflow template folder, either enter the name of a new folder or select an existing folder in the **Group** attribute. Workflow folders are only for the purposes of organizing workflow templates in Alfabet Expand. They are not visible in the Alfabet interface.

10) Click the **Save**  button to save your definitions.

You must now create at least the first workflow step. If the workflow is to begin with an existing object, you must define the **Start Step for Existing Object** attribute. If the workflow is to begin by creating a new object, you must define the **Start Step for New Object** attribute. To carry out these procedures, see:

- Creating a Workflow Step
- Defining the Sequence of the Workflow Steps

Defining the Start of Workflows Based on the Workflow Template

When you configure the start definition for a workflow template, you determine whether workflows may be started manually by specific users in the Alfabet community, whether workflows may be started automatically via a batch process, or whether workflows may be started manually and automatically.

If the workflow template is to be manually started, permissible users may start the workflow in the **My Workflows** functionality, the **Initiate Workflows** functionality, or for a relevant object via the **Workflows** button available in the toolbar of the object profile.

Please note the following about the configuration of the workflow template:

- To configure the manual or automatic start of a workflow, you should first create the first step in the workflow. The workflow step does not need to be configured in detail at this point. However, it must exist and must have a name. For more information about defining a workflow step, see the section Creating a Workflow Step.
- Every workflow will begin with a base object that is the target of the workflow. The configuration of the objects targeted by workflows based on workflow templates configured to start manually or automatically is addressed in the section Specifying the Objects Targeted by the Workflows.
- In order for a workflow to be started manually or automatically, the **Workflow State** attribute of the workflow template must be set to `Active`. Workflows cannot be started for workflow templates that have the **Workflow State** attribute set to `Plan` or `Retired`. For more information about changing the **Workflow State** attribute, see the section Changing the Workflow State and Activating/Deactivating the Workflow Template.

The owner of a workflow is determined as follows:

- If a workflow is started manually in the **My Workflows** functionality, the current user starting the workflow is the workflow owner.
- If a workflow is started automatically via the batch tool `AlfaWorkflowCommandPrompt.exe`, then the workflow owner is based on the owner of the workflow template. If no workflow template owner is defined, then the current user executing the batch job is the workflow owner.
- If a workflow is started in the **Workflow Administration** functionality via the **Start Automatically** button, the workflow owner is based on the owner of the workflow template. If no workflow template owner is defined, then the current user triggering the workflow in the **Workflow Administration** functionality is the workflow owner.

The following information is available:

- Configuring the Manual Start of Workflows
- Configuring the Automatic Start of Workflows

Configuring the Manual Start of Workflows

You can configure the workflow template so that workflows can be started manually by permissible users. In this case, users with access permissions to the workflow template can start workflows for the workflow template in either the **My Workflows** functionality or the **Initiate Workflows** functionality. Only workflow templates with an active workflow state will be displayed in the **My Workflows** functionality and **Initiate Workflows** functionality.

The definition of permissible users is specified via the user group or user profile affiliation. The user who initiates the workflow is the original workflow owner and is responsible for maintaining the workflow as long as he/she remains the workflow owner.

To specify the manual start of workflows for a workflow template:

1) Click the workflow template  to open the attribute window.

2) Define the manual start of workflows for the workflow template. In the **Manual Start** attribute, select `True`.

3) Next, define the users who shall have permissions to initiate workflows for the selected workflow template. You can identify the users via user profiles and user groups. Define one or all of the following attributes:

- **Permitted User Profiles**: Select one or more user profiles that may initiate a workflow that is based on this workflow template.

  If the **Permitted User Profiles** attribute is not defined, then access will only be granted to

  the users that are member of the **Permitted User Groups** attribute.

- **Permitted User Groups**: Select one or more user groups that may initiate a workflow that is based on this workflow template.

  If you define one or more user profiles and one or more user groups, then only users belonging to at least one specified user profile OR at least one specified user group will be able to initiate a workflow based on the workflow template.

4) Next, you should specify the first step that will be used to start the workflow:

- If existing objects should be targeted in the first step of the workflow, specify the first workflow step in the **Start Step for Existing Object** attribute. Leave this attribute empty if existing objects are not targeted in the first workflow step.
- If new objects will be created during the initiation of the workflow, specify the first workflow step in the **Start Step for New Object** attribute. Leave this attribute empty if new objects are not targeted in the first workflow step.

  The workflow step must already be created. However, it does not need to be configured in detail at this point. However, it must exist and must have a name. For more information about defining a workflow step, see the section Creating a Workflow Step.

  Every workflow will begin with a base object that is the target of the workflow. To configure the objects targeted by a workflow template configured to be manually started, see the sections:

  - Configuring the Manual Start of Workflows
  - Configuring Automatically Started Workflows to Start with Existing Objects
  - Configuring Manually-Started Workflows to Start with New Objects
  - Changing to a Different Base Object for a Workflow Step

5) Next, you may define a caption for the initiation of the workflow template that will be displayed in the **My Workflows** functionality, **Initiate Workflows** functionality, and in the drop-down menu displayed via the **Workflows** button in the object profile. This allows you to define a caption that provides basic information to the user (for example, to indicate whether they will start the workflow with a new or existing object).

- If you have defined the **Start Step for Existing Object** attribute, enter a caption in the **Caption for Existing Object** attribute.
- If you have defined the **Start Step for New Object** attribute, enter a caption in the **Caption for New Object** attribute.

6) To provide additional direction for the workflow template in the **My Workflows** functionality and the **Initiate Workflows** functionality, enter a comment in the **Caption** attribute. This information will be displayed in the preview pane that opens when a user selects the workflow template in the table. If you do not define the **Caption for Existing Object** attribute or the **Caption for New Object** attribute, the text defined in the **Caption** attribute will be displayed in table and preview pane in the **My Workflows** functionality and the **Initiate Workflows** functionality.

7) Click the **Save** button to save your definitions.

Configuration Required to Start a Workflow in an Object Profile

A workflow may be manually started via the object profile of an object relevant for the workflow.

Typically, workflows may be manually started via the object profile for a base object in the object class specified in the **Start Base Class** attribute. If the **Source Base Class** attribute has also been specified for the workflow template, a workflow for the source object referencing the base object can also be started via the object profile. The workflow for the source object will be displayed along with the workflow for the base object in the drop-down menu for the **Workflow** button. For more information about the configuration of the **Start Base Class** attribute and the **Source Base Class** attribute, see the section Specifying the Objects Targeted by the Workflows.

If workflows may be started via the object profile, users will see the **Workflow** button in the toolbar of the relevant object profile of the object class specified in the **Start Base Class** attribute of the workflow template. In the case of object class stereotypes, the **Workflow** button will be displayed only in the custom object views configured for the object class stereotype.

To initiate a workflow, the user must click the **Workflow** button in the object profile and select the relevant workflow template listed in the in the drop-down menu. The workflow will be initiated for the selected object. A workflow based on a selected workflow template may only be initiated once for an object.

For users to be able to initiate a workflow for an object via the object profile, the following must be taken into consideration regarding the configuration of the workflow template:

- The workflow template must be configured to start manually. Ensure that the **Manual Start** attribute is set to `True`.
- The workflow template must be configured to start with an existing object. Therefore, the **Start Step for Existing Object** attribute must be defined. (Please note that this does not need to be defined for a source object.) For more information, see the section Specifying the Objects Targeted by the Workflows.
- If the workflow step specified in the **Start Step for Existing Object** attribute is a workflow step for which the **Type** attribute is set to `System`, an informational message can be configured in the

Message for Started Workflow attribute that lets the user know that a workflow has been started when he/she clicks the **Workflow** button. A workflow step for which the **Type** attribute is set to `System` is a system-generated action that does not require user input or user action. If no workflow message is defined, the workflow will begin, but it may not be apparent to the user initiating the workflow that the workflow has actually started.

Configuring the Automatic Start of Workflows

For each workflow template that is to be started automatically, a query must be defined in order to find or create the objects targeted by the workflow.

If the workflow template is to be started automatically, a batch process must be configured and executed by the system administrator in order to trigger the creation of the workflows. For more information about configuring and initiating a batch process for workflows, see the section *Batch Processes for Workflows with AlfaWorkflowCommandPrompt.exe* in the reference manual *System Administration*.

Typically, the system administrator will execute scheduled batch processes in the enterprise. If the workflow template is to be started via a batch process, a workflow administrator can also execute the batch initiation of workflows for a selected workflow template in the **Workflow Administration** functionality in the **Administration** application. For more information about how the workflow administrator can execute a batch process for a specific workflow template, see the chapter *Tracking and Managing Workflows* in the reference manual *User and Solution Administration*.

Workflows can be started via a RESTful service call to the new workflow endpoint of the Alfabet RESTful services. The **Applicable for REST API** attribute is available for workflow templates. The attribute must be set to `True` to start a workflow based on the workflow template via the new endpoint workflow. Only workflow templates with the attribute **Automatic Start** set to `True` can be executed via RESTful service calls. For more information about the configuration required to implement the event capability, see the chapter Configuring Events.

To specify the automatic start of a workflow via a configured batch process:

1) Click the workflow template to open the attribute window.
2) Ensure that `True` is selected in the **Automatic Start** attribute.
3) Next, you should specify the first step that will be used to start the workflow:

- If existing objects should be targeted in the first step of the workflow, specify the first workflow step in the **Start Step for Existing Object** attribute.
- If new objects will be created in the first step of the workflow, specify the first workflow step in the **Start Step for New Object** attribute.

The workflow step must already be created. However, it does not need to be configured in detail at this point. However, it must exist and must have a name. For more information about defining a workflow step, see the section Creating a Workflow Step.

For each workflow template that is to be started automatically, a query must be defined in order to find or create the objects targeted by the workflow. If the workflow template

is configured to automatically start for objects already existing in the database, a workflow will be started for each object found by the query.

To configure the objects targeted by a workflow templates configured to be automatically started, see the sections:

- Configuring Automatically Started Workflows to Start with Existing Objects
- Changing to a Different Base Object for a Workflow Step

4) Click the **Save** button to save your definitions.

Specifying the Objects Targeted by the Workflows

This section focuses on the configuration of the objects for which workflows are to be started. The documentation is relevant for both conventional workflow templates as well as workflow templates that have been configured to trigger sub-workflows via the `TriggerWorkflow` option in the **Type** attribute defined for a workflow step action.

Whether you specify the target objects via the attributes **Start Base Class** or **Source Base Class** will depend on the scenario that you plan to implement to start the workflow template.

- Should workflows be initiated for objects that already exist in the database?
- Should new objects be created as part of the workflow start?
- Should new objects be created for an object found in a base object class. For example, for each application, a corresponding risk object should be created.
- Should new objects be created via an automatically-started workflow. For example, a set of new applications should be created with specified data already defined.
- Should new objects be created that must be approved before they can be added to the Alfabet database. For example, a set of new applications must go through an approval process before they can be added to the Alfabet inventory.

- You must ensure that the object class specified in the **Start Base Class**, **Base Object Query**, and the **Source Object Query** attributes has a `Name` attribute and an `ID` attribute defined. If the object class has no **Name** attribute or **ID** attribute, then you must configure a custom object class property for the `Name` attribute and `ID` attribute. For more information about the configuration of custom object class properties, see the section Configuring Custom Properties for Protected or Public Object Classes.
- If you are creating a workflow template for the object class `ITMapView`, you must configure the workflow template to start with existing objects because a map view is dependent on the object class `ITMasterPlanMap`. A new map view cannot be created in the context of a workflow.

The following information is available:

- Configuring Manually Started Workflows to Start with Existing Objects
- Configuring Automatically Started Workflows to Start with Existing Objects

- Configuring Manually-Started Workflows to Start with New Objects
- Configuring Workflows to Create New Target Objects via the Source Base Class Attribute
- Changing to a Different Base Object for a Workflow Step

Configuring Manually Started Workflows to Start with Existing Objects

If the manually-started workflow is to begin with existing objects, you must define the **Start Base Class** attribute to specify the base object class of the workflow.

Users with permissions to the workflow template will be able to select the object that they want to start a workflow for by means of a configured standard or custom selector. The object that is targeted by the workflow is considered the base object of the workflow. The first workflow step as well as all subsequent workflow steps will target the base object unless a workflow step is explicitly configured to work with an object other than the base object.

To specify existing objects as the targets of manually-started workflows:

1) Click the workflow template  to open the attribute window.
2) Ensure that the **Manual Start** attribute is set to `True`.
3) Define the base object class for the workflow template. This is the object class that the workflow begins with when the first workflow step is performed. In the **Start Base Class** attribute, select the base object class for the workflow template.

   If you select an object class stereotype (`<ObjectClass:ObjectClassStereotype>`) in the **Start Base Class** attribute:

   - The standard object selector that will open will display all objects for all stereotypes configured for the object class. The user starting the workflow must ensure that he/she selects an object of the correct stereotype in the object selector. To avoid this, you can specify a custom selector configured specifically for the object class stereotype. For more information about the configuration of custom selectors, see section Configuring a Custom Selector for Search Functionalities.
   - The **Workflow**  button will be displayed only in the custom object views configured for the object class stereotype.

   For an overview of the object classes for which workflows can be configured, see *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

4) Ensure that the **Start Step for Existing Object** attribute is specified with the first workflow step. When the user clicks the **New Workflow** button in the **My Workflows** functionality or the **Initiate Workflows** functionality, the workflow will begin and the relevant view defined for the first workflow step will open (with the exception of a workflow step action where the **Type** attribute is set to `System`).

The workflow step must already be created. However, it does not need to be configured in detail at this point. However, it must exist and must have a name. For more information about defining a workflow step, see the section Creating a Workflow Step.

5) In the **Define Base or Source Artifact Selector** attribute, specify the standard or custom selector that should be implemented for users to find the relevant object.

- If no selector is defined in the **Define Base or Source Artifact Selector** attribute, then the standard or custom selector defined in the **Selector Definition** attribute of the relevant class setting will be implemented. For more information about defining class settings, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.
- If a custom selector is defined in the **Define Base or Source Artifact Selector** attribute, then the custom selector specification will override the class setting definition. For more information about the configuration of custom selectors, see the section Configuring a Custom Selector for Search Functionalities.

6) In the **Object Permission** attribute, select `AllUsers` if any user with Read/Write access permissions to the workflow template may start the workflow template, regardless of the access permissions defined for the object that is targeted by the workflow template. Select `AuthorizedUsersOnly` if only users with Read/Write access permissions to the object that is targeted by the workflow template may start the workflow.

For an overview of the access permission concept in Alfabet, see the section Overview of Access Permissions for Objects.

7) In the **Allow Multiple Workflows for Same Object** attribute, select `True` if multiple workflows can be created for the same object in the base object class. Select `False` if only one workflow can be created per object for the base object class. To avoid multiple users simultaneously entering conflicting data, it is recommended that this attribute be set to `False`.

8) Click the **Save** button to save your definitions.

Configuring Automatically Started Workflows to Start with Existing Objects

If the automatically-started workflow is to begin with existing objects, you must define the **Start Base Class** attribute to specify the base object class of the workflow. Additionally, you must specify a query in the **Base Objects by Query** attribute that specifies the objects for which workflows should be started.

A workflow will be automatically started for each object found by the query. The first workflow step will be entered once the workflow is started. Depending on the configuration of the workflow step, responsible users may see and thus perform the first workflow step in the **My Workflow Activities** functionalities (`WF_UserWorkflowActivities`, `WF_UserWorkflowActivitiesExt`, and `WF_UserWorkflowActivitiesCommon`) and/or the **Workflow Activities Explorer** (`WFS_Explorer`).

The object that is targeted by the workflow is considered the base object of the workflow. The first workflow step as well as all subsequent workflow steps will target the base object unless a workflow step is explicitly configured to work with an object other than the base object.

If the workflow template is to be started automatically, a batch process must be configured and executed by the system administrator in order to trigger the creation of the workflows. For more information about configuring and initiating a batch process for workflows, see the section *Batch Processes for Workflows with AlfaWorkflowCommandPrompt.exe* in the reference manual *System Administration*.

Typically, the system administrator will execute scheduled batch processes in the enterprise. If the workflow template is to be started via a batch process, a workflow administrator can also execute the batch initiation of workflows for a selected workflow template in the **Workflow Administration** functionality in the **Administration** application. For more information about how the workflow administrator can execute a batch process for a specific workflow template, see the chapter *Tracking and Managing Workflows* in the reference manual *User and Solution Administration*.

To specify existing objects as the targets of automatically-started workflows:

1) Click the workflow template  to open the attribute window.
2) Ensure that `True` is selected in the **Automatic Start** attribute.
3) Define the base object class for the workflow template. This is the object class that the workflow begins with when the first workflow step is performed. In the **Start Base Class** attribute, select the base object class for the workflow template.

   If you select an object class stereotype (`<ObjectClass:ObjectClassStereotype>`) in the **Start Base Class** attribute:

   - The standard object selector that will open will display all objects for all stereotypes configured for the object class. The user starting the workflow must ensure that he/she selects an object of the correct stereotype in the object selector. To avoid this, you can specify a custom selector configured specifically for the object class stereotype. For more information about the configuration of custom selectors, see section Configuring a Custom Selector for Search Functionalities.
   - The **Workflow**  button will be displayed only in the custom object views configured for the object class stereotype.

   For an overview of the object classes for which workflows can be configured, see *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

4) Ensure that the **Start Step for Existing Object** attribute is specified with the first workflow step. When the user clicks the **New Workflow** button in the **My Workflows** functionality or the **Initiate Workflows** functionality, the workflow will begin and the relevant view defined for the first workflow step will open (with the exception of a workflow step action where the **Type** attribute is set to `System`).

   The workflow step must already be created. However, it does not need to be configured in detail at this point. However, it must exist and must have a name. For more information about defining a workflow step, see the section Creating a Workflow Step.

5) In the **Base Objects via Query** attribute, define a query to find the objects for which a workflow should start.

The following Alfabet query is an example of a query defined to find existing applications via the role definition:

```
ALFABET_QUERY_500

FIND

Application

    InnerJoin Role AS RMrole ON RMrole.Object =
    Application.REFSTR

    InnerJoin RoleType AS RMroletype ON RMroletype.REFSTR =
    RMrole.RoleType

    InnerJoin Role AS RErole ON RErole.Object =
    Application.REFSTR

    InnerJoin RoleType AS REroletype ON REroletype.REFSTR =
    RErole.RoleType

WHERE

    (AND

       RMroletype.Name ='Risk Manager'

       REroletype.Name ='Risk Evaluator'

    )
```

For general information about defining an Alfabet query or an SQL query, see the chapter Defining Queries.

6) In the **Allow Multiple Workflows for Same Object** attribute, select `True` if multiple workflows can be created for the same object in the base object class. Select `False` if only one workflow can be created per object for the base object class. To avoid multiple users simultaneously entering conflicting data, it is recommended that this attribute be set to `False`.

7) Click the **Save** button to save your definitions.

Configuring Manually-Started Workflows to Start with New Objects

If the manually-started workflow is to begin with new objects, you must define the **Start Base Class** attribute to specify the base object class of the workflow.

Users with permissions to the workflow template will be able to start a workflow for the workflow template. The editor or wizard configured for the first workflow step will open and the user can enter the required basic data to create the object.

The object that is created via the workflow is considered the base object of the workflow. All subsequent workflow steps will target the base object unless a workflow step is explicitly configured to work with an object other than the base object.

To specify the creation of new objects for manually-started workflows:

1) Click the workflow template to open the attribute window.
2) Ensure that the **Manual Start** attribute is set to `True`.

3) Define the base object class for the workflow template. This is the object class that the workflow begins with when the first workflow step is performed. In the **Start Base Class** attribute, select the base object class for the workflow template.

   If you select an object class stereotype (`<ObjectClass:ObjectClassStereotype>`) in the **Start Base Class** attribute:

   - The standard object selector that will open will display all objects for all stereotypes configured for the object class. The user starting the workflow must ensure that he/she selects an object of the correct stereotype in the object selector. To avoid this, you can specify a custom selector configured specifically for the object class stereotype. For more information about the configuration of custom selectors, see section Configuring a Custom Selector for Search Functionalities.
   - The **Workflow** button will be displayed only in the custom object views configured for the object class stereotype.

   For an overview of the object classes for which workflows can be configured, see *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

4) Ensure that the **Start Step for New Object** attribute is specified with the first workflow step. When the user clicks the **New Workflow** button in the **My Workflows** functionality or the **Initiate Workflows** functionality, the workflow will begin and the relevant editor or wizard defined for the first workflow step will open.

   The workflow step must already be created. However, it does not need to be configured in detail at this point. However, it must exist and must have a name. For more information about defining a workflow step, see the section Creating a Workflow Step.

5) Click the **Save** button to save your definitions.

Configuring Workflows to Create New Target Objects via the Source Base Class Attribute

Software AG provides the possibility of configuring more complex workflow scenarios in terms of creating or finding target objects for workflows. Such configurations require the specification of the **Start Base Class** attribute and the **Source Base Class** attribute as well as associated queries. This configuration is more involved than the previously described scenarios.

The following scenarios can be addressed for both manually and automatically started workflows via the configuration of the **Start Base Class** attribute and the **Source Base Class** attribute:

- New objects should be created for each found object in a base object class. For example, for each application, a corresponding risk object should be created.
- New objects should be created via an automatically-started workflow. For example, a set of new applications should be created with specified data already defined. In this case, you must configure a query that specifies how many objects to create as well as the initial values that should be automatically written to the object class properties.

A query must be configured via the **Source Objects via Query** attribute to create the objects for which workflows must be started. In this case, you must initially specify which object class properties are automatically filled for the new objects (such as `Name`). This must be specified by configuring an **Action on Entered Step** action for the first workflow step.

Typically, workflows may be manually started via the object profile for a base object in the object class specified in the **Start Base Class** attribute. If the **Source Base Class** attribute has also been specified for the workflow template, a workflow for the source object referencing the base object can also be started via the object profile. The workflow for the source object will be displayed along with the workflow for the base object in the drop-down menu for the **Workflow** button. For more information about the configuration of the **Start Base Class** attribute and the **Source Base Class** attribute, see the section Specifying the Objects Targeted by the Workflows.

A workflow will be started for each object created via a query. The first workflow step will be entered once the workflow is started. The object that is created by the workflow is considered the base object of the workflow. The first workflow step as well as all subsequent workflow steps will target the base object unless a workflow step is explicitly configured to work with an object other than the base object.

If the workflow template is to be started automatically, a batch process must be configured and executed by the system administrator in order to trigger the creation of the workflows. For more information about configuring and initiating a batch process for workflows, see the section *Batch Processes for Workflows with AlfaWorkflowCommandPrompt.exe* in the reference manual *System Administration*.

Typically, the system administrator will execute scheduled batch processes in the enterprise. If the workflow template is to be started via a batch process, a workflow administrator can also execute the batch initiation of workflows for a selected workflow template in the **Workflow Administration** functionality in the **Administration** application. For more information about how the workflow administrator can execute a batch process for a specific workflow template, see the chapter *Tracking and Managing Workflows* in the reference manual *User and Solution Administration*.

To specify the creation of new target objects via the **Source Base Class** attribute:

1) Click the workflow template to open the attribute window. The **Automatic Start** attribute and the **Manual Start** attribute may be set `True`.
2) Next define the start base object class for the workflow template. This is either the custom object class or another Alfabet object class that the workflow will begin with when the first workflow step is performed. Objects will be initially created for this object class. In the **Start Base Class** attribute, select the start base object class for the workflow template.
3) Next, define the source base object class for the workflow template. This is the referenced object class containing objects in the Alfabet inventory for which workflows will be started. In the **Source Base Class** attribute, select the source base object class for the workflow template.

   If relevant workflow step actions have been configured, the data for the source base objects will be copied to the transient objects (based on the start base object class) when the workflows are created.

4) If the **Automatic Start** attribute is set to ***True***: In the **Source Objects via Query** attribute, define a query to create the objects for which workflows should be started.

For example, the workflow template is targeted at the creation of workflows for applications that are not yet allocated domains in the enterprise. The **Start Base Class** attribute of the workflow template specifies a custom class that has been created specifically for this workflow template. The **Source Base Class** is Application. The following Alfabet query searches for existing active applications in the inventory that have no domain assignment.

```
ALFABET_QUERY_500
FIND Application
WHERE (AND Application.Domain IS NULL
    Application.ResponsibleUser IS NOT NULL
    Application.ObjectState ='Active')
```

For general information about defining an Alfabet query or SQL query, see the chapter Defining Queries.

5) Next, define an action of the **Type** `Script` for the **Action On Entered Step** defined for the first workflow step. In the **Statements** attribute, you should specify the object class properties that should be automatically written to the transient objects.

For example.

```
Set(WORKFLOWBASE.BASEOBJECT,SOURCE.REFSTR);
Set(WORKFLOWBASE.ID,SOURCE.ID);
Set(WORKFLOWBASE.Name,SOURCE.Name);
```

For more configuration of a property update of the type `Script`, see the section Configuring Automatic Property Updates for a Workflow Step.

6) Configure the remaining workflow steps for the workflow template, as needed.

7) If the **Manual Start** attribute is set to ***True***: In the **Object Permission** attribute, select `AllUsers` if any user with access permissions to the workflow template may start the workflow template, regardless of the access permissions defined for the object that is targeted by the workflow template. Select `AuthorizedUsersOnly` if only users with access permissions to the object that is targeted by the workflow template may start the workflow.

8) Click the **Save**  button to save your definitions.

Changing to a Different Base Object for a Workflow Step

Typically, a workflow will process a single object in the base object class defined for the workflow template. However, it may be necessary during the workflow to change to a different object for a workflow step. This could be necessary, for example, in order to allow objects related to the base object to be defined.

For example, a wizard has been configured in order to capture application data. The enterprise requires that the ICT object that the application is assigned also be defined at the time that the application data is captured. In this case, you can configure a query to find the object in object class ICT object that should be defined in the context of the wizard

In order to change to a different object from the same base object class or a different base object class as that defined in **Start Base Class** attribute for the workflow template, you must configure a query for each workflow step that targets an object that is not the original base object. The query will be executed when the workflow step is entered.

The object found by the query will only be applied to the current workflow step. The subsequent workflow step will return to the original target object found in the **Start Base Class** attribute. If the subsequent workflow step should not return to the original target object, you must enter the same query or a different query for the subsequent workflow step. In other words, if you change from the base object to another object and this new object is the target of 3 workflow steps, the query to find the target object must be specified in all three workflow steps.

To configure the workflow to change to a different object than the original base object.

1) Click the workflow step that should target a different object than the original base object.
2) You can define either an Alfabet query or an SQL query. Define one of the following:

- In the **Base Object via Query** attribute, click the **Browse** button to open the **Alfabet Query Builder** and define the query. For information about defining an Alfabet query or SQL query, see the chapter Defining Queries.
- In the **Base Object via Query as Text** attribute, paste in an Alfabet query or SQL query that you have defined in a text editor.

The following example displays an Alfabet query that searches for the ICT object that the application is assigned to:

```
ALFABET_QUERY_500
FIND
ICTObject
  InnerJoin Application
        ON Application.ICTObject = ICTObject.REFSTR

WHERE
Application.REFSTR CONTAINS  :BASE
```

3) Once you have defined the base object query for the selected workflow step, you can proceed to define the workflow step as described in the section Creating a Workflow Step.

Keep the following in mind when configuring the base object query for a workflow step:

- The object class identified in **Base Object Query** attribute must have an object class property `ID` defined. If the object class has no object class property `ID` defined, then you must configure a custom object class property for the `ID`. For more information about the configuration of custom object class properties, see the section Configuring Custom Properties for Protected or Public Object Classes.
- The query to find the base object for the workflow step is evaluated before the evaluation of queries associated with the workflow step (for example, queries for pre-conditions, post-conditions, or other workflow step actions). Therefore, when using the parameter `BASE` in queries related to a workflow step, `BASE` returns the `REFSTR` of the base object of the workflow step. To refer to the initial base object of the workflow, use the parameter `WORKFLOWBASE`.

- The result of the query entered in the **Base Object Query** attribute should have the `REFSTR` of the object which should be used as Base Object in the first column. Therefore, the `REFSTR` of the object must be defined as the first `SELECT` property in native SQL queries. For Alfabet queries, `SHOW` properties do not need to be defined. The `REFSTR` of the object found by the query is selected automatically as the query result.
- Please note that you must ensure that only one object is found in the query you specify. An error will occur if more than one object is found by the query or if no object is found by the query. If an error occurs, the workflow administrator or workflow owner will need to intervene and redirect the workflow.

4) Click the **Save**  button to save your definitions.

Creating a Workflow Step

A workflow step is an activity or task in a workflow that must be performed. The workflow template may have as many workflow steps as needed to complete the relevant task. Standard and custom editors, wizards, page views, configured reports, and object profiles may be implemented in the workflow step so that users can complete the task. A workflow step may entail entering, modifying or reviewing data, or a workflow step may be configured to be automatically executed by the Alfabet system. If the workflow step is not automatically executed by the system, one or more users may be defined as responsible to perform the workflow step. A user may delegate a workflow step to another user or refuse a workflow step. Depending on the configuration of the workflow step, a user may be required to confirm a workflow step before the workflow can advance to the next workflow steps, or the confirmation of the workflow step may occur automatically. A workflow step may also have one or more associated workflow step actions which allow various operations to be performed when the workflow step is entered, exited, refused, or expired. Additionally, each workflow step may have one or more pre-conditions or post-conditions that must be fulfilled in order to enter or exit the workflow step.

When all workflow steps have been created, they should be sequenced. This is explained in the section Defining the Sequence of the Workflow Steps.

To create a workflow step:

1) In the **Workflow** tab, right-click the workflow template  and select **Add New Workflow Step**. The new workflow step  is displayed in the explorer.

   A workflow step can also be created based on a copy of an existing workflow step. To do so, create a new workflow step without defining any attributes. Select the workflow step you want to copy and right click and select **Copy**. Click the new workflow step and select **Paste**. The attributes are filled as defined and can be modified, as needed.

2) Click the + symbol next to the workflow step to expand the node.

   Below the workflow step you will see workflow step actions. The nodes are labeled **Pre-Conditions**, **Post-Conditions**, **Action on Entered Step**, **Action on Refused Step**, **Action on Expired Step**, and **Action on Exited Step**.

   These nodes will later allow you to define automatic property updates, triggered workflows (sub-workflows), and email notifications that should occur when a workflow step is entered, refused, expired, or exited as well as any pre-conditions or post-conditions that must be fulfilled in order for the workflow step to be entered or exited. These configuration aspects are described in the following sections.

   - Configuring Pre- and Post-Conditions for a Workflow Step
   - Configuring Automatic Property Updates for a Workflow Step
   - Configuring Events to Be Triggered for a Workflow Step
   - Configuring a Workflow Step to Trigger a Subordinate Workflow
   - Configuring Email Notifications for Workflows

3) Click the workflow step  to open the attribute window and begin defining some of the basic attributes.

4) To change the technical name of the workflow step, enter a name for the workflow step in the **Name** attribute. The name will be displayed in the Alfabet interface.

   A validation mechanism checks for correct syntax when defining a technical name. The technical name of a configuration object may not begin with an empty space nor include any of the following special characters: **\ / * ? " > < |':**

   Furthermore, please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.

   Furthermore, if the technical name of a configuration object is changed, the name will be correctly updated in other configuration objects referencing the changed object during design time. Please note that the name of a changed configuration object will not be updated in guide pages nor if the guide pages reference a configuration object listed in the **Show Usage** functionality. If you plan to change the name of a configuration object, the reference in the guide page should be changed prior to changing the configuration object name.

5) In the **Caption** attribute, enter a caption for the workflow step. The caption will be displayed in the Alfabet interface.

6) In the **Comment** attribute, enter an explanation about the workflow step. The comment will be displayed as a workflow step description in the Alfabet interface.

7) Click the **Save** button to save your definitions.

The following information is available:

- Defining the View Implemented for a Workflow Step
- Specifying a Wizard for the Workflow Step
- Specifying an Editor for the Workflow Step
- Specifying a Standard Alfabet Page View for the Workflow Step
- Specifying a Configured Report for the Workflow Step
- Specifying an Object View for the Workflow Step
- Specifying a Dynamically-Generated View for the Workflow Step
- Specifying a System-Generated Action for the Workflow Step
- Defining the Users Responsible for a Workflow Step
- Defining a Deadline and Reminders for a Workflow Step
- Configuring the Action Buttons Available for a Workflow Step
- Specifying the Confirmation of a Workflow Step of the Type Wizard or Editor
- Automatic Confirmation of a Workflow Step (Types Wizard and Editor)
- Confirmation via the Workflow Step Completion Editor (Types Wizard and Editor)
- One-Click Confirmation via the Confirm Button (Types Wizard and Editor)

Defining the View Implemented for a Workflow Step

A workflow step may constitute a wide variety of tasks or actions that are either executed by a single user or a groups of users, or by the system itself. The following types of views can be implemented for a workflow step:

- Standard and/or custom editor
- Standard or custom wizard
- Standard Alfabet page view (technically called a `graphic view`)
- Object view
- Configured report created by your enterprise

If you are defining the first workflow step of the workflow, you must select either `Editor` or `Wizard` in the **Type** attribute for the workflow step. The first workflow step is specified for the workflow in either the **Start Step for Existing Object** attribute or the **Start Step for New Object** for the workflow template. For more information about specifying the first workflow step, see the section Defining the Sequence of the Workflow Steps.

Depending on the task that is to be executed in the workflow step, you can configure the workflow step so that users may be required to enter data for an object or to review data for the object. Thus, if a user is expected to only review and confirm data that has been defined, you can specify the workflow step to be ReadOnly.

Additionally, you can refine the visibility of data or functionalities in the views assigned to a workflow step. For editors, for example, you can hide tabbed pages, and for object views, graphic views, and configured reports, you can hide toolbar functionalities, thereby ensuring that users do not execute an unwanted action.

The following information is available:

One or more post-conditions can later be defined to ensure that the correct data has been completed by a user for a workflow step. For more information about the configuration of post-conditions, see the section Configuring Pre- and Post-Conditions for a Workflow Step.

The following information is available:

- Specifying a Wizard for the Workflow Step
- Specifying an Editor for the Workflow Step
- Specifying a Standard Alfabet Page View for the Workflow Step
- Specifying a Configured Report for the Workflow Step
- Specifying an Object View for the Workflow Step
- Specifying a Dynamically-Generated View for the Workflow Step
- Specifying a System-Generated Action for the Workflow Step

Specifying a Wizard for the Workflow Step

A wizard can be implemented in the workflow capability to capture the initial data required for a new object or to provide a structured process for the definition of references, cost data, project planning, etc. A wizard that is to be implemented in the workflow capability must first be configured in the **Wizards** node. Once the wizard has been configured and tested, it can be assigned to the relevant workflow step.

A wizard cannot be configured in the context of the workflow step and therefore the **Configure View** option will not be available when right-clicking the workflow step. Therefore, the wizard should be configured in the context of the wizard configuration capability before the workflow is configured. The wizard should be designed specifically for the workflow step and should require no further modifications once it is assigned to the workflow step. For detailed information about how to configure a wizard, see the chapter Configuring Wizards.

To specify that a wizard is to be implemented in a workflow:

1) Click the workflow step to open the attribute window.
2) In the **Type** attribute, select `Wizard` in order to implement a wizard for the selected workflow step.
3) In the **View** attribute, select the relevant wizard that should open so that users can carry out the selected step.
4) Click the **Save** button to save the workflow step definition.

Specifying an Editor for the Workflow Step

You can define a standard editor to be displayed for the workflow step. You can also specify a custom editor to be displayed with the standard editor. In this case, you must specify both the standard editor and custom editor to implement.

It is possible to hide interface controls (such as Edit fields) of the editor that are not relevant for the workflow step. By hiding tabbed pages or other interface controls in either the standard or custom editor, you can ensure that the user only sees the interface controls that are necessary for the task at hand. For example, if only the custom editor is relevant for the workflows step, the tabbed pages in the standard editor can be completely hidden from the user in the context of the workflow step.

Any custom editor you plan to implement for a workflow step must be configured before you configure the workflow step. For detailed information about how to configure a custom editor, see the chapter Configuring Custom Editors.

To specify that a standard or custom wizard is to be implemented in a workflow:

1) Click the workflow step to open the attribute window.
2) In the **Type** attribute, select `Editor` in order to implement a standard or custom editor for the selected workflow step.
3) In the **View** attribute, select the relevant standard editor for the workflow step. You must select a standard editor, even if you only want a custom editor to be displayed for the workflow step.
4) If you want to include a custom editor in the workflow step, select the relevant custom editor in the **Custom Editor** attribute.

5) If the data in the editor should only be reviewed and not edited, you can remove Read/Write access permissions to the editor. To do so, select `True` in the **Is Read-Only** attribute.

6) To hide tabbed pages or other interface controls in the standard or custom editor, right-click the workflow step in the explorer and select **Configure View**. The **View Configuration** editor opens. The **Select Object** attribute displays the name of the editor that is being configured in the context of the workflow step and the **Select View Scheme** attribute displays the name of the selected workflow step.

7) All tabs in the standard editor and, if defined, custom editor are displayed in the table. For each tabbed page or interface control that should be hidden in the context of the workflow step, click the relevant cell in the **Excluded** attribute. An X indicates that the interface control will be hidden. To make the interface control visible in the editor, remove the X in the **Excluded** attribute.

8) If a custom selector should replace the standard selector displayed in the **Default Selector** attribute, select the custom selector in the **Custom Selector** attribute.

9) Click **OK** to save the visibility definition and close the editor.

10) Click the **Save** button to save the workflow step configuration.

Specifying a Standard Alfabet Page View for the Workflow Step

You can define a standard Alfabet view to be displayed for the workflow step. You can hide any functionalities available in the toolbar buttons of the page view that are not relevant to the selected workflow step.

For an overview of the meaning of the technical page views listed in the **View** attribute, see *Page Views Assigned to Explorer Root Nodes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

To specify that a standard Alfabet page view is to be implemented in a workflow:

1) Click the workflow step to open the attribute window.

2) In the **Type** attribute, select `Graphic View` to select a standard Alfabet page view for the workflow step.

3) In the **View** attribute, select the relevant view that should open so that users can carry out the selected step. You must ensure that the page view you select is a relevant view for the object being processed.

4) If the data in the view should only be reviewed and not edited, you can remove Read/Write access permissions for the view. To do so, select `True` in the **Is ReadOnly** attribute.

5) To hide toolbar buttons in the standard Alfabet interface, right-click the workflow step in the explorer and select **Configure View**.

6) All functionalities in the view are displayed. In the upper right corner of the editor, click the **Configure** button to open the editor that allows you to hide functionalities in the view.

   The **Current View Scheme** attribute in the **View Configuration** editor displays the name of the selected workflow step.

7) For each tabbed page or interface control that should be hidden in the context of the workflow step, click the relevant cell in the **Excluded** attribute. An X indicates that the interface element will be hidden. To make the element visible in the editor, remove the X in the **Excluded** attribute.

8) If a custom selector should replace the standard selector displayed in the **Default Selector** attribute, select the custom selector in the **Custom Selector** attribute.

9) Click **OK** to save the visibility definition and close the editor.

Specifying a Configured Report for the Workflow Step

Reports of the type `Query` or `Custom` can be included in a workflow step. You can hide any functionalities available in the report's toolbar buttons that are not relevant to the selected workflow step.

It is highly recommended for reasons of interface compatibility, that you do not embed the configured report directly in the workflow step as described below. Instead, you should place the report in a wizard and assign the wizard to the workflow step. When the user clicks the **Perform** button, the wizard will open and display the report. The user can close the report via the wizard buttons. For more information about assigning a configured report to a wizard step, see the section Configuring a Wizard Step to Display a Configured Report in the chapter *Configuring Wizards*. For more information about embedding the wizard in a workflow step, see the section Specifying a Wizard for the Workflow Step.

To specify that a configured report is to be implemented in a workflow:

1) Click the workflow step  to open the attribute window.

2) In the **Type** attribute, select `Report` to select a configured report for the workflow step.

3) In the **View** attribute, select the relevant report that should open so that users can carry out the selected step.

4) To hide toolbar buttons in the report, right-click the workflow step  in the explorer and select **Configure View**.

5) All functionalities in the report are displayed. In the upper right corner of the editor, click the **Configure**  button to open the editor that allows you to hide functionalities in the report.

The **Current View Scheme** attribute in the **View Configuration** editor displays the name of the selected workflow step.

6) For each tabbed page or interface control that should be hidden in the context of the workflow step, click the relevant cell in the **Excluded** attribute. An X indicates that the interface element will be hidden. To make the element visible in the editor, remove the X in the **Excluded** attribute.

7) If a custom selector should replace the standard selector displayed in the **Default Selector** attribute, select the custom selector in the **Custom Selector** attribute.

8) Click **OK** to save the visibility definition and close the editor.

Specifying an Object View for the Workflow Step

Per default, users who navigate to a standard or custom object view in the context of a workflow step will have Read/Write access permissions to the object view. However, you can specify if ReadOnly access permissions should be implemented for the workflow step. You can hide any workspaces, views, and functionalities that are not relevant to the selected workflow step.

Any custom object views you plan to implement for a workflow step must be configured before you configure the workflow step. For detailed information about how to configure a custom object view, see the chapter Configuring Object Views.

To specify that a standard or custom object view is to be implemented in a workflow:

1) Click the workflow step to open the attribute window.
2) In the **Type** attribute select `Navigate` to open an editable standard or custom object view for the object being processed in the workflow step.
3) In the **View** attribute, select the relevant standard object view or custom object view for the workflow step.
4) If a user should have ReadOnly access permissions when navigating to an object profile in a workflow step, select `True` in the **ReadOnly** attribute.
5) To hide entire workspaces or individual views in the object view, right-click the workflow step in the explorer and select **Configure View**.
6) All workspaces and views are displayed. In the upper-right corner of the editor, click the **Configure** button to open the editor that allows you to hide workspaces and views in the object view.

   The **Current View Scheme** attribute in the **View Configuration** editor displays the name of the selected workflow step.

7) For each tabbed page or interface control that should be hidden in the context of the workflow step, click the relevant cell in the **Excluded** field. An X indicates that the interface element will be hidden. To make the element visible in the editor, remove the X in the **Excluded** field.
8) If a custom selector should replace the standard selector displayed in the **Default Selector** field, select the custom selector in the **Custom Selector** field. Click **OK** to save the configuration.
9) Click the **Save** button to save the workflow step configuration.

Specifying a Dynamically-Generated View for the Workflow Step

The workflow step type `Dynamic` is available to dynamically compute the views, editors, wizards, object profiles, and configured reports to be used in a workflow step based on a query that defines a rule. The workflow designer can define a query to dynamically compute the view in either the **View Query** attribute or the **View Query as Text** attribute. Please note that the first workflow step of the workflow cannot be of the type `Dynamic`.

To specify that the view is dynamically generated:

1) Click the workflow step to open the attribute window.
2) In the **Type** attribute, select `Dynamic`.
3) In the **View Query** attribute, select the relevant object class that should be queried and define the query as needed. The query must return a string representing a combination of a workflow step type appended by a colon and the name of the view to be used. The query returning the string should return a dummy value as the first column or the `RefString` and the required string as the second column. For example: Editor:<editor name> or Wizard:<wizard name>, or Navigate:<object view name>, etc. The query may use any of the following parameters:

- @BASE
- @WORKFLOWBASE
- @CURRENT_STEP
- @PREVIOUS_STEP
- @WORKFLOW
- @CURRENT_USER
- @CURRENT_MANDATE
- @CURRENT_PROFILE.

4) Click the **Save** button to save the workflow step configuration.

Specifying a System-Generated Action for the Workflow Step

A system-generated action can be configured for a workflow step. Prior to the Alfabet release 6.0, this definition was made via the **Type** attribute for the workflow step. For reasons of backward compatibility, the **Type** attribute continues to display the value `System`. However, it is highly recommended that the implementation of a system-generated action is defined via the workflow step actions available for each workflow For more information, see the section Configuring the Action Buttons Available for a Workflow Step.

Defining the Users Responsible for a Workflow Step

For each workflow step configured for the workflow template (other than steps of the type `System`), you must identify one or more users who are potentially responsible for the workflow step. You can configure one single user or a set of users to be responsible for the workflow step. A user may edit an object in the context of a workflow even though he/she may not otherwise have access permissions to the object. However, additional access permissions are not granted for the object outside of the workflow step.

The users who are identified via your configuration as responsible for performing the workflow step will automatically see the workflow step in the **My Workflow Activities** functionalities (`WF_UserWorkflowActivities`, `WF_UserWorkflowActivitiesExt`, and `WF_UserWorkflowActivitiesCommon`)

and/or the **Workflow Activities Explorer** (`WFS_Explorer`). Depending on the configuration of the workflow step, the responsible users may also receive an email notification when the workflow has advanced to the workflow step they are responsible for.

When you configure the workflow step, you must specify who the users are that are responsible to perform the workflow step. For example, the responsible user could be the authorized user of the object targeted by the workflow or users with a specified role (such as the role Architect). You can configure the responsible users to be found via any one or a combination of the following:

- The responsible user affiliation. In this case, the user is the authorized user of the object targeted by the workflow step.
- The role affiliation. In this case, users associated with one or more roles defined for the object can be specified. You can configure that only one user or all users with a role specified for the object targeted by the workflow step must carry out the workflow step.
- A set of users find by a query with the base object class `Person` or `User Group`. For example, a query could be defined to find the workflow owner or workflow initiator as the responsible user, or a query could be defined to find the authorized user of an object referenced by the base object (for example, the authorized user of an ICT owning the application that is the target of the workflow step)
- The user group affiliation.
- Users assigned to a user group. Multiple user groups can be specified. In this case, users associated with one or more users groups defined as authorized user groups for an object are responsible. You can configure that only one member of a user group, all members in a user group, or only one member in each user group must carry out the workflow step.
- Users in a user group found via a query based on the object class `UserGroup`.

Finally, you can specify whether the user currently working with an object via a currently active workflow step has exclusive edit rights to the object. This ensures that other users not involved in the current workflow step do not edit the data of the object that is being currently processed in the workflow step. Depending on the configuration of the users responsible for the workflow step, you may be removing the implicit access permissions that an authorized user has to the object targeted by this workflow step.

Please note the following regarding the configuration of the first workflow step:

- For a first workflow step initiated manually for new and existing objects in the **My Workflows** or **Initiate Workflows** functionalities or for a new object via the **Workflow** button in an object profile, the initiator of the workflow will be the responsible user initiating the first workflow step. (With the exception of a workflow step of the type `System`.) The attributes for the first workflow step defined in the **Responsibility** section of the attribute window (**Responsible User**, **Responsibles by Query**, etc.). are not relevant for workflows that are manually started.
- For a first workflow step that is automatically initiated for new and existing objects via the **Workflow Administration** functionality as well as new and existing objects initiated via a triggered sub-workflow or a batch process, the responsible user for the workflow step will be the responsible user(s) defined via the attributes available in the **Responsibility** section of the attribute window (**Responsible User**, **Responsibles by Query**, etc.).

Mandates are ignored in the context of workflows. Regardless of the mandate assignments to the user and the object managed in a workflow step, the user assigned responsibility for a workflow step has all access permissions required to execute the workflow step within the context of the **My Workflow Activities** functionalities (`WF_UserWorkflowActivities`, `WF_UserWorkflowActivitiesExt`, and `WF_UserWorkflowActivitiesCommon`) and/or the **Workflow Activities Explorer** (`WFS_Explorer`).

Therefore, if only users with a specific mandate should be permitted to perform a workflow step, the workflow designer must ensure that mandate assignment is taken into consideration via a query configured for the workflow step.

Once the user responsible for the workflow step has been configured, the following configurations can be specified:

- The automatic dispatch of email notifications informing each user that he/she is responsible for a current workflow step. For more information about the configuration of email notifications, see the section Configuring Email Notifications for Workflows.
- The configuration of the activities permissible for the workflow step. For example, should a user be able to refuse or delegate the workflow step? Is the user required to manually confirm a workflow step when has been executed? Are comments required by a user when performing or confirming a workflow step? Must the workflow step be specified by the user as completed in order to move to the next workflow step? For more information about the configuration of the actions allowed or required for the workflow step, see the section Configuring the Action Buttons Available for a Workflow Step.

To define the users responsible to perform a workflow step:

1) Click the workflow step to open the attribute window.
2) To explicitly define the responsible persons for the workflow step, define one or more of the following:

- **The authorized user of the object targeted by the workflow step**: Set the **Responsible User** attribute to `True` if the responsible user of the object addressed in this workflow step is the user that is responsible for performing the workflow step. Set to `False` if the responsible user of the object is not responsible to perform the workflow step.
- **Users assigned one or more specified roles**: In the **Responsibles by Roles** attribute, click the **Browse** button and select one or more role types to identify the responsible user(s) of the object targeted by this workflow step.
- **One or more users found via a query defined for the base object class** `Person`: If you want to specify that either the workflow owner, workflow initiator, or other set of persons are responsible for performing the workflow step, you can define a query based on the object class `Person` to find the user(s) who are responsible for performing the workflow step. Click the **Browse** button in the **Responsibles by Query** attribute to open the **Alfabet Query Builder** in order to define an Alfabet query or enter an SQL query in the **Responsibles by Query as Text** attribute.

  - The query must return the `REFSTR` property of one or multiple objects of the object class `Person` or the object class `UserGroup`. If you are defining an Alfabet query, no `SHOW` properties must be defined. The `REFSTR` of the object found by the Alfabet query is automatically returned as Alfabet query result. For native SQL queries, the `REFSTR` must be defined as the first property in

the SELECT statement. For information about defining Alfabet queries and native SQL queries in the context of Alfabet configurations, see the chapter Defining Queries.

- If you want to define the workflow owner or the workflow initiator as workflow step responsible, you must refer to the current workflow with the parameter WORKFLOW in the query. Each workflow corresponds to an object Workflow in the Alfabet database. The class Workflow has the object class property Initiator, which specifies the workflow initiator, and the object class property Owner, which specifies the workflow owner. Both object class properties reference the object class Person. To refer to the current workflow in the query, the parameter WORKFLOW can be used to return the REFSTR object class property of the current workflow. For example, the following Alfabet query finds the initiator of the workflow:

```
ALFABET_QUERY_500
FIND
Person
InnerJoin Workflow ON Workflow.Initiator =
Person.REFSTR
WHERE
Workflow.REFSTR CONTAINS:WORKFLOW
```

If no responsible user is found via the configured query, the workflow step will escalate and must then be dealt with by either the workflow owner or the workflow administrator in the **Workflows Administration** functionality.

3) To implicitly find users via their user group affiliation:

- **Users assigned one or more specified user groups**: In the **Responsibles by User Groups** attribute, click the **Browse** button and select one or more user groups to identify the responsible user(s) of the object addressed in this workflow step.
- **Users found for one or more user groups found via a query defined for the base object class User Group**: If you want to specify that users are found via a user group affiliation that cannot be defined via **Responsibles by User Groups** attribute, you can define a query based on the object class UserGroup to find the user(s) who are responsible for performing the workflow step. Click the **Browse** button in the **Responsibles by Query** attribute to open the **Alfabet Query Builder** in order to define an Alfabet query. Or enter an SQL query in the **Responsibles by Query as Text** attribute.

The query must return the REFSTR property of one or multiple objects of the object class Person or the object class UserGroup. If you are defining an Alfabet query, no SHOW properties must be defined. The REFSTR of the object found by the Alfabet query is automatically returned as Alfabet query result. For native SQL queries, the REFSTR must be defined as the first property in the SELECT statement. For information about defining an Alfabet query or native SQL query, see the chapter Defining Queries.

4) To specify that the user currently working with an object via a currently active workflow step has exclusive edit rights to the object, select True in the **Has Exclusive Edit Rights** attribute. This ensures that other users not involved in the current workflow step do not edit the data of the

object that is being currently processed in the workflow step. Depending on the configuration of the users responsible for the workflow step, you may be removing the implicit access permissions that an authorized user has to the object subject to this workflow step.

5) Click the **Save** button to save your definitions.

Defining a Deadline and Reminders for a Workflow Step

You can define due dates and reminders for each workflow step. If a workflow step's target due date is approaching, an email may be sent to the responsible user(s) of the workflow step if a corresponding batch process has been configured.

You can specify the number of days that a user has to complete the workflow step, the number of days before the deadline when the first reminder notification should be sent, and at which frequency the user should be reminded of the impending deadline.

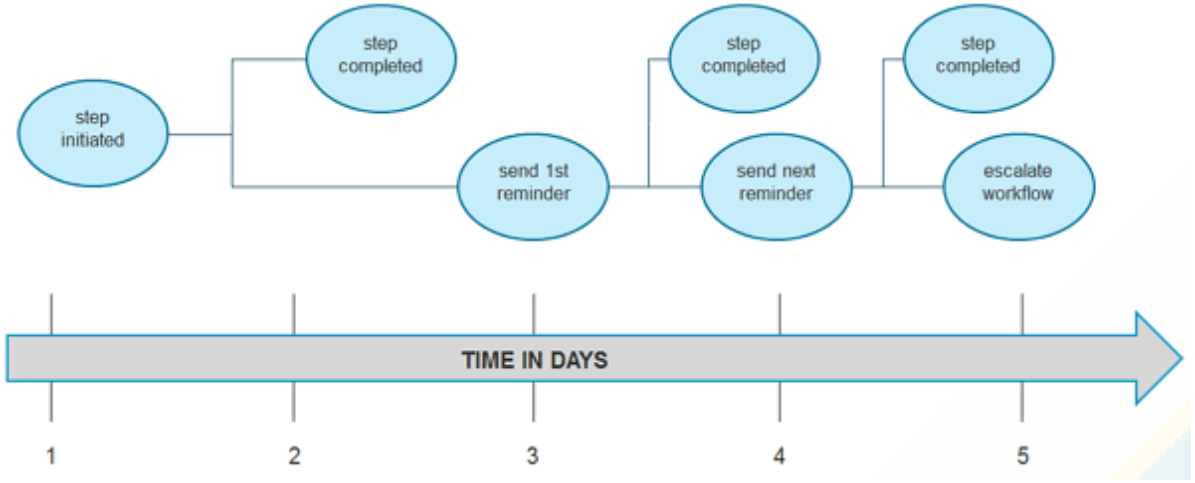


FIGURE: Example of the configuration and execution of a performance duration definition.

If the deadline has passed and an expired workflow step has been configured for the selected workflow step, then an expired workflow step will automatically advance to the configured step.

If this is the case, this series of events will be documented in the **Workflow Event Trace** page view for the selected workflow. If no subsequent step has been configured, the workflow step will be escalated (status: `Expired`) and must be manually redirected to a subsequent workflow step via the **Go To** button in the **Workflow Event Trace** page view

Workflow activities that are finished, expired, cancelled, etc. will remain visible in the **My Workflow Activities** explorers and in the respective sub-folder in the **Workflow Explorer** functionality for 30 days after they have been completed, etc. Thereafter, they will be automatically removed from the views.

If the **Performance Duration** attribute is defined for the workflow template, the batch job `WorkflowCommandPrompt.exe` must be executed in order to check the deadline of the workflow step and change the event type to `StepExpired`. It is recommended that reminder emails are sent every day. Therefore, the batch job `WorkflowCommandPrompt.exe` should be executed on daily basis because the reminders are sent when the specified reminder start, and frequency is matched based on the day that the workflow activity was created. For more information about

configuring a batch job to check workflow deadlines, see the section *Batch Processes for Workflows with AlfaWorkflowCommandPrompt.exe* in the reference manual *System Administration*.

Depending on the extent that workflows are implemented in your enterprise, some users may receive an extraordinary amount of reminder email notifications informing about impending workflow step deadlines. Therefore, you may want to consider concatenating all reminder emails generated for a user into one single email. This can be done by your system administrator via the parameter `SendWorkflowReminderWithOneEmail` in the server alias configuration. If the parameter is set to `True`, all reminder emails will be concatenate into one email for all users when the batch job `WorkflowCommandPrompt.exe` is executed. For more information about configuring the concatenation of reminder email notifications, see the section *Batch Processes for Workflows with AlfaWorkflowCommandPrompt.exe* in the reference manual *System Administration*.

To define a deadline for the workflow step as well as the timing of the reminder notifications to be sent to responsible users that a deadline is approaching or passed:

1) Click a workflow step to open its attribute window.
2) In the **Performance Duration** attribute, define the number of days that the workflow step may require to be completed.

   The integer entered for the **Performance Duration** attribute must be greater than 0 if the **Check Performance Duration** option should be enabled in the **Workflow Administration** functionality. Please note that a workflow step that has its performance duration set to 1 will not expire one day after it was started but the day after that.

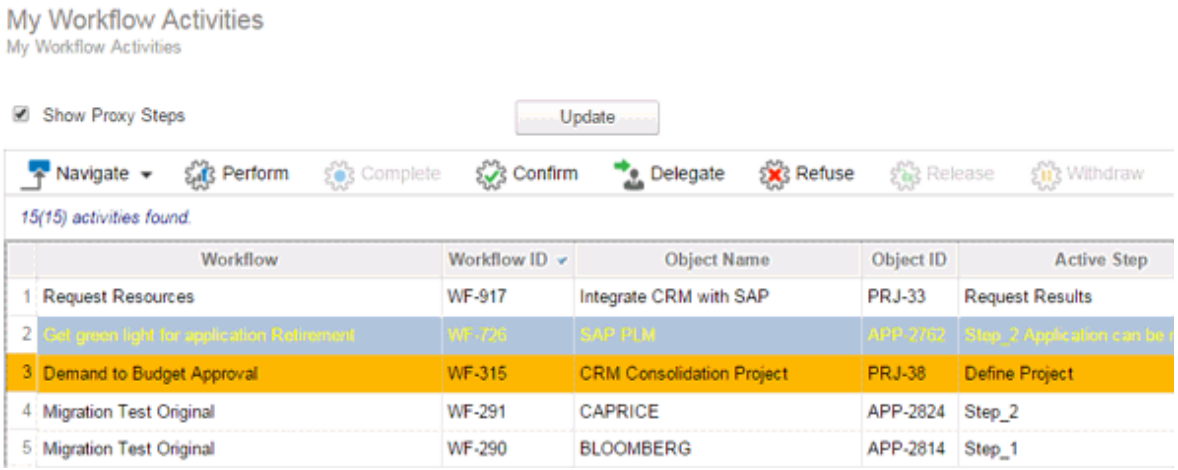
   If you do not change the default value of 0, the responsible user will have an unlimited amount of time to complete the workflow step. If the **Performance Duration** attribute for a workflow step is set to 0 so that the workflow step never expires, reminder email notifications will be sent out based on specifications of the **Performance Reminder Start** and **Performance Reminder Frequency** attributes. For more information, see the section *Checking a Workflow Step's Deadline* in the reference manual *User and Solution Administration*.

3) In the **Performance Reminder Start** attribute, define a number indicating on which day during the duration of the workflow step (day 1 = start of workflow step) the first reminder should be generated and sent to the responsible users. In the **Performance Reminder Frequency** attribute, enter a number indicating in which intervals (number of days) an email notification should be generated and sent to the responsible users reminding them that the workflow step must be completed.
4) In the **Expiration Step** attribute, you can optionally define the workflow step that should be the subsequent workflow step if the selected step is not completed by a user in the timeframe defined in the **Performance Duration** attribute. You may only define one workflow step for this attribute. If you do not define an expired workflow step, the workflow administrator will be required to manually redirect the workflow to a subsequent step if the workflow step has expired.
5) Click the **Save** button to save your definitions.

Configuring the Action Buttons Available for a Workflow Step

For each workflows step, you can specify which options are possible for a user to carry out for a workflow step by making the relevant toolbar buttons available in the toolbar. The accessibility of toolbar buttons must be specified for each workflow step where the **Type** attribute is not set to `System`. Buttons that are not made available for the workflow step will be disabled in the toolbar.

The configuration of the buttons can be made on the level of the workflow template and/or workflow step. If the button attributes are defined for the workflow template, the button definition will be applied to all workflow steps unless a button definition is explicitly made for a workflow step, in which case the workflow step definition will take precedence over the workflow template definition. If a workflow step is to inherit the button configuration from the workflow template, you should ensure that the value `Inherited` is selected for the **Visibility** attribute in the respective attribute (**Complete Button**, **Confirm Button**, **Delegate Button**, **Perform Button**, and **Refuse Button**).



The following buttons are available for configuration in the **My Workflow Activities** functionalities (`WF_UserWorkflowActivities`, `WF_UserWorkflowActivitiesExt`, and `WF_UserWorkflowActivitiesCommon`) as well as the **Workflow Activities Explorer** (`WFS_Explorer`).

- **Perform** button: This button opens the configured editor, wizard, view, etc. needed to process the workflow step. A **Perform** button should be available for ALL workflow steps except those workflow steps where the **Type** attribute is set to `System`.
- **Confirm** button: This button allows an explicit confirmation of the workflow step to be made via one-click without requiring a comment to be entered. In this case, the workflow step will immediately advance to the next workflow step if all post-conditions are satisfied. If the workflow step is based on an object view, graphic view, or custom report, the user must have either the **Confirm** or **Complete** button available
- **Complete** button: This button allows an explicit confirmation, refusal, or continuation of the workflow step to be made via the **Workflow Step Completion** editor. The configuration of the **Complete** button makes the **Confirm** button and **Refuse** button superfluous. Furthermore, a **Comment** attribute is available that may be specified as optional or mandatory. If the workflow step is based on an object view, graphic view, or custom report, the user must have either the **Confirm** or **Complete** button available. The option to refuse the workflow step may be removed from the **Workflow Step Completion** editor.

- **Refuse** button: This button allows a workflow step to be refused by a user. The refusal of a workflow step by a single user results in the overall refusal of the workflow step for all responsible users. Once a workflow step has been refused by one user, the workflow step will be removed from the **My Workflow Activities** functionalities and/or the **Workflow Activities Explorer** or all other responsible users.
- **Delegate** button: This button opens the **Delegate Step** editor in which the responsible user or workflow owner delegating the workflow step must provide an explanation about the delegation. When the user delegates a workflow step, he/she can choose to delegate only his/her responsibility for the workflow step or remove the responsibility from all users identified as responsible users. The step will be removed accordingly from the **My Workflow Activities** functionalities and/or the **Workflow Activities Explorer** for all other responsible users

The following issues should be clarified for each workflow step before configuring the button options and confirmation details:

- Once the workflow step task is performed by a user, will the workflow step be automatically confirmed or must the user explicitly confirm the workflow step?
- If the workflow step is to be automatically confirmed, is a comment required by the user before the automatic confirmation can be triggered?
- If the workflow step must be explicitly confirmed, should the confirmation occur via the one-time click of the **Confirm** button in the toolbar or by selecting an option in the **Workflow Step Completion** editor. By configuring the confirmation via the **Confirm** button, the user can click the button and if all post-conditions are satisfied, the workflow will advance to the next workflow step. By configuring the confirmation via the **Workflow Step Completion** editor, the user has the option in the editor to either confirm the workflow step, refuse the workflow step, continue working on the workflow step, and provide a mandatory or optional comment about the workflow step.
- May the user refuse the workflow step?
- May the user delegate the workflow step to another user? All workflow steps in which a responsible user may delegate the workflow step require the **Delegate** button to be visible in the toolbar. An additional query can be defined for the workflow template that specifies the set of users to whom a workflow step may be delegated.
- Who must confirm the workflow step in order for the workflow to advance to the next workflow step? If more than one user is responsible for the workflow step, is it required that only one user, all users, or, if relevant, one user per responsible user group must confirm the workflow step to advance the workflow?
- Which functionalities will be implemented for the user to process the workflow step. If the user will be working in the **My Workflow Activities** functionalities (`WF_UserWorkflowActivities`, `WF_UserWorkflowActivitiesExt`, and `WF_UserWorkflowActivitiesCommon`), then only the visibility of the buttons can be configured (in other words, whether or not the buttons are displayed in the toolbar). If the users are processing the workflow step in the **Workflow Activities Explorer** (`WFS_Explorer`), the captions on the buttons as well as the icon can be customized. This allows the task at hand to be made more explicit for the user. (For example, the **Perform** button may be captioned **Enter Data**, **Review Data**, or **Approve Project**, depending on the task at hand.)

The following information is available:

- Specifying the Confirmation of a Workflow Step of the Type Wizard or Editor
- Automatic Confirmation of a Workflow Step (Types Wizard and Editor)

- Confirmation via the Workflow Step Completion Editor (Types Wizard and Editor)
- One-Click Confirmation via the Confirm Button (Types Wizard and Editor)
- Specifying the Explicit Confirmation of a Workflow Step of the Types Navigate, GraphicView, and Report
- Confirmation via the Workflow Step Completion Editor (Types Navigation, GraphicView, Report)
- One-Click Confirmation via the Confirm Button (Types Navigation, GraphicView, Report)
- Specifying the Users Required to Confirm a Workflow Step
- Confirmation for Authorized Users, Role Responsibles, and Users Found via Queries of Class Person
- Confirmation for User Group Responsibles and Users Found via Queries of Class User Group
- Removing the Option to Refuse a Workflow Step
- Making the Refuse Capability Available for a Workflow Step
- Making the Delegate Capability Available for the Workflow Step
- Defining Permissible Users That a Workflow Step May Be Delegated To
- Ensuring the Automatic Closure of Performed Workflow Steps
- Configuring the Visualization of the Buttons in the Workflow Activities Explorer

Specifying the Confirmation of a Workflow Step of the Type Wizard or Editor

There are many ways to specify the automatic confirmation of a workflow step. The following information is available:

- Automatic Confirmation of a Workflow Step (Types Wizard and Editor)
- Confirmation via the Workflow Step Completion Editor (Types Wizard and Editor)
- One-Click Confirmation via the Confirm Button (Types Wizard and Editor)

Automatic Confirmation of a Workflow Step (Types Wizard and Editor)

If the **Type** attribute for a workflow step is set to `Wizard` or `Editor`, it is possible to configure the workflow step so that it will be automatically confirmed once the workflow step has been performed.

In this case, the user would perform the required task and when the user clicks the **OK** button (in the editor) or the **Finish** button (in the wizard), the workflow step will be automatically confirmed. If all post-conditions are satisfied, the workflow will advance to the next workflow step. Thus, users are not explicitly required to confirm the workflow step via the **Confirm** button or via a selection in the **Workflow Step Completion** editor.

To configure the automatic confirmation of a workflow step:

1) Click the workflow step  to open the attribute window. Expand the **Visualization** section of the grid by clicking the arrow.

2) Expand the **Perform Button** attribute and ensure that the **Visibility** attribute is set to `Visible` so that the user can perform the relevant workflow task.

3) To activate the automatic confirmation capability, select `AutoConfirm` in the **Completion Type** attribute.

The **Visibility** attribute of the **Complete Button** and **Confirm Button** attributes are typically set to `False` in this case.

4) To specify whether a mandatory comment is required before the workflow step can be automatically confirmed, define the **Completion Comment Required** attribute:

- If a mandatory comment should be provided by the user in the **Workflow Step Completion** editor after the workflow step has been performed, select `True` in the **Completion Comment Required** attribute. Only the **Comment** attribute will be editable in the **Workflow Step Completion** editor. The comments will be displayed in the **Active Workflow Steps** view for the workflow.
- If no comment is required after the workflow step has been performed, select `False` in the **Completion Comment Required** attribute. In this case, the workflow will immediately advance to the next workflow step once the editor/wizard is closed and all post-conditions have been satisfied.

5) Additionally, a post-condition can be defined to ensure that the relevant data has been provided before the workflow step can be automatically confirmed. For more information about the configuration of a post-condition for a workflow step, see the section Configuring Post-Conditions for a Workflow Step.

6) Click the **Save**  button to save your definitions.

Confirmation via the Workflow Step Completion Editor (Types Wizard and Editor)

If the **Type** attribute for a workflow step is set to `Wizard` or `Editor`, it is possible to configure the workflow step so that the **Workflow Step Completion** editor automatically opens after the workflow step is performed. In this case, the user would perform the required task and when the user clicks the **OK** button in the wizard or editor, the **Workflow Step Completion** editor will open and the user can either confirm that the workflow step is complete (whereby the workflow will advance to the next workflow step), continue to perform the workflow step, or refuse the workflow step. If the user continues working on the workflow step, he/she would then subsequently click the **Complete** button to reopen the **Workflow Step Completion** editor and confirm the workflow step and, if all post-conditions are satisfied, advance the workflow to the next workflow step.

To configure the refuse capability so that it is disabled in the toolbar and in the **Workflow Step Completion** editor, see the section Removing the Option to Refuse a Workflow Step.

The **Workflow Step Completion** editor cannot be customized.

To configure the explicit confirmation of a workflow step via the **Workflow Step Completion** editor:

1) Click the workflow step  to open the attribute window. Expand the **Visualization** section of the grid by clicking the arrow.

2) Expand the **Perform Button** attribute and ensure that the **Visibility** attribute is set to `Visible` so that the user can perform the relevant workflow task.

3) Expand the **Complete Button** attribute and ensure that the **Visibility** attribute is set to `Visible` so that the user can continue the confirmation process if he/she interrupts it.

The **Visibility** attribute of the **Confirm Button** attribute is typically set to `Hidden` in this case.

4) To activate the confirmation capability, select `ConfirmPrompt` in the **Completion Type** attribute.

5) To specify whether a mandatory comment is required as part of the confirmation process, define the **Completion Comment Required** attribute:

- If a mandatory comment should be provided by the user in the **Workflow Step Completion** editor after the workflow step has been performed, select `True` in the **Completion Comment Required** attribute. Comments will be displayed in the **Active Workflow Steps** view for the workflow.
- If no comment is required after the workflow step has been performed, select False in the **Completion Comment Required** attribute.

6) Click the **Save**  button to save your definitions.

One-Click Confirmation via the Confirm Button (Types Wizard and Editor)

To configure the explicit confirmation of a workflow step via the **Confirm** button:

1) Click the workflow step  to open the attribute window. Expand the **Visualization** section of the grid by clicking the arrow.

2) Expand the **Perform Button** attribute and ensure that the **Visibility** attribute is set to `Visible` so that the user can perform the relevant workflow task.

3) Expand the **Confirm Button** attribute and ensure that the **Visibility** attribute is set to `Visible` in order to allow the user to advance directly to the next workflow step.

Typically, either only the **Confirm Button** button or the **Complete Button** should be available in the toolbar. Therefore, in this case, the **Visibility** attribute of the **Complete Button** should be set to `Hidden`.

4) To specify the confirmation capability, select `NotSpecified` for the **Completion Type** attribute.

5) To ensure that the **Workflow Step Completion** editor is NOT displayed to the user, select `False` for the **Completion Comment Required** attribute.

If the **Completion Comment Required** attribute is set to `True`, the dialog will be displayed upon clicking the **Confirm** button, whereby the option to refuse the workflow step will be displayed. Thus, the user can provide a comment for his confirmation.

6) Click the **Save**  button to save your definitions.

Specifying the Explicit Confirmation of a Workflow Step of the Types Navigate, GraphicView, and Report

It is possible to configure the explicit confirmation of a workflow step of the type `Navigate`, `GraphicView`, and `Report`. If the workflow step is based on an object view, graphic view, or custom report, the user must have either the **Confirm** or **Complete** button available in the **My Workflow Activities** functionalities and/or the **Workflow Activities Explorer** in order to be able to indicate that the workflow step has been performed and is completed. Please consider the following:

- If the user should be able to simply confirm the workflow step and NOT be required to enter a comment, it is recommended that the workflow step is to be confirmed via the **Confirm** button and that the **Workflow Step Completion** editor is not implemented for the workflow step. In this case, the workflow step will immediately advance to the next workflow step if all post-conditions are satisfied.
- If the user should be able to confirm the workflow step and also is required to enter a comment, it is recommended that the workflow step is to be confirmed via the **Complete** button. In this case, the **Workflow Step Completion** editor will open and the user can either confirm that the workflow step is complete (whereby the workflow will advance to the next workflow step), continue to perform the workflow step, or, if permissible for the workflow step, refuse the workflow step.

The following options are available to configure the explicit confirmation of a workflow step in which an object view, graphic view, or custom report is implemented:

- Confirmation via the Workflow Step Completion Editor (Types Navigation, GraphicView, Report)
- One-Click Confirmation via the Confirm Button (Types Navigation, GraphicView, Report)

Confirmation via the Workflow Step Completion Editor (Types Navigation, GraphicView, Report)

To configure the explicit confirmation of a workflow step via the **Workflow Step Completion** editor:

To configure the refuse capability so that it is disabled in the toolbar and the **Workflow Step Completion** editor, see the section Removing the Option to Refuse a Workflow Step.

The **Workflow Step Completion** editor cannot be customized.

1) Click the workflow step  to open the attribute window. Expand the **Visualization** section of the grid by clicking the arrow.
2) Expand the **Perform Button** attribute and ensure that the **Visibility** attribute is set to `Visible` so that the user can perform the relevant workflow task.
3) Expand the **Complete Button** attribute and ensure that the **Visibility** attribute is set to `Visible` so that the user can confirm the workflow step via the **Workflow Step Completion** editor.

   Typically, either only the **Confirm** button or the **Complete** button should be available in the toolbar. Therefore, in this case, the **Visibility** attribute for the **Confirm Button** attribute should be set to `Hidden`.

4) To activate the confirmation capability, select `ConfirmPrompt` in the **Completion Type** attribute.

5) To specify whether a mandatory comment is required as part of the confirmation process, define the **Completion Comment Required** attribute:

- If a mandatory comment should be provided by the user in the **Workflow Step Completion** editor after the workflow step has been performed, select `True` in the **Completion Comment Required** attribute. Comments are displayed in the **Workflow Event Trace** view for the workflow as well as the views available in the **Step-Related Activities** workspace in the object profile for a workflow step.
- If no comment is required after the workflow step has been performed, select `False` in the **Completion Comment Required** attribute.

6) Click the **Save** button to save your definitions.

One-Click Confirmation via the Confirm Button (Types Navigation, GraphicView, Report)

To configure the explicit confirmation of a workflow step via the **Confirm** button:

1) Click the workflow step to open the attribute window. Expand the **Visualization** section of the grid by clicking the arrow.
2) Expand the **Perform Button** attribute and ensure that the **Visibility** attribute is set to `Visible` so that the user can perform the relevant workflow task.
3) Expand the **Confirm Button** attribute and ensure that the **Visibility** attribute is set to `Visible` in order to allow the user to advance directly to the next workflow step.

   Typically, either only the **Confirm Button** button or the **Complete Button** should be available in the toolbar. Therefore, in this case, the **Visibility** attribute of the **Complete Button** should be set to `Hidden`.

4) To specify the confirmation capability, select `NotSpecified` for the **Completion Type** attribute.
5) To ensure that the **Workflow Step Completion** editor is NOT displayed to the user, select `False` for the **Completion Comment Required** attribute.

   If the **Completion Comment Required** attribute is set to `True`, the dialog will be displayed upon clicking the **Confirm** button, whereby the option to refuse the workflow step will be displayed. Thus, the user can provide a comment for his confirmation.

6) Click the **Save** button to save your definitions.

Specifying the Users Required to Confirm a Workflow Step

Depending on the configuration of the users responsible for a workflow step, the set of responsible users may be quite large. In some cases, it may be necessary that all users actually perform and confirm the workflow step whereas in other cases, it may suffice if one single user performs and confirms the workflow step.

If the responsible users are found via their affiliation to a user group, you can stipulate that one user in each user group must perform and confirm the workflow step. In this case, the workflow step will not be

complete until a user in every responsible user groups have confirmed the workflow step. If a user confirms the workflow step and that user is a member of multiple responsible user groups, the confirmation will fulfill the required confirmation for each user group that the user is a member of.

The following information is available:

- Confirmation for Authorized Users, Role Responsibles, and Users Found via Queries of Class Person
- Confirmation for User Group Responsibles and Users Found via Queries of Class User Group

Confirmation for Authorized Users, Role Responsibles, and Users Found via Queries of Class Person

If you have defined the user responsible definition via either the **Responsible User** attribute, **Responsibles by Roles** attribute, or a query based on the object class `Person`, define one of the following to determine who is responsible for confirming the workflow step in order for it to be completed:

- If only one of the users identified to perform this workflow step must confirm the workflow step to complete it, select `SingleUser` for the attribute **Confirmation Type**. The first user who confirms the workflow step completes the workflow step. If all post-conditions are fulfilled for the workflow step, the workflow step will be removed from the **My Workflow Activities** functionalities and/or the **Workflow Activities Explorer** for all responsible users and the workflow will subsequently advance to the next workflow step.
- If all users identified to perform this workflow step must confirm the workflow step to complete it, select `AllUsers` for the attribute **Confirmation Type**. In this case, the workflow step will only be completed once all responsible users have confirmed the workflow step.

Click the **Save**  button to save your definitions.

Confirmation for User Group Responsibles and Users Found via Queries of Class User Group

If you have defined the user responsible definition via either the **Responsibles by User Groups** attribute or a query based on the object class `User Group`, define one of the following to determine who is responsible for confirming the workflow step in order for it to be completed

- If only one member in each user group selected in the **Responsibles by User Groups** attribute must confirm the workflow step to complete, select `SingleUserOfEachUserGroup` for the **Confirmation Type** attribute. In this case, the workflow step will not be complete until a user in all user groups responsible for the workflow step have confirmed the workflow step. If a user confirms the workflow step and that user is a member of multiple responsible user groups, the confirmation will fulfill the required confirmation for each user group that the user is a member of.
- If only one of the users identified to perform this workflow step must confirm the workflow step to complete it, select `SingleUser` for the **Confirmation Type** attribute. The first user who confirms the workflow step completes the workflow step. If all post-conditions are fulfilled for the workflow step, the workflow step will be removed from the **My Workflow Activities** functionalities and/or the **Workflow Activities Explorer** for all responsible users and the workflow will subsequently advance to the next workflow step.

- If all users identified to perform this workflow step must confirm the workflow step to complete it, select `AllUsers` for the **Confirmation Type** attribute. In this case, the workflow step will only be completed once all responsible users have confirmed the workflow step.

Click the **Save** button to save your definitions.

Removing the Option to Refuse a Workflow Step

In some cases, it may be that a responsible user may NOT refuse a workflow step and instead is required to perform the workflow step. In this case, you can configure the refuse capability so that it is unavailable in the toolbar of the **My Workflow Activities** functionalities (`WF_UserWorkflowActivities`, `WF_UserWorkflowActivitiesExt`, and `WF_UserWorkflowActivitiesCommon`) and/or the **Workflow Activities Explorer** as well as in the **Workflow Step Completion** editor.

To ensure that users may not refuse the workflow step:

1) Click the workflow step to open the attribute window.
2) Select `NotSpecified` for the **Completion Type** attribute.
3) Expand the **Visualization** section of the grid by clicking the arrow. Expand the **Refuse Button** attribute and ensure that the **Visibility** attribute is set to `Hidden` to ensure that the refuse capability is not available in the toolbar.
4) Click the **Save** button to save your definitions.

Making the Refuse Capability Available for a Workflow Step

The refusal of a workflow step by a single user results in the overall refusal of the workflow step for all responsible users. Once a workflow step has been refused by one user, the workflow step will be removed from the **My Workflow Activities** functionalities (`WF_UserWorkflowActivities`, `WF_UserWorkflowActivitiesExt`, and `WF_UserWorkflowActivitiesCommon`) and/or the **Workflow Activities Explorer** for all other responsible users.

Please keep the following in mind when configuring the refusal capability:

- If the **Workflow Step Completion** editor is implemented for the workflow step, the refusal capability will be automatically displayed in the editor. The **Refuse** button can nevertheless be configured to be displayed in the toolbar as an additional means to refuse the workflow step.
- If the **Confirm** button or the automatic confirmation capability is implemented in the workflow step, then the **Refuse** button must be made available in the toolbar if users may refuse the workflow step.

Any user refusing a workflow step must provide an explanation in either the **Workflow Step Completion** editor or the **Refuse Step** editor that opens when he/she clicks the **Refuse** button. Comments are displayed in the **Active Workflow Steps** view for the workflow.

1) Click the workflow step to open the attribute window. Expand the **Visualization** section of the grid by clicking the arrow.

2) Expand the **Refuse Button** attribute and ensure that the **Visibility** attribute is set to `Visible` to display the **Refuse** button in the toolbar of the **My Workflow Activities** functionalities and/or the **Workflow Activities Explorer** as well as in the **Workflow Step Completion** editor. The **Refuse** button allows users to refuse a workflow step that they are responsible for. Select `Hidden` if the user should not have the option to refuse the workflow step.

3) In the **Refuse Step** attribute for the selected workflow step, define the workflow step that the workflow should advance to if the selected workflow step is refused. For more information, see the section Defining the Sequence of the Workflow Steps.

4) Click the **Save** button to save your definitions.

If the configuration of the workflow step allows users to refuse a workflow step, you may specify a property update, email notification, or workflow trigger for an **Action on Refused Step** for the workflow step. For more information, see the following sections:

- Configuring Automatic Property Updates for a Workflow Step
- Configuring Email Notifications for Workflows
- Configuring a Workflow Step to Trigger a Subordinate Workflow

Making the Delegate Capability Available for the Workflow Step

In some cases, you may decide that the responsible users or workflow owner may delegate their workflow step to another user whom they consider more appropriate. For this purpose, the **Delegate** button must be displayed in the toolbar of the **My Workflow Activities** functionalities (`WF_UserWorkflowActivities`, `WF_UserWorkflowActivitiesExt`, and `WF_UserWorkflowActivitiesCommon`) and/or the **Workflow Activities Explorer** in order to enable the user or workflow owner to delegate a workflow step to another user.

A responsible user or workflow owner delegating a workflow step must provide an explanation in the **Delegate Step** editor that opens when he/she clicks the **Delegate** button. When the user delegates a workflow step, he/she can choose to delegate only his/her responsibility for the workflow step or remove the responsibility from all users identified as responsible users. The step will be removed accordingly from the **My Workflow Activities** functionalities and/or the **Workflow Activities Explorer** for all other responsible users.

If the **Delegate** button is not implemented for the workflow step, then the workflow owner and responsible users will not be able to delegate the workflow step to another user.

Please note that if the workflow administrator must have access to the **Delegate** button, he/she must be specified as one of the responsible users for the workflow step. Regardless of whether the workflow administrator has access to the **Delegate** button or not, he/she will still be able to reassign workflows and workflow steps for users that have left the organization in the **Workflow Administration** functionality.

Typically, when the user clicks the **Delegate** button, the standard person selector opens displaying all named users in Alfabet. In order to limit the set of users displayed in the person selector, you can define a query for the associated workflow template in the **Step Delegate Rule** attribute of the workflow template. This option is described below in the section Defining Permissible Users That a Workflow Step May Be Delegated To.

Please note that only users that are qualified named users in Alfabet may be delegated workflow steps. If an external source selector (for example, LDAP) is configured as the selector for the object class `Person`, the configuration will be ignored and the standard person selector will be implemented.

1) Click the workflow step to open the attribute window. Expand the **Visualization** section of the grid by clicking the arrow.
2) Expand the **Delegate Button** attribute and ensure that the **Visibility** attribute is set to `Visible` to display the **Delegate** button in the toolbar of the **My Workflow Activities** functionalities and/or the **Workflow Activities Explorer** as well as the **Active Workflow Steps** view of the relevant workflow. Select `Hidden` if the button should not be disabled in these views.

   Typically, when the user clicks the **Delegate** button, the standard person selector opens displaying all named users in Alfabet. In order to limit the set of users displayed in the person selector, you can define a query for the associated workflow template in the **Step Delegate Rule** attribute of the workflow template. For more information, see the section Defining Permissible Users That a Workflow Step May Be Delegated To.

3) Click the **Save** button to save your definitions.

Defining Permissible Users That a Workflow Step May Be Delegated To

Typically, when the responsible user or workflow owner delegates a workflow step, the standard person selector opens displaying all named users in Alfabet. In order to limit the set of users displayed in the person selector, you can define a query for the workflow template that the workflow step belongs to.

The rule you define applies to all relevant workflow steps generated in the context of the selected workflow template. When a user clicks the **Delegate** button in the **My Workflow Activities** functionalities (`WF_UserWorkflowActivities`, `WF_UserWorkflowActivitiesExt`, and `WF_UserWorkflowActivitiesCommon`) and/or the **Workflow Activities Explorer**, for example, a person selector will open displaying the names of all users found by the query.

For example, to configure a query find only users for which the custom object class property `CustomType` is set to `Workflow`:

```
Person.CustomType="Workflow"
```

All users found via the `WHERE` clause of the query are permissible to be the recipient of a workflow step delegation and will be displayed in the person selector.

For general information about defining an Alfabet query or SQL query, see the chapter Defining Queries.

To define a set of users to whom a workflow step generated in the context of the selected workflow template can be delegated to:

1) Click the workflow template to open the attribute window.
2) In the **Step Delegation Rule** attribute, enter an Alfabet query or an SQL query. The query should be defined like a `WHERE` clause.

3) Click the **Save** button to save your definitions.

Ensuring the Automatic Closure of Performed Workflow Steps

It is possible to specify that a configured batch process should automatically close any workflow steps that are unconfirmed but for which all post-conditions have been satisfied.

In this case, a batch job must be configured by the system administrator. The batch job `alfaWorkflowCommandPrompt.exe` must include the runtime option `StepClosure`. For more information about the configuration of the batch job, see the section *Batch Processes for Workflows with AlfaWorkflowCommandPrompt.exe* in the reference manual *System Administration*.

1) Click the workflow step to open the attribute window.
2) In the **Allow Automatic Closure** attribute, select `True`. Any unconfirmed workflow steps will be checked when the batch process is triggered, and any workflow steps whose post-conditions are fulfilled will automatically be closed and the workflow will advance to the next workflow step.
3) Click the **Save** button to save your definitions.

Configuring the Visualization of the Buttons in the Workflow Activities Explorer

To specify which button should be displayed (and thus which actions are possible for a workflow step) in the **My Workflow Activities** functionalities (`WF_UserWorkflowActivities`, `WF_UserWorkflowActivitiesExt`, and `WF_UserWorkflowActivitiesCommon`), you must define the attributes **Complete Button**, **Confirm Button**, **Delegate Button**, **Perform Button**, and **Refuse Button** in the **Visualization** section of the attribute window.

If these attributes are defined for the workflow template, the button definition will be applied to all workflow steps unless a button definition is explicitly made for a workflow step, in which case the workflow step definition will take precedence over the workflow template definition. For each button, a further table section can be opened displaying the attributes **Visibility**, **Caption**, and **Icon**.

Define as explained below for the attributes **Complete Button**, **Confirm Button**, **Delegate Button**, **Perform Button**, and **Refuse Button**.

1) Click the workflow step to open the attribute window. Expand the **Visualization** section of the grid by clicking the arrow.
2) Expand the grid for the relevant button by clicking the arrow.
3) In the **Visibility** attribute:

- Select `False` if the button should not be available for the respective workflow step (or workflow template).
- Select `True` if the button should be available for the respective workflow step (or workflow template).
- If you select `True` for the **Visibility** attribute, define the following attributes:
- **Caption**: Enter a caption to display on the button

- **Icon**: Select an image from a preconfigured button library to display on the button.

4) Click the **Save**  button to save your definitions.

Configuring Pre- and Post-Conditions for a Workflow Step

For each workflow step, you may define one or more pre-conditions and post-conditions that must be fulfilled before the workflow can enter or exit the workflow step. A workflow step's pre-conditions should define the minimum conditions that should apply for the workflow step to be entered and a workflow step's post-conditions should define the minimum conditions that should apply for the workflow step to be exited.

- The following example describes a scenario for a pre-condition:

A workflow step A requires a proposed project's cost to be estimated. The next workflow steps of workflow step A are workflow step B and C.

Workflow step B has a precondition that all project proposals must be over € 25k to advance to workflow step B. Workflow step C has a pre-condition that all project proposals must be under or equal to € 25k to advance to workflow step C. Therefore, if the estimated cost is over € 25k, the next workflow step will be workflow step B, but if the estimated cost is under or equal to € 25k, the next workflow step will be workflow step C.

- The following example describes a scenario for a post-condition:

A workflow step A requires that a proposed project is reviewed and that specific data is entered. A post-condition requires that the release status of the project proposal is set to Scope Reviewed. The release status must therefore be set to Scope Reviewed by a responsible user of the workflow step (via an editor or wizard) before workflow step A can be confirmed and workflow step B may begin.

The definition of pre-conditions and post-conditions is generally handled in the same way. Both types of conditions require an Alfabet query or SQL query to be defined that checks whether the condition has been fulfilled.

The following information is available:

- Configuring Pre-Conditions for a Workflow Step
- Configuring Post-Conditions for a Workflow Step

Configuring Pre-Conditions for a Workflow Step

A pre-condition is based on an Alfabet query or an SQL query that checks whether a workflow step may or may not be executed. Each workflow step may have multiple pre-conditions defined. All pre-conditions must be fulfilled before the workflow step may be started. Pre-conditions are required, for example, when a workflow step has multiple next workflow steps defined. Each next workflow step must have one or more pre-conditions defined in order to determine which of the potential next workflow steps should be the subsequent next workflow step.

Pre-conditions configured for workflow steps will be evaluated prior to the base object query.

A pre-condition can be created based on a copy of an existing pre-condition. To do so, create a new pre-condition without defining any attributes. Select the pre-condition you want to copy and right click and select **Copy**. Click the new pre-condition and select **Paste**. The attributes are filled as defined and can be modified, as needed.

To define a pre-condition for the selected workflow step:

1) Click the + next to the workflow step  to expand the workflow step node.
2) Right-click **Pre-Conditions** and select **Add New Pre-Condition** to create a pre-condition.
3) Click the pre-condition  to display the attribute window.
4) **If you are defining pre-conditions for the first workflow step**: Define the **Message** attribute explaining what must be done to fulfill the pre-condition. If the pre-condition is not fulfilled and an error occurs during the initialization of the workflow, the message text will be displayed in an error dialog and should provide users with an explanation about how the pre-condition must be fulfilled in order to complete the initialization. The error message will be displayed as follows:

   An error has occurred during the initialization of the workflow'<`WorkflowTemplate.Caption`>'. Reason: <`WorkflowStep.PreCondition.Message`>.
5) The error message will display the following text:
6) Define the query that determines the check that is executed to ascertain whether the pre-condition has been fulfilled. To define an Alfabet query, click the **Browse**  button in the **Query** attribute to open the **Alfabet Query Builder** or paste an SQL query in the **Query as Text** attribute. For information about defining an Alfabet query, see the chapter Defining Queries.

   You must ensure that the pre-conditions are defined so that only one workflow step fulfills the criteria to become the next workflow step. If multiple workflow steps fulfill the criteria, an error will occur in the workflow.
7) In the **Result Type** attribute, select either `Positive` or `Negative` to define what constitutes a fulfilled condition.

- `Positive` means that the condition is fulfilled if the Alfabet query has delivered a result (at least one row.)
- `Negative` means that the condition is fulfilled if the Alfabet query doesn't deliver any results.

8) Click the **Save**  button to save your definitions.
9) To specify the sequence that the pre-conditions should be executed for the workflow step, click the relevant workflow step and order the conditions in the **Pre-Conditions** attribute. Click the **Save**  button.

Configuring Post-Conditions for a Workflow Step

A post-condition is based on a query that checks whether a workflow step has been completed. Each workflow step may have multiple post-conditions defined. All post-conditions must be fulfilled in order to confirm and complete a workflow step and advance to the subsequent workflow step.

A post-condition can be created based on a copy of an existing post-condition. To do so, create a new post-condition without defining any attributes. Select the post-condition you want to copy and right click and select **Copy**. Click the new post-condition and select **Paste**. The attributes are filled as defined and can be modified, as needed.

1) Click the + next to the workflow step to expand the workflow step node.
2) Right-click **Post-Conditions** and select **Add New Post- Condition** to create a post-condition.
3) Click the post-condition to display the attribute window.
4) Define the **Message** attribute explaining what must be done to fulfill the post-condition. The message text will be displayed in an error dialog and should provide users with an explanation about how the post-condition must be fulfilled in order to complete the workflow step.
5) Define the query that determines the check that is executed to ascertain whether the post-condition has been fulfilled. To define an Alfabet query, click the **Browse** button to the **Query** attribute open the **Alfabet Query Builder** or paste an SQL query in the **Query as Text** attribute. For information about defining an Alfabet query or SQL query, see the chapter Defining Queries.
6) In the **Result Type** attribute, select either `Positive` or `Negative` to define what constitutes a fulfilled condition.

- `Positive` means that the condition is fulfilled if the Alfabet query has delivered a result (at least one row.)
- `Negative` means that the condition is fulfilled if the Alfabet query doesn't deliver any results.

7) If the workflow step should be automatically closed once all post-conditions have been satisfied, click the workflow step to open its attribute window. In the **Allow Automatic Closure** attribute, `True` if the workflow step should be automatically closed and advance to the next workflow step via a batch process if all post-conditions have been satisfied. The user responsible for the workflow step does not need to manually confirm the workflow step in this case. Select `False` if the workflow step must be manually confirmed by the responsible user.

- The **Allow Automatic Closure** attribute must be set to `True` if the **Check Post Conditions and Close Activity** option should be enabled in the **Workflow Administration** functionality. If the **Allow Automatic Closure** attribute is set to `False`, the workflow administrator will not be able to check post-conditions and close a workflow step. For more information, see the section *Checking a Workflow Step's Post-Conditions and Closing the Workflow Step* in the reference manual *User and Solution Administration*.
- If the **Allow Automatic Closure** attribute is set to `True`, the batch job `alfaWorkflowCommandPrompt.exe` must include the runtime option `StepClosure`. For form information about the configuration of the batch job, see the section *Batch Processes for Workflows with AlfaWorkflowCommandPrompt.exe* in the reference manual *System Administration*.

8) Click the **Save** button to save your definitions.

9) To specify the sequence that the post-conditions should be executed for the workflow step, click the relevant workflow step and order the conditions in the **Post-Conditions** attribute. Click the **Save**  button.

Configuring Automatic Property Updates for a Workflow Step

You can configure various mechanisms to update property values when a workflow step is entered, exited, refused, or expired. The update to a property value can be configured for the object that is targeted by the workflow step as well as an object referenced by it.

The following workflow step actions targeting the update of a scalar property can be configured when a workflow step is entered, exited, refused, or expired:

- A workflow step action of the type `Script` can be created and statements using the commands `SET`, `CLEAR`, `ADDRELATIONS`, and `REMOVERELATIONS` can be defined.
- One or more values can be removed from a object class property. For example, you could configure that a specified object class property is cleared if a workflow step expires.
- One or more values can be entered for a object class property. For example, you could configure that a status value is automatically set when a workflow step is entered.
- a combination of multiple set and clear workflow step actions is possible.
- A workflow step action of the type `SQL` can be created and the commands `UPDATE` and `INSERT` can be defined.

The following information is available:

- Define a Workflow Step Action of the Type Script
- Defining a Workflow Step Action of the Type SQL

Define a Workflow Step Action of the Type Script

A workflow step action of the type `Script` can be created and statements using the commands `SET`, `CLEAR`, `ADDRELATIONS`, and `REMOVERELATIONS` can be defined. Please note the following regarding the syntax of the statements:

- Scalar properties and reference properties may be updated. An object class property can be defined via multiple relationships.
- Enter one statement for each updating workflow step action. If multiple statements are required, a separate workflow step action must be created for each statement.
- Each statement should begin with the command for the workflow step action. Permissible commands are `SET`, `CLEAR`, `ADDRELATIONS`, and `REMOVERELATIONS`. The correct syntax for a statement is: `INSTRUCTION(<Parameter>);`
- The notation for the base object of the workflow is `@WORKFLFOWBASE` and the notation for the object targeted by a workflow step is `@BASE`.

- The following parameters may be used in the statement:
- To specify the target object class property to be set: `@BASE`, `@WORKFLOW`, `@WORKFLOWBASE`, `@CURRENT_STEP`, `@PREVIOUS_STEP`, `@SOURCE`, `@CURRENT_USER`
- To specify the source object class property: `@BASE`, `@WORKFLOW`, `@WORKFLOWBASE`, `@CURRENT_STEP`, `@PREVIOUS_STEP`, `@SOURCE`, `@CURRENT_USER`, `@CURRENT_MANDATE`, `@CURRENT_PROFILE`, `@TODAY`.
- Dot notation may be implemented to concatenate multiple references in one statement. For example: `<Base.Ref1.Ref2....Target>`.
- Each statement must be terminated by a semi-colon.
- There is no mechanism to verify the syntax of the statement. If an error occurs, it will be written to the **Workflow Event Trace** page view for the workflow in Alfabet.
- There is no mechanism to verify the correctness of the command. If an error occurs (for example, because the source or target reference cannot be found for the `Set` command), the error will be written to the **Workflow Event Trace** page view for the workflow in Alfabet. For more information, see the section *Suspending, Resuming, or Withdrawing the Workflow* in the reference manual *Configuring Alfabet with Alfabet Expand*.

For example:

If the name of the base object of the workflow is to be set via the name of the component that the base object references, the statement would be:

```
SET(@WORKFLOWBASE.Name,@WORKFLOWBASE.Component.Name);
```

If the responsible user of the base object of the workflow is to be set via the responsible user of the component that the base object references, the statement would be:

```
SET(@WORKFLOWBASE.ResponsibleUser,@WORKFLOWBASE.Component.Responsible
User);
```

If the name of the object targeted by the workflow step is to be set to a specified value, the statement would be:

```
SET(@BASE.Name,"SpecifiedValue");
```

If the responsible user of the object targeted by the workflow step is to be removed, the statement would be:

```
CLEAR(@CURRENT_STEP.ResponsibleUser);
```

If a newly created application that is the base object of the workflow should be added to an application group that is the base object of the current workflow step, the statement would be:

```
ADDRELATIONS(@BASE.Applications,@WORKFLOWBASE);
```

To define a property update for a workflow step:

1) Depending on when the property update should occur, right-click one of the following workflow step actions:

- Right-click **Action On Entered Step** and select **Add New Action On Entered Step** in order to configure a property update that should occur when the workflow step is entered.

Please note the following if an **Action On Entered Step** workflow step action is configured for the first workflow step in a workflow template:

The **Action On Entered Step** workflow step action will be executed before the configured editor/wizard opens for the first workflow step. Therefore, if an **Action On Entered Step** workflow step action for the first workflow step is executed at workflow initiation and the workflow is then cancelled, the **Action On Entered Step** workflow step action will have already been executed. The updating workflow step action will not be automatically reversed with the cancellation of the workflow.

- Right-click **Action On Refused Step** and select **New Action on Refused Step** in order to configure a property update that should occur when the workflow step is refused.
- Right-click **Action On Expired Step** and select **New Action on Expired Step** in order to configure a property update that should occur when the workflow step is expired.
- Right-click **Action On Exited Step** and select **New Action on Exited Step** in order to configure a property update that should occur when the workflow step is exited.

2) Click the action to open the attribute window.
3) In the **Technical Name** attribute, enter a name for the workflow step action. This name is not displayed in the Alfabet interface.
4) In the **Type** attribute, select `Script`.
5) In the **Statements** attribute, click the **Browse** button to open the **Statements** editor. Enter one or more statements for the updating workflow step action.
6) Click the **Save** button to save your definitions.

Defining a Workflow Step Action of the Type SQL

A property update via a native SQL statement allows an `UPDATE` or `INSERT` statement to be specified. The SQL statement may be configured for the nodes **Action On Entered Step**, **Action On Expired Step**, **Action On Refused Step**, and **Action On Exited Step**.

Please note that the `INSERT` statement is only allowed to insert new relation entries in the `RELATIONS` table. If you attempt to insert new relation entries in class tables, an error will occur because the `GUID` and `REFSTR` are not automatically created and result in a `NULL` value.

To define a workflow step action of the type `SQL`:

1) Expand the relevant workflow step to display the workflow step actions.
2) Right-click the relevant workflow step action and select either **Action On Entered Step**, **Action On Expired Step**, **Action On Refused Step**, and **Action On Exited Step**.
3) Click the action to open the attribute window.
4) In the **Technical Name** attribute, enter a name for the workflow step action. This name is not displayed in the Alfabet interface.
5) In the **Type** attribute, select `SQL`.

6) In the **Statements** attribute, click the **Browse** button to open the **Statements** editor. Enter one or more statements for the updating workflow step action.

The following examples display the correct syntax to use for statements in this context:

```
UPDATE ICTOBJECT
SET DESCRIPTION='ICTO Description' WHERE ICTOBJECT.REFSTR
=@BASE
```

7) Click the **Save** button to save your definitions.

Configuring Events to Be Triggered for a Workflow Step

You can configure that an event is triggered when a workflow step is entered, exited, refused, or expired. Events allow for real-time automated updates to external systems as well as to data in Alfabet. Events allow, for example, a configured action to be triggered when a user closes the workflow window in the middle of the workflow process. As an alternative to batch jobs which may typically be initiated daily, events can be configured for workflow steps if a specified event occurs in the workflow step, a configured ADIF scheme will be triggered to process the data.

Additional configuration is required in order to trigger events. For more information about the configuration required to implement the event capability, see the chapter Configuring Events.

To configure the workflow step to trigger an event:

1) In Alfabet Expand, navigate to the workflow step that you want the event to be triggered from.
2) Go to the **Presentations** tab, expand the custom workflow and expand the relevant workflow step.
3) Right-click the node of the relevant workflow step action that shall trigger the event and select **Action On Entered Step**, **Action On Expired Step**, **Action On Refused Step**, and **Action On Exited Step**.
4) Click the action to open the attribute window and define the following attributes:

- **Technical Name**. Enter a name for the workflow step action. This name is not displayed in the Alfabet interface.
- **Action Type**: Select `TriggerEvent`.
- **Technical Comment**: Enter any relevant information about the workflow step action.
- **Event Template**: Select the relevant event template from the drop-down list.

5) Click the **Save** button to save your definitions.

Configuring a Workflow Step to Trigger a Subordinate Workflow

Some workflow template configurations will require that sub-workflows are triggered when a particular event occurs (for example, a workflow step is refused) or as the result of data input (for example, a specified value is defined for an object).

In this case, it is possible to configure a workflow step on the base workflow template so that the workflow step references a second workflow template. Depending on the circumstance that will trigger the sub-workflows, a workflow step action of the type `TriggerWorkflow` must be defined on the relevant workflow step action (**Action On Entered Step**, **Action On Expired Step**, **Action On Refused Step**, and **Action On Exited Step**). The following are examples of typical scenarios where triggered sub-workflows might be implemented.

- Sub-workflows can be triggered based on a workflow step action. For example, it may be necessary to trigger sub-workflows if a workflow step is refused by a responsible user. In this case, the `TriggerWorkflow` option must be configured for the workflow step action **Action On Refused Step**. The `TriggerWorkflow` option allows you to stipulate the workflow template that should be triggered to generate sub-workflows if the workflow step is refused.
- Sub-workflows can also be triggered based on whether pre-conditions or post-conditions configured for a workflow step action are/are not fulfilled. If the pre-condition specifies, for example, that the target object of the workflow step must have a specified status, then the workflow template would be triggered if the pre-condition is fulfilled. If the pre-condition is not fulfilled, the workflow step would not be entered and the workflow template would not be triggered. In this case, the workflow would advance to the next workflow step for which the pre-conditions are fulfilled.

A query must be configured to find the objects for which workflows must be generated. If multiple objects are found, a sub-workflow will be triggered for each workflow. The workflow owner of the workflows generated for the triggered workflow template will be the same as the owner of the base workflow template that the triggered workflow template is assigned to.

As part of the configuration, you must also decide how to proceed to the next workflow step of the base workflow template after the sub-workflows are triggered and running. Should the next workflow step of the workflow be entered when all sub-workflows are complete or does it suffice that only one sub-workflow must be complete before the next workflow step can be entered.

Finally, you need to consider what should happen if a sub-workflow must be stopped for some reason. If sub-workflows have been triggered for a workflow step of the base workflow, it may be necessary to specify that the sub-workflows can be stopped under certain circumstances. For example, if multiple sub-workflows initiate an approval process that is carried out by various individuals, and during the course of one of the sub-workflows a user determines that circumstances have changed drastically and the object targeted by the approval process warrants no further consideration, then one or all sub-workflows should be stopped.

The following information is available:

- Defining a Sub-Workflow To Be Triggered
- Defining the Re-Evaluation of Active Workflow Steps in Sub-Workflows
- Defining the Cancellation of Triggered Sub-Workflows
- Understanding Where Sub-Workflows are Implemented

Defining a Sub-Workflow To Be Triggered

The following should be completed before you configure a workflow step action of the type `TriggerWorkflow`:

For the parent workflow template:

- Create and configure the selected workflow step and the subsequent workflow step.
- If pre- or post-conditions are required to determine whether sub-workflows should be triggered, configure the pre-condition or post-condition for the selected workflow step. A query must be configured for each condition. For more information about the configuration of conditions, see the section Configuring Pre- and Post-Conditions for a Workflow Step.
- A query must be configured for the workflow step action when the **Type** attribute is set to `TriggerWorkflow` in order to find the objects for which workflows must be generated. If multiple objects are found, a sub-workflow will be triggered for each object.
- If the query is configured to find multiple objects so that multiple sub-workflows will be triggered, consider what conditions must be fulfilled before the subsequent workflow step can begin. Must all sub-workflows be complete before the next workflow step may begin? Or must only one sub-workflow be complete before the next workflow step may begin? The information is defined in the **Synchronize Triggered Workflows** attribute on the workflow step.

For the sub-workflow template: Create and configure the entire workflow template. Sub-workflows will be generated for the base object class identified in the **Start Base Class** attribute of the sub-workflow template. If the **Source Base Class** attribute is configured for the sub-workflow template, the sub-workflows will be generated for the source object class. The workflow template for sub-workflows is configured as for a conventional workflow template. For more information about configuring the objects targeted by a workflow template, see the section Specifying the Objects Targeted by the Workflows.

To define a workflow step action of the type `TriggerWorkflow`:

1) On the relevant workflow step of the base workflow template, right-click one of the following workflow step actions based on when the workflow trigger should occur:

- Right-click **Action On Entered Step** and select **New Action On Entered Step** in order to specify a workflow that should be triggered when the workflow step is entered.
- Right-click **Action On Refused Step** and select **New Action On Refused Step** in order to specify a workflow that should be triggered when the workflow step is refused.
- Right-click **Action On Expired Step** and select **New Action On Expired Step** in order to specify a workflow that should be triggered when the workflow step is expired.
- Right-click **Action On Exited Step** and select **New Action On Exited Step** in order to specify a workflow that should be triggered when the workflow step is exited.

2) Click the action  to open the attribute window.

3) In the **Technical Name** attribute, enter a name for the workflow step action. This name is not displayed in the Alfabet interface.

4) In the **Type** attribute, select `TriggerWokflow`.

5) In the **Workflow Template** attribute, select the workflow template that should be triggered for the action.

The **Name** attribute of the workflow template should NOT be changed if it is already referenced as a sub-workflow (once it is selected in the **Workflow Template** attribute of a workflow step action of the **Type** `TriggerWorkflow`). If the workflow template name is changed, the sub-workflows may not be correctly triggered.

6) You must now configure an Alfabet query or SQL query that finds the objects for which workflows should be generated when the workflow template is initialized. Enter either an SQL query or Alfabet query in the **Query as Text** attribute, or click the **Browse** button in the **Query** attribute, to open the **Alfabet Query Builder**.

If the **Source Base Class** attribute is defined, the source object class will be automatically be specified in the Alfabet query. For information about defining Alfabet queries and SQL queries, see the chapter Defining Queries.

7) Next, you must specify what conditions must be fulfilled before the subsequent workflow step can begin. In the **Synchronize Triggered Workflows** attribute on the workflow step of the base workflow template, select one of the following:

- Select `NoWait` if the next workflow step can be entered as soon as the workflow step is finished. This essentially represents a "fire & forget" scenario. The sub-workflows are triggered and are executed independent of the parent workflow triggering them. This is most applicable in situations where the parent process is merely used to determine which objects a sub-process should be triggered for.
- Select `WaitForFirstFinished` if the next workflow step can be entered as soon as the first sub-workflow is finished. This essentially represents a "winner takes all" scenario where the parent workflow triggers a number of sub-workflows and will resume activity as soon as the first of the sub-workflows has finished. The first response will be used to determine how to proceed in the parent workflow. This approach is best used in a situation where it is important to take a decision quickly rather than waiting indefinitely for a good decision.
- Select `WaitForAllCompleted` if the next workflow step can only be entered once all sub-workflows are finished. This essentially represents a "quorum" scenario where the parent workflow will wait for all opinions to be returned and derive the decision to proceed with the workflow based on all the results received. This approach is best suited for any type of majority vote or proportional vote decision-making scenario that is typically the case when deciding about introducing new technologies, launching new services, or introducing new policies that need to be endorsed by a number of decision bodies.

8) Click the **Save** button to save your definitions.

Defining the Re-Evaluation of Active Workflow Steps in Sub-Workflows

In some cases, it may be necessary to trigger a re-evaluation of active workflow steps for sub-workflows if, for example, they are only valid while a specified workflow step of the parent workflow is active.

In this case, a workflow step action of the **Type** `AutoClosureSubworkflowSteps` may be defined for the relevant workflow step of the parent workflow. If the workflow step of the parent workflow is confirmed, the re-evaluation of all active workflow steps of all sub-workflows will be triggered. If the **Allow Automatic Closure** attribute has been set to `True` for an active workflow step of a sub-workflow and all post-conditions have been met for that workflow step, the active workflow step will be completed and the sub-workflow will proceed to the next workflow step.

Defining the Cancellation of Triggered Sub-Workflows

If sub-workflows have been triggered for a workflow step of the base workflow, it may be necessary to specify that the sub-workflows can be stopped under certain circumstances.

Typically, the first workflow step in the triggered workflow template will have a pre-condition specifying that if a condition is met (for example, a specific value is defined for a object class property), then the workflow will proceed to a next workflow step that serves to cancel the workflow. In this case, you must create a workflow step that will stop the sub-workflow.

Furthermore, you must specify how the parent workflow template should respond if a sub-workflow is stopped. Should the stopped sub-workflow be ignored so that the other active sub-workflows continue to run their course until completion? Or should all running sub-workflows be immediately stopped?

Finally, what should then occur in the base workflow template after sub-workflows have been stopped. This must be configured in a next workflow step defined to follow the workflow step that triggered the sub-workflows.

To define the cancellation of triggered sub-workflows:

1) On the workflow template specifying the workflow steps of the triggered sub-workflows, create a new workflow step.
2) Right-click **Action On Entered Step** and select **Add Action On Entered Step** in order to specify a workflow step to stop the sub-workflow.
3) Click the action  to open the attribute window.
4) In the **Technical Name** attribute, enter a name for the workflow step action. This name is not displayed in the Alfabet interface.
5) In the **Technical Comment** attribute, enter an explanation for the abortion.
6) In the **Type** attribute, select `StopWorkflow`.
7) On the workflow step of the base workflow template configured to trigger the sub-workflows, specify what should occur to the running sub-workflows. In the **Sub-Workflow Stop Propagation** attribute on the workflow step, select one of the following:

- Select `NoStop` if no running sub-workflows should be aborted.
- Select `StopAll` if all running sub-workflows should be aborted if one parallel sub-workflow is aborted.

For more in formation about the configuration of sub-workflows, see the section Configuring a Workflow Step to Trigger a Subordinate Workflow.

8) Click the **Save**  button to save your definitions.

Understanding Where Sub-Workflows are Implemented

To understand which workflow template has implemented a selected workflow template, right-click the workflow template and select **Show Usage**. The **Objects Usage** dialog opens. All workflow templates in which the selected workflow template is implemented as a sub-workflow are displayed in the **Using Path** column. Click **Close** to close the dialog.

Defining the Sequence of the Workflow Steps

Once some of the workflow steps have been created, you can begin to define the sequence of the workflow steps. When you define the sequence of workflow steps for the workflow, you must define all possible subsequent workflow steps for each workflow step.

Furthermore, you should specify which workflow steps are to follow a workflow step that is refused by a user or a workflow step that has not been completed by its configured deadline.

The possible next workflow steps will depend on whether pre-conditions that have been defined for any subsequent workflow step are fulfilled for the object that is the target of the workflow step. Therefore, you may opt to define pre-conditions for each workflow step before you define the sequence of workflow steps. For more information, see Configuring Pre- and Post-Conditions for a Workflow Step.

The **Next Steps** attribute should remain empty for the last workflow step(s) in the workflow.

You can define the order of how the workflow steps should be structured below the workflow template node via the **Steps** attribute on the workflow template. This is only a visual structuring of the workflow steps. The actual sequence of workflow steps as they should occur in a running workflow is described below.

To define the sequence of the workflow steps:

1) Click a workflow template  to open its attribute window.

2) First, you must specify which workflow step is the first workflow step for the workflow template:

- If the workflow template has been configured to target existing objects when workflows are started, select the first workflow step in the **Start Step for Existing Object** attribute for the workflow template.
- If the workflow template has been configured to target new objects when workflows are started, select the first workflow step in the **Start Step for New Object** attribute for the workflow template.

If you have not already done so, you may define a caption for the initiation of the workflow template that will be displayed in the **My Workflows** functionality and the **Initiate Workflows** functionality. This allows you to define basic instruction to the user (for example, to indicate whether they will start the workflow with a new or existing object). If you do not define a caption for the initiation, the caption of the workflow template defined in the **Caption** attribute will be displayed in the **My Workflows** functionality and the **Initiate Workflows** functionality.

- If you have defined the **Start Step for Existing Object** attribute, enter a caption in the **Caption for Existing Object** attribute.
- If you have defined the **Start Step for New Object** attribute, enter a caption in the **Caption for New Object** attribute.

Next, you must define the sequencing of workflow steps for all workflows. For each workflow step, carry out the following.

3) Select the workflow step  to open its attribute window. In the **Next Step** s attribute, click in the cell and select all potential workflow steps that may follow the initial step.

4) If you have defined multiple workflow steps in the **Next Steps** attribute and would like to allow multiple workflow steps that fulfill their pre-conditions to be executed at the same time in parallel to one another, you must ensure that `True` is selected in the **Allow Simultaneous Execution of Steps** attribute of the workflow template. If workflow steps may not be simultaneously executed, you must ensure that `False` is selected in the **Allow Simultaneous Execution of Steps** attribute of the workflow template.

If multiple workflow steps are defined in the **Next Steps** attribute for a workflow step and the pre-conditions are fulfilled for more than one of these workflow steps, the following scenarios are possible:

- If the **Allow Simultaneous Execution of Steps** attribute is set to `False`, an error message will occur. In other words, you must ensure that only one workflow step listed in the **Next Steps** attribute of each workflow step can have its pre-conditions fulfilled and be entered.
- If the **Allow Simultaneous Execution of Steps** attribute is set to `True`, then all workflow steps in the **Next Steps** attribute that fulfill their pre-conditions will be executed in parallel. The subsequent behavior of the workflow will depend on how the **Next Steps** attributes are defined for the parallel workflow steps. There are two alternatives for the configuration of the subsequent workflow steps:
  - All parallel running workflow steps will merge directly or indirectly into the same subsequent workflow step (defined in the **Next Steps** attribute of the merging workflow steps). In this case, the subsequent workflow step will only be entered when all preceding parallel-running steps have been completed.
  - Each parallel running workflow step forms its own independent branch with subsequent workflow steps. The branches do not merge into a common workflow step. The workflow will be completed only when ALL branches reach their last workflow step. **Please note that this workflow design is not recommended. Best practice recommendation for designing workflows with parallel branches is to merge all those branches back into a single lane of execution prior to ending the workflow.**

Please note that if a workflow includes parallel workflow steps and one of the branches forms a cycle that returns to the workflow step where the branching occurred, the availability of that workflow step will be checked prior to recreating the workflow step. If the workflow step is still active, it will not be recreated.

In Alfabet, the workflow step status `Entering` indicates that a workflow step has several parallel workflow steps proceeding it. If one of the parallel branches has been completed, the workflow step that the branches merge into will acquire the status `Entering`. Once all relevant parallel branches have been completed, the workflow step where the branches merge will acquire the status `Pending`. Email notifications will then be sent, if relevant, and the workflow step can be performed by the responsible users.

5) For each workflow step  (including the first workflow step), you may optionally define the **Expiration Step** attribute in order to specify that a specific workflow step should follow the selected step if it surpasses its deadline. In the **Expiration Step** attribute, define the workflow step that should be the next workflow step if the selected step is not completed by a user in the timeframe defined in the **Performance Duration** attribute. You may only define one workflow step for this attribute.

If no workflow step is defined and the workflow step expires, an error will occur in the workflow ad a workflow administrator or workflow owner will have to intervene and re-direct the workflow.

Alternatively, you can configure another workflow to be triggered if a workflow step is expired. In this case, you must define an action for the **On Step Expired** action of the selected workflow step. The action must be of the type `TriggerWorkflow`. The **Expiration Step** attribute should be left empty in this case. For more information about configuring a workflow to follow an expired workflow step, see the section Configuring a Workflow Step to Trigger a Subordinate Workflow.

For more information about defining the **Performance Duration** attribute, see the section Defining a Deadline and Reminders for a Workflow Step.

6) For each workflow step (including the first workflow step), you may optionally define the **Refuse Step** attribute. In the **Refuse Step** attribute for each workflow step, define the workflow step that should be the next workflow step if the selected step is refused by a user. You may only define one workflow step for this attribute.

If no workflow step is defined and the workflow step is refused, an error will occur in the workflow and a workflow administrator or workflow owner will have to intervene and re-direct the workflow.

Alternatively, you can configure another workflow to be triggered if a workflow step is refused. In this case, you must define an action for the **On Refuse Step** action of the selected workflow step. The action must be of the type `TriggerWorkflow`. The **Refuse Step** attribute should be left empty in this case. For more information about configuring a workflow to follow a refused workflow step, see the section Configuring a Workflow Step to Trigger a Subordinate Workflow.

Regardless of the setting defined in the **Confirmation Type** attribute, the refusal of a workflow step by a single user results in the overall refusal of the workflow step. Once a workflow step has been refused by a responsible user, all other responsible users will no longer be required to confirm the workflow step. For more information about defining configuring the refusal capability, see the section Configuring the Action Buttons Available for a Workflow Step.

7) Click the **Save**  button to save your definitions.

Configuring Email Notifications for Workflows

The workflow capability in Alfabet allows for the configuration of automatically-generated email notifications for a wide variety of events that could occur in a workflow. Email notifications can be automatically generated and sent to the relevant users for the following workflow actions:

- a workflow step has been entered (with the exception of the first workflow step)
- a workflow step has been refused
- a workflow step has been completed
- a workflow step has been delegated
- the targeted deadline for a workflow step is approaching
- the targeted deadline for a workflow step has expired
- an error has occurred in the workflow
- a workflow has been suspended
- a suspended workflow has been resumed
- the workflow owner has been changed
- a workflow has been completed

As the workflow designer, you can either enable the email functionality for all workflows generated for a workflow template or completely disable the email functionality for the workflow template. If you disable the email functionality, no email notifications will be generated in the context of the workflow. Please note that once a workflow step has been confirmed and thus completed, the workflow step can no longer be opened via the link in the email notification.

All email notifications sent in the context of a workflow are based on text templates that specify the text in the email as well as information about and hyperlinks to the workflow step triggering the notification and, if relevant, the object targeted by the workflow step in which the email is being generated. Software AG provides standard text templates for all contexts in which emails are generated. The standard text templates may be modified or new custom text templates can be created based on a standard text template.

The sender of the email notification will be determined based on the following, in the order given below:

1) Email address configured in the **System Mail Account** attribute in the server alias configuration.
2) Email address configured in the **Notification Sender Address** attribute for the workflow template.
3) Email address of the workflow owner.
4) Email address of the workflow template owner.

The following configuration requirements must be completed by the system administrator in the tool Alfabet Administrator:

- All Alfabet functionalities for which the email capability is to be implemented require the setup of a connection to an SMTP server for outgoing email. For more information, see the

section *Activating the Dispatch of Email Notifications in Alfabet* in the reference manual *System Administration*.

- The user profile used to open the Alfabet views targeted by hyperlinks in the email is, by default, the user profile that the sender was logged in with when the view was sent/triggered. However, your system administrator may configure that Alfabet views in email notifications are to be opened using the recipient's user profile. For more information, see the section *Configuring the Setup To Open the Alfabet Interface via Links in Email Notifications* in the reference manual *System Administration*.

For email notifications triggered via batch processes (such as reminder notifications or automatically-started workflows), a batch process must be configured by your system administration in the tool Alfabet Administrator. For more information about configuring a batch job for workflows, see the section *Batch Processes for Workflows with AlfaWorkflowCommandPrompt.exe* in the reference manual *System Administration*.

The following steps are required in order to configure and implement email notifications for a workflow template:

- Activate the email capability for the workflow template via the **Use Email Notification** attribute.
- Specify whether to concatenate messages per user via the **Notification Type** and, if necessary, **Notification Subject** attributes.
- Review and edit protected text templates available for workflows or create customized public text templates, as needed.
- Assign the relevant protected or public text template to each workflow step or relevant workflow step action.

The following information is available:

- Implementing the Email Notification Functionality for a Workflow Template
- Configuring Email Notifications When a Workflow Step Is Entered
- Configuring Email Notifications When a Workflow Step Is Refused
- Configuring Email Notifications When a Workflow Step Is Expired
- Configuring Email Notifications When a Workflow Step Is Exited
- Configuring Reminder Email Notifications About Approaching Deadlines
- Configuring Email Notifications About Delegated Workflow Steps
- Configuring Email Notifications About Changes in Workflow Ownership
- Configuring Email Notifications About Errors Occurring in a Workflow
- Configuring Email Notifications When a Workflow Is Paused and Resumed
- Configuring Email Notifications When a Workflow Is Completed

Implementing the Email Notification Functionality for a Workflow Template

The following steps are required in order to configure the implementation of email notifications for a workflow template:

1) Select `True` in the **Use Email Notification** attribute for the workflow template. If this attribute is set to `False`, no emails will be generated, regardless of any further configuration of email notification.

2) In the **Notification Type** attribute, specify if multiple emails should be concatenated for notifications generated via batch processes. Select one of the following:

Please note that if email notifications are consolidated for workflow templates and the **Is HTML** attribute is set to `True` for the associated text template, the consolidated email will also have HTML formatting. For information about configuring text templates, see the section Configuring Text Templates for Email Notifications.

- `NoConsolidation`: Select if an email should be generated for each message associated with an event triggering the notification. This is the default value.
- `ConsolidateAllMessages`: Select if all messages generated for a workflow template for a user should be concatenated in one email.
- `ConsolidateMessagesByStep`: Select if all messages generated for a workflow step for a user should be concatenated in one email.
- `ConsolidateMessagesByTemplate`: Select if all messages generated based on a text template in the context of a workflow for a user should be concatenated in one email. If this value is selected, the user will reserve all messages based on a specific type of activity, regardless of the workflow step in which they were generated.

3) If `ConsolidateAllMessages` or `ConsolidateMessagesByStep` is selected in the **Notification Type** attribute, the subject line of the email notification can be customized in the **Notification Subject** attribute.

4) In the **Executing User Profiles** attribute for the workflow template, define which user profile should be used when a user accesses the **My Workflow Activities** functionalities (`WF_UserWorkflowActivities`, `WF_UserWorkflowActivitiesExt`, and `WF_UserWorkflowActivitiesCommon`) and/or the **Workflow Activities Explorer** via the hyperlink in an email notification.

5) In the toolbar, click the **Save**  button to save your changes.

6) Define the text that should be displayed in the email notifications. Software AG provides a predefined default text template for each context in which a notification can be triggered. These text templates can be used as-is, modified, or used as a basis to create new text templates. This is done on the **Text Templates** node in the **Presentation** tab of Alfabet Expand. For general information about defining text templates, see the section Configuring Text Templates for Email Notifications in the chapter Configuring Alfabet Functionalities Implemented in the Solution Environment.

Configuring Email Notifications When a Workflow Step Is Entered

Email notifications may be configured and automatically generated when a workflow step has been entered.

Please note however that this is not possible for the first workflow step. The **Type** attribute may not be set to `Notification` for a workflow step action of the type **Action On Entered Step**.

You can configure each workflow step so that email notifications will be sent to the users specified in the associated query.

```
Dear {Person:Name} {Person:FirstName},

You are responsible for the workflow
step: {WorkflowStep:Name} '{WorkflowStep:Caption}'.

The object targeted by the workflow step is: '{Object:RefImage}'.
The workflow was created by the user: {Sender:Name} {Sender:FirstName}.

Please follow this link
        {Link:ObjectView}
to access the My Workflow Activities functionality.
```

FIGURE: Example of the standard text template WorkflowActivityNew

You can use the standard text template ***WorkflowActivityNew*** or a custom text template that you create for your enterprise. For more information about the standard text template ***WorkflowActivityNew*** and the permissible variables for the text template, see the section *Text Templates for Workflows* in the chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

To configure email notifications to be sent when a workflow step is entered:

1) Expand the relevant workflow step node  to display its subordinate workflow step actions.
2) Right-click the workflow step action **Action On Entered Step** and select **New Action On Entered Step**.

   If you define an action of the type `Notification` for the workflow step action **Action On Entered Step**, then the **Email Template by Enter Step** or **Email Template by Enter Step from Refuse** attributes for the workflow step may not be defined. Otherwise an error will occur.

3) Click the action  to open the attribute window and select `Notification` in the **Type** attribute.
4) In the **Technical Name** attribute, enter a name for the notification action.
5) In the **Text Template** attribute, select the standard text template ***WorkflowActivityNew*** or a customized text template that should be sent when the workflow step is entered.
6) In the **Query as Text** attribute, enter an Alfabet query or native SQL query returning the recipient email addresses to which email notifications shall be sent. The email addresses are entered in the relevant **To:** field, **CC:** field, **BCC:** field.:

   Emails can be sent to any email addresses stored in custom object class properties of the property type `Email` or the standard property `Email` of the object class `Person`. There are two ways to specify the recipient emails

- Define a query finding the relevant objects only. Emails will be sent to all email addresses stored in any object class property of the property type `Email` defined for the returned objects. The query can be either a native SQL query or an Alfabet query.

  For example to send emails to any email addresses defined for Alfabet users, you must at least define:

  ```
  Select REFSTR from PERSON
  ```

  or

  ```
  ALFABET_QUERY_500
  FIND PERSON
  ```

- If an object class has multiple properties of the property type `Email` and you would like emails to be send to the email addresses stored in one of these object class properties only, you must specify an Alfabet Query. The Alfabet query must return the email addresses in the **Show Properties** definition. The `FIND` class is not relevant.

  For example, the following query triggers sending of emails to the email addresses stored in a custom property of the type `Email` defined for the object class `Application`, although `Application` is not the `FIND` class. The information about the relevant object class and object class property are read from the **Show Properties**:

  ```
  ALFABET_QUERY_500
  FIND Person
  InnerJoin Application ON Person.REFSTR =
  Application.ResponsibleUser
  WHERE
  Application.MyEmailProp IS NOT NULL
  QUERY_XML
  <QueryDef>
  <ShowProperty Type="Property" ClassName="Application"
  Name="MyEmailProp">
  </QueryDef>
  ```

  Please note that the Alfabet Query Builder and the native SQL specific editor are not available in this attribute. If you would like to define the query in one of the specific editors, you can define and test the query in a configured report of the type `Query` or `NativeSQL` and past it into this attribute afterwards.

7) In the **CC Query as Text** attribute, enter an Alfabet query or SQL query to find the users who should be in the CC list of the email.
8) In the **BCC Query as Text** attribute, enter an Alfabet query or SQL query to find the users who should be in the BCC list of the email.
9) Click the **Save**  button to save your definitions.

Configuring Email Notifications When a Workflow Step Is Refused

You can configure each workflow step so that email notifications will be sent to the users specified in the associated query.

```
Dear {Person:Name} {Person:FirstName},

{Sender:Name} {Sender:FirstName} has refused the workflow
step: {WorkflowStep:Name} '{WorkflowStep:Caption}'

The object targeted by the workflow step is:  '{Object:RefImage}'

The following message provides explanation: {Message:Text}.
```

FIGURE: Example of the standard text template WorkflowStepRefused

You can use the standard text template ***WorkflowStepRefused*** or a custom text template that you create for your enterprise. For more information about the standard text template ***WorkflowStepRefused*** and the permissible variables for the text template, see the section *Text Templates for Workflows* in the chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

The refusal of the workflow step by one user results in the refusal of the workflow step for all responsible users. If a workflow step is refused, the workflow would advance to the workflow step defined in the **Refuse Step** attribute for the refused workflow step. If email notification is configured for the workflow step defined in the **Refuse Step** attribute, then the users responsible for that workflow step would receive email notification when that workflow step is entered.

To configure email notifications to be sent when a workflow step is refused:

1) Expand the relevant workflow step node  to display its subordinate workflow step actions.
2) Right-click the workflow step action **Action On Refused Step** and select **New Action On Refused Step**.
3) Click the action  to open the attribute window and select `Notification` in the **Type** attribute.
4) In the **Technical Name** attribute, enter a name for the notification action.
5) In the **Text Template** attribute, select the standard text template ***WorkflowStepRefused*** or a customized text template that should be sent when the workflow step is refused.
6) In the **Query as Text** attribute, enter an Alfabet query or native SQL query returning the recipient email addresses to which email notifications shall be sent. The email addresses are entered in the relevant **To:** field, **CC:** field, **BCC:** field.:

   Emails can be sent to any email addresses stored in custom object class properties of the property type `Email` or the standard property `Email` of the object class `Person`. There are two ways to specify the recipient emails

- Define a query finding the relevant objects only. Emails will be sent to all email addresses stored in any object class property of the property type `Email` defined for the returned objects. The query can be either a native SQL query or an Alfabet query.

For example to send emails to any email addresses defined for Alfabet users, you must at least define:

```
Select REFSTR from PERSON
```

or

```
ALFABET_QUERY_500
FIND PERSON
```

- If an object class has multiple properties of the property type `Email` and you would like emails to be send to the email addresses stored in one of these object class properties only, you must specify an Alfabet Query. The Alfabet query must return the email addresses in the **Show Properties** definition. The `FIND` class is not relevant.

For example the following query triggers sending of emails to the email addresses stored in a custom property of the type `Email` defined for the object class `Application`, although `Application` is not the `FIND` class. The information about the relevant object class and object class property are read from the **Show Properties**:

```
ALFABET_QUERY_500
FIND Person
InnerJoin Application ON Person.REFSTR =
Application.ResponsibleUser
WHERE
Application.MyEmailProp IS NOT NULL
QUERY_XML
<QueryDef>
<ShowProperty Type="Property" ClassName="Application"
Name="MyEmailProp">
</QueryDef>
```

Please note that the Alfabet Query Builder and the native SQL specific editor are not available in this attribute. If you would like to define the query in one of the specific editors, you can define and test the query in a configured report of the type `Query` or `NativeSQL` and past it into this attribute afterwards.

7) In the **CC Query as Text** attribute, enter an Alfabet query or SQL query to find the users who should be in the CC list of the email.

8) In the **BCC Query as Text** attribute, enter an Alfabet query or SQL query to find the users who should be in the BCC list of the email.

9) Click the **Save**  button to save your definitions.

Configuring Email Notifications When a Workflow Step Is Expired

You can configure each workflow step so that email notifications will be sent to the users specified in the associated query.

```
Dear {Person:Name} {Person:FirstName},

The workflow step: {WorkflowStep:Name} '{WorkflowStep:Caption}'

has escalated for the object targeted by the workflow
step: '{Object:RefImage}'

The following message provides explanation: {Message:Text}.
```

FIGURE: Example of the standard text template WorkflowStepEscalated

You can use the standard text template ***WorkflowStepEscalated*** or a custom text template that you create for your enterprise. For more information about the standard text template ***WorkflowStepEscalated*** and the permissible variables for the text template, see the section *Text Templates for Workflows* in the chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

If the current step is not performed in the number of days defined in the **Performance Duration** attribute starting with the day the workflow step is entered, the email notifications will be triggered. Please note however that either batch process must be executed by the system administrator in order for the deadline check to be executed. Alternatively, the workflow administer can trigger the **Check Performance Duration** functionality for a selected workflow in the **Workflow Administration** view.

For more information about configuring a batch job to check workflow deadlines, see the section *Batch Processes for Workflows with AlfaWorkflowCommandPrompt.exe* in the reference manual *System Administration*.

To configure email notifications to be sent when a workflow step has escalated due to, for example, a surpassed deadline:

1) Expand the relevant workflow step node  to display its subordinate workflow step actions.
2) Right-click the workflow step action **Action On Expired Step** and select **New Action On Expired Step**.
3) Click the action  to open the attribute window and select `Notification` in the **Type** attribute.
4) In the **Technical Name** attribute, enter a name for the notification action.
5) In the **Text Template** attribute, select the standard text template ***WorkflowStepEscalated*** or a custom text template that should be sent when the workflow step surpasses its deadline.
6) In the **Query as Text** attribute, enter an Alfabet query or native SQL query returning the recipient email addresses to which email notifications shall be sent. The email addresses are entered in the relevant **To:** field, **CC:** field, **BCC:** field.:

   Emails can be sent to any email addresses stored in custom object class properties of the property type `Email` or the standard property `Email` of the object class `Person`. There are two ways to specify the recipient emails

- Define a query finding the relevant objects only. Emails will be sent to all email addresses stored in any object class property of the property type `Email` defined for the returned objects. The query can be either a native SQL query or an Alfabet query.

  For example, to send emails to any email addresses defined for Alfabet users, you must at least define:

  ```
  Select REFSTR from PERSON
  ```

  or

  ```
  ALFABET_QUERY_500
  FIND PERSON
  ```

- If an object class has multiple properties of the property type `Email` and you would like emails to be send to the email addresses stored in one of these object class properties only, you must specify an Alfabet Query. The Alfabet query must return the email addresses in the **Show Properties** definition. The `FIND` class is not relevant.

  For example, the following query triggers sending of emails to the email addresses stored in a custom property of the type `Email` defined for the object class `Application`, although `Application` is not the `FIND` class. The information about the relevant object class and object class property are read from the **Show Properties**:

  ```
  ALFABET_QUERY_500
  FIND Person
  InnerJoin Application ON Person.REFSTR =
  Application.ResponsibleUser
  WHERE
  Application.MyEmailProp IS NOT NULL
  QUERY_XML
  <QueryDef>
  <ShowProperty Type="Property" ClassName="Application"
  Name="MyEmailProp">
  </QueryDef>
  ```

  Please note that the Alfabet Query Builder and the native SQL specific editor are not available in this attribute. If you would like to define the query in one of the specific editors, you can define and test the query in a configured report of the type `Query` or `NativeSQL` and past it into this attribute afterwards.

7) In the **CC Query as Text** attribute, enter an Alfabet query or SQL query to find the users who should be in the CC list of the email.
8) In the **BCC Query as Text** attribute, enter an Alfabet query or SQL query to find the users who should be in the BCC list of the email.
9) Click the **Save**  button to save your definitions.

Configuring Email Notifications When a Workflow Step Is Exited

You can configure email notifications to be sent to the users specified in the associated query when the workflow step has been completed.

You can create a custom text template for your enterprise. For more information about the standard text templates and their variables, see the section *Text Templates for Workflows* in the chapter *Overview of Pre-configured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

To configure email notifications to be sent when a workflow step has been completed:

1) Expand the relevant workflow step node to display its subordinate workflow step actions.
2) Right-click the workflow step action **Action On Exited Step** and select **New Action On Exited Step**.
3) Click the action to open the attribute window and select `Notification` in the **Type** attribute.
4) In the **Technical Name** attribute, enter a name for the notification action.
5) In the **Text Template** attribute, select the email template that should be sent to the workflow owner when the workflow step is exited.
6) In the **Query as Text** attribute, enter an Alfabet query or native SQL query returning the recipient email addresses to which email notifications shall be sent. The email addresses are entered in the relevant **To:** field, **CC:** field, **BCC:** field.:

   Emails can be sent to any email addresses stored in custom object class properties of the property type `Email` or the standard property `Email` of the object class `Person`. There are two ways to specify the recipient emails

- Define a query finding the relevant objects only. Emails will be sent to all email addresses stored in any object class property of the property type `Email` defined for the returned objects. The query can be either a native SQL query or an Alfabet query.

  For example to send emails to any email addresses defined for Alfabet users, you must at least define:

  ```
  Select REFSTR from PERSON
  ```

  or

  ```
  ALFABET_QUERY_500
  FIND PERSON
  ```

- If an object class has multiple properties of the property type `Email` and you would like emails to be send to the email addresses stored in one of these object class properties only, you must specify an Alfabet Query. The Alfabet query must return the email addresses in the **Show Properties** definition. The `FIND` class is not relevant.

  For example, the following query triggers sending of emails to the email addresses stored in a custom property of the type `Email` defined for the object class `Application`, although `Application` is not the `FIND` class. The information about the relevant object class and object class property are read from the **Show Properties**:


```
ALFABET_QUERY_500

FIND Person

InnerJoin Application ON Person.REFSTR =
Application.ResponsibleUser

WHERE

Application.MyEmailProp IS NOT NULL

QUERY_XML

<QueryDef>

<ShowProperty Type="Property" ClassName="Application"
Name="MyEmailProp">

</QueryDef>
```

Please note that the Alfabet Query Builder and the native SQL specific editor are not available in this attribute. If you would like to define the query in one of the specific editors, you can define and test the query in a configured report of the type `Query` or `NativeSQL` and past it into this attribute afterwards.

7) In the **CC Query as Text** attribute, enter an Alfabet query or SQL query to find the users who should be in the CC list of the email.

8) In the **BCC Query as Text** attribute, enter an Alfabet query or SQL query to find the users who should be in the BCC list of the email.

9) Click the **Save**  button to save your definitions.

Configuring Reminder Email Notifications About Approaching Deadlines

If a workflow step's target due date is approaching, an email may be sent to the responsible user(s) of the workflow step. You can define email notifications for both the responsible users informing them of the pending deadline as well as the workflow owner informing him/her that reminders have been sent to the responsible users. This configuration can be made on the level of the workflow template as well as the workflow step.

The definitions made for the workflow step will have precedence over the definition defined for the workflow template.

```
Person: {Person:Name} {Person:FirstName}
Sender: {Sender:Name} {Sender:FirstName}

Dear {Person:Name} {Person:FirstName},

Workflow Activity: {WorkflowStep:Name} '{WorkflowStep:Caption}'

The object targeted by the workflow step is: '{Object:RefImage}'.

Please follow this link
        {Link:ObjectView}
to access the My Workflow Activities functionality.
```

FIGURE: Example of the standard text template WorkflowReminder

You can use the standard text template ***WorkflowReminder*** or a custom text template that you create for your enterprise. For more information about the standard text template ***WorkflowReminder*** and the permissible variables for the text template, see the section *Text Templates for Workflows* in the chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

The workflow step deadline is configured via the **Performance Duration** attribute and includes the number of days that a user has to complete the workflow step, the number of days before the deadline when the first reminder notification should be sent, and the how often the frequency that the user should be reminded of the impending deadline. Reminder frequency is defined on the level of an individual workflow step. For more information, see Defining a Deadline and Reminders for a Workflow Step.

If the **Performance Duration** attribute is defined for the workflow template, it is recommended that reminder emails are sent every day. Therefore, the batch job `WorkflowCommandPrompt.exe` should be executed on daily basis because the reminders are sent when the specified reminder start and frequency is matched based on the day that the workflow activity was created. For more information about configuring a batch job to check workflow deadlines, see the section *Batch Processes for Workflows with AlfaWorkflowCommandPrompt.exe* in the reference manual *System Administration*.

To configure the reminder emails that should be sent out to inform users about impending deadlines:

1) To define reminder notifications for the workflow template as a whole, click the workflow step  or workflow template  to display the attribute window.
2) In the **Email Template by Reminder** attribute, select the standard text template ***WorkflowReminder*** or a customized text template that should be sent as a reminder of the workflow step's impending target date.

   If an email template has been defined in the **Email Template by Reminder** or **Email Template by Reminder Alert** attributes for a workflow step, then the template defined for a workflow step will be used instead of the one defined for the workflow template. Reminder frequency is defined on the level of an individual workflow step. For more information, see Defining a Deadline and Reminders for a Workflow Step.

3) In the **Email Template by Reminder Alert** attribute, select the email template that should be sent to the workflow owner and initiator of the workflow if a reminder email is sent to the users responsible for the current workflow step. If an email template is not selected, the workflow owner and initiator will not be informed about impending target dates
4) Click the **Save**  button to save your definitions.

Configuring Email Notifications About Delegated Workflow Steps

You can configure the workflow template so that email notifications will be sent to the workflow owner and the user to whom the workflow step is being delegated.

```
Dear {Person:Name} {Person:FirstName},

{Sender:Name} {Sender:FirstName}

has delegated the following workflow step to you.

The workflow step is: {WorkflowStep:Name} '{WorkflowStep:Caption}'

The object targeted by the workflow step is: '{Object:RefImage}'.

The following message provides explanation: {Message:Text}.

Please follow this link
        {Link:ObjectView}
to access the My Workflow Activities functionality.
```

FIGURE: Example of the standard text template WorkflowStepDelegated

You can use the standard text template ***WorkflowStepDelegated*** or a custom text template that you create for your enterprise. For more information about the standard text template ***WorkflowStepDelegated*** and the permissible variables for the text template, see the section *Text Templates for Workflows* in the chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

To configure email notifications to be sent when a workflow is delegated:

1) Click the workflow template  to display the attribute window.
2) In the **Email Template by Delegate** attribute, select standard text template ***WorkflowStepDelegated*** or a custom text template that should be sent to the workflow owner and the user to whom the workflow step is being delegated.
3) Click the **Save**  button to save your definitions.

Configuring Email Notifications About Changes in Workflow Ownership

The owner of a workflow is typically the user that has created the workflow based on the workflow template. If necessary, the workflow administrator can change the owner of the workflow to another user that has access to the **My Workflows** or **Initiate Workflows** functionality via his/her user profile.

You can configure the workflow template so that email notifications will be sent to the original workflow owner and the new workflow owner when the workflow administrator has changed the ownership of a workflow


```
Dear {Person:Name} {Person:FirstName},

{Sender:Name} {Sender:FirstName}
has changed the workflow owner.

The following message provides explanation: {Message:Text}.

Please follow this link
        {Link:ObjectView}
to access the My Workflows functionality.
```

FIGURE: Example of the standard text template WorkflowChangeOwner

You can use the standard text template ***WorkflowChangeOwner*** or a custom text template that you create for your enterprise. For more information about the standard text template ***WorkflowChangeOwner*** and the permissible variables for the text template, see the section *Text Templates for Workflows* in the chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

For more information about changing the ownership of a workflow, see the section *Changing the Owner of a Workflow Template or Workflow* in the reference manual *User and Solution Administration*.

To configure email notifications to be sent when the workflow owner is changed:

1) Click the workflow template  to display the attribute window.
2) In the **Email Template by Change Owner** attribute, select the standard text template ***WorkflowChangeOwner*** or a custom text template that should be sent to the new and original workflow owners when the ownership of a workflow has been changed by the system administrator.
3) Click the **Save**  button to save your definitions.

Configuring Email Notifications About Errors Occurring in a Workflow

Errors can occur in a workflow that prevent the workflow from advancing to a next workflow step. For example, if a workflow step's target date has expired, the workflow step may escalate (`EscalateByTime`), or if no responsible user is assigned to a workflow step or found via a predefined Alfabet query, the workflow step will escalate (`EsclatedByError`). The errors must then be dealt with and resolved by either the workflow owner or the workflow administrator.

You can configure the workflow template so that email notifications will be sent to the workflow owner when an error has occurred and the state of the workflow has escalated.


```
Dear {Person:Name} {Person:FirstName},

The workflow step: {WorkflowStep:Name} '{WorkflowStep:Caption}'

has escalated for the object targeted by the workflow
step: '{Object:RefImage}'

The following message provides explanation: {Message:Text}.
```

FIGURE: Example of the standard text template WorkflowStepEscalated

You can use the standard text template ***WorkflowStepEscalated*** or a custom text template that you create for your enterprise. For more information about the standard text template ***WorkflowStepEscalated*** and the permissible variables for the text template, see the section *Text Templates for Workflows* in the chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

For more information about error resolution, see the section *Fixing a Workflow with an Error State* in the reference manual *User and Solution Administration*.

To configure email notifications to be sent when an error has occurred in the workflow:

1) Click the workflow template to display the attribute window.
2) In the **Email Template by Escalate** attribute, select the email template that should be sent to the workflow owner and workflow administrator when a workflow step is escalated.
3) Click the **Save** button to save your definitions.

Configuring Email Notifications When a Workflow Is Paused and Resumed

You can configure the workflow template so that email notifications will be sent to the responsible users of the workflow step that is impacted when the workflow owner or workflow administrator suspends or resumes a workflow.

For more information about the standard text templates and their variables, see the section *Text Templates for Workflows* in the chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

For more information about how a workflow can be suspended or continued, see the section *Suspending, Resuming, or Withdrawing the Workflow* in the reference manual *User and Solution Administration*.

To configure email notifications to be sent when a workflow is suspended and continued:

1) Click the workflow template to display the attribute window.
2) In the **Email Template by Pause** attribute, select the standard text template ***WorkflowPaused*** or a custom text template that should be sent to the responsible user(s) when the workflow is suspended by the workflow owner or workflow administrator.

3) In the **Email Template by Resume** attribute, select the email template that should be sent to the responsible user(s) when the workflow is resumed by the workflow owner or workflow administrator.

4) Click the **Save**  button to save your definitions.

Configuring Email Notifications When a Workflow Is Completed

You can configure the workflow template so that email notifications will be sent to the workflow owner whenever his/her workflow based on the workflow template reaches the state `Finished`.

For more information about the standard text templates and their variables, see the section *Text Templates for Workflows* in the chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

To configure email notifications to be sent when a workflow is completed:

1) Click the workflow template  to display the attribute window.
2) In the **E-Mail Template by Finish** attribute, select the standard text template ***WorkflowResumed*** or a custom text template that should sent to workflow owners when workflows based on the selected workflow template reaches the state `Finished`.

3) Click the **Save**  button to save your definitions.

Specifying the Customized Workflow Activities Explorer (WFS_Explorer) or a Custom Explorer

Software AG provides several options to implement an explorer for users to process their workflow activities.

1) Implement the standard datasets for workflow activities. The **My Workflow Activities** functionalities (`WF_UserWorkflowActivities`, `WF_UserWorkflowActivitiesExt`, and `WF_UserWorkflowActivitiesCommon`) are standard tabular datasets that may be made available to the users who will be responsible for performing the workflow steps. These views are preconfigured and cannot be modified. For more information about these standard functionalities, see the section *Performing and Tracking Your Workflow Steps in the My Workflow Activities Functionality* in the reference manual *Getting Started with Alfabet*. The implementation of the standard functionalities is described in the section Making the Workflow Capability Available to the User Community.

2) Implement the **Workflow Activities Explorer** (`WFS_Explorer`). The **Workflow Activities Explorer** functionality (`WFS_Explorer`) is a standard explorer available to support users in processing their workflow activities. The **Workflow Activities Explorer** functionality features a design and layout of data in Alfabet similar to established email management systems so that users can more easily and efficiently accomplish their workflow tasks. The explorer structure is preconfigured and cannot be changed for the **Workflow Activities Explorer** (`WFS_Explorer`) but the data displayed in the right pane when a workflow activity is selected in the explorer is configurable and users can be provided with precisely the data they need in order to process the task-at-hand with a minimum number of clicks and navigation to other views. The **Workflow Activities Explorer** functionality (`WFS_Explorer`) includes the following:

- The **Workflow Activities Explorer** is displayed in the left pane. It has a root node **My Workflow Activities** followed by the **Start New Workflow** node, which when clicked displays the **My Workflows** (`WF_UserWorkflows`) dataset in which a new workflow can be started. Below the **Start New Workflow** node, the subordinate nodes **Open Activities**, **Finished Activities**, **Refused Activities**, **Cancelled Activities**, and **Withdrawn Activities** and **Expired Activities** are available. Please note the following about the explorer:
- The nodes will only be displayed if the user has relevant activities. For example, the explorer node **Expired Activities** will only be displayed in the explorer if the user has workflow activities that have not been completed by the deadline.
- The workflow activities that the user is responsible for are displayed below the respective node. For each workflow activity, the name of the workflow activity is displayed followed by the name of the object targeted by the workflow activity. The captions for the objects targeted by the workflow are based on the **Image Properties** attribute for the relevant class setting.
- Standard symbols indicate the respective completion status of each workflow activity.
- Once the workflow activity is completed, it will automatically be removed from the **Open Activities** node.
- Completed activities will be displayed below the **Finished Activities** node for 30 days after they have been completed.
- Users can click the + symbol to expand a list and click the - symbol to collapse a node.

- The explorer search functionality is available and supports the user in finding a specific workflow activity in the explorer. For more information about using the explorer search functionality, see the section *Working with Explorers* in the reference manual *Getting Started with Alfabet*.
- Users can click a workflow activity in the explorer to display detailed information about the task in the reading pane. Please note the following:
- The toolbar above the reading pane displays only the action buttons relevant for the selected workflow activity. Users can click a button in the toolbar to perform the relevant action. The captions and symbols on the buttons are configurable and thus may vary among workflow activities. The display of buttons is determined by the definition of the attributes in the **Visualization** section of the workflow step's attribute grid as described in the section Configuring the Action Buttons Available for a Workflow Step.
- The content in the right pane is configured via an HTML template which specifies the content, navigation, and layout of the data. An HTML template may be configured for each individual workflow step defined for a workflow template if necessary. A HTML template may be reused for multiple workflow steps or assigned to the workflow template and used for all workflow steps. The configuration of the HTML template is described in the section Configuring HTML Templates for the Workflow Activities Explorer. Please note the following:
    - The header of the reading pane can be configured in the HTML template. For example, you could configure to display the name of the workflow activity and targeted object as well as the description configured for the workflow step.
    - You can specify one or more groups of information in the reading pane. For example, one section could display task-related information about the workflow activity including, for example, the status and target date of the workflow activity, other users responsible for the workflow activity, the name of the previous or next workflow activity as well as information about the workflow and user who triggered the workflow. Another section could display details about the object targeted by the workflow activity such as the name, release status, object state, start and end dates or other data relevant to the workflow activity.
    - The information displayed may be hyperlinked so that users can navigate to additional information in, for example, another object view, page view, or configured report.
    - The data may display a list of items (for example, responsible users). The view can be configured so that the list is abbreviated if the number of items exceeds a specified limit. Only the specified number of users will be displayed in the reading pane followed by the line**... + <number> more...**.

3) Implement a custom explorer. In order to implement a custom explorer for workflows, you must specify the content of the explorer tree as well as the content, layout, and navigation for the reading pane of the view. Unlike the **Workflow Activities Explorer** (`WFS_Explorer`), which only includes workflow activities, you can configure a custom explorer to display nodes with workflows as well nodes with workflow activities. By implementing a custom explorer instead of the standard **Workflow Activities Explorer** (`WFS_Explorer`), you can configure the content of the explorer tree as well as the information available in the reading pane described above. The custom explorer may consist of a single node and display one type of workflow activity such as all workflow activities that have been confirmed. Or the custom explorer may have multiple nodes, each of which shows a different type of workflow activity such as workflow activities past their deadline, workflow activities due in a week, workflow activities due in a month, and workflow activities that are not urgent.

The following information is available:

Configuring the Workflow Activities Explorer (WFS_Explorer)

In order to implement the **Workflow Activities Explorer** (`WFS_Explorer`), you must specify the content, layout, and navigation for the reading pane of the view as well as make the **Workflow Activities Explorer** (`WFS_Explorer`) available to the user community via a user profile or guide page/guide view. The visualization of the data in the reading pane of the **Workflow Activities Explorer** (`WFS_Explorer`) can be customized for each workflow step, if needed. Users will only see the workflow activities that they are responsible for in the **Workflow Activities Explorer** (`WFS_Explorer`).

The configuration of guide views and guide pages may contain a hyperlinked text or button displaying the number of workflow steps that the user is responsible for and that when clicked opens the **Workflow Activities Explorer**. Instead of displaying the default **Workflow Activities Explorer**, a custom explorer can be configured to open instead. The custom explorer must be specified in the **Custom Workflow Activities Explorer** attribute of the relevant user profile. For more information about configuring custom explorers, see the chapter Configuring Standard Business Functions and Custom Explorers.

The following must be completed in order to implement the **Workflow Activities Explorer** (`WFS_Explorer`):

1) Configure the workflow template and workflow steps as outlined in this chapter on workflow configuration. All configurations described for workflow templates and workflow steps should be completed before you begin specifying the implementation of the **Workflow Activities Explorer** (`WFS_Explorer`). Please note that the following:

- The **Perform**, **Confirm**, **Delegate**, etc. are specified either for the workflow template or for the workflow step via the options (**Perform** button, **Confirm** button, **Delegate** button, etc.) in the **Visualization** section of the attribute grid for the workflow template and/or workflow step. Please note that the workflow step definition will have precedence over the workflow template definition.
- The toolbar buttons specified for the workflow step/workflow template will be displayed above the reading pane in the **Workflow Activities Explorer** functionality.

2) Configure the HTML template to display the content, navigation, and layout of the data for the workflow steps. The HTML must be XML-conform HTML, compliant with HTML 5, and use standard HTML tags. For more information about how to specify the HTML template as well as the proprietary HTML elements available for the HTML template, see the section Guidelines for HTML Template Syntax in Workflows.

3) The HTML template must be assigned to the workflow template (if relevant for all or some of the workflow steps) or to the relevant workflow step via the **HTML Template for Step** attribute of either the workflow template or relevant workflow step.

If the **HTML Template for Step** attribute is defined for the workflow template, the HTML template will be applied to all workflow steps unless an HTML template is explicitly defined for a workflow step, in which case the workflow step definition will take precedence over the workflow template definition. All HTML templates listed below the **Visualization Items** node are displayed in the drop-down list for the **HTML Template for Step** attribute.

4) A CSS file must be configured to specify the styles for the HTML template. A reference to the CSS file must be included in the specification of the HTML template. The CSS file that you reference in the HTML template must be stored in the **Internal Document Selector**. For more information about configuring the CSS file, see the section Guidelines for the CSS File.

5) The **Workflow Activities Explorer** (`WFS_Explorer`) must be assigned as a business function to the user profile or implemented in a guide view to make it accessible to the user community.

6) You can specify whether the **Workflow Activities Explorer** functionality (`WFS_Explorer`) opens per default when the **Workflow Activities** link defined in a navigation page is clicked and the **Open Workflow Activities** link in an object profile is clicked. This is specified in the XML attribute `WFGuiVersion` in the XML object ***UserPersonalSettings***. Users can later specify their individual setting in the **User Settings** editor in the Alfabet interface. For more information, see the section Configuring Default User Settings for the User Community in the chapter Executing Administrative Tasks in Alfabet Expand.

7) In the XML object **SolutionOptions**, specify the maximum number of days that finished, refused, and expired workflow activities shall be displayed in the **Workflow Activities Explorer** functionality. This ensures that obsolete workflows are no longer cluttering the **My Workflow Activities** explorer.

The following information is available:

- Configuring HTML Templates for the Workflow Activities Explorer
- Guidelines for HTML Template Syntax in Workflows
- Guidelines for the CSS File
- Configuring the Maximum Days to Display Finished Workflow Activities

Configuring HTML Templates for the Workflow Activities Explorer

HTML templates contain basic elements to capture information about the workflow step as well as the object targeted by the workflow step, provide navigation to relevant object views, and specify the layout and visualization of the data. HTML templates can be implemented the **Workflow Activities Explorer** functionality (`WFS_Explorer`) as well as custom explorers configured for workflow activities.

The **Workflow Activities Explorer** in the `Showcase` database displays an example of an HTML template configuration. In this example, the caption of the workflow step is configured to provide navigation to the object profile of the workflow step. The header also displays the caption of the object that is targeted by the workflow step including relevant object class properties and navigation to the object profile of the targeted object. In the reading pane, two boxes grouping data are displayed side-by-side. The first box is captioned **Task-Related Details** and display information about the workflow step including the workflow

step's description, status, deadline, responsible users as well as information about the previous step and it's responsible users, and the workflow owner. The second box is captioned **<Object>-Related Details** and displays information about the object targeted by the workflow step, including relevant data such as the start and end date of the object, release status, contact persons, etc.

In the `Showcase` database provided by Software AG, the node **Visualization Items** contains preconfigured HTML templates that can be used as-is for workflow steps targeting applications, components, demands, and projects. The preconfigured HTML template `Generic_Standard`, a generic template that is not tailored to a specific object class, can also be used as a basis for configuring a custom HTML template. You can copy the HTML template `Generic_Standard` and paste it to a new HTML template and refine the definition in the **XML Def** attribute of the new HTML template, as needed.

An HTML template must first be configured in the **Visualization Items** node in the **Presentation** tab and then assigned to a workflow template or to one or more workflow steps. The HTML templates are based on XHTML using HTML 5 standards. HTML templates implement standard HTML elements as well as proprietary HTML elements (`AlfaHtmlElement` and `AlfaHtmlEvent`). The HTML can be defined in a standard XHTML editor and pasted into the HTML template. The placeholders for the proprietary HTML elements will then be replaced at run-time in Alfabet with the proper content. The syntax for the HTML template is described in detail in the section Guidelines for HTML Template Syntax in Workflows.

Please note the following:

- The HTML must be XML-conform HTML, compliant with HTML 5, and use standard HTML tags.
- We strongly recommend that JavaScript is not used in HTML templates since JavaScript code is not always compatible with all browsers supported for Alfabet. Using JavaScript in HTML templates may lead to errors in the display of HTML pages.

If an HTML template is assigned to a workflow template, it will be used for all relevant workflow steps defined for the workflow template. However, an HTML template can also be explicitly assigned to one or more workflow steps. In this case, the HTML template assigned to the workflow step has precedence over the HTML template assigned to the workflow template. Thus, HTML templates can be reused and assigned to multiple workflow steps and can even be assigned to other workflow templates.

To create and define an HTML template:

1) Go to the **Presentation** tab and expand the **Visualization Items** folder by clicking the + symbol. You will see the preconfigured templates `APP_Standard`, `COM_Standard`, `DEM_Standard`, `PRJ_Standard` and `Generic_Standard` (a class-independent template). You can copy a preconfigured HTML template as a basis for a custom HTML template or you can create a new HTML template from scratch.
2) Right-click the **Visualization Items** node and select **New HTML Template**. A new HTML template  is added to the **Visualization Items** folder.
3) Click the new HTML template to open the attribute window. In the attribute window, define the following attributes:

- **Name**: Enter a name.
- **Description**: Enter a description of the HTML template.
- **Sub-Type**: Select `Workflow`.

- **Group**: If the HTML template should be structured in a folder, enter the name of the folder. You can create a new HTML group this way or move the HTML template to an existing folder.

4) To use an existing HTML template as the basis for a new HTML template, right-click the template you want to use and select **Copy**. Right-click the new HTML template and click **Paste**.

5) To edit the HTML template, right-click it and select **Edit HTML Template**: The editor in the center pane opens. You can edit the HTML text directly in the editor or work in a standard HTML editor and copy the HTML text into the HTML template. The HTML must be XML-conform HTML, compliant with HTML 5, and use standard HTML tags. For details about the permissible syntax in the HTML template, see the section Guidelines for HTML Template Syntax in Workflows.

6) Click the **Save** button to save the HTML template.

7) To assign the HTML template to the relevant workflow template or workflow step, navigate to the relevant workflow template or workflow step node in the Workflows tab. Click the node to open the attribute window. In the **Visualizations** section of the table, select the HTML template in the **HTML Template for Step** attribute.

> If the **HTML Template for Step** attribute is defined for the workflow template, the HTML template will be applied to all workflow steps unless an HTML template is explicitly defined for a workflow step, in which case the workflow step definition will take precedence over the workflow template definition. All HTML templates listed below the **Visualization Items** node are displayed in the drop-down list for the **HTML Template for Step** attribute.

8) To specify which button should be displayed (and thus which actions are possible for a workflow step) in the **Workflow Activities Explorer** functionality (`WFS_Explorer`), you must define the attributes **Complete Button**, **Confirm Button**, **Delegate Button**, **Perform Button**, and **Refuse Button** in the **Visualization** section of the attribute window. If these attributes are defined for the workflow template, the button definition will be applied to all workflow steps unless a button definition is explicitly made for a workflow step, in which case the workflow step definition will take precedence over the workflow template definition. For each button, a further table section can be opened displaying the attributes **Visibility**, **Caption**, and **Icon**.

- Select `False` in the **Visibility** attribute for all toolbar buttons that should not be available for the respective workflow step (or workflow template).
- Select `True` in the **Visibility** attribute for all toolbar buttons that should be available for the respective workflow step (or workflow template). Define the following attributes:
- **Caption**: Enter a caption to display on the button
- **Icon**: Select an image from a preconfigured button library to display on the button.

> For a detailed description about the configuration of buttons for workflow steps including such issues as the users required to confirm a workflow step, the permissibility of the refusal or delegation of a workflow step, the automatic closure of a workflow step, etc., see the section Configuring the Action Buttons Available for a Workflow Step.

9) Click the **Save** button to save the changes made to the workflow template or workflow step.

Guidelines for HTML Template Syntax in Workflows

HTML templates are configured in the **Visualization Items** node in the **Presentation** tab. The HTML templates that you implement in your enterprise can be designed as needed. The HTML templates allow you to specify the information to display about the workflow step as well as the object targeted by the workflow step, provide navigation to relevant object views, page views, or configured reports, and specify the layout and visualization of the data. The preconfigured templates include a wrapper item displayed above the reading pane that displays a title for the workflow activity.

In the `Showcase` database provided by Software AG, the node **Visualization Items** contains preconfigured HTML templates that can be used as-is for workflow steps targeting applications, components, demands, and projects. The preconfigured HTML template `Generic_Standard`, a generic template that is not tailored to a specific object class, can also be used as a basis for configuring a custom HTML template. You can copy the HTML template `Generic_Standard` and paste it to a new HTML template and refine the definition in the **XML Def** attribute of the new HTML template, as needed.

We strongly recommend that JavaScript is not used in HTML templates since JavaScript code is not always compatible with all browsers supported for Alfabet. Using JavaScript in HTML templates may lead to errors in the display of HTML pages.

Please note the following about specifying the HTML template:

- The HTML templates are based on XHTML using HTML 5 standards. The HTML must be compliant with XHTML The HTML must be XML-conform HTML, compliant with XHTML and HTML 5, and use standard HTML tags. The HTML templates implement standard HTML elements as well as proprietary XML elements. All HTML and proprietary XML elements must be closed. For example, if you include an image, the element `<img>` must be written with a closing slash like for example: `<img src="image.gif"/>`.
- Each HTML template must begin with document type definition: `<!DOCTYPE HTML>`
- The root element of the HTML document must be `<xhtml>`.
- Standard HTML elements must be written in lower-case letters in order to be correctly parsed: `<xhtml>`, `<html>`, `<body>`, and `<culture_>`
- In the header of the XML document, an element `script` must be specified with the attributes as defined in the example:

```
<xhtml>
    <html>
        <head>
            <meta http-equiv="content-type" content="text/html;charset=utf-
            8" />
            <script language="javascript"
            src="alfaembeddedhtml.js"></script>
            <link type="text/css" rel="stylesheet"
            href="IDOC:\CSS\my_style_file.css"></link>
        </head>
        ...
    </html>
</xhtml>
```


- In the header of the XML document, an element `link` must reference the stylesheet to be used by the HTML template. The CSS file is stored in the **Internal Document Selector** available in the **Internal Documents** functionality in Alfabet. Details about the configuration of the CSS file is described in the section Guidelines for the CSS File. For example:

```
<xhtml>
    <html>
        <head>
            <meta http-equiv="content-type" content="text/html;charset=utf-
            8" />
            <script language="javascript"
            src="alfaembeddedhtml.js"></script>
            <link type="text/css" rel="stylesheet"
            href="IDOC:\CSS\my_style_file.css"></link>
        </head>
        ...
    </html>
</xhtml>
```

- Two proprietary elements are available to retrieve data from the Alfabet database. The XML element `<AlfaHtmlElement>` and the function `AlfaHtmlEvent()`.

- Each XML `<AlfaHtmlElement>` should be added as a child element to a standard HTML element such as `<td>`, `<span>`, `<div>`, etc. in order to retrieve data from the Alfabet. The following can be included in the XML element `<AlfaHtmlElement>`:

- The XML attributes `ApplyTo`, `ShowProps`, `PropName`, and `FormatString` can be used to retrieve and display data. For example, to display the comments of the current workflow step, you would specify the current workflow step in the XML attribute `ApplyTo` and the object class property `Comment` in the XML attribute `ShowProps`.

```
<span><AlfaHtmlElement ApplyTo="@CURRENT_STEP"
ShowProps="Comment"/></span>
```

To display previous workflow step of the current workflow step, you would specify the name of the referenced object class property in the XML attribute `PropName`, and the object class property `Caption` in the XML attribute `ShowProps`:

```
<span><AlfaHtmlElement ApplyTo="@CURRENT_STEP"
PropName="PreviousStep" ShowProps="Caption"/></span>
```

- The following Alfabet query language parameters can be used in the XML attribute `ApplyTo` to define the object whose data is to be displayed:
    - `@BASE`: to retrieve data about the object that the workflow step references
    - `@CURRENT_STEP`: to retrieve data about the current workflow step
    - `@PREVIOUS_STEP`: to retrieve data about the previous workflow step
    - `@WORKFLOW`: to retrieve data about the current workflow

- The XML attribute `SqlText` allows you to include an Alfabet query or native SQL query to retrieve data. Please note that if the XML attribute `SqlText` is included in the XML element `<AlfaHtmlElement>`, the XML attribute `ShowProps` will be ignored. In the example below, multiple users will be returned for

the query. The XML attribute `MaxCount` allows you to limit the number of rows (results) displayed. If the number of results is greater than the value in the XML attribute `MaxCount`, an additional line **+{X} more** will be displayed whereby X is determined by the number of results minus the value in the XML attribute `MaxCount`. For example:

```
<AlfaHtmlElementMaxCount ="5"
ApplyTo="@CURRENT_STEP,@CURRENT_USER"
onclick="AlfaHtmlEvent(this,'Navigate_OpenObjectView','@REF',
null);">
    <SqlText> <![CDATA[
        SELECT per.REFSTR, per.TECH_NAME,'(' + per.PHONE +')'
        FROM RELATIONS rel, PERSON per
        WHERE rel.FROMREF = @CURRENT_STEP
            AND rel.PROPERTY ='SignOnUsers'
            AND rel.TOREF = per.REFSTR
            AND per.REFSTR != @CURRENT_USER
        ORDER BY per.TECH_NAME
        ]]>
    </SqlText>
</AlfaHtmlElement>
```

- The XML attribute `Filter` allows you to include color instructions, picture instructions, or filter instructions written in an Alfabet query or SQL query. For example:

```
<AlfaHtmlElement ApplyTo="@CURRENT_STEP" ShowProps="State">
    <Filter> <![CDATA[COLORASSIGNMENT(EqualTo, "Pending", #c0c000,
    Transparent);]]> </Filter>
    <Filter> <![CDATA[COLORASSIGNMENT(EqualTo, "Confirmed",
    #00c000, Transparent);]]> </Filter>
    <Filter> <![CDATA[COLORASSIGNMENT(EqualTo, "Cancelled",
    #c00000, Transparent);]]> </Filter>
</AlfaHtmlElement>
```

- `AlfaHtmlEvent()` is a function that allows you to specify navigation behavior. The function must be specified in the XML attribute `onclick` of the XML element `AlfabetHTMLElement`. The XML attribute `onclick` allows you to specify where the user will navigate if he/she clicks on the element in the HTML specification. The user must have permissions to therelevant view. For example:

    - To navigate to the object view of the object, use the following parameter:

    ```
    <AlfaHtmlElement ApplyTo="@CURRENT_STEP" ShowProps="State"
        onclick="AlfaHtmlEvent(this,'Navigate_OpenObjectView','@REF'
        , null);" >
    </AlfaHtmlElement>
    ```

    - To navigate to the object cockpit of the object, use the following parameter:

    The user must have permissions to the specified object view and the object cockpit. If the user is not permissible for the specified object view and

object cockpit, the object view defined for the class settings of the object class/object class stereotype will be used.

```
<AlfaHtmlElement ApplyTo="@CURRENT_STEP" ShowProps="State"
   onclick="AlfaHtmlEvent(this,'Navigate_OpenObjectView',’@REF'
   ,’<Object View Name:Object Cockpit Name>');" >
</AlfaHtmlElement>
```

- To navigate to a page view, use the following parameter, replacing the `null` with the technical name of the relevant view:

```
<AlfaHtmlElement ApplyTo="@CURRENT_STEP" ShowProps="State"
   onclick="AlfaHtmlEvent(this,'Navigate_OpenGraphicView',’@REF
   ',’WF_Diagram');" >
</AlfaHtmlElement>
```

- To navigate to a configured report, use the following parameter, replacing the `null` with the technical name of the relevant configured report:

```
<AlfaHtmlElement ApplyTo="@CURRENT_STEP" ShowProps="State"
   onclick="AlfaHtmlEvent(this,'Navigate_OpenReportView',’@REF'
   ,’NameOfCustomReport');" >
</AlfaHtmlElement>
```

- To navigate to an `ALFA_IDOCUMENT` object as input and open it for download, use the following parameter:

  For security reasons, a blacklist and whitelist concept is available to restrict the uploading and downloading of files with permissible file extensions. For more information, see the section Configuring the Permissibility of Files and Web Links in Alfabet.

```
<AlfaHtmlElement ApplyTo="@CURRENT_STEP" ShowProps="State"
   onclick="AlfaHtmlEvent(this,'Object_Attachment',’@REF',
   null);" >
</AlfaHtmlElement>
```

- If translation to a secondary language is required for the HTML template, you must provide the translation in the HTML specification. Please note that HTML texts cannot be translated via the **Translation Editor** available in Alfabet Expand. Please consider the following:

- If the HTML description is required in additional languages, each language text must be defined within language elements (For example, `<culture_1031>` for English, `<culture_1033>` for German, etc.)
- The element `<culture_xxx>` must be specified as a child of the root element `<xhtml>`. The element `<html>` must be specified as a child of the element `<culture_xxx>`. The element `<html>` contains the HTML specification in the relevant language. For example, to provide information in English and German:


```
<xhtml>
   <culture_1033>
      <html>
         <body>
            <h3>Glossary: Application</h3>
            <p>An application is a fully-functional integrated IT
            product that provides functionality to end users and/or
            to other applications. As such, an application supports
            the business to accomplish its mission. Applications
            operate on a platform made up of hardware and software
            components necessary to run the application.</p>
         </body>
      </html>
   </culture_1033>
   </culture_1031>
      <html>
         <body>
            <h3>Glossar: Applikation</h3>
            <p>Eine Applikation ist ein voll funktionsfähiges,
            integriertes IT-Produkt, das Funktionalitäten für
            Endanwender und/oder für andere Applikationen bietet.
            Eine Applikation unterstützt das Unternehmen bei der
            Zielerreichung. Applikationen werden auf einer Plattform
            betrieben, die aus den für die Ausführung der
            Applikation erforderlichen Hardware- und Software-
            Komponenten besteht.</p>
         </body>
      </html>
   </culture_1031>
</xhtml>
```

- If only the default language is used in the Alfabet interface, the element `<culture_xxx>` is not required and the `<xhtml>` is immediately followed by its child element `<html>`.

Guidelines for the CSS File

A stylesheet must be defined that can be referenced by one or more HTML templates. A default stylesheet labelled `template_styles.css` is stored in the **Internal Document Selector** of the `Showcase` database in the root folder of the **IDOC** folder. Please note the following in order to implement your own stylesheet:

- The CSS file you specify should contain all necessary styles to display the content of the HTML template as well as the styles required to display the buttons in the toolbar of the **Workflow Activities Explorer** functionality (`WFS_Explorer`). If you configure a new style sheet, you must define the class selectors.`wfButtons`,.`wfButton`, `wfButton td`, and.`wfButtonSeparator` in order to visualize the buttons. The information below displays the class selector, a description of its purpose, and the standard definition in the `template_styles.css` file:


```
.wfButtons{
/* styles for the surrounding table that contains all cells containing
individual button tables */
text-align:center;
    color:#989FA9;
    font-family:Arial;
    font-size:12pt;
    font-weight:bold;
    vertical-align:middle;
    margin-bottom:7px;
    margin-left:7px;
    }
.wfButton{
/* styles for the cell of the table that contains all buttons (td
element) */
    background-color:#efefef;
    cursor:pointer;
    vertical-align:middle;
    }
.wfButton td{
/* styles for the individual td elements inside the table that contain a
button (2 elements if icon and text is used, otherwise one) */
    vertical-align:middle;
    padding:3px;
    padding-right: 5px;
    }
.wfButtonSeparator{
/* styles for the horizontal separator between buttons and the remaining
content of the page. */
    width:100%;
    height:2px;
    background-color:#989FA9;
    overflow:hidden;
```

- The CSS file you specify must be uploaded to the **Internal Document Selector** Stylesheets stored in the **Internal Document Selector** must be located in a document folder that is subordinate to the root folder of the **IDOC** explorer. To upload a file to the **Internal Document Selector**, see the section *Uploading Documents and Managing User Permissions to Document Folders in the Internal Document Selector* in the reference manual *User and Solution Administration*.

  Please note that unlike the default CSS file `template_styles.css`, your CSS may not be stored in the root folder of the **IDOC** explorer.

- The CSS file stored in the **Internal Document Selector** available must be referenced in the XML element `<html>` in an XML attribute `<link>`. For example:

```
<xhtml>
    <html>
        <head>
            <meta http-equiv="content-type" content="text/html;charset=utf-
            8" />
            <dscript language="javascript"
            src="alfaembeddedhtml.js"></script>
            <link type="text/css" rel="stylesheet"
            href="IDOC:\CSS\my_style_file.css"></link>
        </head>
        ...
    </html>
</xhtml>
```

Configuring the Maximum Days to Display Finished Workflow Activities

You can specify the maximum number of days that finished, refused, and expired workflow activities shall be displayed in the **Workflow Activities Explorer** functionality. This ensures that obsolete workflows are no longer cluttering the **My Workflow Activities** explorer.

1) Go to the **Presentation** tab and expand the **XML Objects** folder.
2) Right-click the XML object ***SolutionOptions*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) In the XML attribute `ActiveWorkflowHistory`, enter an integer for the maximum number of days that finished, refused, and expired workflow activities shall be displayed in the **Workflow Activities Explorer** functionality..
4) In the toolbar, click the **Save**  button to save the XML definition.

Configuring a Custom Explorer for a Workflow Step

In order to implement a custom explorer for workflows, you must specify the content of the explorer tree as well as the content, layout, and navigation for the reading pane of the view. Unlike the **Workflow Activities Explorer** (`WFS_Explorer`), which only includes workflow activities, you can configure a custom explorer to display nodes with workflows as well nodes with workflow activities. The custom explorer can be simple and display one node with one type of workflow activity such as workflow activities that have been confirmed or may have multiple nodes, each which shows a different type of workflow activity such as workflow activities past their deadline, workflow activities due in a week, workflow activities due in a month, and workflow activities that are not urgent.

The configuration of custom explorers is described in detail in the chapter Configuring Standard Business Functions and Custom Explorers. The information below addresses only particularities about configuring a custom explorer for the workflow context.

The following must be completed in order to implement custom explorer for workflows and/or workflow activities:

1) Configure the workflow template and workflow steps as outlined in this chapter on workflow configuration. All configurations described for workflow templates and workflow steps should be completed before you begin specifying the implementation of the custom explorer. Please note that the following:

- The **Perform**, **Confirm**, **Delegate**, etc. are specified either for the workflow template or for the workflow step via the options (**Perform** button, **Confirm** button, **Delegate** button, etc.) in the **Visualization** section of the attribute grid for the workflow template and/or workflow step. Please note that the workflow step definition will have precedence over the workflow template definition.
- The toolbar buttons specified for the workflow step/workflow template will be displayed above the reading pane in the **Workflow Activities Explorer** functionality.

2) If the custom explorer shall display multiple root-level nodes for the object classes `WorkflowStep`, `Workflow`, or `WorkflowTemplate`, then you must configure object class stereotypes for the object class `GenericReferenceData`. Please note that this is not necessary if the custom explorer includes only one node. For example, to display an explorer with one root-level node with workflow templates and four root-level nodes with different workflow activities such as workflow activities past their deadline, workflow activities due in a week, workflow activities due in a month, and workflow activities that are not urgent, you would specify one object class stereotype for the workflow template node and four other object class stereotypes to represent each type of workflow activity. This must be defined via the **Stereotype** attribute for the object class `GenericReferenceData`. For example:

```
<ClassStereotypes>


    <Stereotype Name="WFExpTemplates" Caption="Workflow Explorer Template"
    CaptionPlural="Workflow Explorer Templates" Comments=""
    HasMandates="false" />

    <Stereotype Name="WFExpExpired" Caption="Workflow Activities Expired"
    CaptionPlural="Workflow Activities Expired" Comments=""
    HasMandates="false" />

    <Stereotype Name="WFExpDueWeek" Caption="Workflow Activities Due in a
    week" CaptionPlural="Workflow Activities Due in a Week" Comments=""
    HasMandates="false" />

    <Stereotype Name="WFExpDueMonth" Caption="Workflow Activities Due in a
    month" CaptionPlural="Workflow Activities Due in a month" Comments=""
    HasMandates="false" />

    <Stereotype Name="WFExpDueAny" Caption="Workflow Activities Not
    Urgent" CaptionPlural="Workflow Activities Not Urgent" Comments=""
    HasMandates="false" />

</ClassStereotypes>
```

3) Specific the custom explorer as described in the chapter Configuring Standard Business Functions and Custom Explorers. Please note the following:

- Each object based on the object class stereotype created for the object class `GenericReferenceData` will be a root node in the custom explorer. The query specified in the **XML Def** attribute of the custom explorer must be specified to search for the objects that are subordinate to each root node. Thus an XML element `ClassEntry` must be created for each object class stereotype with a query that finds the object subordinate to the node. The following example displays the specification for one root-level node:

```
<ExplorerDef
   Name="My Customized Workflow Activities Explorer">
      <Query ClassName="GenericReferenceData:WFExpDueWeek"
      Query="Stereotype ='WFExpDueWeek'" />
         <ClassEntry ClassName="GenericReferenceData:WFExpDueWeek"
         ShowProps="Name" SortProps="Name" >
         <Query ClassName="WorkflowStep"
            Query="SELECT ws.REFSTR, w.CAPTION +'-'
            +ISNULL(dbo.SCF_GetObjectFullName(w.SOURCEARTIFACT),ISNU
            LL(dbo.SCF_GetObjectFullName(ws.ARTIFACT),'No/Unknown
            Artifact'))
            FROM WORKFLOWSTEP ws, WORKFLOWTEMPLATE wt, WORKFLOW w,
            RELATIONS rel, PERSON per
            WHERE ws.WORKFLOW = w.REFSTR
            AND w.TEMPLATE = wt.REFSTR
            AND ws.STATE ='Pending'
            AND rel.FROMREF = ws.REFSTR
            AND rel.TOREF = per.REFSTR
            AND rel.PROPERTY ='SignOnUsers'
            AND per.REFSTR = @CURRENT_USER
            AND DATEDIFF(dd,GETDATE(),ws.DUEDATE) &lt; 8
            AND DATEDIFF(dd,GETDATE(),ws.DUEDATE) &gt; 0
            />
         <Query ClassName="GenericReferenceData:WFExpDueMonth"
         Query="Stereotype ='WFExpDueMonth'" />
      </ClassEntry>
      <ClasEntry...>
      ...
      </ClassEntry>
   <AlfaExplorerNode ClassName="WorkflowStep" UpdateViews=""
   NodeView="GraphicView:CustomExplorerHtmlView" />
</ExplorerDef>
```

- Specify the presentation object that is used for the right pane when a node is clicked. Whereas this is implicit in the configuration of the **Workflow Activities Explorer** (`WFS_Explorer`), you must explicitly specify the graphic view `CustomExplorerHtmlView` in the custom explorer definition in order to ensure that the correct presentation object is available. This ensures that the relevant HTML template configuration is displayed for the workflow activities. If the custom explorer is configured to display

workflow activities. This is specified in the **XML Def** attribute of the custom explorer via the XML element `AlfaExplorerNode`. For example:

```
<AlfaExplorerNode ClassName="WorkflowStep"
UpdateViews=""NodeView="GraphicView:CustomExplorerHtmlView"/>
```

4) Configure the HTML template to display the content, navigation, and layout of the data for the workflow steps. The HTML must be XML-conform HTML, compliant with HTML 5, and use standard HTML tags. For more information about how to specify the HTML template as well as the proprietary HTML elements available for the HTML template, see the section Guidelines for HTML Template Syntax in Workflows.

5) The HTML template must be assigned to the workflow template (if relevant for all or some of the workflow steps) or to the relevant workflow step via the **HTML Template for Step** attribute of either the workflow template or relevant workflow step.

If the **HTML Template for Step** attribute is defined for the workflow template, the HTML template will be applied to all workflow steps unless an HTML template is explicitly defined for a workflow step, in which case the workflow step definition will take precedence over the workflow template definition. All HTML templates listed below the **Visualization Items** node are displayed in the drop-down list for the **HTML Template for Step** attribute.

6) A CSS file must be configured to specify the styles for the HTML template. A reference to the CSS file must be included in the specification of the HTML template. The CSS file that you reference in the HTML template must be stored in the **Internal Document Selector**. For more information about configuring the CSS file, see the section Guidelines for the CSS File.

7) The custom explorer must be assigned as a business function to the user profile or implemented in a guide view to make it accessible to the user community.

Configuring and Visualizing a Workflow in a Diagram

A diagram capability is available for workflow templates that allows you to configure the workflow template in a diagram format. This allows you to visualize the workflow as you configure it, thus simplifying the task of sequencing the workflow steps. You can click any workflow step visualized in the diagram and configure its attributes in the attribute window.

You can carry out the following in the diagram capability:

- Create and define workflow steps for a workflow template.
- Define the sequence of the workflow steps including next workflow steps, expired workflow steps, and refused workflow steps.
- Create pre- and post-conditions as well as actions for a workflow step.
- Graphically design the layout of the workflow steps and the connector items representing their sequence.
- Graphically design the visualization of the workflow by adding a pool with swim lanes, shapes, text, and color.

There is no Undo/Redo function in the workflow diagram capability. If you delete a workflow step, the workflow step, any workflow step sequence definitions, and any pre- or post-conditions as well as pre- or post-actions will be irrevocably deleted from the database.

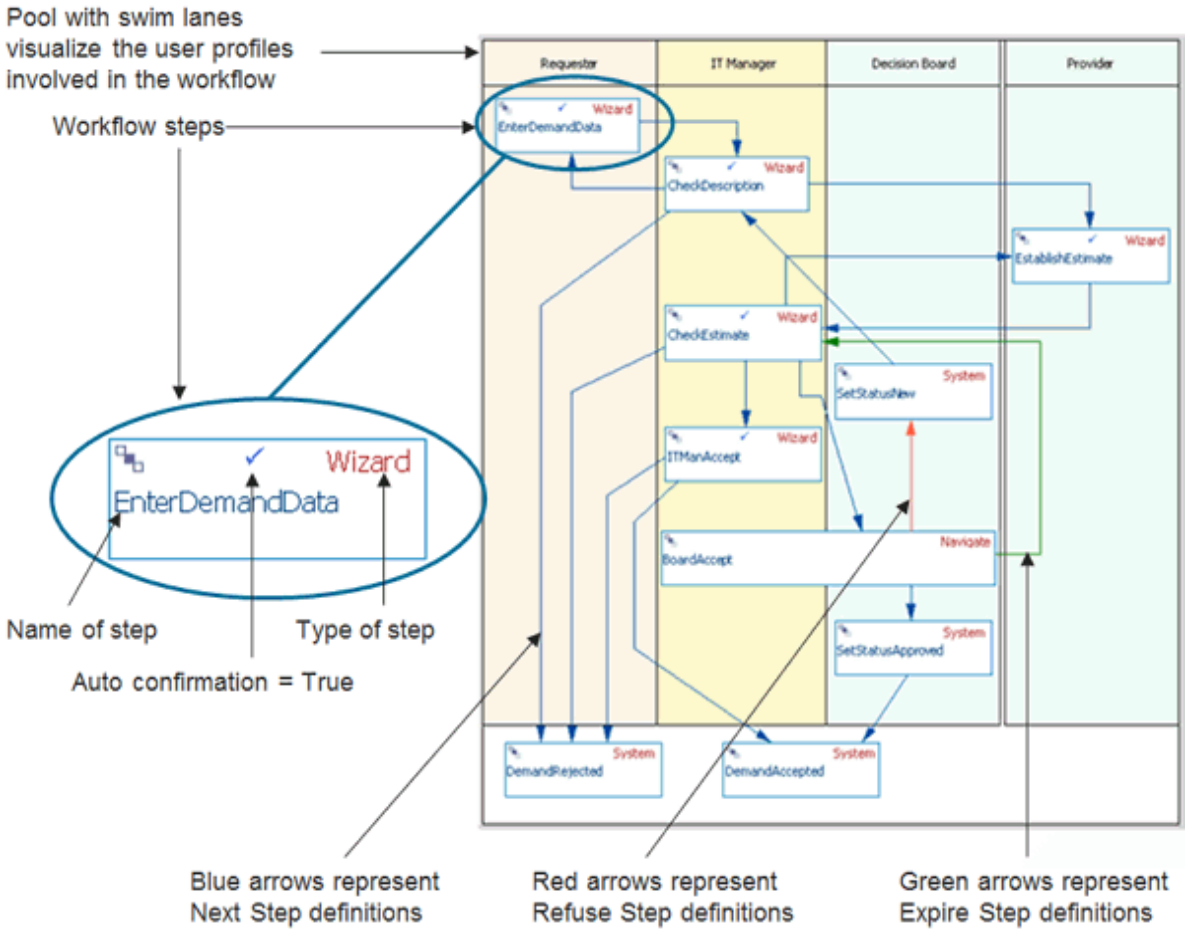


FIGURE: Example of a workflow template configured and visualized in a diagram

The workflow diagram you design in Alfabet Expand can be viewed by the workflow administrator as well as the workflow owners of all workflows associated with the workflow template. Additionally, users processing a workflow step that they are responsible for can also see the workflow diagram in order to understand the workflow step in the context of the entire workflow. Once a workflow step has been performed and confirmed by a user, a timestamp will be displayed on the workflow step in the workflow diagram displayed in the Alfabet interface.

The timestamp on a workflow step in a workflow diagram will display the hour and minutes when the workflow step was confirmed. Users using US culture settings will see AM or PM appended to the timestamp.

If you do not explicitly design a diagram, a rudimentary diagram will be automatically generated. Once the **Workflow State** attribute of the workflow template has been set to `Active`, the diagram will be displayed in the **Workflow Diagram** page view, which is available in the object profile of the respective workflow. The workflow step that is being performed for the selected workflow at the time that the **Workflow Diagram** page view is accessed will be highlighted in the diagram. For more information about using the diagram, see the section *Understand the Course of the Workflow* in the reference manual *User and Solution Administration*.

The following information is available about the configuration and design of workflow diagrams:

- Opening the Workflow Diagram Capability
- Specifying the Diagram Attributes
- Creating a Pool with Swim Lanes for the Diagram
- Adding a Workflow Step to the Diagram
- Deleting a Workflow Step
- Specifying the Sequence of Workflow Steps in the Diagram
- Editing an Existing Workflow Diagram
- General Guidelines for Graphically Refining the Diagram
- Visually Refining the Display of Connection Items
- Adding a Graphic Element to a Diagram
- Deleting a Graphic Element from the Diagram
- Adding a Graphic Image (GIF, BMP, etc.) to a Diagram
- Moving Diagram Elements in a Diagram
- Changing the Size of a Diagram Element
- Simultaneously Changing the Size of Multiple Diagram Elements
- Aligning Diagram Elements in a Diagram
- Changing the Background Color of Graphic Elements

Opening the Workflow Diagram Capability

To work with the diagram capability:

1) In the **Workflow** tab, right-click the **Workflow Templates** icon at the top of the explorer and select **Create New Workflow Template**. The new workflow template is displayed in the explorer.

   Alternatively, you can copy an existing workflow template if the **Workflow State** attribute is set to `Plan` and select **Create New Workflow Template as Copy**. The existing workflow template including workflow steps, pre-conditions, etc. will be copied. The existing diagram will also be copied and can be modified, as needed.

2) Edit the attributes as described in the section Creating a Workflow Template.

3) To open the diagram capability in order to configure the workflow template from scratch or modify an existing workflow template, right-click the workflow template in the explorer and select **Open Diagram**. A tab for the workflow template will open in the workspace area.

   The **Open Diagram** option is only available for workflow templates for which the **Workflow State** attribute is set to `Plan`. You cannot open the diagram capability in Alfabet Expand for a workflow template for which the **Workflow State** attribute is set to `Active` or `Retired`.

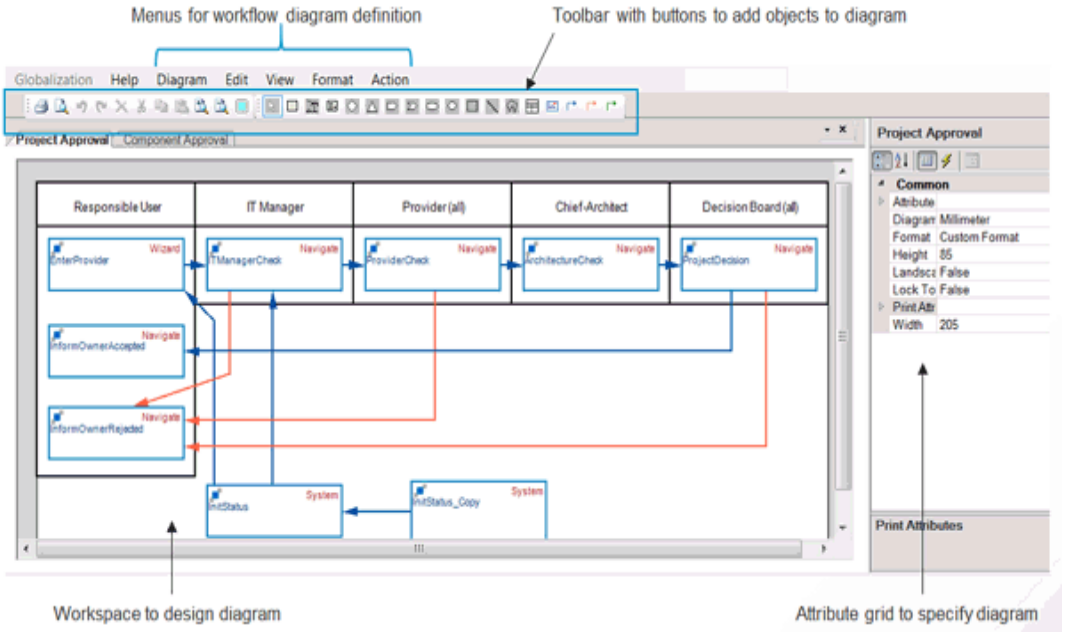


FIGURE: Workflow diagram designed in the workflow diagram capability

4) Click in the diagram workspace. The menus **Diagram**, **Edit**, **View**, **Format**, and **Action** are displayed in the menu bar as well as a toolbar with objects that can be added to the diagram are displayed in the toolbar when the diagram is active. You will also see the attribute window that allows you to define the basic attributes of the diagram. To set up the diagram in order to configure the workflow template, see the section Specifying the Diagram Attributes.

For an overview of the available functionalities in the menus as well as the attributes in the diagram attribute window, see *Workflow Diagram Capability* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

5) Click the **Save** button to save the diagram.

Specifying the Diagram Attributes

Before you can begin to design a workflow diagram, you must set up the diagram and define the diagram properties including, for example, the format and orientation of the diagram as well as the inclusion of drawing elements such as rulers or a grid. Additionally, you can define print attributes such as margins and print mode.

To display the attribute window for the grid, you must click in the diagram workspace. If a diagram already exists, ensure that you do not click an existing object or shape in the diagram. If the diagram is filled with a pool, for example, you may need to click in the grey area outside of the diagram.

An explanation of all attributes is available in the table *Diagram Property Window* in the section *Workflow Diagram Capability* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

To specify the diagram attributes:

1) In the diagram attribute window, define the relevant attributes for the diagram. Some of the most important are listed below:

- **Format**: Enter one of the following attributes to determine the paper format to be used as the default for diagrams. Permissible options are:
- `A4 210x297`: Standard A4 paper size
- `A3 297x420`: Standard A3 paper size
- `A2 420x594`: Standard A2 paper size
- `A1 594x840`: Standard A1 paper size
- `A0 840x1186`: Standard A0 paper size
- `Letter`: US Letter paper size (216x279 cm or 8.5x11 in)
- `Legal`: US Legal paper size (216x356 cm or 8.5x14 in)
- `11x17`: US 11x17 paper size (279x432 cm)
- `Custom Format`: If you select this format, you must define the **Height** and **Width** attributes
- **Landscape**: Enter `True` if the diagram should use the specified paper size in landscape mode. Enter `False` if the diagram should use the specified paper size in portrait mode.

2) Click the + symbol next to **Attributes** to expand the table and define diagram options that are useful for designing the diagram. For example, enter values for the following useful options:

- **Center Connection**: Enter `True` if the end points of connection items should be determined by the user via the drag-and-drop action. Enter `False` if the end points of connection items should always be drawn from the center of one diagram object to the center of the other diagram object.
- **Draw Grid**: Enter `True` if a grid should be displayed. Enter `False` if a grid should not be displayed. The grid version can be defined in the attribute **Grid Mode**.
- **Elbowed Connection**: Enter `True` if the connection items (information flows, sequence flows, etc.) should be drawn as an elbow line (with 90° angle). Enter `False` if the connection items should be created as a straight line.
- **Rulers**: Enter `True` if a rulers should be displayed. Enter `False` if rulers should not be displayed.

3) After you have finished defining the necessary object class properties, click in the diagram workspace to reactivate the menu bar and toolbar.

4) Click the **Save** button to save your definitions.

Creating a Pool with Swim Lanes for the Diagram

You can optionally create one or more pools in order to provide additional detail in the diagram such as which user profiles responsible for certain workflow steps. (Pool and swim lanes are derived from Business Process Modeling Notions). The pools can be added to the diagram at any time during the design process, however it is easiest to add the them before workflow steps have been added to the workflow template.

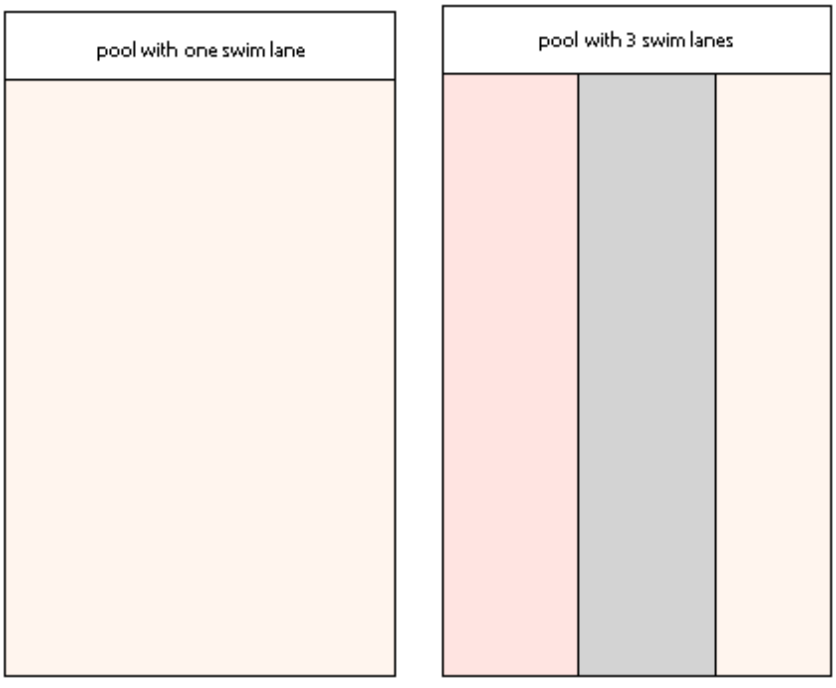


FIGURE: Example of different possibilities of designing a pool for a diagram

A pool can represents major participants in the workflow process and swim lanes could be visualized to depict different categories of the workflow process.

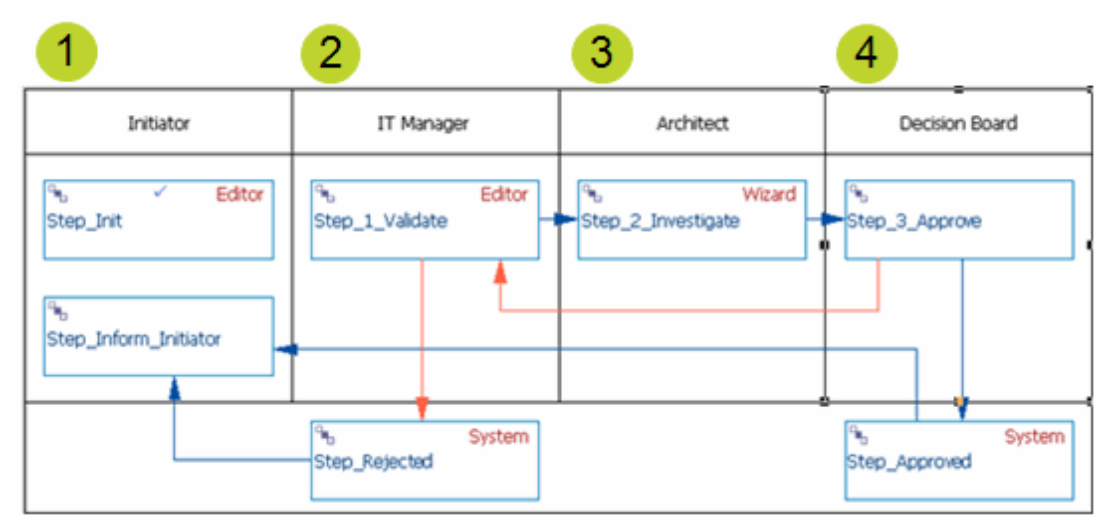


FIGURE: Simple workflow diagram with four pools representing the four user profiles

Although swim lanes can also be added in order to increase the amount of detail, it is advisable to keep the diagrams simple. Typically, however, multiple pools would be added to the diagram for each participant in the workflow. Therefore, their implementation will not be covered in the description below.

To create a pool with swim lanes for the selected workflow template:

1) In the toolbar, click the **Pool** icon and click a blank space in the diagram.
2) In the diagram, drag the pool to the required size.
3) Click in the pool to display the attribute window. Modify the necessary object class properties:

- **Background Color**: Define the background color of the pool
- **Grid Pen Style**: Define the style of the lines for the lanes in the pool.
- **Grid Pen Width**: Define the width of the lines for the lanes in the pool.
- **Pen Attributes**: Define the attributes for the lines of the pool title.
- **Pool Text**: Enter the text that should be displayed for the pool title.
- **Pool Text Attribute**: Define the attributes for the text displayed for the pool title.
- **Pool Text Height**: Define the height of the text displayed for the pool title.
- **Title Color**: Define the color of the pool title.
- **Title Position**: Define the position of the pool title

4) To add swim lanes to the pool, edit the following object class properties:

- **Lanes**: Click the **Browse** button to create swim lanes to display in the pool. In the **Edit Object Collection** dialog box that opens, click the **Add** button. You will see the entry `NewColumn` in the **Members** attribute. Edit the object class properties of this new column in the **Properties** attribute to the right. Continue this procedure until you have the number of swim lanes you want. Click **OK**.
- **Lane Text Attribute**: Define the attributes for the text displayed for the swim lane title(s).
- **Lane Text Height**: Define the height of the text displayed for the swim lane title.

5) To adjust the size of the pool or the size of individual swim lanes, drag the line separating the lanes to the appropriate width.

6) If the pool is covering existing workflow steps, right-click the pool and select **Send to Back**. The pool is moved to the background and any existing workflow steps are on top of the pool and can be dragged to their appropriate position in the diagram.

7) Click the **Save**  button to save your definitions.

Adding a Workflow Step to the Diagram

A workflow step is an activity or task in a workflow that must be performed. The workflow template may have as many workflow steps as needed to complete the relevant task. Standard and custom editors, wizards, page views, configured reports, and object profiles may be implemented in the workflow step so that users can complete the task. A workflow step may entail entering, modifying or reviewing data, or a workflow step may be configured to be automatically executed by the Alfabet system. If the workflow step is not automatically executed by the system, one or more users may be defined as responsible to perform the workflow step. A user may delegate a workflow step to another user or refuse a workflow step. Depending on the configuration of the workflow step, a user may be required to confirm a workflow step before the workflow can advance to the next workflow steps, or the confirmation of the workflow step may occur automatically. A workflow step may also have one or more associated workflow step actions which allow various operations to be performed when the workflow step is entered, exited, refused, or expired. Additionally, each workflow step may have one or more pre-conditions or post-conditions that must be fulfilled in order to enter or exit the workflow step.

You can add a workflow step to the diagram via the workflow step button in the toolbar. After a workflow step is added to the diagram, you can graphically modify it by moving it, changing its size, or adding text. The color of the workflow steps cannot be changed.

To add a workflow step to the diagram:

1) In the toolbar, click the workflow step  icon that you want to add to the diagram and click a blank space in the diagram. A workflow step has been added to the diagram.

2) Edit the attributes as described in the section Creating a Workflow Step. The attributes **Expiration Step**, **Next Steps**, and **Refuse Step** can be ignored in the attribute window. The means to define the sequence of steps within the diagram capability is described in the section Specifying the Sequence of Workflow Steps in the Diagram.

3) Repeat the procedure described above until all workflow steps have been created. You can move the workflow steps to their approximate positions in the diagram or pools by via drag-and-drop.

4) For each workflow step created, you can right-click the workflow step and select either:

- **Add New Pre-Condition**
- **Add New Post-Condition**
- **New Action On Entered Step**
- **New Action On Refused Step**
- **New Action On Expired Step**
- **New Action On Exited Step**

For detailed information about the definition of conditions and actions, see the following sections:

- Configuring Pre- and Post-Conditions for a Workflow Step
- Configuring Automatic Property Updates for a Workflow Step
- Configuring Events to Be Triggered for a Workflow Step
- Configuring a Workflow Step to Trigger a Subordinate Workflow
- Configuring Email Notifications for Workflows

5) Click the **Save** button to save your changes.

Deleting a Workflow Step

You can delete a workflow step from the workflow configuration. The workflow step will be deleted from the database.

There is no Undo/Redo function in the workflow diagram capability. If you delete a workflow step, the workflow step, any workflow step sequence definitions, and any pre- or post-conditions as well as pre- or post-actions will be irrevocably deleted from the database.

Specifying the Sequence of Workflow Steps in the Diagram

You can configure and visualize the sequence of the workflow steps. When you define the sequence of workflow steps, you must define all possible subsequent workflow steps for each workflow step as well as the workflow steps that follow when a workflow step is refused by a user or the deadline to complete the workflow step has expired. The possible next workflow steps will depend on whether pre-conditions that have been defined for any subsequent workflow step are fulfilled for the object that is the target of the workflow step. Therefore, you may opt to define pre-conditions for each workflow step before you define the sequence of workflow steps.

For detailed information relevant to the configuration of workflow steps, see Defining the Sequence of the Workflow Steps.

The diagram capability allows you to define next workflow steps, expired workflow steps, and refused workflow steps by selecting the relative connection item in the toolbar and dragging the connection item from the source workflow step to the target workflow step.

The following connection items are available:

Icon	Purpose
To Next Step	For each workflow step (including the initial workflow step and the first workflow step), define all potential workflow steps that may be executed next for each workflow step. The last workflow step in the workflow should not have a **To Next Step** connection item defined.
To Refuse Step	Define the workflow step that is to be triggered if the current step is refused by the user. A refused workflow step should be configured for each workflow step for which the attribute **Refuse Button** is set to `True`.
To Expire Step	Define the workflow step that is to be triggered if the current step is not performed in the period defined in the **Performance Duration** attribute.

To define the connection items between workflow steps:

1) In the toolbar, click the relevant connection item icon for the workflow step you want to define. The workflow step that you are define will be the source workflow step in the connection item definition.
2) Click the source workflow step in the diagram and drag the connection to the target workflow step.
3) Click the **Save** button to save your definitions.
4) Repeat this procedure as required for all workflow steps.

You can now refine the display of the connection items, as needed. For more information, see Visually Refining the Display of Connection Items.

Editing an Existing Workflow Diagram

You can edit an existing workflow diagram and add or remove workflow steps or connection items between workflow steps. The definitions you make in the workflow diagram will be included in the workflow template definition.

If the workflow diagram is not open, ensure that the **Workflow State** attribute of the workflow template is set to `Plan`. Right-click the workflow template in the explorer and select **Open Diagram**. A tab for the workflow template will open in the workspace area. Edit the diagram as described above.

The **Open Diagram** option is only available for workflow templates for which the **Workflow State** attribute is set to `Plan`. You cannot open the diagram capability in Alfabet Expand for a workflow template that is active or retired. However, if the workflow diagram is already displayed in the workspace area and the **Workflow State** attribute of the workflow template is set to `Active` or `Retired`, you will be able to edit the workflow diagram by adding workflow steps and connection items. In this case, the workflow template does not have to be changed to `Plan`.

General Guidelines for Graphically Refining the Diagram

The following information about how to refine the visualization of the diagram:

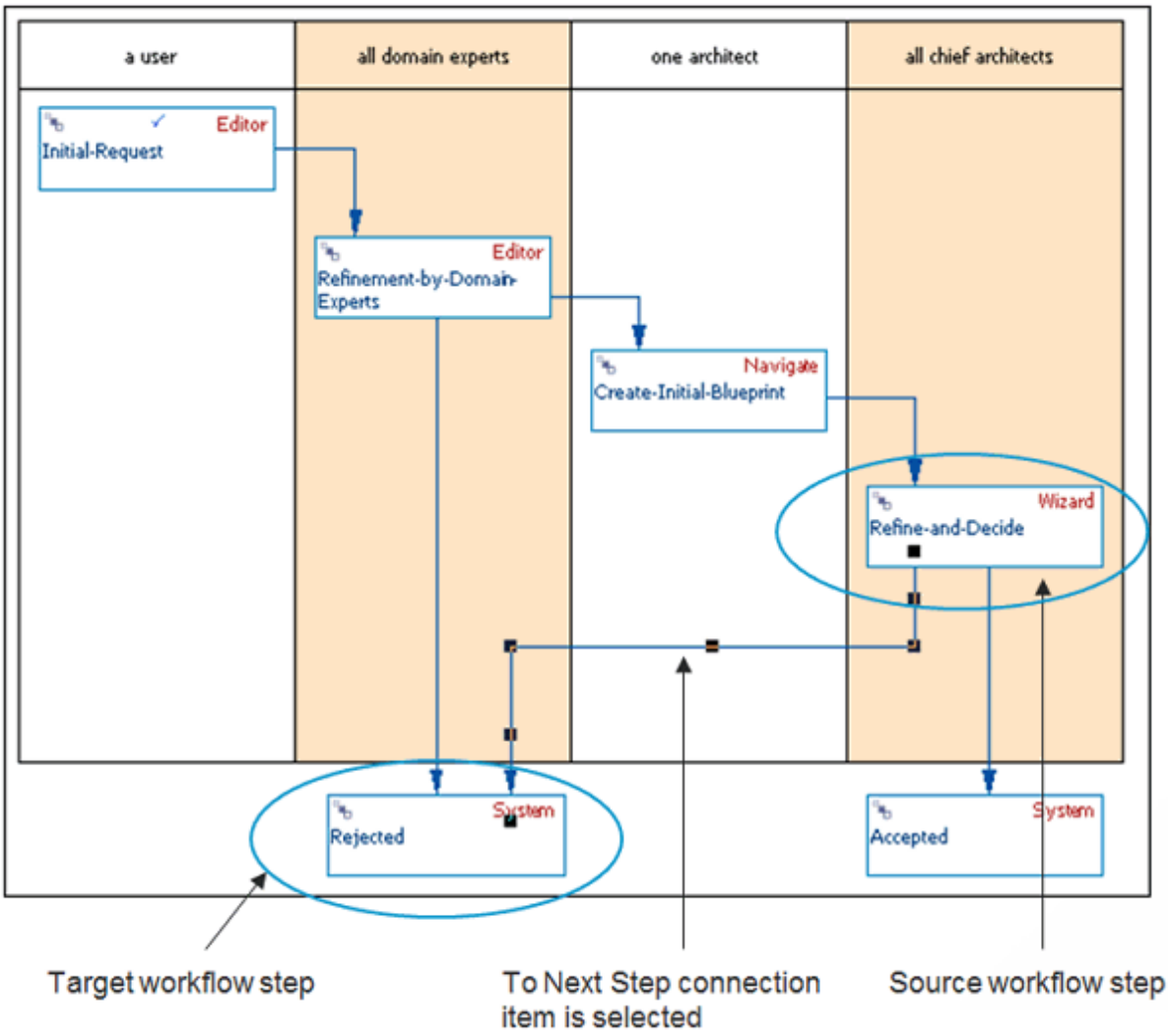
- Visually Refining the Display of Connection Items
- Adding a Graphic Element to a Diagram
- Deleting a Graphic Element from the Diagram
- Adding a Graphic Image (GIF, BMP, etc.) to a Diagram
- Moving Diagram Elements in a Diagram
- Changing the Size of a Diagram Element
- Simultaneously Changing the Size of Multiple Diagram Elements
- Aligning Diagram Elements in a Diagram
- Changing the Background Color of Graphic Elements

Visually Refining the Display of Connection Items

The attributes available in the **Information Flow Attribute** attribute are configured by your solution designer in the XML object ***DiagramInformationFlowDef*** in the configuration tool Alfabet Expand. For more information, see the section Configuring the Visualization of Connection Items and Subordinate Object in Diagrams in the reference manual *Configuring Alfabet with Alfabet Expand*.

Sometimes you may have a confusing network of connection items or you may want to move the connection items so that they are easier to understand graphically. For example, you can define whether the connection items are displayed as elbowed or straight connections.

When you create new connection items, you should first define whether you want elbowed connections or straight connections to be automatically created. To do so, click in the diagram to open the attribute window. In the **Elbowed Connection** attribute, select `True` to create elbowed connections and select `False` to create straight connections. For more information about the attributes in the attribute window, see the table *Diagram Property Window* in *Workflow Diagram Capability* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.



Sometimes connection items may appear entangled or are hard to see. To understand where a connection item starts and ends, click the connection item to see which diagram elements are the source and target objects of the connection item. To move a connection item, grab a center handle and drag the connection to the place you want it to be visualized. Adjust all sections of the connection item until the connection item is correctly positioned. In some cases, you may need to right-click an element in the diagram and select **Move to Front** or **Send to Back** to ensure that a diagram element is adequately visualized. If a connector item becomes mis-shapen by an erroneous drag-and-drop action, you may need to delete the connector item and redefine it.

Adding a Graphic Element to a Diagram

You can add any graphic element that is located in the toolbar in the diagram capability. After a graphic element is placed in the diagram, you can later modify it including moving it in the diagram, changing its size, and adding color, borders, shadow, text, etc. To add a graphic image, see Adding a Graphic Image (GIF, BMP, etc.) to a Diagram.

To add a graphic element to the diagram:

1) In the toolbar, click the graphic element that you want to add to the diagram and click a blank space in the diagram. The graphic element appears on the spot that you click.

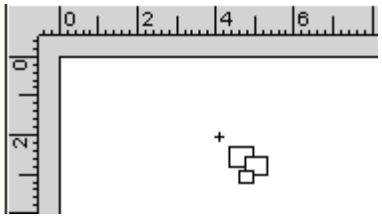


FIGURE: Cursor shows that the toolbar item is selected

To draw a polygon, select the polygon icon. Click in the diagram and drag-and-drop to create a line. Select the CTRL key, click on the line at the point where you want to create an angle, and drag-and-drop to create the desired polygon shape.

2) You can further define the graphic element by changing its color, size, position, or adding text. For more information, see General Guidelines for Graphically Refining the Diagram.

3) Click the **Save** button to save your definitions.

4) Repeat the process in order to add other graphic elements to the diagram, as needed.

Deleting a Graphic Element from the Diagram

There is no Undo/Redo function in the workflow diagram capability. If you delete a workflow step, the workflow step, any workflow step sequence definitions, and any pre- or post-conditions as well as pre- or post-actions will be irrevocably deleted from the database.

There are many different ways to delete one or more graphic elements from the diagram:

You can select several objects at once by holding down the CTRL key while selecting the objects.

To delete one or more graphic elements from the diagram:

- Select the graphic element and click the DELETE key, or
- Select the graphic element and right-click the workspace and select **Delete Selected (Graphics)**, or
- Select the graphic element and click CTRL + G.

Adding a Graphic Image (GIF, BMP, etc.) to a Diagram

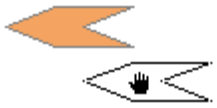
Any graphic files that you want to add to a diagram must first be uploaded to the **Internal Document Selector**. For more information about uploading graphic files to the **Internal Document Selector**, see the section *Uploading Documents and Managing User Permissions to Document Folders in the Internal Document Selector* in the reference manual *User and Solution Administration*. Graphic files are uploaded to the **Internal Document Selector** in the same manner as documents. You can upload a BMP, GIF, JPG, JPE, PNG, DIB, or ICO (icon) file.

To add a graphic image to a diagram:

1) In the toolbar, click the **Bitmap** icon and click a blank space in the diagram. The **Internal Document Selector** will open.
2) Navigate to the directory where the graphic file is located. Click the file and click **OK**. The graphic image is visible in the diagram.
3) To adjust the size of the graphic image, click a corner handle and drag it to the appropriate size.
4) Click the **Save** button to save your changes.

Moving Diagram Elements in a Diagram

You can move any element that is in the diagram.



To move the element to a different spot in the diagram, click the diagram element and drag to the appropriate place.

To move several elements at once, click the diagram elements while holding the CTRL key. Move the group to the appropriate place. After the group has been moved, click each diagram element to deselect it.

Changing the Size of a Diagram Element

You can change the default size of any elements that are in the diagram.

FIGURE: Increasing the size of a rectangle

To increase the size of a diagram object or graphic element, click a corner handle and drag it to the appropriate size.

Simultaneously Changing the Size of Multiple Diagram Elements

You can simultaneously define the size for multiple elements in the diagram.

To define the size of multiple diagram elements:

1) Press the SHIFT key and select all diagram elements in the diagram that should be the same size.
2) In the menu bar, select **Format** > **Size**. The **Size** dialog box opens.
3) Enter values, as needed:

   Width Box:

   - **No change**: The width of the currently selected diagram elements will not be changed.
   - **Shrink to Smallest:** The width of the selected diagram elements is based on the diagram element smallest in width.
   - **Grow to Largest**: The width of the selected diagram elements is based on the item largest in width.
   - **Width**: Enter a size in millimeter to define the width of the selected diagram elements.

   Height Box:

   - **No change**: The height of the currently selected diagram elements will not be changed.
   - **Shrink to Smallest:** The height of the selected diagram elements is based on the diagram element smallest in height.
   - **Grow to Largest**: The height of the selected diagram elements is based on the diagram element largest in height.
   - **Height**: Enter a size in millimeter to define the height of the selected diagram elements.
4) Click **OK**. The size of each selected diagram element adjusts to the size defined.
5) Click the **Save**  button to save your changes.

Aligning Diagram Elements in a Diagram

You can simultaneously define the alignment for multiple elements in the diagram.

To align diagram elements:

1) Press the SHIFT key and select all diagram elements in the diagram you want to align with one another.
2) In the menu bar, select **Format** > **Align**. The **Alignment** dialog box opens.
3) Enter values, as needed:

   Horizontal Box:

   - **No change**: The currently selected diagram elements will not be changed on the horizontal line.
   - **Left sides**: All selected diagram elements are left-aligned according to the selected diagram element closest to the left edge of the diagram.
   - **Centers**: All selected diagram element are left-aligned according to the selected diagram element closest to the left edge of the diagram.
   - **Right sides**: All selected diagram elements are right-aligned according to the selected diagram element closest to the right edge of the diagram.

- **Space Equally**: The distance between the selected diagram elements is equal.
- **Center in diagram**: The selected diagram elements will be placed in the middle of the diagram without changing its horizontal line.

- **Vertical Box:**
  - **No change**: The selected diagram elements will not be changed.
  - **Tops**: All selected diagram elements are top-aligned according to the selected diagram element closest to the top edge of the diagram.
  - **Middles**: All selected diagram elements are center-aligned according to average middle axis.
  - **Bottoms**: All selected diagram elements are bottom-aligned according to the selected diagram element closest to the bottom edge of the diagram.
  - **Space equally**: The distance between the selected diagram elements is equal.
  - **Center in diagram**: The selected diagram element will be placed in the middle of the diagram without changing its vertical line.

4) Click **OK**. The diagram elements are aligned as defined.

5) Click the **Save**  button to save your changes.

Changing the Background Color of Graphic Elements

As you design your diagram, you may want to define color a for various graphic elements in order to make the information more accessible and visually appealing.

The color of workflow steps is preconfigured for Alfabet and cannot be altered.

It is recommended that color assignments are made using explicit color codes or Web colors (and NOT system colors as these will result in differences in display from one client computer to another).

To change the background color of a graphic element in the diagram:

1) Click the graphic element that you want to define a color for to call up its attribute window.
2) Click in the **Background Color** attribute and then click the arrow to open the drop-down box of colors.
3) Click a color in either the `Custom`, `Web`, or `System` tabs. The color fills the selected graphic element in the diagram.

Depending on the kind of graphic element you are currently working with, you can also define other object class properties. For example, for the double arrow rectangle shown above, you can define a double border or choose to point the arrow in the other direction.

4) Click the **Save**  button to save your changes.

Editing an Active Workflow Template

In some cases, you may need to make changes to the configuration of a workflow template after it has been made active and available to the user community. To edit the workflow template, you must return the state of the workflow template to `Plan` in the **Workflow State** attribute. Workflow templates may not be edited if the **Workflow State** attribute of the workflow template is set to either `Active` or `Retired`.

Any modifications made to the workflow template will not impact workflow steps that have already been initiated, performed, or confirmed. It may be necessary to re-initiate all workflows if a workflow template is modified after it has been activated.

If changes are required to a workflow template that is already active in the production environment, see the section Creating a Migration Definition to Update Running Workflows.

To make the workflow template editable, right-click the workflow template and select **Set Workflow State to Plan**. Once the workflow template displays `Plan` in the **Workflow State** attribute, you can edit the workflow template. Any changes you make to the workflow template must be saved by clicking the

Save  button in the toolbar.

Deleting a Workflow Template

Workflow templates may be deleted in Alfabet Expand if the **Workflow State** attribute of the workflow template is set to either `Plan` or `Retired`. A workflow template should not be deleted if there are running workflows. If a workflow is deleted with running workflows, all workflow instances will also be deleted. Before a workflow template is deleted, you should check to see if the workflow template is implemented in other workflow templates. To do so, right-click the workflow template and click **Show Usage**. A window will be displayed showing the configuration objects using the selected workflow template. For more information regarding the **Show Usage** functionality, see the section Using the Show Usage Functionality in the chapter Getting Started with Alfabet Expand.

To delete the workflow template, right-click the workflow template in the explorer and select **Delete**. The deleted workflow template will no longer be displayed in the **Workflows Administration** functionality that can be accessed in the user interface via an administrative user profile.

Validating the Workflow Template

Once you have completed the configuration of the workflow template, it must be validated. You can validate the workflow template at any time by clicking the **Validate** option available for the workflow template. The validation is also automatically executed when you set the workflow template's **Workflow State** attribute to `Active`. When the workflow template is validated, errors will be checked in the workflow template including invalid or missing configurations for wizards, views, and editors specified in the workflow template. Errors in the workflow template definition that may result in failure during execution of the workflow will be reported in a dialog. All errors that are reported must be corrected before the **Workflow State** attribute of the workflow template can be set to `Active`.

A validation will also be executed for running workflows each time a workflow is initiated. If configurations that are relevant to the workflow template are no longer available, an error message

will be displayed to the user stating that the workflow configuration is invalid and that the workflow administrator should be contacted.

To validate all workflow templates in a bulk operation, right-click the **Workflow Templates** explorer root node and select **Validate All**. The validation will review all workflow templates for which the **Workflow State** attribute is set to `Plan` or `Active`. If an error is found in the configuration of a workflow template, an information message will be displayed. If the error was found in a workflow template for which the **Workflow State** attribute was set to `Plan`, a **Set Plan to Workflow State** button will be displayed. An erroneous workflow template must have the **Workflow State** attribute set to `Plan` to be corrected.

To validate a selected workflow template, right-click the workflow template  and select **Validate**.

Changing the Workflow State and Activating/Deactivating the Workflow Template

Workflows can only be initiated for a workflow template for which the **Workflow State** attribute has been set to `Active`. A workflow cannot be initiated for any workflow template that has the **Workflow State** attribute set to `Retired` or `Plan`. Therefore, once the configuration of the workflow template is complete and the workflow template has been validated, you must make it available to the user community.

If you attempt to activate the workflow template and have not yet validated the workflow template, the validation will be performed automatically. If errors exist in the workflow template definition, you will not be able to set the **Workflow State** attribute to `Active`. An error message will be displayed listing the errors that must be corrected before the workflow template can be set to `Active`. For more information about validating workflow templates, see the section Validating the Workflow Template.

To change the workflow state of a selected workflow template:

1) Click the workflow template to open the attribute window.

2) Click the **Save**  button in the toolbar to ensure that all modifications made to the workflow template are saved to the Alfabet database.

3) Right-click the workflow template and select **Validate** to ensure that existing errors will not prevent the workflow template from being activated.

4) Right-click the workflow template and select **Set Workflow State to Active** to activate the workflow template so that workflows can be initiated. The workflow template will now be displayed in the **My Workflows** functionality of all users that have been defined as permitted users of the workflow template.

If you need to modify the workflow template, you must set the **Workflow State** attribute to `Plan`.

To change the workflow state of all workflow templates in a selected workflow group: Right-click the workflow group and click one of the following:

- **Set Workflow State to Active** to activate the workflow templates so that workflows can be initiated.

- **Set Workflow State to Plan** to deactivate the workflow templates so that they can be edited.
- **Set Workflow State to Retired** to deactivate the workflow templates and disable the editing capability.

The state of all workflows directly located in the workflow group and all workflows located in sub-groups of the selected group will be set to the selected state.

Please note that if a workflow template in the workflow group has errors and is invalid, an error message will be displayed for the workflow template. Any workflow templates following the invalid workflow template in the explorer will not be updated.

To activate all workflow templates in all workflow groups. Right-click the **Workflow Templates** node at the top of the explorer and click one of the following:

- **Set Workflow State to Active** to activate the workflow templates so that workflows can be initiated.
- **Set Workflow State to Plan** to deactivate the workflow templates so that they can be edited.
- **Set Workflow State to Retired** to deactivate the workflow templates and disable the editing capability.

Creating a Migration Definition to Update Running Workflows

It may be that a workflow template must be changed even though running workflows based on the workflow template are in progress. For example, if new guidelines have been implemented by the enterprise for cost management, it may be necessary to update the enterprise's workflow templates targeting budgeting processes as well as update any running workflows based on the modified workflow templates. In the context of the workflow migration definition, you can thus edit the workflow template's attributes, add, delete, or modify its workflow steps and/or workflow step actions, etc.

Depending on the extent of changes that need to be made, you can do one of the following:

- For only minor modifications to a workflow template (such as changes to existing pre- or post-conditions, editors/wizards for data capture, etc.) you can modify the existing workflow template without the need to create a new workflow template.
- For major modifications to a workflow template (such as the addition or deletion of a workflow step), you should create a new workflow template based on the existing workflow template and make the necessary changes to the new workflow template.

If the name of a custom editor, class setting, or user profile is changed and that configuration object is implemented in a workflow template, the changed name will be correctly updated in the workflow templates where that configuration object is implemented. A warning message will be generated during the migration process if a configured report, wizard or text template that is used in a workflow template has been renamed. **The modified names of user profiles, configured reports, wizards, or text templates that are referenced in workflow templates must be explicitly retrofitted for running workflow instances by configuring an appropriate workflow migration definition.**

The **Workflow State** attributes of the source workflow template (the workflow template to be changed) and the target workflow template (the updating workflow template), must be set to `Plan`. For the workflow

migration definition, each workflow step in the source workflow template should be mapped to the correct workflow step in the target workflow template. If no mapping is defined for a workflow step, the workflow will continue based on the definition of the source workflow template. Please note the following about the migration process:

- Any running workflows based on the source workflow template will be locked and no further action can be performed for the workflow step. Users will see an error message stating that the current workflow step is blocked.
- The state defined for each workflow prior to the migration will be maintained after the migration to the target workflow template. This is particularly relevant for workflows with the **Workflow State** `Suspended`. Workflows with the **Workflow State** `Finished` will not be updated.
- The state defined for each workflow step prior to the migration will be maintained after the migration. Please note that if the workflow step has the state Error prior to the migration, the workflow owner ow workflow administrator must explicitly redirect the workflow step to another workflow step after the migration. This is explained in the section *Fixing a Workflow with an Error State* in the reference manual *User and Solution Administration*. Please note the following regarding workflow steps:

- **Previous workflow steps**: Workflow steps that have already been completed prior to the migration will be unaffected by the target workflow template.
- **Current workflow steps**:
- Pre-conditions defined for the current workflow step in the target workflow template will not be evaluated.
- Any workflow activity that has already taken place in the current workflow step (such as data capture) of the source workflow template will be ignored. The user must perform the current workflow step anew.
- If the current workflow step has been delegated to another user, the workflow step delegation will be maintained for the current workflow step.
- **Future workflow steps**: The workflow steps that have not yet been completed in the running workflows will be updated to reflect the changes in the target workflow template.

Once the **Workflow State** attribute of the target workflow template is changed to `Active`, users can continue with their workflow activities as well as create new workflows based on the new workflow template. The **Workflow State** attribute of the obsolete workflow template should be set to `Retired` to prevent new workflows from being initiated based on the old workflow template.

Once the target workflow template is activated, the workflow owner can review the migration's consequences to the running workflows and take any necessary action. The **Workflow Event Trace** page view available for each workflow will indicate that the source workflow template of the running workflow has been migrated to the target workflow template. The following information will be available to the workflow owner:

- The **Event** column will show `WorkflowUpdated` and the **Message** column will display a comment indicating that the source workflow template has been updated to the target workflow template.
- The **Event** column will display `GoToStep` and `StepCancelled` for the source workflow steps that were active and blocked when the migration was executed. The **Message** column will display a comment explaining that the source workflow step was updated.

- The **Event** column will display `StepEntered` and `StepPerformed` for all target workflow steps that are subsequently performed after the execution of the workflow template migration.
- All workflow information captured prior to the migration will be unchanged in the **Workflow Event Trace** page view.

.

The following steps are required to migrate workflows to a new workflow template:

- Create a new workflow template based on the source workflow template. This will be the target workflow template. (If you are only making minor modifications, you can edit the existing workflow template. In this case, the source workflow template and the target workflow template will be the same.) The **Workflow State** attribute of the source and target workflow templates must be set to `Plan`.
- Configure the target workflow template as needed. If workflow steps are being added or deleted, be sure to review that the **Next Step** attributes are correctly defined for each workflow step.
- Validate the target workflow template and correct any errors in the configuration.
- Define the migration of the source workflow template to the target workflow template by mapping the source workflow steps to the target workflow steps. Execute the migration.
- Set the **Workflow State** attribute of the target workflow template to `Active` in order to implement the changes to the workflow and allow users to access the workflow step that they are currently working with.
- Set the **Workflow State** attribute of the obsolete workflow template to `Retired` in order to prevent new workflows being created based on the old workflow template.

To create a new target workflow template:

1) Right-click the source workflow template in the explorer and select **Set Workflow State to Plan**.

   If you are only modifying the source workflow template and not creating a new target workflow template, modify the workflow template as needed. Skip steps 2-4. Be sure that the **Workflow State** attribute is set to `Active` before executing the migration definition.

2) Right-click the source workflow template in the explorer and select **Create New Workflow Template as Copy**. The new target workflow template is displayed in the explorer.
3) Define the necessary attributes of the target workflow template. The workflow template should have a different technical name in the **Name** attribute. The **Caption** attribute, however, which is the name that users see in the interface, can stay the same.
4) Make the necessary changes to the target workflow template. If you are adding or deleting workflow steps, review the **Next Step** attribute of all workflow steps to ensure that they are correct.
5) When you have finished defining the target workflow template, right-click the target workflow template and select **Validate**. Review and correct any configuration errors displayed in the dialog.

6) Once the definition of the target workflow template is complete, right-click the target workflow template and select **Set Workflow State to Active** and click the **Save**  button in the toolbar.

7) Right-click source workflow template in the explorer and select **Create Migration Definition**. The **Workflow Template Migration** editor opens.

8) In the **Select Target WorkflowTemplate** attribute, select the new target workflow template that the source template should be migrated to.

9) Now you must map the source workflow steps to the target workflow steps. For each workflow step displayed in the **Source** column, select one workflow step in the **Target** column that it should be mapped to. Select `NONE` if the source workflow step should not be mapped to a target workflow step. In this case, a running workflow that is currently in a source workflow step that is not mapped to a target workflow step will continue with the definition of the source workflow template.

10) Once the mapping of the workflow steps is complete, you can do one of the following:

- Click **Execute** to initiate the migration of the source workflow template to the target workflow template.
- If there are errors in the workflow template definition, the hourglass symbol will be continually displayed. If this is the case, you must click **Close** in the editor, confirm the warning by clicking **Yes**, and review the target workflow template definition to determine where the error lies.
- If there are no errors in the workflow template definition, a message will be displayed describing how many workflows have been successfully migrated. Click **OK** to confirm this message.
- Once the migration has been executed, click the source workflow template and change the **Workflow State** attribute to `Retired` to ensure that new workflows are not initiated for the obsolete workflow template.
- Please note that you can copy an existing migration definition for a selected workflow template and paste it to another workflow template as a basis to plan the workflow migration for that workflow template.
- Click **Save As** to save the configuration to an XML file. This mechanism can be used to define the migration of the workflows in a development or test environment and apply it to the production environment at a later time. To do so, the following procedure should be carried out in the development/test environment:

  1) Create and configure the target workflow template.
  2) Right-click the source workflow template and click **Create Migration Definition**.
  3) Specify the source workflow template and target workflow template.
  4) Map the workflow steps for the migration and click **OK**. The migration definition  is saved below the source workflow template
  5) Right-click the migration definition  and select **Save As** and save the migration definition to an XML file.

The following procedures should be carried out in the production environment:

1) Right-click the source workflow template and select **Read Migration Definition** to import the migration definition.

2) Right-click the migration definition and select **Open Migration Definition**.

3) Ensure that the source workflow template and target workflow template are selected and that the workflow steps are correctly mapped. Click the **Execute** button to update the running workflows for the workflow template.

4) Set the **Workflow State** attribute for the target workflow template to `Active`.

5) Set the **Workflow State** attribute for the obsolete workflow template to `Retired`.

6) Save your changes by clicking the **Save** button.

Translating the Name and Description of Workflows

Captions, descriptions, technical comments, and user messages configured for workflows, workflow steps, workflow step actions, and pre- and post-conditions can be translated to a secondary language for the user community. For more information about translating the terminology to a secondary language, see the chapter Localization and Multi-Language Support for the Alfabet Interface.

Workflow step states (`Confirmed`, `Pending`, `Running`, `Finished`, etc.) cannot be translated.

To update the translated strings for workflows:

- To update the translated strings available in the vocabularies to all running workflows for a selected workflow template for which the **Workflow State** attribute is currently set to `Plan`, right-click the workflow template and select **Update Translation**. A message will be displayed stating that the translated strings available for the selected workflow template and its workflow steps, workflow step actions, and pre- and post-conditions have been updated. If the **Update Translation** option is disabled, ensure that the **Workflow State** attribute is set to `Plan` and that the most recent changes have been saved by clicking the **Save** button.
- To update the translated strings available in the vocabularies for all workflow templates, regardless of the current value of the **Workflow State** attribute. Go to the menu bar and click **Globalization** > **Update Workflow Translation**. A message will be displayed stating that the translated strings available for workflow templates, workflow steps, workflow step actions, and pre- and post-conditions have been updated to all running workflows for all workflow templates

Making the Workflow Template Available to the AlfaBot Capability

Workflow templates can be configured to allow a workflow to be started for new and existing objects via the AlfaBot. Workflows can be started via the AlfaBot for all workflow templates that have been enabled to start via the AlfaBot. In the AlfaBot, a user can ask to start a workflow for an existing object by entering the object class stereotype and object name. If multiple workflows can be started for the object, these will be listed in the AlfaBot and the user can start the relevant workflow. Alternatively, the user can ask to start a workflow by entering the workflow caption. Multiple captions defined to start a workflow with new objects or with existing objects will be taken into account. In this case, the AlfaBot will request information about the object that the workflow shall be started with. The caption of the workflow started via the AlfaBot is a

concatenation of the caption configured for the workflow and the name of the object that the workflow has been started for. For more information about the configuration of the AlfaBot capability, see the section Implementing the AlfaBot for Navigation via a Full-Text Command.

To implement a workflow template in the AlfaBot capability, select the relevant workflow template define the following attributes as needed:

- **Start Base Class**: Please note that if object class stereotypes have been defined for a base class that is the target of the workflow, then the **Start Base Class** attribute should specify <Class:Stereotype>.
- **Allow AlfaBot to Create New Workflows for Existing Objects**: Set to `True` to allow workflows for existing objects to be started via the AlfaBot.
- **Allow AlfaBot to Create New Workflows for New Objects**: Set to `True` to allow workflows for new objects to be started via the AlfaBot.
- **Message for Started Workflow**: Specify a message to be specified when a new workflow is triggered for the workflow template via either the AlfaBot or via the **Workflow** button in an object profile. The message will be displayed in an information dialog after the workflow has been triggered if the workflow template does not define a start step.

Making the Workflow Capability Available to the User Community

Once the workflow template has been configured, validated, and the **Workflow State** attribute set to `Active`, it can be made available for the user community. As with most capabilities in Alfabet, you must ensure that the relevant functionalities are available to the relevant user profiles. Please keep the following in mind:

- Ensure that the users responsible to perform the workflow steps have the relevant functionality available. Assign either the configured **Workflow Activities Explorer** functionality (`WFS_Explorer`), custom explorer, or the relevant standard **My Workflow Activities** functionalities (`WF_UserWorkflowActivities`, `WF_UserWorkflowActivitiesExt`, and `WF_UserWorkflowActivitiesCommon`) to the user profiles of the users who will be responsible for performing the workflow steps. For more information, see the section Making Functionalities Accessible to a User Profile in the chapter Configuring User Profiles for the User Community.
- Users responsible for workflow steps should also have access to a standard or custom object view for the object class `WorkflowStep` via the class setting associated with their user profile. The standard object view is `WFS_ObjectView`. For more information about the configuration of a custom object view, see the chapter Configuring Object Views as well as the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.
- Ensure that potential workflow owners (users who will initiate the workflow or may later have maintenance responsibility for the workflow) have access to the **Initiate Workflows** functionality (`Initiate Workflow` or `Workflows`).
- Users responsible to maintain workflows should also have access to a standard or custom object view for the object classes `Workflow` and `WorkflowTemplate`. The standard object views are `WF_ObjectView` and `WFT_ObjectView`.

- Please note that configured reports, custom object views and object cockpits, and class settings may be configured for the relevant object classes associated with the workflow capability.

Structuring Workflow Templates in Folders

You can structure and organize workflow templates in group folders. The following is possible:

- To create a group for an existing workflow template, enter a name for the group in the **Group** attribute and press ENTER. The group appears as a folder ascendant to the selected workflow template.
- To create a new workflow template for an existing group, right-click the group and select **Create New Workflow Template**.
- To change the group that a workflow template is assigned to, click the **Group** attribute and select the relevant workflow template group.

Any changes you make to the workflow template must be saved by clicking the **Save**  button

Chapter 14: Configuring Alfabet Functionalities Implemented in the Solution Environment

Alfabet Expand provides many options for configuring the solution environment. In some cases, configurations are defined in an XML object whereas in other cases, you configure changes directly in the Alfabet solution that you access from within Alfabet Expand.

The following information is available:

Configuring the Compliance Management Capability

The **Compliance Management** capability in Alfabet provides support for the definition of compliance inquiries, management of compliance projects, evaluation of target objects in the context of a compliance project as well as the auditing of compliance projects. The following business functions constitute the Compliance Management capability:

- **Compliance Configuration** functionality (`ComplianceConfiguration`): Allows compliance control sets, compliance domains, and compliance policies to be defined. Compliance policies require queries to be configured that specify the rules to find target objects and users responsible for answering questions about the target objects.
- **Compliance Projects** functionality (`ComplianceInstances`): Allows compliance projects to be created, activated, and managed.
- **Compliance Evaluation** functionality (`Home_ComplianceObjects`: Allows users to answer questions about the target objects that they are responsible for in the compliance project.

Before the user community can work with this capability, you must first configure the object classes that are the target of the compliance evaluation as well as the color-coding to represent the state of completion of an object's evaluation. These aspects of the configuration are configured in the XML object ***ComplianceManager***.

Furthermore, you must configure queries for the compliance policies that are relevant for a compliance control set. Queries must be defined in order to find the objects that are the target of a compliance project.

Additionally, you may need to define a query in order to find the persons to perform the evaluations of objects, if the persons are not authorized users of the target objects or have a role defined for the target objects.

The following information is available:

- Configuring the Object Classes to Evaluate
- Configuring the Color-Coding for Completion States
- Configuring Queries for Compliance Policies
- Configuring Release Status Definitions for Compliance Projects
- Making the Compliance Management Functionalities Available to the User Community

Configuring the Object Classes to Evaluate

All classes for objects that are to be evaluated in the context of the **Compliance Management** functionalities must be defined in the XML object ***ComplianceManager***.

1) Go to the **Presentations** tab, expand the **XML Objects** folder, and expand the **SolutionManagers** folder.
2) Right-click the XML object ***ComplianceManager*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) In the XML attribute `Classes`, enter the object classes that are to be included in a compliance evaluation. The object classes should be comma-separated. All objects in the defined object classes may be targeted by a compliance policy, which will determine the relevant set of objects that are to be evaluated. The **Affecting Compliance Projects** workspace (`ObjectAffectingCompliancesProjects`) and *Affecting Compliance Projects Page View* (`AffectingCompliancesProjects`) will be added to the object profile for any object class defined.

The following classes may be the target of a compliance project:

- `Application`
- `ApplicationGroup`
- `BusinessData`
- `BusinessFunction`
- `BusinessObject`
- `BusinessProcess`
- `Component`
- `ComponentGroup`
- `Demand`
- `DemandGroup`
- `ICTObjectGroup`
- `Location`
- `MasterPlatform`
- `Peripheral`
- `PeripheralGroup`
- `Project`:<Stereotype>
- `MarketProduct`
- `MarketProductGroup`
- `StandardPlatform`
- `Technology`

- Deployment
- Device
- DeviceGroup
- Domain
- ICTObject
- TechnologyGroup
- Person
- Vendor
- VendorProduct

Please note that names of object class stereotypes may NOT be entered in the `Classes` attribute. You may only enter the names of base object classes in the `Classes` attribute. If object class stereotypes are defined for an object class, you must explicitly add the *Affecting Compliance Projects Page View* (`ObjectAffectingComplianceProjects`) to the custom object view(s) for each object class stereotype that should be included in the compliance evaluation.

4) In the toolbar, click the **Save** button to save the XML definition.

Configuring the Color-Coding for Completion States

The *Manage Objects Page View* (`CMPLI_ObjectsOverview`) displays the objects that are evaluated in a compliance project. The page view displays the objects in a selected class and shows their percentage of completeness. The values displayed in the **Complete** column are color-coded in green, yellow, and red. The threshold values for the colors must be defined in the XML object ***ComplianceManager***.

1) Go to the **Presentation** tab, expand the **XML Objects** folder, and expand the **Solution Managers** folder.
2) Right-click the XML object ***ComplianceManager*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) In the XML attribute `GreenThreshold` and `YellowThreshold` attribute, specify the threshold values for the color. The value entered for the `GreenThreshold` attribute must be larger than the value entered for the `YellowThreshold` attribute. All objects with a value above the `GreenThreshold` value will be highlighted green in the *Manage Objects Page View* (`CMPLI_ObjectsOverview`). All objects with a value below the `GreenThreshold` value will be highlighted yellow. All objects with a value below the `YellowThreshold` attribute will be highlighted red.

   If no values are entered for either attribute in the XML object ***ComplianceManager***, the default value of 66 for the `GreenThreshold` attribute and 33 for the `YellowThreshold` attribute will be applied.

4) In the toolbar, click the **Save** button to save the XML definition.

Configuring Queries for Compliance Policies

Queries may be required in the following contexts in the Compliance Management capability:

- Mandatory: In order to specify which target objects are to be evaluated in the context of a compliance project as well as the persons to perform the evaluation. This is achieved by defining queries to find the objects that are the target of the compliance project. All queries that are relevant to find target objects for a compliance project must be assigned to compliance policies that have been created for the compliance control set.
- Optional: If the users that are responsible for evaluating the target objects are either the authorized user or a user defined via a role for the target object, then queries do not need to be defined to find the responsible users. If however the responsible users are neither the authorized user or user found via a role definition, then queries must also be defined to find these users responsible. All queries that are relevant to find responsible users in a compliance project must be assigned to compliance policies that have been created for the compliance control set.
- Optional: If compliance controls (the questions about target objects) are to be prioritized in terms of the order in which they are executed in a compliance project, then queries must be configured for each rule needed to find the compliance controls that are relevant for the compliance project. For example, you could configure that only compliance controls on with a **Level ID** attribute of 1, 2, and 3 are relevant to a compliance project. Once the first compliance project is completed, you could then initiate the compliance project again and specific that compliance controls with a **Level ID** attribute of 4, 5, and 6 are relevant to a compliance project. The queries must return the relevant compliance controls that should be included in the compliance project. The relevant queries to execute for the compliance project can be selected in the **Compliance Control Prioritization Policy** field in the **Compliance Project** editor.

Please note the following:

- You must create a configured report for each query required for the compliance policies. The **Category** attribute of the configured report must be set to the value defined for the use case `Compliance` in the XML object ***UseCaseCategories***. For more information, see Assigning a Category for Specific Functional Use to a Configured Report.
- Configured reports to be used in the **Compliance Policy** editor must be of the type `Query` and based on an Alfabet query. Configured reports used in the **Compliance Project** editor can be of the type `Query` or `NativeSQL` and define a native SQL query or an Alfabet query. For detailed information about configuring queries, see the chapter Defining Queries. For detailed information about configuring reports, see the chapter Configuring Reports.
- Which objects are found as base objects of the query of the report is relevant for the availability of the report in the editor fields for compliance management:
  - All Alfabet queries specified to find the target objects will be available for selection in the **Object Queries** tab in the **Compliance Policy** editor. Please note the following:
    - The base class of the query should be the object class that shall be evaluated. A base class may be any of the object classes specified in the XML element `Classes` in the XML object ***ComplianceManager***. When a user selects an object class in the **Class** field in the **Compliance Policy** editor, the relevant queries defined for that base class will be displayed in the **Object Queries** tab in the **Compliance Policy** editor.
    - `JOIN` and `WHERE` clauses may be used to define a subset of objects of the base class to be evaluated.

Example of an Alfabet query to find SOX-relevant applications for a compliance project:

```
ALFABET_QUERY_500

FIND DISTINCT

Application

WHERE Application.Sox_Relevant = 1

QUERY_XML

<QueryDef>

    <ShowProperty Type="Property"
    ClassName="Application" Name="Name" />

    <ShowProperty Type="Property"
    ClassName="Application" Name="Version" />

    <ShowProperty Type="Property"
    ClassName="Application" Name="Sox_Relevant" />

    <ShowProperty Type="Property"
    ClassName="Application" Name="Sox_Software_Type" />

</QueryDef>
```

- All Alfabet queries specified for the object class `Person` will be available in the **Permission Rules** tab in the **Compliance Policy** editor:
  - The base class `Person` must be defined in the query. The relevant queries defined for the base class `PERSON` will be displayed in the **Permission Rules** tab in the **Compliance Policy** editor.
  - The relation to the object class of the objects that are to be evaluated must be defined via `JOIN` and `WHERE` clauses.
  - The parameter:`BASE` refers to the current compliance project and not to the artifact that is evaluated.
  - The `REFSTR` property of the base object class `Person` and of the object class of the objects to be evaluated by the person found by the query must be included in the result dataset of the query. You can optionally include other properties in the result data set to test the query.
- All queries will be available in the **Compliance Control Prioritization Policy** field in the **Compliance Project** editor:
  - The query must return objects of the specified for the object class `ComplianceControl`.
  - The property `CompliancePrioritization` is available for the object class `ComplianceControlInstance` in order to allow compliance controls to be prioritized.
  - When the query is executed, all compliance controls found by the query will be displayed in the *Manage Objects Page View* (`CMPLI_ObjectsOverview`) of the compliance project.

For example, a native SQL query to find compliance controls with a **Level ID** attribute of 1, 2, and 3:

```
SELECT REFSTR
FROM COMPLIANCECONTROL
WHERE LEVELIDNUM LIKE '0001%'
   OR LEVELIDNUM LIKE '0002%'
   OR LEVELIDNUM LIKE '0003%'
```

- It is recommended to use a naming convention for the configured reports that clearly mark the configured reports as reports selectable in the **Compliance Project** editor. The **Compliance Control Prioritization Policy** field will display a drop-down list containing also configured reports only selectable in the **Compliance Policy** editor.

To specify the queries relevant for the **Compliance Management** capability:

1) In the **Reports** tab, right-click the **Reports** folder or a sub-folder and select **Create New Report**. You will see the new custom report listed below the selected folder.
2) In the attribute window, define the following attributes for the report:
   - **Name**: Enter a name for the report. The name you define here will be displayed in the relevant editor in the Alfabet user interface. The name of the report may exceed the conventional 64 character limit.

     In the **Permission Rules** tab of the **Compliance Policy** editor, all queries defined to select persons are listed regardless of the object class for which the person is searched for in the query. Therefore, the name of the report should provide adequate information about the relevant object class specified in the query as well as the relationship of the user to that object class.

   - **Type**: The type must be set to `Query` for an Alfabet query, and to `NativeSql` for a native SQL query. Please note that only configured reports of the type `Query` will be selectable in the editor **Compliance Policy**.
   - **Category**: Select the category name defined in the XML object ***UseCaseCategories*** for the use case `Compliance`. For more information about setting categories in the XML object ***UseCaseCategories***, see Assigning a Category for Specific Functional Use to a Configured Report.

     If the **Category** is not correctly defined, the query cannot be selected in the context of the **Compliance Management** capability.

   - **Alfabet Query** / **Native SQL Query** / **Query as Text**: Define the query as needed. For information about defining queries, see the chapter Defining Queries.
   - **Comment**: Optionally provide information about the report that is useful to the person configuring the reports. This comment will not be displayed in the Alfabet interface.
3) To make the query available in the Alfabet user interface, right-click the configured report in the **Reports** explorer and select **Set Report State to 'Active'**. If further changes are required to the configured report or query, you must return the **Report State** attribute to `Plan`.

4) In the toolbar, click the **Save**  button.

Configuring Release Status Definitions for Compliance Projects

A release status definition must be created for the object classes `ComplianceControlSet` (Compliance Control Set) and `ComplianceControlSetInstance` (Compliance Project) in the XML object ***ReleaseStatusDefs*** in the configuration tool Alfabet Expand.

For example, a release status definition for `ComplianceControlSet` (Compliance Control Set):

```
<ReleaseStatusDef
    ClassNames = "ComplianceControlSet"
    StatusSet = "Draft,Approved,Retired,Discarded"
    RetiredStatusSet = "Retired,Discarded"
    EditableStatusSet = "Draft,Approved,Retired,Discarded"
    DefaultStatus = "Draft"
    ApprovedStatus = "Approved">
    <StatusTransition ToStatus="Draft" FromStatuses="" />
    <StatusTransition ToStatus="Discarded" FromStatuses="any"/>
    <StatusTransition ToStatus="Approved" FromStatuses="Draft" />
    <StatusTransition ToStatus="Retired" FromStatuses="Approved" />
    <Status Name="Draft" Hint="Compliance control set is not
    approved." />
    <Status Name="Approved" Hint="Compliance control set is approved
    and can be implemented." />
    <Status Name="Retired" Hint="Compliance control set should not be
    used any more." />
    <Status Name="Discarded" Hint="Compliance control set is marked
    for deletion." />
</ReleaseStatusDef>
```

Please note the following regarding the release status definition for the Compliance Management capaility:

- You should carefully consider the definition of the StatusTransition XML attributes. For example, if a compliance project haw been set ot the release status specified in the XML attribute `ApprovedStatus`, should it be possible to return the compliance project back to a release status that preceeds the approved status. Please note the following changes that can be made to a compliance project that has been approved.
  - If a compliance project has been approved, the compliance policies may be edited. No new compliance policies may be added to the compliance project.
  - If additional objects that wer enot found by the object queries need to be added to the compliance project, a functioanlity is available that allows the ocmpliance project to be amended.

- The XML attribute `ApprovedStatus` must be specified for the object classes `ComplianceControlSet` (Compliance Control Set) and `ComplianceControlSetInstance` (Compliance Project). This is required to indicate that the compliance control set and compliance project have been approved.
- The XML attribute `RetiredReleaseStatusSet` must be specified for the object classes `ComplianceControlSet` (Compliance Control Set) and `ComplianceControlSetInstance` (Compliance Project). Please note however that only one release status may be specified for the XML attribute `RetiredReleaseStatusSet` configured for the objec class `ComplianceControlSetInstance` (Compliance Project).
- For general information about how to configure release statuses, see the section Configuring Release Status Definitions for Object Classes in the chapter Configuring the Class Model.

Making the Compliance Management Functionalities Available to the User Community

You must ensure that the **Compliance Management** functionalities are assigned to the relevant user profiles. For details about how to make the functionalities available to a user profile, see the section Making Functionalities Accessible to a User Profile in the chapter Configuring User Profiles for the User Community.

- **Compliance Configuration** functionality (`ComplianceConfiguration`): The **Compliance Configuration** functionality supports the definition and configuration of a compliance project. When the user configures the project, he/she must define the catalog of questions and the metrics for answering the questions, the rules to find the targeted objects, and the users who are responsible to answer the questions about the objects, which is captured in the compliance control se. The topical or geographical areas in the enterprise for which the compliance project is relevant are defined via the compliance domain. This functionality should be assigned to a user profile made up of advanced Alfabet users who are familiar with solution configuration and the configuration of Alfabet queries or native SQL queries.
- **Compliance Projects** functionality (`ComplianceInstances`): The **Compliance Projects** functionality supports the management of compliance projects. This functionality allows the user to create compliance projects for specific compliance domains and specific timeframes. This functionality should be assigned to a user profile made up users who are responsible for compliance projects
- **Compliance Evaluation** functionality (`Home_ComplianceObjects`): The **Compliance Evaluation** functionality allows users to view and assess the objects assigned to them in a compliance project. This functionality should be assigned to user profiles made up of users responsible to evaluate the objects targeted by the evaluation.
- The *Affecting Compliance Projects Page View* (`AffectingComplianceProjects`) may be added to the custom object view(s) for each object class/object class stereotype that will be targeted by compliance evaluations.

Configuring the Project Management Capability

Because the project planning capability is highly configurable and each project stereotype will be configured with a name unique in your enterprise, all documentation provided by Alfabet uses the generic term "project" when referring to any object created for any project stereotype.

Alfabet 's project management capability allows you to create and configure a project framework that suits the needs of your enterprise. You can define a strict hierarchy of project levels that serves your enterprise's needs in terms of structuring tasks, defining deliverables, monitoring existing projects, scoping project costs and business cases, aligning projects with the enterprise's strategic intentions and demands, designing alternative architecture scenarios, and planning and implementing the enterprise's to-be architecture.

A typical project management hierarchy might have the following levels:

- Program
- Statement of Work
- Project
- Sub-Project
- Project Activity

A project stereotype represents a level in the project management framework. The project hierarchy is made up of an arbitrary number of object class stereotypes defined for the class `Project`. An unlimited number of objects can be created in Alfabet for each project stereotype.

The project hierarchy can be configured as a linear hierarchy or as a branched tree structure that allows complex project structures and hierarchies of stereotypes to be captured. Each project stereotype may define multiple subordinate project stereotypes. The top-level project stereotype represents the most abstract level of a project and the lowest level project stereotype represents the most granular level of project tasks that must be acted upon in order to realize the project. For example, a typical linear project might be made up of the following project stereotypes: Level 1: Program, Level 2: Statement of Work, Level 3: Project, Level 4: Sub-Project, Level 5: Project Activity.

The name, number, and hierarchical ordering of the project stereotypes are configurable. A project stereotype is similar to an object class. Like an object class, the projects based on a project stereotype are instances of the project stereotype. For example, the project stereotype Project Activity will have objects called project activity. Each project stereotype may have only one ascendant project stereotype and only one subordinate project stereotype. However, projects of a specific stereotype do not have to be to be associated with a project of the ascendant project stereotype nor with projects of the subordinate project stereotype.

The definitions that can be made for each project stereotype must be configured. Therefore, the views that are available may differ for each project stereotype in the hierarchy. For example, the configuration will determine for which project stereotype users can bundle demands to projects, document the as-is architecture, plan a to-be architecture, assess alternative project scenarios, track costs via cost centers, and monitor projects via milestones.

In order to configure project stereotypes that make up the project management hierarchy, you must first create object class stereotypes for the object class `Project`. Once the project stereotypes have been created in the class model for the object class `Project`, you can order the hierarchy of project stereotypes in

the XML object ***ProjectManager***. All project management functionalities in Alfabet are available by default for each project stereotype. However, you should refine which functionalities should be available for each project stereotype by means of the XML object ***ProjectManager***. Questions to consider include:

- For which, if any, project stereotypes should project baselining occur? Project baselines capture the scope of a project at the time that the project baseline is created and allows the deviation of the original scope of the base project to be tracked over time in terms of the total project costs and income, resource costs, project evaluations, scheduling and achievement of project milestones, and the changes to the to-be architecture.
- For which, if any, project stereotypes should project scenarios be allowed? Project scenarios allow multiple what-if scenarios of the to-be architecture, project budgeting and cost calculation, project evaluations, high-level resource planning, and project scheduling to be explored, planned, and potentially implemented.
- For which, if any, project stereotypes should the as-is architecture be defined? The as-is architecture includes the IT architecture elements such applications, components, etc. that are impacted or will be impacted in some way by the project. Typically, the same project stereotype is used to define the as-is architecture and the to-be architecture.
- For which, if any, project stereotypes should the to-be architecture be defined? The to-be architecture allows the user to plan and update the existing as-is architecture by planning new objects, updating existing objects, and retiring objects. The planned to-be architecture can then be merged to the project via a project scenario thus updating the as-is architecture. This would typically be the same project stereotype that you define the as-is architecture for.
- For which project stereotypes should business cases and costs be defined?
- For which, if any, project stereotypes should costs centers be defined? A cost center is a means to centrally define costs for a specified period of time and allocate the costs to projects according to a defined allocation scheme.
- For which project stereotype should be used when a project is created for a demand?
- For which, if any, project stereotypes should project planning and project milestones to track the progress of the projects be implemented?
- For which, if any, project stereotypes should resource planning occur? Resource planning includes the planning of skills and resources over time, including the assignment of the persons or organizations to perform the necessary tasks and track the resource costs.

For each project stereotype, you should carefully consider which functionalities should be available and configure the relevant custom object views and object cockpits necessary for the user community to capture and analyze the necessary data. Once defined, a project stereotype is similar to a standard object class. Correspondingly, you can configure the mandate capability to control user access to each project stereotype and to the capability as a whole. Additionally, you can configure custom editors, wizards, class settings, workflows, and reports for each project stereotype, as needed. All project stereotypes can be configured to be searchable and thus selected in the list of searchable object classes displayed in the **Search for** field in the **Simple Search** and **Browse** functionalities.

The configuration is relevant for the functionalities **Portfolio Overview** (`PRJ_Management`), **Capture Projects** (`PRJ_CaptureProjects`), and the **Project Groups** explorer (`PRJG_Explorer`). These functionalities must be assigned to the relevant user profile in order to make the project management functionalities available in the Alfabet interface. For more information, see the section Making Functionalities Accessible to a User Profile in the chapter Configuring User Profiles for the User Community.

The following information is available:

Creating Project Stereotypes for the Project Hierarchy

Project stereotypes must first be created in the **Stereotype** attribute of the object class `Project`. You can define the project hierarchy as a linear hierarchy or as a branched tree structure that allows complex project structures and hierarchies of stereotypes to be captured. Each project stereotype may define multiple subordinate project stereotypes.

For example, if an enterprise supports Agile release trains, the following project structure could be configured:

- The project stereotype "Program" has the sub-projects "Project" and "Agile Release Train".
- The project stereotype "Project" has only the sub-project "Project Step".
- The project stereotype "Agile Release Train" has only the sub-project "Program Increment".

In this case the XML definition in the XML element ***Stereotype*** for the object class `Project` would be configured as follows:

```
<ClassStereotypes>
```



```
        <Stereotype Name="Program" Caption="Program"
        CaptionPlural="Programs" Comments="" HasMandates="true" />

        <Stereotype Name="Project" Caption="Project"
        CaptionPlural="Projects" Comments="" HasMandates="true" />

        <Stereotype Name="ProjectStep" Caption="Project Step"
        CaptionPlural="Project Steps" Comments="" HasMandates="false" />

        <Stereotype Name="AgileReleaseTrain" Caption="Agile Release
        Train" CaptionPlural="Agile Release Trains" Comments=""
        HasMandates="false" />

        <Stereotype Name="ProgramIncrement" Caption="Program Increment"
        CaptionPlural="Program Increments" Comments=""
        HasMandates="false" />

    </ClassStereotypes>
```

Each project stereotype must only be specified once in the XML object ***ProjectManager***. If a project stereotype is specified more than once, only the first occurrence will be used and any subsequent occurrences will be ignored. If project refinements in different hierarchies require the same terminology, different project stereotypes must be created in the **Stereotype** property of the class `Project`. The XML attribute `Caption` of the stereotypes may be the same for different stereotypes and in this way you can address your enterprise's terminology requirements. For more information about specifying project stereotypes, see the section Creating Project Stereotypes for the Project Hierarchy.

Before you create the project framework, you should carefully consider the number of levels required for your enterprise’s project hierarchy as well as at which level such objects such as project scenarios, costs, milestones, etc. should be defined. **Once the enterprise begins to work with the project management capability, you cannot add new or remove existing project stereotypes.**

Once a project stereotype has been created, you can configure its class settings and specify whether the project stereotype is searchable, and which icon and preview properties should be displayed for the project stereotype. For more information about configuring class settings, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes.

To create project stereotypes for the project management capability:

1) Go to the **Meta-Model** tab and navigate to the protected class node  for the object class `Project` in the **Class Model** explorer.
2) Click the object class `Project` in the explorer to open the attribute window.
3) Click the **Browse**  button available for the **Stereotypes** attribute to open the XML editor. Define the following XML attributes in the XML element ***ClassStereotypes***.

An error message will be displayed upon closing the text editor if syntax errors are written in the XML definition of the **Stereotypes** attribute.

- `Stereotype Name`: Enter the technical name for each object class stereotype. You may define a name of up to 64 characters. This name should not be changed once the object class stereotypes are implemented.

Changing the technical name after objects have been created based on the object class stereotype requires such objects to be updated by means of an ADIF scheme. For more information, see the reference manual *Alfabet Data Integration Framework*.

- `Caption`: Enter the caption for the object class stereotype. This is the name that users will see in the Alfabet user interface.

  Please note that the following:

  - The caption of the object class stereotype should be different than the caption of its object class. This is particularly important for the implementation of the search functionality for the object class stereotype. For more information, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.
  - The values defined for the XML attribute `Caption` can be translated whereas the values defined for the XML attribute `Stereotype Name` cannot be translated. However, in the table of the **Simple Search** functionality as well as other search selectors, the values defined for the XML attribute `Stereotype Name` of the object class stereotype are displayed in the **Stereotypes** column. If the values defined for the XML attribute `Caption` should be displayed, a custom selector should be defined. For more information, see the section Configuring a Custom Selector for Search Functionalities.
  - If you want to enter a caption (`Caption`) or plural form for the caption (`CaptionPlural`) that contains special characters (for example, a greater then (>) or lesser then (<) symbol), you must replace the special characters with respective XML compliant code, for example:
    - > for >
    - < for <
    - " for "
    - [for [
    -] for]
  - The caption will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.

- `Description`: Provide a brief description about the object class stereotype. The description will be displayed in the **Stereotype Selector** that opens when the user must select which stereotype a new object is to be based on. The description should help the user to understand which stereotype the new object should be based on. The description will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.
- `CaptionPlural`: Enter a plural form for the caption. This is the name in plural that users will see in the Alfabet user interface.

- `IDPrefix`: Enter no more than 5 characters for the ID prefix used to generate the unique identification number for new objects based on the object stereotype. If the XML attribute `IDPrefix` is not specified, the ID prefix of the object class that the object class stereotype is based on will be used to generate the unique ID. Please note that the integer generated for the unique ID is based on the number of instances created for the object class that the object class stereotype is based on, and includes all instances of all object class stereotypes created for the object class.

  The prefix ID of existing objects based on an object class stereotype can be updated via the ADIF import scheme `SetStereotypeID` available in the `Alfabet Standard Jobs` folder. For more information, see the section *Predefined ADIF Schemes* in the reference manual *Configuring Alfabet with Alfabet Expand*.

- `Comments`: Enter text in order to provide information about the object class stereotype. This information is only for the purposes of configuration. It is not displayed in the Alfabet user interface.

- `HasMandates`: Enter "`true`" if the mandate capability should be available for the object class stereotype. Enter "`false`" if the mandate capability should not be available for the object class stereotype. If the XML attribute `HasMandates` is defined as "`false`" for an object class stereotype, objects based on the object class stereotype will be visible to users with any mandate implemented in your enterprise. For information about additional configuration requirements necessary to implement the mandate capability, see the section *Implementing the Mandate Capability for a Federated Architecture* in the chapter Configuring Access Permissions for Alfabet .

- `EnableStatutoryLanguage`: Set to `True` if the statutory language capability shall be available for the object class stereotype. The statutory language capability is available in order to address regulatory requirements that mandate that objects are captured in a specific language. When a user creates an object based on a relevant object class/object class stereotype, a statutory language selector will open in which the language can be selected. In the object editor that opens, the user can capture all attributes that support automated translation in the statutory language. If the XML attribute `EnableStatutoryLanguage` is not set, the object class stereotype will inherit the definition of the **Enable Statutory Language** attribute specified for the object class. Specifying a Statutory Language for the Enterprise

4) Click the **Save**  button to save your changes.

Configuring Mandates for the Project Management Capability

The mandate functionality is typically configured when the project stereotypes are created. However, the mandate assignment can be edited, as needed. The mandate capability can be activated or deactivated for the entire project planning capability as well as explicitly activated or deactivated for specific project stereotypes.

The mandate configuration may differ for the project stereotypes defined. Per default, the XML attribute `HasMandates` is defined as "`false`" for all project stereotypes. This means that the default setting is that projects based on a project stereotype will be visible to users with any mandate implemented in your enterprise. Therefore, if visibility should be controlled for the projects based on a project stereotype, you must explicitly define the mandate capability for each project stereotype.

For more information about the mandates capability as well as how to create mandates for your enterprise, see the section *Implementing the Mandate Capability for a Federated Architecture* in the chapter Configuring Access Permissions for Alfabet .

To configure the mandate capability for your project hierarchy:

1) Go to the **Meta-Model** tab and navigate to the protected class node for the object class `Project` in the **Class Model** explorer.
2) Click the object class `Project` in the explorer to open the attribute window.
3) Set the **Can Have Mandates** attribute to `True` if the mandate capability should be available for the project management capability as a whole. Select `False` if the mandate capability should not be available.
4) Next, you must explicitly define the mandates capability for each project stereotype. Click the **Browse** button available for the **Stereotypes** attribute to open the XML editor.
5) To implement the mandate capability for a project stereotype, the XML attribute `HasMandates` must be defined as "`true`". If you do not want to implement the mandate capability for a project stereotype, the XML attribute `HasMandates` must be defined as "`false`". If the XML attribute `HasMandates` is defined as "`false`" for a project stereotype, projects based on the stereotype will be visible to permissible users with any mandate implemented in your enterprise.
6) Close the XML definition of the **Stereotypes** attribute by clicking **OK** in the XML editor.
7) In the toolbar, click the **Save** button to save your changes.

Configuring the Attributes and Functionalities Available for a Project Stereotype

Once the project stereotypes have been created for the object class `Project`, you can enable the respective functionalities for project scenarios and project baselines and specify functionality for each project stereotype in the XML object ***ProjectManager***. You can define the project hierarchy as a linear hierarchy or as a branched tree structure that allows complex project structures and hierarchies of stereotypes to be captured. Each project stereotype may define multiple subordinate project stereotypes.

For example, if an enterprise supports Agile release trains, the following project structure could be configured:

- The project stereotype "Program" has the sub-projects "Project" and "Agile Release Train".
- The project stereotype "Project" has only the sub-project "Project Step".
- The project stereotype "Agile Release Train" has only the sub-project "Program Increment".

The order of the project stereotypes listed in the XML object ***ProjectManager*** determines the order of the levels in the project hierarchy. Each project stereotype begins with the XML element ***Stereotype***. The first level of project stereotypes in the XML object represents the most ascendant level in the hierarchy. To change the order of the project stereotypes, select the entire project stereotype and cut and paste it to the

correct position in the order. Be sure to include the syntactic constructs (< and />), otherwise you will see an error message at the bottom of the XML editor.

If more than one project stereotype can be defined as a subordinate project, the < **Sub-Projects** > page views (`PRJ_SubProjects`) in the user interface will display the captions of all possible project stereotypes that may be created as a subordinate project of the selected project that the user is working with. The project stereotypes will be separated by a slash (/). Likewise, if a user is creating a new sub-project, the captions of all permissible project stereotypes that may be created for the selected project will be displayed in the **Create...** option in the **New** menu. When a user creates a project, the stereotype selector will open if the new project may be based on more than one project stereotype. The user must select the relevant stereotype that the new project shall be based on and then the relevant **Project** editor will open. If only one project stereotype is permissible, the new project will automatically be based on the project stereotype and the relevant **Project** editor will open

Please note that the visibility of workspaces and page views was configured via the XML attributes `HasSolutions`, `HasAsIsArchitecture`, and `HasToBeArchitecture` in releases prior to Alfabet release 10.0. The visibility of workspaces and page views is now configured via the XML object ***ObjectViewCustomFilters*** and is explained in the section Making the Project Scenarios Capability Available. If you require further explanation about the XML attributes `HasSolutions`, `HasAsIsArchitecture`, and `HasToBeArchitecture`, please see the reference manual *Configuring Alfabet with Alfabet Expand* published for Alfabet release 9.12.

To edit the XML object ***ProjectManager***:

1) Go to the **Presentations** tab, expand the **XML Objects** folder, and expand the **SolutionManagers** folder.
2) Right-click the XML object ***ProjectManager*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see the section Working with XML Objects.
3) Specify the following XML elements below the root node ***ProjectManager***:
   - `ProjectClassName`: Displays the object class name `Project`. This value should not be changed.
   - `Solutions`: Enter "`true`" to implement the project solutions capability. Please note that it is highly recommended that you work with project scenarios rather than project solutions when planning and implementing alternative scenarios for your projects. Project scenarios are an extension of the project solutions concept but provide much more flexibility and functionality in the planning and implementation of what-if scenarios. For more information, see the section Making the Project Scenarios Capability Available.

     If the XML attribute `Solutions` is set to "`true`", then the XML attribute `Scenarios` should be set to "`false`".

   - `Scenarios`: Enter "`true`" to implement the project scenarios capability. Project scenarios allow multiple what-if scenarios to be explored, planned, and potentially implemented for the to-be architecture, project budgeting and cost calculation, project evaluations, high-level resource planning, and project scheduling. For more information about the functionality and configuration of project scenarios, see the section Making the Project Scenarios Capability Available.

If the XML attribute `Scenarios` is set to "`true`", then the XML attribute `Solutions` should be set to "`false`".

- `Baselines`: Enter "`true`" to implement the project baselines capability. Project baselines capture the scope of the base project at the time that the project baseline is created and allows the deviation of the current project from its original scope to be measured in terms of the total project costs and income, resource costs, project evaluations, scheduling and achievement of project milestones, and the changes to the to-be architecture. For more information about the functionality and configuration of project scenarios, see the section Making the Project Scenarios Capability Available.

- `ConflictStatuses`: Specify the release statuses configured for the project stereotypes specified in the XML object ***ReleaseStatusDefs*** that may be displayed in the **Project Release Status** filter in the **Check-In To-Be Architecture** page view. If no release statuses are specified here, then all release statuses available for all project stereotypes specified in the XML object ***ReleaseStatusDefs*** will be displayed. For more information about configuring release statuses for the project stereotypes, see the section Configuring Release Status Definitions for Projects.

  The **Project Release Status** filter allows users to select one or more release statuses to specify that only projects with the selected release status(es) will be checked for possible conflicts before the to-be architecture is checked in to the inventory. Defining this filter improves performance by reducing the potentially excessive number of projects that are scanned for conflicts prior to check-in.

- `WorkdayHours`: Enter a number that specifies the number of hours that constitute one workday. The default is 8. This definition allows the hours specified as time-track entries in the *My Tasks and Time Reporting Functionality* to be computed as days in relevant project planning views.

4) Create an XML element ***Stereotype*** for each top-level node of the tree structure.

- `ProjectClassName`: Displays the object class name `Project`. This value should not be changed.

- `Name`: Enter the technical name of the object class stereotype defined for the object class `Project`. This is the name defined in the XML attribute `Name` specified in the **Stereotypes** attribute for the object class `Project`. The stereotype name is passed on to the objects that users create for the selected project stereotype.

  You must spell the technical name of the project stereotype exactly as it is spelled in the **Stereotypes** attribute of the object class `Project`.

- `RDS_Expand`: Enter "`true`" to display the expand/collapse functionality in the dataset in the **Capture Projects** functionality (`PRJ_CaptureProjects`). Enter "`false`" to hide the expand/collapse functionality in the dataset. This will allow users to see only the current project stereotype selected in the **Project Type** filter in the **Capture Projects** functionality.

- `RightsPath`: Enter "`Parent`" if a project stereotype is to inherit the access permissions of its ascendant project stereotype. If the project stereotype should not inherit access permissions from another project stereotype, leave the `RightsPath` attribute undefined.

- `HasMonitoring`: Enter "`true`" if milestones should be created and tracked for objects of this project stereotype. In this case, the **Create Milestones** functionality will be available in the

Project Milestones Page View (`PRJ_Milestones`) for the relevant project stereotype and the **Milestone** editor will be available in the *Project, Skill Request and Resource Request Time Schedule Page View* (`PRJ_TimeSchedule`) in Alfabet. Enter "`false`" if milestones should not be created and tracked for the project stereotype.

Milestone templates must be configured in order to track milestones in the project management capability. For more information, see the section Configuring Milestone Templates to Track Projects.

- `HasCostCenters`: Enter "`true`" if cost centers may be defined for projects of this project stereotype. The costs center allows costs to be defined and then allocated to projects for budgeting purposes. In this case, the *Cost Centers Page View* (`PRJ_CostCenters`) available in the **Finance** workspace will be displayed for the relevant project stereotype. Enter "`false`" if cost centers should not be implemented for the project stereotype.
- `DefaultForDemand`: Enter "`true`" if this project stereotype is the default project stereotype for the creation of a new project for a selected demand. If this attribute is set to "`true`", the *Input Demands Page View* (`PRJ_InputDemands`) available in the **Project Alignment** workspace will be displayed for the relevant project stereotype. Additionally, projects based on this stereotype can be created for or added to a demand in the **Demand Management** functionality (`DEM_Management`) or the demand's object profile. Enter "`false`" if projects based on this project stereotype may not be created for demands.

  Only one project stereotype should be mapped to demands. If this attribute is set to "`true`" for more than one project stereotype, the user will only be able to create/add a project of the first project stereotype in the **Demand Management** functionality (`DEM_Management`) or demand's object profile.

- `IsDefault`: Enter "`true`" if the project stereotype is the default project stereotype to display in the **Capture Projects** functionalities. The default project stereotype will be automatically selected when the **Capture Projects** functionalities are opened for the first time.
- `ArchitectureClasses`: Enter a comma-separated list of object classes and objects class stereotypes that may be added as affected architecture for the project stereotype. A menu entry with the caption **Add <Object Class>** or **Add <Object Class Stereotype>** will be added to the *Affected Architecture Page View*. If you are defining an object class stereotype in the XML attribute `ArchitectureClasses`, you must use the syntax ObjectClass:ObjectClassStereotype. If you are defining an object class stereotype in the XML attribute `ArchtectureClasses`, you must use the syntax ObjectClass:ObjectClassStereotype. If the XML attribute `ArchitectureClasses` is not defined, the standard object classes preconfigured by Software AG will be displayed.

5) For each subordinate project stereotype, create an XML element ***Stereotype*** as a child element of the parent XML element ***Stereotype*** and define the XML attributes described in the previous step. Continue creating XML elements to structure the project hierarchy. You can define the project hierarchy as a linear hierarchy or as a complex branched tree structure.

   Each project stereotype must only be specified once in the XML object ***ProjectManager***. If a project stereotype is specified more than once, only the first occurrence will be used and any subsequent occurrences will be ignored. If project refinements in different hierarchies require the same terminology, different project stereotypes must be created in the **Stereotype** property of the class `Project`. The XML attribute `Caption` of the stereotypes may be the same for different stereotypes and in this way you can address your

enterprise's terminology requirements. For more information about specifying project stereotypes, see the section Creating Project Stereotypes for the Project Hierarchy.

6) In the toolbar, click the **Save**  button to save the XML definition.

Making the Project Baselines Capability Available

You may enable the project baselining capability for one or more project stereotypes. A project baseline captures the scope of its base project at the time that the project baseline is created and allows the deviation of the base project from its original scope to be tracked over time in terms of the total project costs and income, resource costs, project evaluations, scheduling and achievement of project milestones, and the changes to the to-be architecture. When a project baseline is created, it is created as an independent project where the attribute **Type** = `Baseline`. Multiple project baselines may be created at different points in time for the selected project. The project baseline is for documentation purposes and cannot be modified in any way.

The project baseline is a deep copy of the project that it is based on and will inherit the project stereotype of the project as well as the following:

- Subordinate projects
- Evaluations and milestones
- Business case including planned and budgeted costs, bucket allocation, cashout plan, and cost center allocation
- As-is architecture and to-be architecture definition
- Solution maps and application diagrams
- High-level resource plan. This includes the general definition of skill requests and resource requests but not the granular project resource plan that includes the detailed planning and scheduling of tasks.
- Value nodes
- Documents and web links

Users will be able to navigate to a ReadOnly version of the object view for the project baseline in order to view the details of the original scope of the project. The object view available for a project baseline will be the same custom object view including object cockpits available for the project stereotype that the project baseline is based on, except that the custom object view for the project baseline will consist of a preconfigured subset of workspaces that are relevant for project baselines. The standard subset of available views can be further configured so that specified workspaces, page views, configured reports, and object cockpits may be hidden in the context of the project baseline. An overview of the standard set of workspaces and page views that are displayed per default for the project baseline is available in the section *Standard Workspaces and Views Available for Project Baselines* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

The following is required to configure the availability of the project baselines capability:

- Enable the project baselines capability in the XML object ***ProjectManager***.

- If necessary, configure visibility of available workspaces, page views, configured reports, or object cockpits in the XML object ***ObjectViewCustomFilters***.
- Ensure that no class keys to implement uniqueness constraints have been defined for the object class `Project`. For more information about the definition of class keys, see the section Configuring Class Keys for Object Classes.

To implement the project baselines capability, specify "`true`" for the XML element `Baselines` in the XML object ***ProjectManager***.

```
<ProjectManager ProjectClassName="Project" Solutions="false"
Scenarios="true"Baselines="true" >
...
</ProjectManager>
```

You may decide that some of the object cockpits and configured reports as well as the workspaces and page views should not be displayed in the context of the project baseline. In this case, you can specify which workspaces, page views, configured reports, and object cockpits should be hidden in the custom objects views configured for a specified project stereotype.

The XML object ***ObjectViewCustomFilters*** allows you to explicitly specify which workspaces, page views, configured reports, or object cockpits should be hidden from the custom object views defined for a specified project stereotype. Please note that the XML object ***ObjectViewCustomFilters*** is a blacklist that allows you to specify the content that shall **not** be available when users access custom object views for the specified project stereotype.

To edit the XML object ***ObjectViewCustomFilters***:

1) Go to the **Presentations** tab, expand the **XML Objects** folder, and expand the **SolutionManagers** folder.
2) Right-click the XML object ***ObjectViewCustomFilters*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see the section Working with XML Objects.
3) In the XML attribute ***ObjectViewFilter***, specify the following XML attributes for each object class stereotype relevant for the project baseline configuration:

For example, to hide an object cockpit displaying cost information:

```
<ObjectViewFilter ClassName="Project" Stereotype="Project"
Property="Type" Value="Baseline" >
<Item Type="Cockpit"
Name="PRJ_ObjectView_Project_PS_Financials" />
</ObjectViewFilter>
```

- `ClassName`: Enter "`Project`"
- `Stereotype`: Enter the name of the project stereotype that the project is based on.
- `Property`: Enter "`Type`"
- `Value`: Enter "`Baseline`"

4) Add a child XML element ***Item*** for each workspace, page view, configured report, or object cockpit that should be hidden for the specified project stereotype:

   - `Type`: Enter any of the following:
     - "`WS`" to specify a workspace that should be hidden.
     - "`View`" to specify a page view or configured report that should be hidden.
     - "`Cockpit`" to specify an object cockpit that should be hidden.
   - `Name`: Enter the technical name of the relevant workspace, page view, configured report, or object cockpit that should be hidden.

5) In the toolbar, click the **Save** button to save the XML definition.

Making the Project Scenarios Capability Available

In the XML object ***ProjectManager***, you can specify whether project scenarios should be implemented to capture what-if scenarios for projects. Project scenarios are an alternative to the project solution concept and it is highly recommended that you work with project scenarios rather than project solutions when planning and implementing alternative scenarios for your projects. Project scenarios allow multiple alternative what-if scenarios to be explored, planned, and potentially implemented for the to-be architecture, project budgeting and cost calculation, project evaluations, resource planning, and project scheduling. Project scenarios are an extension of the project solutions concept but provide much more flexibility and functionality in the planning and implementation of what-if scenarios. A project scenario is an independent copy of the project that it is based on.

Unlike a project solution, which is created from scratch, a project scenario copies relevant attributes, custom properties, and other definitions from the base project to the project scenario. Furthermore, unlike the check-in process of a project solution whereby the project solution irrevocably replaces the base project upon the final check-in, the project scenario concept allows the project scenario to be merged to the base project so that only relevant aspects are updated. In this case, objects in the to-be architecture, evaluations, costs, high-level resources of the project scenario that have been added, removed, or modified are updated to the base project but the rest of the base project is not changed or replaced. The project scenario as well as the base project continue to exist and both can be further modified, if needed. The project scenario can be merged multiple times.

Please note that merging a project scenario with its base project can be done at any time by anyone in with access permissions to the project scenario. If the merge action should be governed in any way, then it is recommended that you configure a workflow that ensures some type of an approval process before the merge can be executed. For more information about configuring a workflow, see the chapter Configuring Workflows.

When a project scenario is created, it is created as an independent project where the attribute **Type** = `Scenario`. The project scenario is a deep copy of the project that it is based on and will inherit the project stereotype of the project as well as the following:

- Authorized user and user groups, mandates, roles, and providing organization
- Evaluations and milestones
- Business case including planned and budgeted costs, bucket allocation, and cashout plan

- As-is architecture and to-be architecture definitions. Please note the following:
  - Only solution objects in the base project that do not have a corresponding inventory object with an **Active** object state will be added to the to-be architecture of the project scenario when it is created. If a solution object in the base project has a corresponding inventory object with the object state **Active** set, the inventory object will be added to the as-is architecture of the project scenario.
  - The project scenario cannot be merged if the to-be architecture has been checked in for the base project. If your enterprise is planning projects via project scenarios, the to-be architecture should be planned in the context of the project scenario rather than the base project. When the project scenario is merged to the base project, the to-be architecture will be updated and the solution objects will be added to the inventory.
  - For more information about additional configuration required for the to-be architecture, see the section Making the As-Is Architecture and To-Be Architecture Capabilities Available.
- Solution maps and application diagrams
- High-level resource plan. This includes the general definition of skill requests and resource requests but not the granular project resource plan tasks are planned and scheduled in detail.
- Migrations
- Input demands
- Contract deliverables
- Value nodes
- Documents and web links

Users will be able to navigate to a ReadWrite version of the object view for the project scenario in order to define the details of the project scenario. The object view available for a project scenario will be the same custom object view including object cockpits available for the project stereotype that the project scenario is based on, except that the custom object view for the project scenario will include only a subset of the workspaces. An overview of the standard set of workspaces and page views that are displayed per default for the project scenario is available in the section *Standard Workspaces and Views Available for Project Scenarios* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

The following is required to configure the availability of the project scenarios capability:

- Enable the project scenarios capability in the XML object ***ProjectManager***.
- If necessary, configure visibility of available workspaces, page views, configured reports, or object cockpits in the XML object ***ObjectViewCustomFilters***.
- Ensure that no class keys to implement uniqueness constraints have been defined for the object class `Project`. For more information about the definition of class keys, see the section Configuring Class Keys for Object Classes.

To implement the project scenarios capability, specify "`true`" for the XML element `Scenarios` in the XML object ***ProjectManager***.

```
<ProjectManager ProjectClassName="Project"
Solutions="false"Scenarios="true"Baselines="true">

...

</ProjectManager>
```

Or, to implement the project solutions capability, specify "`true`" for the XML element `Solutions` in the XML object ***ProjectManager***. Please note that it is highly recommended that you work with project scenarios rather than project solutions when planning and implementing alternative scenarios for your projects. Project scenarios are an extension of the project solutions concept but provide much more flexibility and functionality in the planning and implementation of what-if scenarios.

Please note the following regarding the configuration of the XML object ***ProjectManager***:

- Your enterprise may not simultaneously enable project scenarios and project solutions. If both the XML attribute `Scenarios` and the XML attribute `Solutions` are set to "`true`", the system will automatically set the XML attribute `Scenarios` to "`false`".
- The visibility of workspaces and page views for project solutions was configured via the XML attribute `HasSolution` in releases prior to Alfabet 10.0. If the XML element `Solutions` is set to "`false`" and the XML attribute `HasSolutions` is set to "`true`", the XML element `Solutions` = "`false`" will have precedence. Furthermore, if the XML attributes `HasSolutions` is set to "`true`" in the XML object ***ProjectManager*** and the solution designer then configures to hide the workspace `PRJ_SolutionAssesstment` via the XML object ***ObjectViewCustomFilters***, then the XML object ***ObjectViewCustomFilters*** definition will take precedence and the workspace will not be available for the relevant project stereotypes.

You may decide that some of the object cockpits and configured reports as well as the workspaces and page views should not be displayed in the context of the project scenario. In this case, you can specify which workspaces, page views, configured reports, and object cockpits should be hidden in the custom objects views configured for a specified project stereotype.

The XML object ***ObjectViewCustomFilters*** allows you to explicitly specify which workspaces, page views, configured reports, or object cockpits should be hidden from the custom object views defined for a specified project stereotype. Please note that the XML object ***ObjectViewCustomFilters*** is a blacklist that allows you to specify the content that shall **not** be available when users access custom object views for the specified project stereotype.

To edit the XML object ***ObjectViewCustomFilters***:

1) Go to the **Presentations** tab, expand the **XML Objects** folder, and expand the **SolutionManagers** folder.
2) Right-click the XML object ***ObjectViewCustomFilters*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see the section Working with XML Objects.
3) In the XML attribute ***ObjectViewFilter***, specify the following XML attributes for each object class stereotype relevant for the project scenario configuration:

For example, to hide an object cockpit displaying cost information:


```
<ObjectViewFilter ClassName="Project" Stereotype="Project"
Property="Type" Value="Scenario" >

<Item Type="Cockpit"
Name="PRJ_ObjectView_Project_PS_Financials" />

</ObjectViewFilter>
```

- `ClassName`: Enter `"Project"`
- `Stereotype`: Enter the name of the project stereotype that the project is based on.
- `Property`: Enter `"Type"`
- `Value`: Enter `"Scenario"`

4) Add a child XML element ***Item*** for each workspace, page view, configured report, or object cockpit that should be hidden for the specified project stereotype:
   - `Type`: Enter any of the following:
     - `"WS"` to specify a workspace that should be hidden.
     - `"View"` to specify a page view or configured report that should be hidden.
     - `"Cockpit"` to specify an object cockpit that should be hidden.
   - `Name`: Enter the technical name of the relevant workspace, page view, configured report, or object cockpit that should be hidden.

5) In the toolbar, click the **Save**  button to save the XML definition.

Making the As-Is Architecture and To-Be Architecture Capabilities Available

When creating project stereotypes, you should consider for which project stereotypes the as-is architecture definition and to-be architecture definition functionalities should be available.

The as-is architecture comprises all architecture elements that are actively used as well as architecture elements that are planned to be used in the future. The as-is architecture serves as the baseline for the process of planning and proposing new architecture elements for the IT landscape. In the context of a project, the as-is architecture defines the scope of the IT environment that the project addresses. This is typically the set of architecture elements that is the subject of the change or that may be affected by the changes proposed.

The to-be architecture allows the user to plan and update the existing as-is architecture by planning new objects, updating existing objects, and retiring objects. A project's to-be architecture may include inventory objects from the as-is architecture that may be affected by the proposed architectural changes as well as new solutions objects such as solution applications, solution information flows, and solution business supports created in the context of the to-be architecture. The solution objects are considered to be "shadow" objects — copies of the base applications, information flows, etc. that they were derived from. Solution objects may be modified within the context of the to-be architecture without affecting the inventory objects they are based on. Any architectural changes defined in the context of the proposed to-be architecture can be checked in to the inventory. All architecture elements added to the inventory via the proposed to-be architecture will have an object state indicating that the object is planned. The object state

must be manually changed to an active object state when operational use of the to-be architecture element begins, which is typically upon completion of the project's implementation.

Architecture elements assigned to the as-is architecture or to-be architecture will not be inherited from one project stereotype to the next in Alfabet. Architecture elements must be manually assigned to a relevant object for each level in the project hierarchy.

The following information is available:

- Configuring the As-Is Architecture Capability
- Configuring the To-Be Architecture Capability

Configuring the As-Is Architecture Capability

The as-is architecture includes the IT architecture elements such applications, components, etc. that are impacted in some way by the project. Objects belonging to any of the following classes can be defined as architecture elements that are affected by a project: `Application`, `BusinessData`, `BusinessFunction`, `BusinessObject`, `BusinessProcess`, `Component`, `CustomerSegment`, `Device`, `Domain`, `ICTObject`, `InformationFlow`, `MarketProduct`, `MasterPlatform`, `OrgaUnit`, `Peripheral`, `SalesChannel`, `SystemBuildingBlock`, `StandardPlatform` and, `VendorProduct`. In order to make these object classes available for assignment to the as-is architecture, you must ensure that the exclusion of an object class via a class setting definition does not pose a conflict.

All project management functionalities in Alfabet are available by default for each project stereotype. However, you can refine the availability of these functionalities by adding or removing them from the relevant custom object views defined for a specific project stereotype In order for users to configure and analyze the as-is architecture, the **As-Is Architecture Definition** (`PRJ_AsIsArchitecture`) and **Affected Architecture Analysis** (`PRJ_AffectedArchitectureAnalysis`) workspaces should be available for the relevant project stereotype. For an overview of the standard views available for each workspace, see the section *Standard Workspaces and Views Available for Projects* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

Please note that the availability of the as-is architecture definition capability was configured via the XML attribute `HasAsIsArchitecture` in releases prior to Alfabet release 10.0. If the XML attributes `HasAsIsArchitecture` is set to "`true`" in the XML object ***ProjectManager*** and the solution designer then configures to hide the workspaces `PRJ_AsIsArchitecture` and `PRJ_AffectedArchitectureAnalysis` via the XML object ***ObjectViewCustomFilters***, then the XML object ***ObjectViewCustomFilters*** definition will have precedence and the workspaces will not be available for the relevant project stereotypes. If you require further explanation about the XML attribute `HasAsIsArchitecture`, please see the reference manual *Configuring Alfabet with Alfabet Expand* published for Alfabet release 9.12.

Configuring the To-Be Architecture Capability

When planning the to-be architecture, users work with solution objects, which are copies of inventory objects that exist only in the context of the to-be architecture and are not visible in the inventory. In the context of the to-be architecture, users can manipulate these objects without making any actual changes to the inventory. Solution objects can be created for the following object classes: `SolutionApplication`, `SolutionBusinessData`, `SolutionBusinessService`, `SolutionBusinessSupport`, `SolutionComponent`, `SolutionDevice`,`SolutionFunctionalModule`, `SolutionInformationFlow`,

SolutionLocalComponent, SolutionPlatform, SolutionPlatformElement, SolutionPlatformInformationFlow, SolutionService, SolutionServiceOperation, and SolutionStandardPlatform.

Once all architectural conflicts have been resolved for the proposed to-be architecture, the changes to the to-be architecture can be checked in to the inventory. At this point, all new objects created in the context of the to-be architecture will be added to the inventory, objects that have been modified in the context of the to-be architecture will overwrite their corresponding inventory objects, and the object states of inventory objects that have been retired in the to-be architecture will be set to retired.

Please note the following configuration requirements in order to work with solution objects:

- If object class stereotypes are configured for an object class that can be defined in the *To-Be Architecture Page View* (such as Application, Component, Device, or StandardPlatform) then you must create an identical configuration for the corresponding solution object class (for example, for SolutionApplication or SolutionComponent, SolutionDevice, or SolutionStandardPlatform). For example, the configuration defined in the **Stereotypes** attribute for the object class Application should be copied and pasted to the **Stereotypes** attribute for the object class SolutionApplication. If the configuration is not the same, errors may occur when solution objects are checked in to the Alfabet inventory in the context of project and solution planning. For more information about configuring object class stereotypes, see the section Configuring Object Class Stereotypes for Object Classes in the chapter Configuring the Class Model.

- Users must be able to add any inventory objects that are either to be modified, retired, or used as a basis for new solution objects to the architectural scope of the relevant project. (Objects belonging to any of the following classes can be defined as architecture elements that are affected by a project: Application, BusinessData, BusinessFunction, BusinessObject, BusinessProcess, Component, CustomerSegment, Device, Domain, ICTObject, InformationFlow, MarketProduct, MasterPlatform, OrgaUnit, Peripheral, SalesChannel, SystemBuildingBlock, StandardPlatform and, VendorProduct.)

- All project management functionalities in Alfabet are available by default for each project stereotype. However, you can refine the availability of these functionalities by adding or removing them from the relevant custom object views defined for a specific project stereotype In order for users to configure and analyze the as-is architecture, the **As-Is Architecture Definition** (PRJ_AsIsArchitecture) and **Affected Architecture Analysis** (PRJ_AffectedArchitectureAnalysis) workspaces should be available for the relevant project stereotype. Additionally the PRJ_ToBeArchitecture and PRJ_ToBeArchitectureAnalysis workspaces must be available for the relevant project stereotype. For an overview of the standard views available for each workspace, see the section *Standard Workspaces and Views Available for Projects* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

- You should specify the release statuses configured for the project stereotypes specified in the XML object ***ReleaseStatusDefs*** that may be displayed in the **Project Release Status** filter in the **Check-In To-Be Architecture** page view. If no release statuses are specified here, then all release statuses available for all project stereotypes specified in the XML object ***ReleaseStatusDefs*** will be displayed. The **Project Release Status** filter allows users to select one or more release statuses to specify that only project's with the selected release status(es) will be checked for possible conflicts before the to-be architecture is checked in to the inventory. Defining this filter improves performance by reducing the potentially excessive number of projects that are scanned for conflicts prior to check-in.

Please note that the availability of the to-be architecture definition capability was configured via the XML attribute HasToBeArchitecture in releases prior to Alfabet release 10.0. If the XML attributes HasToBeArchitecture is set to "true" in the XML object ***ProjectManager*** and the

solution designer then configures to hide the workspaces `PRJ_ToBeArchitecture` and `PRJ_ToBeArchitectureAnalysis` via the XML object ***ObjectViewCustomFilters***, then the XML object ***ObjectViewCustomFilters*** definition will take precedent and the workspaces will not be available for the relevant project stereotypes. If you require further explanation about the XML attribute `HasToBeArchitecture`, please see the reference manual *Configuring Alfabet with Alfabet Expand* published for Alfabet release 9.12.

Making Skill Capacity Planning Available

In order to implement the **Assign Capacity** option in the **Project Resource Planning** page view of a project as well as the **Skill Capacity Planning** page view of an organization or organization group, a custom wizard must be configured with the **Assign Capacity** `(SKLR_AssignedCapacity)` page view embedded in the wizard. The custom wizard must be assigned to the class setting for the class `SkillRequest` to be implemented in the views.

For more information about configuring custom wizards, see Configuring Wizards.

Configuring Release Status Definitions for Projects

The release status definition for project stereotypes is specified in the XML object ***ReleaseStatusDefs*** in Alfabet Expand.

A release status definition must be created for the object class `Project` and for each object class stereotype created for the object class `Project`. The release status definition for the object class `Project` must include the complete set of release statuses implemented in the project management functionalities. These must be specified in the XML attribute `StatusSet` in the release status definition for the object class `Project`. Furthermore, an element `Project:<ProjectStereotype>` should be defined in the XML object ***ReleaseStatusDefs*** for each project stereotype in your enterprise's project framework.

Please note the following regarding the release status definition for projects:

- **Multiple release status set definitions should not be defined for the project management functionalities.** You should conceptualize one release status set that includes all possible release statuses for all project stereotypes. The release statuses in the XML attribute `StatusSet` can be configured as needed for each project stereotype specified in the XML attribute `ClassNames`. Release statuses that are irrelevant for a given project stereotype may be excluded in the XML attribute `StatusSet` of that stereotype.

  For example, the XML attribute `StatusSet`of the overall project is conceptualized to include the release statuses: `New, Discarded, Described, InReview, ReviewFailed, Reviewed, Planned, Closed, Discarded`.

  - For the object class Project, the following is specified: `StatusSet = "New,Discarded,Described,InReview,ReviewFailed,Reviewed,Planned,Closed,Discarded"`
  - For the object class Project:Program, the following is specified: `StatusSet = "New,Discarded,Described,Reviewed,Planned,Closed"`

- For the object class Project:Project, the following is specified:
  `StatusSet = "New,Described,InReview,ReviewFailed,Reviewed,Planned,Closed,Discarded"`
- For the object class Project:ProjectStep, the following is specified:
  `StatusSet = "New,Described,Planned,Closed,Discarded"`

- It is recommended that the release status values `Draft` and `Closed` are not included in the XML attribute `RetiredStatusSet`. The status `Draft` typically indicates a new request for a project and the status `Closed` typically indicates that a project is completed. When configuring the retired release status definition for projects, it is recommended that the statuses `Discarded` or `Archived` are implemented in the `RetiredStatusSet`.
- Each project stereotype requires a release status to indicate that the project is approved. This is configured in the XML attribute `ApprovedStatus`. For more information about defining an approval workflow for projects, see the section Configuring the Release Status for Objects Requiring Approval.
- For general information about how to configure release statuses in the XML object ***ReleaseStatusDefs***, see the section Configuring the XML Object ReleaseStatusDefs.

Configuring Cost and Budgeting Data for Projects

Default values for business cases including the period, tax rate, and cost of capital as well as the cost definition types for project cost assessment and reporting must be configured in the XML object ***CostManagerDef*** in the configuration tool Alfabet Expand. The configuration applies to all project stereotypes configured for the project planning capability. The following cost views are relevant for projects:

- *Cost Report Page View*
- *Cost Accrual Page View*
- *Business Case Page View*
- *Business Case Comparison Page View*
- *Project, Skill Request and Resource Request Time Schedule Page View*
- *Cash Out Planning Page View*

Please note that the `Obligation` cost definition is a core concept for cash-out planning in projects. Therefore, the configuration of the cost definition `Obligation` is not relevant for the cash-out planning capability. In other words, if you specify that the `Obligation` cost definition should not be visible in the XML object ***CostManagerDef***, it will nevertheless be visible in the *Cash Out Planning Page View*.

For more information about configuring the default values for business cases as well as the cost definition types for project cost assessment and reporting, see the section Configuring Cost Management Capabilities.

All cost types and income types relevant for cost planning and analysis must be defined for the object class `Project` in the **Reference Data** functionality in the **Configuration** module. For more information, see

Configuring Cost Types and Income Types for Cost Management Capabilities in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.

Configuring Milestone Templates to Track Projects

In Alfabet, users can create milestones in order to track and manage projects. The XML object ***MilestoneManager*** allows you to configure milestone templates that users can select when defining milestones for a project.

Please note that enterprise milestones are also configured in the XML object ***MilestoneManager***. There is no need to use a similar or identical set of milestones for enterprise releases and projects. However, it is recommended that the two milestone concepts are aligned with one another. For more information about configuring enterprise milestones, see the section Configuring Milestone Templates to Track Enterprise Releases.

Milestone templates are defined according to the project methodology pursued by the enterprise in order to assess the stage gates or milestones of the project methodology or enterprise release cycle.

A milestone template bundles a set of milestones including the definition of the color scheme used for tracking and reporting milestones and the sequence and tolerance period for each milestone. The color of the milestone serves as an indicator providing information about the difference between the original planned date of the milestone and the current target date. Users may either assign a configured milestone template directly to a project or enterprise release or, in the case of projects, they may select a configured project template and assign it to the project.

Projected target dates are automatically set for each milestone based on the configuration defined in the milestone template, but these dates may be manually edited for each milestone. Alfabet users may delete any milestone from a project or enterprise release that they deem unnecessary. Only one milestone template may be assigned to a project or enterprise release. The realization of milestones can be tracked for the enterprise release or, in the case of project milestones, on the level of the project, project group, or bucket.

In order to track projects in a consistent manner, it is recommended that all project stereotypes in a project hierarchy are assigned the same milestone template. Milestone templates are assigned to a project in Alfabet via the *Project Milestones Page View* (`PRJ_Milestones`) of the relevant project and can be tracked in the *Project Tracking Overview Report Page View* (`PRJ_TimeScheduleReport`).

The captions and descriptions configured in the XML object ***MilestoneManager*** will be available in the vocabularies and can be translated. For more information about translating terms for the Alfabet interface, see the section Modifying, Translating and Managing the Vocabularies in the chapter Localization and Multi-Language Support for the Alfabet Interface.

To edit the XML object ***MilestoneManager***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder by clicking the + symbol. You will see all XML objects that can be edited.
2) Right-click the XML object ***MilestoneManager*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) The table below displays the XML attributes that can be edited for the XML object ***MilestoneManager***:

XML Element (bold) / XML Attribute	Explanation
MilestoneTemplate	The milestone template bundles a set of milestones. The XML element ***MilestoneTemplate*** contains the XML attributes that allow you to configure the name, caption, type of duration (date or percentage of completion) that the milestone measures, and a description of the milestone template. You can configure multiple ***MilestoneTemplate*** elements.
`Name`	Enter text to define the name of the milestone template. This is the name that users will see in the Alfabet interface. The name may not exceed 16 characters.
`DurationType`	The completion of milestones can be configured to represent either the number of days or the percentage of the completed milestone: • Enter "`Days`" if the milestone dates should be computed based on the number of days defined in the XML attribute `Duration` in the XML element ***MilestoneDef***. The milestone dates are computed based on the value specified in the XML attribute `Duration` plus the date of the previous milestone. • Enter "`Percents`" if the milestone dates should be computed based on the percentages of number of days completed as defined in the XML attribute `Duration` in the XML element ***MilestoneDef***. The dates are computed based on the project's start and end date and the percentage specified in the XML attribute `Duration`. The computed milestone dates will be displayed in the *Project Milestones Page View* (`PRJ_Milestones`), where they can be modified if necessary.
`Caption`	Enter text to define the caption of the milestone template. The caption serves as explanatory text about the milestone template when it is selected in the **Select Milestone Template** field in the **Create Milestones** editor available in the *Project Milestones Page View* (`PRJ_Milestones`).
`Description`	Enter text to define a description about the milestone template. The description is displayed in the **Create Milestones** editor available in the *Project Milestones Page View* (`PRJ_Milestones`) and should help the user to understand the milestone template.
MilestoneColorDef	This element allows you to configure the information to display on the milestones to indicate the milestone status. Each XML element ***MilestoneColorDef*** includes, a caption, colors, the interval of time when the status color should be displayed. The color serves as an indicator for the difference between the original planned date of the milestone and the current target date. For example,

XML Element (bold) / XML Attribute	Explanation

	you could configure the following information for the milestone statuses `Completed`, `Normal`, `Warning`, and `Danger`:
	
	`...`
	
	`<MilestoneColorDef Caption="Completed" Completed="true" BackColor="LightGrey" ForeColor="Green" TextColor="Blue" />`
	
	`<MilestoneColorDef Caption="Normal" ToDays="0" BackColor="Green" ForeColor="Black" TextColor="Blue" />`
	
	`<MilestoneColorDef Caption="Warning" FromDays="0" ToDays="9" BackColor="Yellow" ForeColor="Green" TextColor="Blue" />`
	
	`<MilestoneColorDef Caption="Danger" FromDays="10" BackColor="Red" ForeColor="Green" TextColor="Blue" />`
	
	`...`
	
	In this example definition, the following would occur:
	
	• A milestone that is marked as `Completed` in the **Milestone** editor will be colored light grey.
	• If there is no difference (0 days) between the original target date and the current target date, the milestone will be colored green.
	• If there is a difference of 0-9 days between the original target date and the current target date, the milestone will be colored yellow.
	• If there is a difference of 10 or more days between the original target date and the current target date, the milestone will be colored red.
	
	A legend will be displayed in the *Project Tracking Overview Report Page View* (`PRJ_TimeScheduleReport`) describing the color scheme.

`Caption`	Enter text to define the caption that should be displayed to describe the milestone status in the legend in the *Project Tracking Overview Report Page View* (`PRJ_TimeScheduleReport`). For example, a milestone status could be "Normal" or "Warning", etc.
`Completed`	Enter "`true`" if the color set (defined in the attributes `BackColor`, `ForeColor`, and `TextColor`) should be displayed when a user checks the **Completed** field in the **Milestone** editor to indicate that the milestone has been completed. Enter "`false`" if the color set should not be displayed when the milestone has been completed. The attribute `Completed` should only be set to "`true`" for one

XML Element (bold) / XML Attribute	Explanation
	XML element ***MilestoneColorDef*** definition. In this case, the `FromDays` and `ToDays` attributes are not necessary.
`FromDays`	Enter an integer that defines the period for which the color set should be displayed. The period of validity is the interval between the initial date of the milestone plus the `FromDays` value and the initial date of the milestone plus the `ToDays` value.
`ToDays`	Enter an integer that defines the period for which the color set should be displayed. The period of validity is the interval between the initial date of the milestone plus the `FromDays` value and the initial date of the milestone plus the `ToDays` value.
`BackColor`	Enter text to define the color of the outline of the box representing the milestone in the *Project Tracking Overview Report Page View* (`PRJ_TimeScheduleReport`). The value entered should be a Windows, Web, hexadecimal, or RGB color value.
`ForeColor`	Enter text to define the color of the interior of the box representing the milestone in the *Project Tracking Overview Report Page View* (`PRJ_TimeScheduleReport`). The value entered should be a Windows, Web, hexadecimal, or RGB color value.
`TextColor`	Enter text to define the color of the short name text in the box representing the milestone in the *Project Tracking Overview Report Page View* (`PRJ_TimeScheduleReport`). The value entered should be a Windows, Web, hexadecimal, or RGB color value.
MilestoneDef	This element allows you to configure the milestones that are bundled in the milestone template. An XML element ***MilestoneDef*** should be created for each milestone. Whenever a user selects a milestone template for a project, the milestone dates will be initially computed based on the `DurationType` and the `Duration` definitions. The computed dates can be manually adjusted. The milestones will be added automatically to the project and a legend will be displayed in the *Project Tracking Overview Report Page View* (`PRJ_TimeScheduleReport`).
`Priority`	Enter an integer that defines the position of the milestone in the set of milestones. The position also determines the priority ranking for milestones that have the same value and must Therefore, be stacked in the *Project Milestones Page View* (`PRJ_Milestones`). For example, a value of 1 indicates that this milestone should be the first that must be completed and will be stacked on top if milestones have the same

XML Element (bold) / XML Attribute	Explanation
	value. (If the milestone template consists of 5 milestones, you should define an integer in the range 1-5 for each XML element ***MilestoneDef***.)
`Duration`	Enter an integer that defines the default period between milestones. The milestone dates will be automatically defined in Alfabet, but one or more can be manually changed by a user. If no integer is defined for the XML attribute `Duration`, then the default is the integer "`1`" (`Duration` = one day or 1%.) The duration you define will depend on the definition of the XML attribute `DurationType`. See the explanation of the XML attribute `DurationType` to understand how the duration values are computed. If a user manually changes a milestone date, the subsequent dates will NOT be automatically adjusted.
`Name`	Enter text to define the name of the milestone. The name of the milestone is not displayed in the Alfabet interface.
`Caption`	Enter text to define the caption of the milestone. The caption typically describes the milestone and will be displayed in the *Project Milestones Page View* (`PRJ_Milestones`) and the legend of the *Project Tracking Overview Report Page View* (`PRJ_TimeScheduleReport`). For example, if the milestones are based on the XML attribute `DurationType` = "`Percents`", then it would be informative to the user to define such captions describing the percentage of project completion (for example, `25% Completed`, `50% Completed`, etc.) If the milestones are based on the XML attribute `DurationType` = "`Days`", then it would be informative to define a caption that describes the phase of the project completed by the milestone (for example, "`Design Complete`" and "`Test Complete`")
`ShortName`	Enter text to define a short name for the milestone. The short name is displayed in the box representing the milestone in the *Project Tracking Overview Report Page View* (`PRJ_TimeScheduleReport`). The milestone's short name and caption will be displayed in the legend. **Standard Milestones - Abbreviation** D *Design Complete* B *Build Complete* T *Test Complete* For reasons of usability, a short name should only be 1 or 2 characters long. Thus, for a milestone `25% Completed`, it is recommended to implement a short name such as `25`. For milestones `Design Complete` and `Test Completed`, it is

XML Element (bold) / XML Attribute	Explanation
	recommended to implement a short name using initials such as `D` and `T`.
`Description`	Enter text that describes the milestone. This text will be displayed to the user in the milestone's object profile and preview in the *Project Milestones Page View* (`PRJ_Milestones`).

4) In the toolbar, click the **Save**  button to save the XML definition.

Configuring Class Settings for Project Stereotypes

If you want to define class settings for a project stereotype, you must manually add the project stereotypes configured by your enterprise to the **Class Settings** folder in the **Presentation** tab. To add a project stereotype to the **Class Settings** folder, navigate to and right-click the relevant class setting for the object class `Project` and select either:

- **New Class Settings as Copy** to create a class setting as copy for the selected class, or
- **New Class Settings for Stereotype as Copy** to create a class setting for a stereotype of the selected class. In the dialog that opens, select the project stereotype and click **OK**. You must use this function if there are no class settings already defined for the new stereotype.

The project stereotype will be added with the naming convention `Project:ProjectStereotype`. The class settings can be edited, as needed. The class setting for the project stereotype can be configured in the same manner as class settings for other object classes.

Please keep in mind that excluding an object class from the Alfabet interface via the class setting definition may have consequences for the views in the **As-Is Architecture Definition** and **Affected Architecture Analysis** workspaces that are determined by the attribute `HasAsIsArchitecture`. For example, if you exclude the object class ICT Object via a class setting, ICT objects cannot be defined or displayed in the views in the **As-Is Architecture Definition** and **Affected Architecture Analysis** workspaces.

Please note that the colors of the projects displayed in *Project, Skill Request and Resource Request Time Schedule (Gantt) Page View*, Project Time Schedule (Gantt) page view, Skill Offer page view, and Skill Analysis page view are based on the class setting for the relevant project stereotype. The color of the outline of the box representing the start and end of the project stereotype is configured in the attribute `BackColor` in the XML object ***ProjectManager***.

For more information about configuring class settings, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.

Configuring Wizards for Project Stereotypes

Wizards can be defined for the project management capability. Like class settings for other object classes, each class setting defined for a project stereotype may have a different wizard assigned to it.

Please note, however, that when a wizard is created and defined, it is first defined independent of the project stereotype that it is relevant for. Thus, when defining the wizard, you must select the value `Project` for the **Class Name** attribute. The wizard is then assigned to a class setting for the relevant project stereotype. A wizard may be reused for multiple project stereotypes.

For more information about configuring wizards, see the chapter Configuring Wizards.

Configuring Projects for the Strategy Deduction Capability

Project stereotypes can be mapped to a specific value node stereotype in the **Strategy Deduction** capability. By mapping a project stereotype to a value node stereotype, users will be able to define the relevant projects as impacted by the value node in the *Affected Architecture Page View* of a value node. Typically, projects are associated with the leaf-level value node stereotype in the value node hierarchy because the projects usually represent the strategic initiatives that must be acted upon in order to realize the abstract strategic intentions.

To map a project stereotype to a value node, you must include the project stereotype in the `MappingClasses` attribute. The project stereotype must be written with the correct naming convention: `Project:ProjectStereotype`

For more information about configuring the **Strategy Deduction** capability, see the section Configuring the Strategy Deduction Capability.

About Project Stereotypes in Alfabet Query-Based Reports

To define the release status for a project stereotype in a report view, you must define the project stereotype that the release status definition is for in the **Custom Attribute** field in the attribute window.

The object class `Project` can be defined for the base class or `JOIN` of an Alfabet query. A project stereotype is an attribute for the object class `Project`. If a report is to be configured for the object class `Project` and the report should show objects of a specific project stereotype, then a `WHERE` clause must be added to the Alfabet query. For more information about configuring Alfabet queries, see the chapter Defining Queries.

For example:

```
WHERE Project.Stereotype LIKE 'StatementOfWork'
```


Configuring Domain Models and Domain Planning

The domain model capability in Alfabet is highly configurable and each domain stereotype will be configured with a name unique to your enterprise. Please note that all documentation provided by Software AG uses the generic term "domain" when referring to any object created for any domain stereotype.

The information below describes the configuration required so that users can work with domain models. A domain model describes a hierarchy of domains in the enterprise. Each domain represents a functional entity in the domain model and allows the enterprise architecture to be hierarchically partitioned into disjoint segments and structured, for example, from a functional or technological point of view. An enterprise may have only one domain model or may have multiple domain models. Each domain model consists of one or more configured domain stereotypes.

You can configure each domain stereotype and specify the permissible object classes (`BusinessProcess`, `BusinessObject`, `BusinessFunction`, `ICTObject`, `Application`, `Component`, `MarketProduct`, `StandardPlatform`, and `VendorProduct`) that mayare are permissible to be a domain owner may create for a domain as well as which object classes a domain owner may associate with a domain.

Once defined, a domain stereotype is similar to a standard object class. Correspondingly, you can configure the mandate capability to control user access to the domain model as a whole or to each domain stereotype in the domain model hierarchy. All domain stereotypes can be configured to be searchable and thus selected in the list of searchable object classes displayed in the **Search for** field in the **Simple Search** and **Browse** functionalities. Additionally, you can configure custom editors, wizards, custom object views, class settings, workflows, and reports for each domain stereotype, as needed.

Users may work with domains in the following functionalities.

- **Domains** explorer (`DOM_Explorer`): This business function allows users to create and define one or more domain models that reflect the critical areas in the enterprise. The explorer displays all domain models with their domains and the object classes specified as visible in the explorer for each domain. For more information about the configuration of evaluation groups, evaluation types (and their indicators), etc. required to evaluate business capabilities, see the chapter *Documenting the Enterprise's Functional Domains* in the reference manual *Portfolio Management Basic*.
- **Domain Planning** functionality (`DomainPlanning`). This business function allows users to plan and restructure a section of the domain model in the context of a solution domain project. In the solution domain project, users can restructure the domain hierarchy by creating new domains, deleting existing domains, and proposing successor domains for the objects referenced by the deleted domains. This occurs in the context of the solution domain project so that the actual domain model is not actually modified during the planning phase. Once the solution domain project is checked in to the Alfabet inventory, the solution domain project will overwrite the corresponding section of the domain model. This feature requires that the object class stereotypes that are configured for the object class `Domain` are also configured for the object class `SolutionDomain`. The specific configuration required to implement the **Domain Planning** functionality is explained in the section Configuring Solution Domains and the Domain Planning Capability
- **Capability Management** functionality (`CapabilityManagement` or `CAPM_Explorer`):This business function allows the business to be described by means of business capabilities instead of domains. Please note that there is no explicit class for capability in Alfabet. Capabilities are represented by the domains and business functions that have been defined in domain model. Therefore, to work with business capabilities, a domain model should be configured with domain stereotypes whereby business functions must be allowed for the domain stereotype on the lowest level of the domain

hierarchy. Permissible users must then first define the domains. Once the domains are defined, they are added by a permissible user to a business capability map via the **Capability Maps** explorer. The domains are then considered business capabilities and all functionalities relevant for business capabilities will be available. Other than the configuration of domain stereotypes, no additional configuration is required in Alfabet Expand. For more information about the configuration of evaluation groups, evaluation types (and their indicators), etc. required to evaluate business capabilities, see the chapter *Documenting the Enterprise's Business Capabilities* in the reference manual *Portfolio Management Basic*.

The following configuration steps are necessary to work with domain models:

- Create all relevant domain stereotypes for the object class `Domain` via the **Stereotypes** attribute.
- Specify the hierarchy of the domain stereotypes for each domain model in the XML object ***DomainManager***. Here you can also specify whether a domain stereotype is recursive, which architecture elements may be created or assigned to the domains based on the domain stereotypes, and which architecture elements that are owned by a domain are displayed in the **Domains** explorer.
- Configure mandates for the domain model, if necessary. In contrast to the implementation of mandates for other object classes, the mandate definition for the object class `Domain` is propagated from the root domains in the domain model to all subordinate domains. In other words, sub-domains will inherit the mandate definition from their parent domain and users cannot explicitly define a mandate for any domain subordinate to the root domain. If the mandate definition is changed for a root domain, then the mandate definition will be propagated to its sub-domains.
- Configure the domain glossary capability, if necessary. A domain glossary can be created in order to allow users to define and search for domain-specific terminology for objects assigned to a selected domain.
- Define the necessary custom editors, wizards, custom object views, class settings, workflows, and reports for each domain stereotype, as needed.
- Assign the relevant functionalities and views to the relevant user profile.

To map domain stereotypes to the **Strategy Deduction** capability, see the section Configuring the Strategy Deduction Capability.

The following information is available:

- Creating Domain Stereotypes for the Domain Model
- Configuring the Sequence of the Domain Stereotypes in the Domain Model
- Configuring Functionality for Domain Stereotypes in the XML Object DomainManager
- Implementing Mandates for the Domain Model
- Configuring the Domain Glossary Capability
- Configuring Solution Domains and the Domain Planning Capability
- Making the Domain Management and Domain Planning Functionalities Available to User Profiles

- Configuring Capability Maps for the Domains Explorer

Creating Domain Stereotypes for the Domain Model

Before you create one or more domain models, you should carefully consider the number of levels required for each domain model as well as which object classes may be mapped to a domain stereotype. Once the enterprise begins to work with the domain planning functionalities, you cannot easily add new or remove existing domain stereotypes.

You must create all object class stereotypes relevant for all domain models in the **Stereotypes** attribute for the object class `Domain`. For example, you plan to create two domains models, one domain to capture functionality relevant to the business and another to capture functionality relevant to the technology. In the example below, the business domain model consists of the domain stereotypes Area, Sub-Area, and Domain and the technology domain model consists of the domain stereotype Technology Domain. All domain stereotypes for all domain models must be configured in the **Stereotypes** attribute for the object class `Domain`:

Example of the XML definition in the **Stereotypes** attribute for the object class `Domain`:

```
<ClassStereotypes>
   <Stereotype Name="Area" Caption="Area" CaptionPlural="Areas"
   Comments="" HasMandates="false"/>
   <Stereotype Name="SubArea" Caption="Sub-Area" CaptionPlural="Sub-
   Areas" Comments="" HasMandates="false" />
   <Stereotype Name="Domain" Caption="Domain"
   CaptionPlural="Domains" Comments="" HasMandates="false" />
   <Stereotype Name="TechnologyDomain" Caption="Technology Domain"
   CaptionPlural="Technology Domains" Comments=""
   HasMandates="false" />
</ClassStereotypes>
```

When you create the domain stereotypes in the **Stereotypes** attribute for the object class `Domain`, you also specify which domain stereotypes the mandate capability applies to. Once a domain stereotype has been created, you can configure its class settings and specify whether the domain stereotype is searchable, and which icon and preview properties should be displayed for the domain stereotype. For more information about configuring class settings, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes.

Later on when the domain models are configured in the XML object ***DomainManager***, the solution designer will configure the business domain model to have the hierarchy levels Area > Sub-Area > Domain and the technology domain model to have the domain stereotype Technology Domain to be recursive so that a technology domain can have subordinate technology domains.

To define the domain stereotypes:

1) Go to the **Meta-Model** tab and navigate to the protected class node  for the object class `Domain` in the **Class Model** explorer.
2) Click the object class `Domain` in the explorer to open the attribute window.

3) Set the **Can Have Mandates** attribute to `True` if the mandate capability should be available for the domain planning capability. Select `False` if the mandate capability should not be available.

4) Click the **Browse** button available for the **Stereotypes** attribute to open the XML editor. For each domain stereotype in your domain model hierarchy, define the following XML attributes:

   - `Stereotype Name`: Enter the technical name for each domain stereotype. You may define a name of up to 64 characters. This name cannot be changed once the stereotypes are implemented.
   - `Caption`: Enter the caption for the domain stereotype. This is the name that users will see in the Alfabet interface.
   - `CaptionPlural`: Enter a plural form for the caption. This is the name in plural that users will see in the Alfabet interface.
   - `Comments`: Enter text in order to provide information about the domain stereotype. This information is only for the purposes of configuration. It is not displayed in the Alfabet user interface.
   - `HasMandates`: Enter True if the mandate capability should be available for the domain stereotype. To implement the mandate capability for a domain model, the root domain stereotype must be set to `HasMandates="true"` and all subordinate domain stereotypes must also be set to `HasMandates="true"`. Enter False if the mandate capability should not be available for the stereotype.

     The configuration of mandates for a domain model is determined by the domain stereotype that is at the root node of the domain hierarchy. All subordinate domain stereotypes in the domain model require the same mandate configuration as the root domain stereotype. For a more detailed explanation, see the section Implementing Mandates for the Domain Model.

5) Click the **Save** button to save your changes.

Configuring the Sequence of the Domain Stereotypes in the Domain Model

The sequence of the domain stereotypes in the domain model is determined in the XML object ***DomainManager***. The order of the domain stereotypes listed in the XML object determines the order of the levels in the domain hierarchy. Each domain model begins with the root domain stereotype specified in the first ***Stereotype*** element in the XML object ***DomainManager***, followed by its child domain stereotype in the next ***Stereotype*** element, followed by its child domain stereotype in the next ***Stereotype*** element, etc.


```
<DomainManager DomainClassName="Domain">
    <Stereotype Name="Area"
                AllowedArtifacts=""
                AllowAssociatedObjects="false">
        <Stereotype Name="SubArea"
                    AllowedArtifacts="BusinessProcess"
                    AllowAssociatedObjects="false">
            <Stereotype Name="Domain"
                        AllowedArtifacts="BusinessFunction,Application,
                        ICTObject,MarketProduct,BusinessProcess,BusinessObject"
                        CreateClasses="BusinessFunction,MarketProduct,BusinessObject"
                        AllowAssociatedObjects="false"
                        IsRecursive="true">
            </Stereotype>
        </Stereotype>
    </Stereotype>
<Stereotype Name="TechnologyDomain"
            AllowedArtifacts="Component,StandardPlatform,VendorProduct"
            CreateClasses="Component,StandardPlatform,VendorProduct"
            AllowAssociatedObjects="true"
            IsRecursive="true">
</Stereotype>
</DomainManager>
```

In the example above, the first domain model structures in the domain stereotypes in the hierarchy levels Area > Sub-Area, and Domain. The domain stereotype Technology Domain does not have subordinate domain stereotypes but has been defined as recursive so that any technology domain can have subordinate technology domains.

For more information about defining all elements in XML object ***DomainManager***, see the section Configuring Functionality for Domain Stereotypes in the XML Object DomainManager.

Configuring Functionality for Domain Stereotypes in the XML Object DomainManager

Once the domain stereotypes have been created for the object class `Domain`, you can specify one or more domain stereotype hierarchies in the XML object FeatureManager***DomainManager***. A domain stereotype may have multiple subordinate domain stereotypes. Furthermore, a domain stereotype may be configued to be recursive, thereby allowing subordinate domains to be defined that are based on the same domain stereotype. The order of the domain stereotypes listed in the XML object determines the order of the levels in the domain hierarchy. When a user creates a domain in the user interface, the stereotype selector will open if the new domain may be based on more than one domain stereotype. The user must select the relevant stereotype that the new domain shall be based on and then the relevant **Domain** editor will open. If only one domain stereotype is permissible, the new domain will automatically be based on the domain stereotype and the relevant **Domain** editor will open.

Additionally, you can specify whether business functions, business objects, business processes, applications, ICT objects, functional modules, components, standard platforms, market products and vendor products may be created for or associated with the domain stereotype.

To edit the XML object ***DomainManager***:

1) Go to the **Presentations** tab, expand the **XML Objects** folder, and expand the **SolutionManagers** folder.
2) Right-click the XML object ***DomainManager*** and select **Edit XML**. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.

3) Edit the XML attributes, as needed The table below displays the XML attributes that can be edited for the XML object ***DomainManager***:

XML Elements (bold) and Attributes	Explanation
DomainManager	
Domain-Class-Name	Enter `Domain`.

Stereotype	An XML element ***Stereotype*** should be created for each domain stereotype. The root domain stereotype should be the first ***Stereotype*** element, with the child stereotype element nested below the XML element ***Stereotype*** of its parent.
	For example:
	`<Stereotype Name="Area"AllowedArtifacts=""AllowAssociatedObjects="false">`
	`<Stereotype Name="SubArea" AllowedArtifacts="BusinessProcess" AllowAssociatedObjects="false">`
	`<Stereotype Name="Domain" AllowedArtifacts="BusinessFunction,Application" CreateClasses="BusinessFunction,MarketProduct,BusinessObject" AllowAssociatedObjects="false" IsRecursive="true">`
	`</Stereotype>`
	`</Stereotype>`
	`</Stereotype>`

Name	The domain stereotype name must be spelled exactly as defined in the XML attribute `Name` in the **Stereotypes** attribute for the object class `Domain`.
AllowedArtifacts	Enter the technical names of the object classes or their object class stereotypes that may be created for or associated with the domain stereotype. Possible object classes include `BusinessProcess`, `BusinessObject`, `BusinessFunction`, `FunctionalModule`, `ICTObject`, `Application`, `Component`, `MarketProduct`, `StandardPlatform`, and `VendorProduct`.

XML Elements (bold) and Attributes	Explanation
	The `AllowedArtifacts` attribute should include the following: • All object classes that can be created by an authorized user of a domain based on the domain stereotype. The object classes must also be specified in the XML attribute `CreateClasses`. • All object classes that can be associated with a domain based on the domain stereotype. If you set the XML attribute `AllowAssociatedObjects` to `"true"`, the domains based on this domain stereotype can be defined as associated domains for all object classes specified in the XML attribute `AllowedArtifacts`. All objects based on object classes specified in the XML attribute `AllowedArtifacts` that are owned by a domain will be displayed below that domain in the **Domains** explorer.
`CreateClasses`	Enter all relevant object classes listed in the XML attribute `AllowedArtifacts` that may be created for the domain stereotype. Possible object classes include `BusinessProcess`, `BusinessObject`, `BusinessFunction`, `FunctionalModule`, `ICTObject`, `Application`, `Component`, `MarketProduct`, `StandardPlatform`, and `VendorProduct`. **NOTE**: The object class must also be defined in the XML attribute `AllowedArtifacts` in order to be created for the domain stereotype. Please note that object class stereotypes are not allowed in the XML attribute `CreateClasses`. For every object class specified in the XML attribute `CreateClasses`, the **Create <ObjectClass>** menu option will be displayed in the respective page view in the domain stereotype's object profile. For example, if the object class `BusinessProcess` is defined in the XML attribute `CreateClasses`, then the **New** > **Create New Business Process** will be displayed in the *Business Processes Page View* (`DOM_BusinessProcesses`). For the object classes `ICTObject`, `Application`, `Component`, and `StandardPlatform`, the **Create New <ObjectClass> as Copy** menu option will also be displayed. Please note that for object classes not specified in the XML attribute `CreateClasses`, the relevant page view will be available in the domain's object profile but the **Create <ObjectClass>** menu option will be disabled. This page view may be deleted from a custom object view or removed from a user profile if it is not needed. For more information about hiding page views, see the section Refining Visibility Issues in the View Scheme in the chapter Configuring User Profiles for the User Community.
`AllowAssociatedObjects`	Enter `"true"` if objects based on any object class or object class stereotype listed in the XML attribute `AllowedArtifacts` may be added to the domain stereotype as associated objects. The menu option **New** > **Associate Existing <ObjectClass> to <DomainStereotype>** will be displayed in the respective page view in the relevant domain stereotype's object profile. If you define `"false"` for this attribute, the option **New** >

XML Elements (bold) and Attributes	Explanation
	Associate Existing <ObjectClass> to <DomainStereotype> will not be displayed in the respective page view. **NOTE**: Please note that the *Associated Domains Page View* (`OBJ_AssociatedDomains`) will be available per default in standard object views of the object classes `BusinessProcess`, `BusinessObject`, `BusinessFunction`, `FunctionalModule`, `ICTObject`, `Application`, `Component`, `MarketProduct`, `StandardPlatform`, and `VendorProduct`. If the XML attribute `AllowAssociatedObjects` is set to "`false`" for an object class, the *Associated Domains Page View* (`OBJ_AssociatedDomains`) should not be available in the relevant user profile. In this case, the view should be explicitly hidden from the relevant standard or custom object views. For more information about hiding page views, see the section Refining Visibility Issues in the View Scheme in the chapter Configuring User Profiles for the User Community. **NOTE**: Please note that the object class or object class stereotype must also be defined in the XML attribute `AllowedArtifacts` in order to be associated with the domain stereotype.
`IsRecursive`	Enter "`true`" if a child domain based on the same domain stereotype may be created for this domain stereotype. Enter "`false`" if a child domain based on the same domain stereotype may not be created for this domain stereotype.

4) In the toolbar, click the **Save** button to save the XML definition.

Implementing Mandates for the Domain Model

Some enterprises have a federated architecture. If this is the case in your enterprise, then mandates may be implemented to control the visibility of Alfabet objects. If you want the visibility of the domains in your domain model to also be controlled by the mandates implemented in your enterprise, then you will need to explicitly configure the mandate capability for all domain stereotypes in the domain hierarchy.

In contrast to the implementation of mandates for other object classes, the mandate definition for the object class `Domain` is propagated from the root domains in the domain model to all subordinate domains. In other words, sub-domains will inherit the mandate definition from their parent domain and users cannot explicitly define a mandate for any domain subordinate to the root domain. If the mandate definition is changed for a root domain, then the mandate definition will be propagated to its sub-domains.

The mandate configuration is typically defined when the domain stereotypes are created. However, the mandate configuration can be changed, as needed. Please keep the following in mind regarding the implementation of mandates for the domain model:

- The **Can Have Mandates** attribute of the object class `Domain` must be set to "`true`" in order to implement the mandate capability as a whole for the domain models in your enterprise as well as to display the *Mandates Page View* (`ObjectMandates`) in the object view of the root domain stereotype. The *Mandates Page View* will only be displayed on the root domains in the domain model.

You need to ensure that the *Mandates Page View* is assigned to all relevant object views for the user profiles that may define mandates. For general information about the mandates concept, see the section *Implementing the Mandate Capability for a Federated Architecture* in the chapter Configuring Access Permissions for Alfabet.

- The configuration of mandates for a domain model is determined by the domain stereotype that is at the root node of the domain hierarchy. All subordinate domain stereotypes in the domain model require the same mandate configuration as the root domain stereotype.
  - To implement the mandate capability for the domain model, the `HasMandates` attribute in the XML definition of the **Stereotypes** attribute of the object class `Domain` must be defined as "`true`" for the root domain stereotype as well as all subordinate domain stereotypes.
  - To deactivate the mandate capability for the domain model, the `HasMandates` attribute in the XML definition of the **Stereotypes** attribute of the object class `Domain` must be defined as "`false`" for the root domain stereotype as well as all subordinate domain stereotypes. Additionally, the solution designer must ensure that the **Mandates** page view (`ObjectMandates`) is removed from all relevant object views.
- For an overview of mandate configuration in the context of the various access permission concepts in Alfabet, see the chapter Configuring Access Permissions for Alfabet.

To implement the mandate capability for the object class `Domain`:

1) Go to the **Meta-Model** tab and navigate to the object class `Domain` in the **Class Model** explorer. Click the object class `Domain` in the explorer to open the attribute window.
2) Set the **Can Have Mandates** attribute to `True` if the mandate capability should be available for the domain model functionalities. Select `False` if the mandate capability should not be available.
3) Next, define the mandate capability for the object class. Click the **Browse** button available for the **Stereotypes** attribute to open the XML editor.
4) To implement the mandate capability for a domain model, the attribute `HasMandates` must be defined as "`true`" for the root domain stereotype and all subordinate domain stereotypes. If you do not want to implement the mandate capability for a domain model, the attribute `HasMandates` must be defined as "`false`" for the root domain stereotype and all subordinate domain stereotypes.
5) Close the editor by clicking **OK** in the editor.
6) Click the **Save** button to save your changes.

Configuring the Domain Glossary Capability

Alfabet provides a capability that allows a domain glossary to be created in order to allow users to define and search for domain-specific terminology for objects assigned to a selected domain. The domain glossary is defined by a relevant user in the *Domain Glossary Page View* in Alfabet and users can then search for the domain glossary term in the *Full-Text Search* page view. In order to implement this capability, you must ensure that the *Domain Glossary Page View* (`DOM_Glossary`) and *Full-Text Search* page view (`SRCH_FullTextSearch`) is included in the relevant object views.

If a domain glossary is defined, a search group must be configured in order to index, search, and display results for the domain glossary. In the XML object ***SearchManager***, you need to create at least one search group for the base class `Domain`. The search group should contain an Alfabet query for each object class that can be associated with a domain and for which users will be defining glossary items. These classes include: `Application`, `BusinessFunction`, `BusinessObject`, `Component`, `ICTObject`, `BusinessProcess`, `StandardPlatform`, and `VendorProduct`.

If domain stereotypes have been configured in your enterprise, the **Domain Glossary** functionality will be available for all domain stereotypes as long as the *Domain Glossary Page View* and *Full-Text Search* page view are included in the domain stereotype's object view. Therefore, if the domain glossary capability should not be available for a domain stereotype, then the *Domain Glossary Page View* and *Full-Text Search* page view should not be included in the domain stereotype object view. For more information about adding views to an object view, see the section Adding Page Views to the Object Profile Independent of a Workspace in the chapter Configuring Object Views.

- You must ensure that the attribute `Active` in the ***SearchManager*** element of the XML object ***SearchManager*** is defined as "`true`". This ensures that an index can be created for all existing search groups. If an index cannot successfully be created in the *Domain Glossary Page View*, users will not be able to use the capability. For more information about enabling the creation of an index, see the section Configuring the Full-Text Search Capability.
- In order to create an index and execute a search, a path must be defined by your system administrator describing where the search indexes needed for the full-text search functionalities are found. For more information, see the section *Installation* in the reference manual *System Administration* or contact your system administration.
- Please note that Alfabet provides another glossary capability that can be searched via the **Full-Text Search** functionality in the **Search** module. This glossary and its glossary items must be created in the **Glossary** functionality in the **Search** module. For more information about configuring a search group for the **Glossary** functionality, see the section Configuring the Glossary Search Functionality.

To edit the XML object ***SearchManager*** for a global search:

1) Go to the **Presentation** tab and expand the **XML Objects** folder by clicking the + symbol. You will see all XML objects that can be edited.
2) Right-click the XML object ***SearchManager*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) Define the remaining XML attributes, as needed. The table below provides a general explanation about the XML attributes that can be edited for the XML object ***SearchManager***:

XML Element (bold) / XML Attribute	Explanation
SearchGroup	
`Name`	Enter text that defines the name of the search group. This is the name that users will see when they select the search group in order to execute a search.
`Type`	Enter "`Object`" to define an object-centric search. **NOTE**: Relevant only for the definition of an object-centric search. The search is by default a global search. To define a global search, leave the **Type** attribute empty.
`BaseClass`	Enter "`Domain`" as the base class for the search. **NOTE**: If domain stereotypes have been configured in your enterprise, the domain glossary functionality will be available for all domain stereotypes as long as the *Domain Glossary Page View* and *Full-Text Search* page view are included in the domain stereotype's object view. Therefore, if the domain glossary capability should not be available for a domain stereotype, then the *Domain Glossary Page View* and *Full-Text Search* page view should not be included in the domain stereotype object view.
Query	You should create a query for each object class that can be associated with a domain and for which users will be defining glossary items. These classes may include: `Application`, `BusinessFunction`, `BusinessObject`, `Component`, `ICTObject`, `BusinessProcess`, `StandardPlatform`, and `VendorProduct`.
`Name`	Enter the name of the Alfabet query that shall be implemented in the object-centric search.
`ClassName`	Enter "`GlossaryItem`".
`Query`	Enter the Alfabet query to search for the objects that the index should be generated for (via the object class properties defined in the `ExportProps` attribute). For more information about defining an Alfabet query, see the chapter Defining Queries.
`ShowProps`	This attribute specifies the defined object class properties for all objects found by the search. Enter a comma-separated list of object class properties to be displayed in the search results for the objects found by the search. For example, you may want `Application.Version` or `Application.ShortName` to be displayed for all applications displayed in the results. The object

XML Element (bold) / XML Attribute	Explanation
	class properties are visualized in the results displayed via a data set as well as record-by-record. Please keep the following in mind. • Any object class properties defined in the `ImageProps` attribute should not be redefined again in the `ShowProps` attribute since the object class properties defined in the `ImageProps` attribute will already be displayed. • The results will display matches for any of the object class properties defined in the `ExportProps` attribute. For example, if `Application.Description` is defined in the `ExportProps` and the search term is found in value defined for an application's `Description` property, then the description will be displayed in the results with the matching words highlighted, even if `Application.Description` is not defined in the `ShowProps` attribute.
`ExportProps`	The attribute specifies the object class properties to be indexed. Enter a comma-separated list of object class properties to define all object class properties of the object class to be indexed. The `ExportProps` definition must include all object class properties defined in the `ShowProps` and `ImageProps` attributes.
`ImageProps`	Because the title line should be brief, the object class property `Description` should not be included in the `ImageProps` attribute. The attribute specifies the object class properties to display in the title line of the search results. Enter a comma-separated list of object class properties to create a concatenation that serves as the title line for each search result. The sequence of the object class properties listed in the `ImageProps` attribute determines the sequence displayed in the concatenated title. The object class properties defined in the `ImageProps` attribute are read from the Alfabet database when the index is created. The concatenation is displayed in the results displayed via a data set as well as record-by-record.
`ExportReference`	Enter "`GlossaryItem.ArtifactReference`". The attribute specifies that the search results should not display the glossary item found by the search, but rather the object specified in the Alfabet query that references the glossary item. (For example, the applications, business functions, etc.)

4) In the toolbar, click the **Save** button to save the XML definition.

Configuring Solution Domains and the Domain Planning Capability

If the **Domain Planning** functionality (`DomainPlanning`) is implemented in your enterprise, it is highly recommended that changes to the relationships between domains in the domain model are executed solely in the context of the domain planning capability.

If a sub-tree of the domain model is checked out of the inventory for the purposes of domain planning, the targeted domain and its subordinate domains can still be accessed and modified by users in the inventory. However, any changes that these users make to the inventory domain or its subordinate domains will NOT be written to the corresponding solution domain project.

Therefore, if the domain planning capability is implemented in your enterprise, it is recommended that users do not create or delete domains in the **Domains** explorer (`DOM_Explorer`) or *<Sub-Domains> Page View* (`DOM_SubDomains`) available via the domain object profile. Instead, domains should only be created and deleted in the context of a solution domain project in the *<Sub-Domains> Page View* (`SLNDOM_SubDomains`) of a solution domain.

If you plan to implement the **Domain Planning** functionality (`DomainPlanning`) in Alfabet, then you should configure the object class `SolutionDomain` so that it has the same configuration as the object class `Domain`. Consider the following aspects in order to configure the domain planning capability.

- Release statuses must be configured for the object class `SolutionDomainProject` in the XML object ***ReleaseStatusDefs*** in the configuration tool Alfabet Expand. Please note that the definition of the `ApprovedStatus` attribute is required so that a solution project domain can be checked in to the inventory. Be sure to click the **Save** button to save your changes. For more information about the configuration of release status definitions, see the section Configuring Release Status Definitions for Object Classes in the chapter Configuring the Class Model.
- If domain stereotypes are implemented in your enterprise, then you must configure the same domain stereotype definitions for the object classes `Domain` and `SolutionDomainProject`. To do so, copy the definition in the **Stereotypes** attribute for the object class `Domain` in the **Meta-Model** tab and paste the definition to the **Stereotypes** attribute for the object class `SolutionDomain`. Be sure to click the **Save** button to save your changes. For more information about the configuration of the stereotypes, see the section Configuring Object Class Stereotypes for Object Classes in the chapter Configuring the Class Model.
- You must define the class settings for each domain stereotype. It is recommended that the same icons are configured for the solution domains as for their corresponding domain. For more information about configuring class settings, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.
- If custom object class properties have been configured for the object class `Domain` and users should view or edit the custom object class properties in the context of a solution domain, then the relevant custom object class properties must be recreated for the object class `SolutionDomain`. The custom object class properties must have the same **Name** attribute definition for the object class `SolutionDomain` as the object class `Domain`. Be sure to click the **Save** button to save your changes. For more information about creating a custom object class property, see the section Configuring Custom Properties for Protected or Public Object Classes in the chapter Configuring the Class Model.

- Once the custom object class properties have been created, you can create a custom editor for the object class `SolutionDomain`. Be sure to click the **Save** button to save your changes. For more information about configuring custom object class properties, see the chapter Configuring Custom Editors.
- If necessary, you can also create a wizard for the object class `SolutionDomain`. Please note, however, that when a wizard is defined, it is first defined for the object class `SolutionDomain` and is independent of the solution domain stereotype that it is intended for. Thus, when defining the wizard, you must select the value `SolutionDomain` for the **Class Name** attribute. The wizard is alter assigned to a class setting for the relevant solution domain stereotype. Be sure to click the **Save** button to save your changes. For more information about configuring custom wizards, see the chapter *Configuring Wizards*.
- If custom editors or custom wizards are to be implemented for a solution domain stereotype or, if no stereotypes are implemented, for the object class `SolutionDomain`, then you must configure the class settings for the relevant object class (`SolutionDomain`:Stereotype or `SolutionDomain`). Be sure to click the **Save** button to save your changes. For more information about configuring class settings, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.

It is recommended that you configure reports in order to track the changes made to the domain model. For example, reports could be configured for the object classes `SolutionDomain`, `SolutionDomainProject`, and/or `Domain` in order to capture the data relevant to the enterprise. Users could then export the data to a Microsoft Excel, Microsoft Word or PDF file for archiving purposes and attach the document to the relevant object via the *Attachments Page View*. For more information about configuring reports, see the chapter Configuring Reports.

Making the Domain Management and Domain Planning Functionalities Available to User Profiles

Please take the following functionalities and views into consideration when configuring object views and user profiles:

- **Domains** explorer (`DOM_Explorer`): The **Domains** explorer displays all domain models with their domain hierarchies as well as the objects owned by the domains that have been configured to be displayed in the **Domains** explorer. This explorer should be available to all user profiles that work with domains.
- **Domain Admin Desktop** functionality (`DOM_AdminDesktop`): This functionality is available to the user profile `Admin` only and allows a user or solution administrator to address modifications in the ownership of objects that result from changes to the domain structure. For more information about the **Domain Admin Desktop** functionality, see the section *ManagingAlfabetObjects in the Context of Changes in the Enterprise* in the reference manual *User and Solution Administration*.
- *Domain Glossary Page View* (`DOM_Glossary`) and *Full-Text Search* page view (`SRCH_FullTextSearch`): These views should be available to the relevant user profiles if a domain glossary is implemented.

- *Domain Exchange Desktop Page View* (`DOM_ExchangeDesktop`):This view is an administrative view available in the standard object view for the object class `Domain` and is used for the quick maintenance of the entire domain model and allows the user to move business processes, applications, business functions, business objects, or functional modules from one domain to another. Please note that there is no validation mechanism regarding permissibility for this view and Therefore, should only be available to user profiles that are authorized to modify the domain model.

- Relevant page views needed to create or associate objects with a domain. In the XML object ***DomainManager***, the solution designer can specify that objects in the object classes `BusinessProcess`, `BusinessObject`, `BusinessFunction`, `FunctionalModule`, `ICTObject`, `Application`, `Component`, `MarketProduct`, `StandardPlatform`, and `VendorProduct` may be created for a domain stereotype and associated with a domain stereotype. For example, if the object class `BusinessProcess` is defined in the `CreateClasses` attribute, then the **New** > **Create New Business Process** will be displayed in the *Business Processes Page View* (`DOM_BusinessProcesses`) and if a business process may be associated with the domain, the menu option **New** > **Associate Existing <ObjectClass> to <DomainStereotype>** will be displayed in the *Business Processes Page View*. If an object class cannot be created or associated with a domain, you may want to remove this view from the domain's object profile. For more information about hiding page views, see the section Refining Visibility Issues in the View Scheme in the chapter Configuring User Profiles for the User Community.

- *Associated Domains Page View* (`OBJ_AssociatedDomains`): This page view will be available per default in standard object views of the object classes `BusinessProcess`, `BusinessObject`, `BusinessFunction`, `FunctionalModule`, `ICTObject`, `Application`, `Component`, `MarketProduct`, `StandardPlatform`, and `VendorProduct` regardless of the specification in the XML object ***DomainManager***. If the `AllowAssociatedObjects` attribute in the XML object is set to "`false`", the *Associated Domains Page View* (`OBJ_AssociatedDomains`) should not be available in the object profiles of the object classes `BusinessProcess`, `BusinessObject`, `BusinessFunction`, `FunctionalModule`, `ICTObject`, `Application`, `Component`, `MarketProduct`, `StandardPlatform`, and `VendorProduct` in the relevant user profile. In this case, the view should be explicitly hidden from the relevant standard or custom object views. For more information about hiding page views, see the section Refining Visibility Issues in the View Scheme in the chapter Configuring User Profiles for the User Community.

- **Domain Planning** functionality (`DomainPlanning`): The **Domain Planning** functionality allows users to plan and design an alternative hierarchy of domains for a section of the domain model in the context of a solution domain project. This feature requires that the same object class stereotypes that are configured for the object class `Domain` are also configured for the object class `SolutionDomain`.

For more information about assigning functionalities to user profiles, see the section Making Functionalities Accessible to a User Profile in the chapter Configuring User Profiles for the User Community. For more information about hiding page views in user profiles, see the section Refining Visibility Issues in the View Scheme in the chapter Configuring User Profiles for the User Community.

Configuring Capability Maps for the Domains Explorer

The XML object ***SolutionOptions*** allows you to define whether domains and business functions should be displayed in the **Domains** explorer or whether business capability maps should be displayed in the **Domains** explorer.

To edit the XML object ***SolutionOptions***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder.
2) Right-click the XML object ***SolutionOptions*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) In the XML attribute `BusinessFunctionType`, enter `BusinessFunction` to display domains and business functions in the **Domains** explorer tree. Enter `Capability` to display business capability maps in the **Domains** explorer tree.
4) In the toolbar, click the **Save** button to save the XML definition.

Configuring the Strategy Deduction Capability

The **Strategy Deduction** capability in Alfabet is highly configurable and each value node stereotype will be configured with a name unique to your enterprise. Therefore, all documentation provided by Software AG uses the generic term "value node" when referring to any object created for any value node stereotype.

The aim of the **Strategy Deduction** capability is to close the gaps between abstract strategic intentions and the day-to-day activities that are relevant to the architecture. After a high-level strategic intention is specified, strategic planners define a network of interrelated and increasingly more refined strategic intentions. This process of refinement enables planners to identify the areas of the enterprise requiring change and thus formulate the projects or initiatives needed to realize the corporate strategy.

In Alfabet Expand, you can configure the strategy network that will be made available to your Alfabet community in the **Strategy Deduction** functionality (`VM_StrategyDeduction`).

A strategy network is made up of an arbitrary number of levels called value node stereotypes. The top-level value node stereotype represents the most abstract level of strategic intention, and the lowest leaf-level value node stereotype represents those strategic initiatives that must be acted upon in order to realize the abstract strategic intentions. For example, a typical strategy network might be made up of the following value node stereotypes: Level 1: Vision, Level 2; External Trends, Level 3: Business Drivers, Level 4: Business Requirements, Level 5: Architectural Requirements, Level 6: Initiatives. The name, number, and hierarchical ordering of value node stereotypes is configurable.

A value node is thus defined for a specific value node stereotype. Any pair of value nodes that are related to one another in the strategy network must belong to different value node stereotypes that are adjacent to each other in the hierarchy. In other words, value nodes are related via a parent-child relationship. The relationship between two value nodes is represented by value node arcs, which allow the relative ranking of the strategic intentions to be computed.

Depending on the configuration of each value node stereotype, objects in specified object classes may be defined as impacted by a value node in the *architecture elements* of the relevant value node. One or more object classes can be mapped to a value node stereotype.

The following steps should be carried out for the configuration of the **Strategy Deduction** capability.

- First, you must create the value node stereotypes that make up the strategy network. A value node stereotype must be configured for each hierarchy level in the strategy network. Multiple value node stereotypes may be created for the root node of the value node hierarchy and multiple child value node stereotypes may be created for a parent value node stereotype. The object class stereotypes must be created as an XML definition in the **Stereotypes** attribute for the object class `ValueNode` in the **Meta-Model** tab. For more information about configuring the value node stereotypes, see the section Creating Value Node Stereotypes for the Strategy Network.
- Next, you must specify the hierarchical order and mapping classes for each value node stereotype in the XML object ***ValueManager***. For more information, see the section and Configuring the Value Node Hierarchy and Attributes via the XML Object ValueManager.
- Configure solution requirements for each value node stereotype as you would for a conventional object class. You can configure one or more custom editors, custom selectors, custom wizards, custom object views, custom reports, etc. for each value node stereotype. You must ensure that the *Affected Architecture Page View* (`VMND_ArchitectureElements`) is assigned to the custom object view of the relevant value node stereotype and the *Impacting Value Nodes Page View* (`ObjectValueNodes`) is assigned to the standard or custom object views of the object classes defined in the `MappingClasses` attribute in the XML object ***ValueManager***.
- Configure the names for the object states implemented for value nodes if they should differ from the standard object state names `Plan`, `Active`, and `Retired`. For more information about the configuration of object states, see the section Configuring Object State Definitions for Object Classes in the chapter Configuring the Class Model.
- Configure the class settings for each value node stereotype as you would for a conventional object class. A node for each value node stereotype will be automatically added to the **Class Settings** folder in the **Presentation** tab. For more information about configuring class settings, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.
- Ensure that the **Strategy Deduction** functionality (`VM_StrategyDeduction`) is available in the relevant user profile in the **User Profiles** folder in the **Admin** tab. For more information, see the section Making Functionalities Accessible to a User Profile in the chapter Configuring User Profiles for the User Community.

The following information is available:

- Creating Value Node Stereotypes for the Strategy Network
- Configuring the Value Node Hierarchy and Attributes via the XML Object ValueManager
- Configuring Alfabet to Capture Legal Ownership of Organizations
- Configuring Affected Architecture for Measure Types

- Configuring Icons for the Value Nodes Displayed in the Strategy Network Explorer
- Making the Strategy Deduction Capability Available to User Profiles

Creating Value Node Stereotypes for the Strategy Network

Before you create a strategy network, you should carefully consider the number of levels required as well as which object classes may be mapped to a value node stereotype. Once the enterprise begins to work with the **Strategy Deduction** capability, you cannot easily add new or remove existing value node stereotypes.

You must create a value node stereotype for each level in the strategy network. Multiple value node stereotypes may be created for the root node of the value node hierarchy and multiple child value node stereotypes may be created for a parent value node stereotype.

The value node stereotypes are hierarchically structured in the XML object ***ValueManager***. For more information, see the section Configuring the Value Node Hierarchy and Attributes via the XML Object ValueManager.

Once a value node stereotype has been created, you can configure its class settings and specify whether the value node stereotype is searchable, and which icon and preview properties should be displayed for the value node stereotype. For more information about configuring class settings, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes.

To define the value node stereotypes:

1) Go to the **Meta-Model** tab and navigate to the protected class node  for the object class `ValueNode` in the **Class Model** explorer.
2) Click the object class `ValueNode` in the explorer to open the attribute window.
3) Set the **Can Have Mandates** attribute to `True` if the mandate capability should be available for the value node hierarchy as a whole. Select `False` if the mandate capability should not be available.
4) Click the **Browse**  button available for the **Stereotypes** attribute to open the XML editor. You will see a text template that defines the value node stereotypes that make up the value node hierarchy. For each value node stereotype in your value node hierarchy, define the following XML attributes:

An error message will be displayed upon closing the text editor if syntax errors are written in the XML definition of the **Stereotypes** attribute.

- `Stereotype Name`: Enter the technical name for each object class stereotype. You may define a name of up to 64 characters. This name should not be changed once the object class stereotypes are implemented.

Changing the technical name after objects have been created based on the object class stereotype requires such objects to be updated by means of an ADIF scheme. For more information, see the reference manual *Alfabet Data Integration Framework*.

- `Caption`: Enter the caption for the object class stereotype. This is the name that users will see in the Alfabet user interface.

  Please note that the following:

  - The caption of the object class stereotype should be different than the caption of its object class. This is particularly important for the implementation of the search functionality for the object class stereotype. For more information, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.
  - The values defined for the XML attribute `Caption` can be translated whereas the values defined for the XML attribute `Stereotype Name` cannot be translated. However, in the table of the **Simple Search** functionality as well as other search selectors, the values defined for the XML attribute `Stereotype Name` of the object class stereotype are displayed in the **Stereotypes** column. If the values defined for the XML attribute `Caption` should be displayed, a custom selector should be defined. For more information, see the section Configuring a Custom Selector for Search Functionalities.
  - If you want to enter a caption `(Caption)` or plural form for the caption (`CaptionPlural`) that contains special characters (for example, a greater then (>) or lesser then (<) symbol), you must replace the special characters with respective XML compliant code, for example:
    - > for >
    - < for <
    - " for "
    - [for [
    -] for]
  - The caption will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.

- `Description`: Provide a brief description about the object class stereotype. The description will be displayed in the **Stereotype Selector** that opens when the user must select which stereotype a new object is to be based on. The description should help the user to understand which stereotype the new object should be based on. The description will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.
- `CaptionPlural`: Enter a plural form for the caption. This is the name in plural that users will see in the Alfabet user interface.
- `IDPrefix`: Enter no more than 5 characters for the ID prefix used to generate the unique identification number for new objects based on the object stereotype. If the XML attribute `IDPrefix` is not specified, the ID prefix of the object class that the object class stereotype is based on will be used to generate the unique ID. Please note that the integer generated for the unique ID is based on the number of instances created for the object class that the object

class stereotype is based on and includes all instances of all object class stereotypes created for the object class.

The prefix ID of existing objects based on an object class stereotype can be updated via the ADIF import scheme `SetStereotypeID` available in the `Alfabet Standard Jobs` folder. For more information, see the section *Predefined ADIF Schemes* in the reference manual *Configuring Alfabet with Alfabet Expand*.

- `Comments`: Enter text in order to provide information about the object class stereotype. This information is only for the purposes of configuration. It is not displayed in the Alfabet user interface.
- `HasMandates`: Enter "`true`" if the mandate capability should be available for the object class stereotype. Enter "`false`" if the mandate capability should not be available for the object class stereotype. If the XML attribute `HasMandates` is defined as "`false`" for an object class stereotype, objects based on the object class stereotype will be visible to users with any mandate implemented in your enterprise. For information about additional configuration requirements necessary to implement the mandate capability, see the section *Implementing the Mandate Capability for a Federated Architecture* in the chapter Configuring Access Permissions for Alfabet .
- `EnableStatutoryLanguage`: Set to `True` if the statutory language capability shall be available for the object class stereotype. The statutory language capability is available in order to address regulatory requirements that mandate that objects are captured in a specific language. When a user creates an object based on a relevant object class/object class stereotype, a statutory language selector will open in which the language can be selected. In the object editor that opens, the user can capture all attributes that support automated translation in the statutory language. If the XML attribute `EnableStatutoryLanguage` is not set, the object class stereotype will inherit the definition of the **Enable Statutory Language** attribute specified for the object class. Specifying a Statutory Language for the Enterprise

5) Click the **Save** button to save your changes.

Configuring the Value Node Hierarchy and Attributes via the XML Object ValueManager

Once the value node stereotypes have been created for the object class `ValueNode`, you can define the value node hierarchy in the XML object ***ValueManager***. You can define the value node hierarchy as a linear hierarchy or as a branched tree structure that allows complex value node structures to be captured. A value node stereotype may have multiple subordinate value node stereotypes. Furthermore, a value node stereotype may be configured to be recursive, thereby allowing subordinate value nodes to be defined that are based on the same value node stereotype. The order of the value node stereotypes listed in the XML object determines the order of the levels in the value node hierarchy.

When a user creates value node in the Alfabet user interface, the stereotype selector will open if the new value node may be based on more than one value node stereotype. The user must select the relevant stereotype that the new value node shall be based on and then the relevant **Value Node** editor will open. If only one value node stereotype is permissible, the new value node will automatically be based on the value node stereotype and the relevant **Value Node** editor will open. A *(Subordinate Value Nodes) Page View* (`VMND_SubNodes`) page view will be displayed for each value node stereotype subordinate to a root value node stereotype, whereby the caption of the *(Subordinate Value Nodes) Page View* will automatically

display the caption of the relevant subordinate value node stereotype that may be assigned to the selected value node.

In addition to defining the value node hierarchy, you must also define the object classes that can be mapped to a value node stereotype in order to specify which objects are impacted by a value node. One or more object classes can be mapped to a value node stereotype. Any object class representing an artifact in Alfabet can be assigned to a value node stereotype in the `MappingClasses` attribute in the XML object ***ValueManager***. It is recommended to map the lowest level value node stereotype to the object classes `Demand`, `Project`, or `Principle` in order to operationalize the strategy expressed by the value nodes in that branch of the value node hierarchy. However, you can specify any value node stereotype as permissible for the creation of demands, projects, or principles.

To edit the XML object ***ValueManager***:

1) Go to the **Presentations** tab, expand the **XML Objects** folder, and expand the **SolutionManagers** folder.
2) Right-click the XML object ***ValueManager*** and select **Edit XML**. The XML object can be edited in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) Specify the following XML attributes as child elements of the root node `ValueManager`:
   - `Gaps`: Enter texts or symbols that represent the relative gap or difference in importance between adjacent value nodes. The number of gaps required depends on the number of value node stereotypes defined. The gaps should be ordered from least important to most important. Please note that if you want to enter a string that contains special characters, you must replace the special characters with respective XML compliant code. For example, a greater than (>) must be entered as > or lesser than (<) symbol must be entered as <
   - `TimeSeriesGroup`: Enter the name of the time series group to implement for the strategy network. A time series group bundles a set of time series evaluation types for which target values can be defined for value nodes that the time series evaluation type is assigned to. For detailed information about working with and configuring time series evaluations, see the section *Creating a Time Series Period* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*. Please note the following:
     - The time series definition is uniform for all measure types and an editor allows the target and rated values to be maintained for the measures based on a measure type. Time series entries may be skipped for those measure types where values are not available.
     - The time series group must first be specified in the protected enumeration `TimeSeriesGroup`. For more information about the protected enumeration `TimeSeriesGroup`, see the chapter *Overview of Protected Enumerations* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.
4) Specify a child XML element `StereotypeDefinition` should be created for each value node stereotype in the value node hierarchy:
   - For each value node stereotype that may be created for the root node of the value node hierarchy, create an XML element `StereotypeDefinition` as a child of the XML element `ValueManager`. Multiple value node stereotypes may be created for the root node of the value node hierarchy.

The first level of value node stereotypes in the XML object represents the most ascendant level in the hierarchy. To change the order of the value node stereotypes, select the entire value node stereotype and cut and paste it to the correct position in the order. Be sure to include the syntactic constructs (< and />), otherwise you will see an error message at the bottom of the XML editor.

- Define the following XML attributes for the XML element `StereotypeDefinition`:
  - `Stereotype`: Enter the technical name of the value node stereotype. This is the name defined in the with the XML attribute `Name` in the XML specification of the **Stereotypes** attribute for the object class `ValueNode`. The stereotype name is passed on to the objects that users create for the selected value node stereotype.
  - `MappingClasses`: Users can define objects in the object classes specified here as impacted by value nodes based on the value node stereotype. Enter a comma-separated list of object classes that may be mapped to the value nodes assigned to the value node stereotype. Please note that if the class `Demand`, `Project`, or `Principle` shall be added to the XML attribute `CreateClasses`, then it must also be included in the XML element `MappingClasses`. Please note the following:
    - The following object classes may be defined in the `MappingClasses` attribute: `Application`, `Brand`, `BusinessFunction`, `BusinessData`, `BusinessObject`, `BusinessProcess`, `Component`, `CustomerSegment`, `Demand`, `Device`, `Domain`, `ICTObject`, `InformationFlow`, `ITStrategy`, `ITStrategyMap`, `Market`, `MarketProduct`, `MasterPlan`, `MasterPlanMap`, `MasterPlatform`, `OrgaUnit`, `Peripheral`, `Principle`, `Project`, `SalesChannel`, `SystemBuildingBlock`, `StandardPlatform`, `StrategicBusinessSupport`, `TacticalBusinessSupport`, and `VendorProduct`. You must enter the value of the `Name` attribute of the object classes.

      If master plan maps and IT strategy maps are to be defined for a value node, then the object classes `BusinessProcess`/`Domain`, `OrgaUnit`/`MarketProduct` must be mapped to the relevant value node stereotype because objects in these classes are used in business support maps.

    - To map object class stereotypes to a value node (for example, a project stereotype or an application stereotype), add each relevant object class stereotype to the `MappingClasses` attribute for the value node stereotype. The object class stereotype must be written using the correct naming convention (for example: `Project`:ProjectStereotype).
    - The specified object classes will automatically be added to the **New** drop-down menu in the *Affected Architecture Page View* (`VMND_ArchitectureElement`). If the `MappingClasses` attribute is not defined for a value node stereotype, the *Affected Architecture Page View* (`VMND_ArchitectureElement`) will be displayed in the object profile but will not be functional. If no object classes are specified in the `MappingClasses` attribute, then the *Affected Architecture Page View* will be automatically removed from the object views of all value nodes.
    - For each relationship defined between an object in a mapped class and a value node, an object of the object class `ValueNodeArch` will be created. If the definition of the `MappingClasses` attribute is changed AFTER objects of the object class `ValueNodeArch` have been created, the existing relationships

defined (in other words, the objects of the object class `ValueNodeArch`) will continue to be available even though the mapped object class may no longer be referenced for the value nodes stereotype. In this case, the user will still be able to detach the mapped object from the value node.

- Please ensure that the exclusion of an object class via a class setting definition does not conflict with the definition of the XML attribute `MappingClasses`.

- `CreateClasses`: Demands, projects, and principles or their object stereotypes may be created in the *Affected Architecture Page View* as objects that operationalize that value node. Enter a comma-separated list of the object classes `Demand`, `Project`, or `Principle` or their object stereotypes that may be created for the value node stereotype. Please note that any classes specified for the XML attribute `CreateClasses` must also be specified in the XML attribute `MappingClasses`. The options **Create New Demand**, **Create New Project**, and **Create New Principle** (or any of their stereotypes) will be available in the *Affected Architecture Page View* of the relevant value node.

- `HasObjectives`: Enter "`true`" if objectives may be defined for the value node stereotype. Objectives allow the value node to be refined and specified.

- For each permissible child value node of a parent value node, create an XML element `ChildStereotype` as a child of the parent XML element `Stereotype Definition`. Multiple child XML elements `ChildStereotype` may be created for a parent value node stereotype. Add the XML attribute `Name` to each XML element `ChildStereotype` and enter the technical name of the value node stereotype.

5) In the toolbar, click the **Save**  button to save the XML definition.

Configuring Alfabet to Capture Legal Ownership of Organizations

Alfabet provides a functionality that allows the legal ownership of organizations to be captured. Please note the following configuration required to implement this functionality:

- Custom properties may be captured for the object class `LegalOwnership` in order to capture relevant data regarding the legal ownership relationships in your enterprise. For more information, see the section Configuring Custom Properties for Protected or Public Object Classes.

- The **Legal Owners** page view (`ORG_LegalOwnerships`) and the **Legally-Owned Organizations** page view (`ORG_LegalSubordinations`) must be assigned to the custom object views configured for the relevant object class stereotypes configured for the object class `OrgaUnit`. The **Legal Owners** page view (`ORG_LegalOwnerships`) should be available for the organization stereotype that owns other legal entities and the **Legally-Owned Organizations** page view (`ORG_LegalSubordinations`) should be available for the organization stereotype that represents organizations owned by other legal entities. The percentage that various organizations own a given organization is defined for the owned organization in the **Legal Owners** page view (`ORG_LegalOwnerships`). For more information, see the sections Configuring Object Class Stereotypes for Object Classes and Configuring Object Views.

- The values available for the **Ownership Type** attribute in the **Legal Ownership** editor (`LEGOWN_Editor`) may be configured via the enumeration ***OwnershipType***. For more information, see the section Defining Protected and Custom Enumerations.

Configuring Affected Architecture for Measure Types

You can specify the object classes and objects class stereotypes that may be added as affected architecture for measure types.

1) Go to the **Presentations** tab, expand the **XML Objects** folder, and expand the **Solution Managers** folder
2) Right-click the XML object ***AffectedArchManager*** and select **Edit XML**. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) Create an XML element `ClassEntry` and define the following XML attributes:
   - `ClassName`: Enter the technical name `VMMeasureType`.
   - `ArchitectureClasses`: Enter a comma-separated list of object classes and objects class stereotypes that may be added as affected architecture for risk mitigation templates. A menu entry with the caption **Add <Object Class>** or **Add <Object Class Stereotype>** will be added to the *Architecture Elements Page View*. If you are defining an object class stereotype in the XML attribute `ArchitectureClasses`, you must use the syntax ObjectClass:ObjectClassStereotype. If the XML attribute `ArchitectureClasses` is not defined, the standard object classes preconfigured by Software AG will be displayed.
4) In the toolbar, click the **Save** button to save the XML definition.

Configuring Icons for the Value Nodes Displayed in the Strategy Network Explorer

The icons to display for value nodes in the **Strategy Network** explorer (`VMND_Explorer`) which is implemented in the **Strategy Deduction** functionality (`VM_StrategyDeduction`) are configured via the class settings of each value node stereotype. You must configure a class setting for each value node stereotype.

Please note that the icon definition for a value node stereotype will only be displayed for value nodes with an `Active` object state. Value nodes that do not have an `Active` object state are automatically displayed with different icons in the explorer tree. All value nodes with the object state `Plan` will be displayed with the icon and all value nodes with the object state `Retired` will display the icon.

For more information about configuring class settings, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community. For more information about the configuration of object states, see the section Configuring Object State Definitions for Object Classes in the chapter Configuring the Class Model.

Making the Strategy Deduction Capability Available to User Profiles

Please take the following functionalities and views into consideration when configuring object views and user profiles:

- Ensure that the **Strategy Deduction** functionality (`VM_StrategyDeduction`) is available in the relevant user profile in the **User Profiles** folder in the **Admin** tab. For more information, see the section Making Functionalities Accessible to a User Profile in the chapter Configuring User Profiles for the User Community.

- Ensure that the *Affected Architecture Page View* (`VMND_ArchitectureElements`) is assigned to the relevant object view implemented for each value node stereotype.

  The visibility of the **Add Existing <Object Class>** options available in the **Affected Architecture** (`VMND_ArchitectureElements`) page view for a value node can be controlled in the **Customization Editor** available for the view. For each object class mapped to a value node stereotype via the XML attribute `MappingClasses` in the XML object `ValueManager`, an entry will be added below the **New** menu entry in the **Customization Editor**. The solution designer may exclude the menu option from the use profile as well as specify a custom selector for the option. For more information about configuring the visibility of menu options, see the section Hiding Functionalities in a Page View or Configured Report .

- Ensure that the *Objectives Page View* (`VMND_Objectives`) is assigned to the relevant object view implemented for this relevant value node stereotype.
- If desired, assign the *Impacting Value Nodes Page View* (`ObjectValueNodes`) to the object views of the object classes defined in the `MappingClasses` attribute.

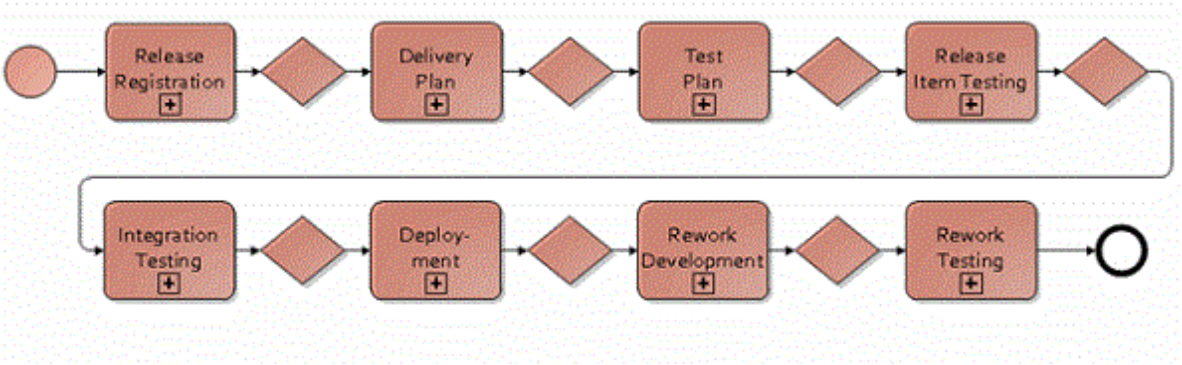
Configuring the Capture Enterprise Releases Capability

The **Capture Enterprise Releases** functionality supports the enterprise in capturing and managing project deliverables designed to provide architectural change to the IT landscape in a coordinated and managed process. In the context of the **Capture Enterprise Releases** functionality, requirements for each enterprise release cycle can be documented and enforced, thus reducing the risk of failure, inadequate performance, and security breaches or disruption to services resulting from change. Proposed projects associated with the enterprise release are required to go through a formal assessment and approval process before they can be registered as part of the enterprise release. The implementation of stage gates ensures that each phase in the execution of the enterprise release is managed and governed in alignment with the enterprise release calendar.

The enterprise release consists of enterprise release items which express the deliverables that will be provided with the enterprise release. Release statuses are implemented in order to manage the different stages of the enterprise release and the approval process for each individual delivery comprised in the enterprise release. Typically, an enterprise release item is based on a project that has been defined alongside with the architectural changes it provides. Alternatively, enterprise release items can be based on applications, components, or standard platforms. For each enterprise release, a set of milestones with target dates constitute the stage gates for the approval and execution of the release and allow the enterprise release cycle to be tracked and controlled. In order to manage the activities that must be realized for each milestone in the enterprise release, the Enterprise Release capability is typically implemented in conjunction with Alfabet 's workflow capability.

To implement the **Capture Enterprise Releases** functionality (`ENTRLS_CaptureReleases`), the following must be configured:

- The **Capture Enterprise Releases** functionality (`ENTRLS_CaptureReleases`) must be assigned to the relevant user profile(s).
- Release status definitions must be configured for the object class `EnterpriseRelease`. The release status definition for the enterprise release specifies the statuses of the enterprise release cycle as a whole. Suggested statuses for the enterprise release are:
  - New: Initial status after enterprise release is created.
  - Planned: Enterprise release is being considered for implementation.
  - InExecution: Enterprise release is being actively pursued. Defined milestones are relevant for an enterprise release in the InExecution status.
  - Closed. Enterprise release item has been executed and is complete.
- A milestone template must be configured for the enterprise release. The milestones configured are implemented during the InExecution status of the enterprise release. Typical milestones that could be configured for the **Enterprise Release** milestone template are:



- Release Registration: Initial execution phase. Enterprise release items must be submitted for registration in the enterprise release. Submission of enterprise release items is limited to this phase.
- Delivery Plan: Participating enterprise release items must submit a delivery plan.
- Test Plan: Test Plan document must be attached to the enterprise release items
- Release Item Testing: Stand-alone testing of enterprise release items must be completed and documented
- Integration Testing: Integration testing of all enterprise release items completed and documented.
- Deployment: Enterprise release items must be ready for deployment.
- Rework Development: Development revisions to enterprise release items that have been conditionally approved.
- Rework Testing: Testing of revisions to conditionally approved items.
- Completion: All milestone phases completed and all items in status Closed.

- Release status definition must be configured for the object class `ReleaseItem`. The release status definition for enterprise release items should include the following status definitions:
  - The statuses relevant for an enterprise release item to be approved/rejected for the release during the phase prior to the Release Registration milestone.
  - The statuses relevant for an enterprise release item when the enterprise release is completed.
  - The statuses relevant for the milestones representing the execution stages of the release cycle (for example, Delivery Plan, Test Plan, Testing, etc.)

    Please note that if enterprise release items are based on projects, the projects may also have milestones configured that are relevant for the approval and execution process of the project.

Suggested statuses for enterprise release items are:

- Accepted: Enterprise release item has been accepted for participation in release execution.
- Rejected: Not accepted during registration phase.
- In Preparation: Pending delivery for requirements of Registration milestone.

  - Under Review: Project Manager/release item responsible has submitted the required delivery for the current stage. The delivery is under review by the Release Manager.
  - Rework required: The delivery of the enterprise release item must be revised. The enterprise release item is still active.
  - Passed: Current stage passed, ready to enter next.
  - Cond. Passed: Current stage not OK, pass anyway. May require rework in later stage.
  - Retracted: Project Manager withdraws.
  - Excluded: Kicked out during execution, not passed.
  - Closed: Reached final milestone, open for postmortem changes.
  - Archived: Final state. No more changes possible.
- A workflow should be configured for each enterprise release milestone in the enterprise release cycle in order to standardize the tasks required to fulfill the milestone by the specified target date.
- Typical roles involved in the enterprise release management process include Release Manager, Project Manager, and Application Owner. You must ensure that relevant user profiles are available and that the views relevant to the user profile and your enterprise's implementation of the **Capture Enterprise Releases** functionality (`ENTRLS_CaptureReleases`) are available.

The following configurations are available for the **Capture Enterprise Releases** functionality (`ENTRLS_CaptureReleases`). The configuration should be carried out according to conventional configuration procedures described in this manual.

- Searchability of the object classes `EnterpriseRelease` and `ReleaseItem`.
- Mandate configuration for the object classes `EnterpriseRelease` and `ReleaseItem`.
- Monitor configuration for relevant target dates, review dates, etc. for the object classes `EnterpriseRelease` and `ReleaseItem`.
- Configuration of custom editors, wizards, custom object views, custom class settings, workflows, and configured reports.

The following information is available:

- Configuring Release Status Definitions for the Capture Enterprise Releases Capability
- Configuring Milestone Templates to Track Enterprise Releases
- Configuring User Profiles and Visibility of Views in the Enterprise Release Capability

Configuring Release Status Definitions for the Capture Enterprise Releases Capability

A release status definition must be created for the object classes `EnterpriseRelease` and `ReleaseItem` in the XML object ***ReleaseStatusDefs*** in the configuration tool Alfabet Expand. The release status definition includes the complete set of release statuses implemented in the context of the `ENTRLS_CaptureReleases` functionality as well as the possible sequences permissible for the release statuses to reach a specific target status.

Please note that the `RetiredStatusSet` property for the object class `EnterpriseRelease` should be defined. Otherwise, all release statuses will be considered a retired status for an enterprise release and can be deleted.

For information about how to configure the release status definition in the XML object ***ReleaseStatusDefs***, see the section Configuring Release Status Definitions for Object Classes in the chapter Configuring the Class Model.

Configuring Milestone Templates to Track Enterprise Releases

In Alfabet, users can create milestones in order to track and manage enterprise release. The XML object ***MilestoneManager*** allows you to configure milestone templates that users can select when defining milestones for an enterprise release.

Please note that project milestones are also configured in the XML object ***MilestoneManager***. There is no need to use a similar or identical set of milestones for enterprise releases and projects. However, it is recommended that the two milestone concepts are aligned with one another. For more information about configuring project milestones, see the section Configuring Milestone Templates to Track Projects.

Milestone templates are defined according to the project methodology pursued by the enterprise in order to assess the stage gates or milestones of the project methodology or enterprise release cycle.

A milestone template bundles a set of milestones including the definition of the color scheme used for tracking and reporting milestones and the sequence and tolerance period for each milestone. Users may either assign a configured milestone template directly to a project or enterprise release or, in the case of projects, they may select a configured project template and assign it to the project.

Projected target dates are automatically set for each milestone based on the configuration defined in the milestone template, but these dates may be manually edited for each milestone. Users may delete any milestone from a project or enterprise release that they deem unnecessary. Only one milestone template may be assigned to a project or enterprise release. The realization of milestones can be tracked for the enterprise release or, in the case of project milestones, on the level of the project, project group, or bucket.

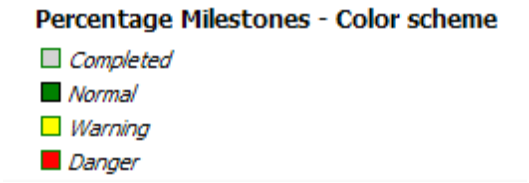
Enterprise release milestones must be configured in the XML object ***MilestoneManager***. The enterprise release milestones represent the stage gates and the requirements that must be fulfilled to pass from one stage to the next. If enterprise release items are based on projects, the projects may also have milestones configured that are relevant for the approval and execution process of the project. There is no need to use a similar or identical set of milestones for enterprise releases and projects. However, it is recommended that the two milestone concepts are aligned with one another.

To edit the XML object ***MilestoneManager***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder by clicking the + symbol. You will see all XML objects that can be edited.
2) Right-click the XML object ***MilestoneManager*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects. The following is an example of a typical definition of milestones for the **Capture Enterprise Releases** capability.

3) The table below displays the XML attributes that can be edited for the XML object ***MilestoneManagerxml***:

XML Element (bold) / XML Attribute	Explanation
MilestoneTemplate	The milestone template bundles a set of milestones. The ***MilestoneTemplate*** element contains the attributes that allow you to configure the name, caption, type of duration (date or percentage of completion) that the milestone measures, and a description of the milestone template.
`Name`	Enter text to define the name of the milestone template. This is the name that users will see in the Alfabet interface.
`DurationType`	Enter "`Days`" to indicate that the duration of the milestone completion should be computed based on the number of days defined in the `Duration` attributes in the `MilestoneDef` elements. The milestone dates are computed based on the enterprise release's start date plus the value specified in the `Duration` attribute. The computed milestone dates will be displayed in the *Enterprise Release Milestones Page View*, where they can be modified if necessary. **NOTE**: The value "`Percents`" is typically only relevant in the context of project milestones and should not be implemented in the context of enterprise release milestones.
`Caption`	Enter text to define the caption of the milestone template. The caption serves as explanatory text about the milestone template when it is selected in the **Select Milestone Template** field in the **Create Milestone** editor available in the *Enterprise Release Milestones Page View*.
`Description`	Enter text to define a description about the milestone template. The description is displayed in the **Create Milestones** editor available in the *Enterprise Release Milestones Page View* and should help the user to understand the milestone template.
MilestoneColorDef	This element allows you to configure the milestone statuses to display on the milestones. Each ***MilestoneColorDef*** element includes a caption, color definition, and the period of time that the milestone is valid. For example, you could configure the milestone statuses Completed, Normal, Warning, and Danger. A legend will be displayed in the *Project Milestones Page View* describing the color scheme.

XML Element (bold) / XML Attribute	Explanation
	
`Caption`	Enter text to define the caption that should be displayed to describe the milestone status in the legend in the *Project Milestones Page View*. For example, a milestone status could be "Normal" or "Warning", etc.
`Completed`	Enter "`True`" if the color set (defined in the attributes `BackColor`, `ForeColor`, and `TextColor`) should be displayed when the milestone has been completed. Enter "`False`" if the color set should not be displayed when the milestone has been completed. The attribute `Completed` should only be set to "`True`" for one ***MilestoneColorDef***. In this case, the `FromDays` and `ToDays` attributes are not necessary.
`FromDays`	Enter an integer that defines the period for which the color set should be displayed. The period of validity is the interval between the initial date of the milestone plus the `FromDays` value and the initial date of the milestone plus the `ToDays` value.
`ToDays`	Enter an integer that defines the period for which the color set should be displayed. The period of validity is the interval between the initial date of the milestone plus the `FromDays` value and the initial date of the milestone plus the `ToDays` value.
`BackColor`	Enter text to define the color of the outline of the box representing the milestone in the *Enterprise Release Milestones Page View*. The value entered should be a Windows, Web, hexadecimal, or RGB color value.
`ForeColor`	Enter text to define the color of the interior of the box representing the milestone in the *Enterprise Release Milestones Page View*. The value entered should be a Windows, Web, hexadecimal, or RGB color value.
`TextColor`	Enter text to define the color of the short name text in the box representing the milestone in the *Enterprise Release Milestones Page View*. The value entered should be a Windows, Web, hexadecimal, or RGB color value.
MilestoneDef	This element allows you to configure the milestones that are bundled in the milestone template. Whenever a user selects a milestone template for an enterprise release, the milestone dates will be initially computed based on the

XML Element (bold) / XML Attribute	Explanation
	`DurationType` and the `Duration` definitions. The computed dates can be manually adjusted. The milestones will be added automatically to the enterprise release and a legend will be displayed in the *Enterprise Release Milestones Page View*.
`Priority`	Enter an integer that defines the position of the milestone in the set of milestones. The position also determines the priority ranking for milestones that have the same value and must Therefore, be stacked in the *Enterprise Release Milestones Page View*. For example, a value of 1 indicates that this milestone should be the first that must be completed and will be stacked on top if milestones have the same value. (If the milestone template consists of 5 milestones, you should assign an integer from the range 1-5 to each ***MilestoneDef***.)
`Duration`	Enter an integer that defines the default period between milestones. The milestone dates will be automatically defined in Alfabet, but any or all can be manually changed by a user. For example, if you define a value of 50, the first milestone's start date will be automatically set to 50 days after the start date of the enterprise release, the second milestone's start date will be automatically set to 50 days after the first milestone's start date, etc. **NOTE**: If a milestone date is manually changed, the subsequent dates will NOT be automatically adjusted. If no integer is defined for the `Duration` attribute, then the default is the integer 1 (`Duration` = one day)
`Name`	Enter text to define the name of the milestone. The name of the milestone is not displayed in the Alfabet interface.
`Caption`	Enter text to define the caption of the milestone. The caption typically describes the milestone and will be displayed in the *Project Milestones Page View* and the legend of the *Enterprise Release Milestones Page View*. Since the milestones are based on the `DurationType` = "`Days`", then it would be informative to define a caption that describes the phase of the enterprise release completed by the milestone (for example, "Release Registration" and "Release Delivery Date")
`ShortName`	Enter text to define a short name for the milestone. The short name is displayed in the box representing the milestone in the *Enterprise Release Milestones Page View*. The milestone's short name and caption will be displayed in the legend.

XML Element (bold) / XML Attribute	Explanation
	Standard Milestones - Abbreviation D *Design Complete* B *Build Complete* T *Test Complete* **WARNING**: For reasons of usability, a short name should only be 1 or 2 characters long. Thus, for milestones Release Registration and Release Delivery Plan, it is recommended to implement a short name using initials such as RR and RD.
`Description`	Enter text that describes the milestone. This text will be displayed to the user in the milestone's preview in the *Project Milestones Page View* when the milestone is selected.

4) In the toolbar, click the **Save**  button to save the XML definition.

Configuring User Profiles and Visibility of Views in the Enterprise Release Capability

Please keep the following in mind regarding the configuration of user profiles and visibility issues.

- Ensure that the **Capture Enterprise Releases** functionality (`ENTRLS_CaptureReleases`) is available to the relevant user profile. For more information about assigning functionalities to user profiles, see the section Making Functionalities Accessible to a User Profile in the chapter Configuring User Profiles for the User Community.
- It is highly recommended that the enterprise release items in your enterprise are based on projects. Per default, the *Enterprise Release Items Page View* (`ENTRLS_Objects`) allows projects, applications, components, and standard platforms to be defined for enterprise release items. It is recommended that the menu items **New** > **Add** <Object Class> are excluded for all classes not used in the context of the enterprise release.
- The *Associated Enterprise Releases Page View* (`ENTRLS_ObjectReleases`) should also be removed from the object views of the object classes `Project`, `Application`, `Component`, or `StandardPlatform` if enterprise release items cannot be created for these classes.
- If enterprise release items are NOT based on projects, then the *Project Dependency Map Page View* (`ENTRLS_PRojectDependencyMap`) should be removed from the **Enterprise Release** object view.
- If enterprise release items are NOT based on projects or applications, then the *Business Support Map Report Page View* (`EITMPM_ObjectMatrixReport`) should be removed from the **Enterprise Release** object view.

Configuring the Contract Management Capability

The **Contract Management** functionality supports the enterprise in managing contracts and thus provides transparency to the use of contracts and the enterprise architecture. This capability ensures that the responsibility for contracts or aspects of a contract is understood in the enterprise and that necessary measures required to maintain or terminate contracts are timely and cost-effective. The **Contract Management** capability provides transparency regarding the contracts targeting architecture elements in the Alfabet database. The enterprise can document the deliverables required to fulfill a contractual agreement as well as the architecture elements in the IT landscape that consume the contract deliverables.

Users can define contract deliverables that specify the architecture elements and resources needed to fulfill the contract agreement, individual contract items that different organizations are responsible for, and a payment schedule relevant to each contract. Commercial contracts as well as free and open source contracts can be captured and managed. The contract deliverables for commercial contracts are typically captured directly for the contract or contract item in the *Contract Deliverables Page View* (`CNTR_Deliverables`) of the relevant contract or contract item. n the case of free and open-source contracts, contract deliverables are typically defined from the perspective of a vendor, component, local component or standard platform. To this end, the contract deliverables would be captured in the *Contract Deliverables Page View* (`CNTR_ArtifactDeliverables`) In this context, the referenced contract would be considered the free or open source license type being used for the delivery or use of the object.

Multiple non-hierarchical stereotypes can be configured for the object classes `Contract`, `ContractItem`, and `ContractDeliverable`. Contract stereotypes could be, for example, license contracts, maintenance contracts, Help Desk management contracts, etc. Custom object views, custom editors, custom wizards, custom selectors, configured reports, and custom class settings may be configured for these object classes/object class stereotypes. If free and open source contracts will be captured, for example, it is recommended that a solution designer configure custom selectors that ensure that the contract stereotypes representing free and open software are associated with the relevant contract deliverable.

The **Contract Management** functionality (`ContractManagement`) must be made available to the relevant user profiles. The mandate capability can be implemented for any configured contract stereotype in order to control user access to contracts in the `ContractManagement` functionality. Workflows may be configured for the **Contract Management** functionality (`ContractManagement`), as needed.

The following information is available:

- Creating Object Class Stereotypes for Contracts
- Configuring Release Status Definitions for the Contract Management Object Classes
- Configuring the Protected Enumerations Implemented in Contract Management
- Configuring Monitors or Workflows for Maintaining Contracts
- Configuring Class Settings for Contracts/Contract Stereotypes and Contract Items/Contract Item Stereotypes
- Configuring Wizards for the Contract Management Capability

Creating Object Class Stereotypes for Contracts

You may create an unlimited number of contract stereotypes for the object classes `Contract` and `ContractItem`. Please note that these are not hierarchical stereotypes.

Example of the XML definition in the **Stereotypes** attribute for the object class `Contract`:

```
<ClassStereotypes>
   <Stereotype Name="LicenseContract" Caption="License Contract"
   CaptionPlural="License Contracts" Comments="Contracts for
   software licences" HasMandates="true" />
   <Stereotype Name="MaintenanceLicense" Caption="Maintenance
   License" CaptionPlural="MaintenanceLicense" Comments="Contracts
   for maintenance agreements" HasMandates="false" />
</ClassStereotypes>
```

To create object class stereotypes for the object classes `Contract` and `ContractItem`:

1) Go to the **Meta-Model** tab and navigate to the protected class node for the object class `Contract` or `ContractItem` in the **Class Model** explorer.
2) Click the object class in the explorer to open the attribute window.
3) Click the **Browse** button available for the **Stereotypes** attribute to open the XML editor. Define the following XML attributes:

   An error message will be displayed upon closing the text editor if syntax errors are written in the XML definition of the **Stereotypes** attribute.

   - `Stereotype Name`: Enter the technical name for each object class stereotype. You may define a name of up to 64 characters. This name should not be changed once the object class stereotypes are implemented.

     Changing the technical name after objects have been created based on the object class stereotype requires such objects to be updated by means of an ADIF scheme. For more information, see the reference manual *Alfabet Data Integration Framework*.

   - `Caption`: Enter the caption for the object class stereotype. This is the name that users will see in the Alfabet user interface.

     Please note that the following:

     - The caption of the object class stereotype should be different than the caption of its object class. This is particularly important for the implementation of the search functionality for the object class stereotype. For more information, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.
     - The values defined for the XML attribute `Caption` can be translated whereas the values defined for the XML attribute `Stereotype Name` cannot be translated. However, in the table of the **Simple Search** functionality as well as other search selectors, the values defined for the XML attribute `Stereotype Name` of the object class stereotype are displayed in the **Stereotypes** column. If the values defined for the XML attribute `Caption` should be displayed, a custom selector should be defined. For more

information, see the section Configuring a Custom Selector for Search Functionalities.

- If you want to enter a caption (`Caption`) or plural form for the caption (`CaptionPlural`) that contains special characters (for example, a greater then (>) or lesser then (<) symbol), you must replace the special characters with respective XML compliant code, for example:
  - > for >
  - < for <
  - " for "
  - [for [
  -] for]
- The caption will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.

- `Description`: Provide a brief description about the object class stereotype. The description will be displayed in the **Stereotype Selector** that opens when the user must select which stereotype a new object is to be based on. The description should help the user to understand which stereotype the new object should be based on. The description will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.
- `CaptionPlural`: Enter a plural form for the caption. This is the name in plural that users will see in the Alfabet user interface.
- `IDPrefix`: Enter no more than 5 characters for the ID prefix used to generate the unique identification number for new objects based on the object stereotype. If the XML attribute `IDPrefix` is not specified, the ID prefix of the object class that the object class stereotype is based on will be used to generate the unique ID. Please note that the integer generated for the unique ID is based on the number of instances created for the object class that the object class stereotype is based on and includes all instances of all object class stereotypes created for the object class.

  The prefix ID of existing objects based on an object class stereotype can be updated via the ADIF import scheme `SetStereotypeID` available in the `Alfabet Standard Jobs` folder. For more information, see the section *Predefined ADIF Schemes* in the reference manual *Configuring Alfabet with Alfabet Expand*.

- `Comments`: Enter text in order to provide information about the object class stereotype. This information is only for the purposes of configuration. It is not displayed in the Alfabet user interface.
- `HasMandates`: Enter "`true`" if the mandate capability should be available for the object class stereotype. Enter "`false`" if the mandate capability should not be available for the object class stereotype. If the XML attribute `HasMandates` is defined as "`false`" for an object class stereotype, objects based on the object class stereotype will be visible to users with any mandate implemented in your enterprise. For information about additional configuration requirements necessary to implement the mandate capability, see the section *Implementing the Mandate Capability for a Federated Architecture* in the chapter Configuring Access Permissions for Alfabet.

- `EnableStatutoryLanguage`: Set to `True` if the statutory language capability shall be available for the object class stereotype. The statutory language capability is available in order to address regulatory requirements that mandate that objects are captured in a specific language. When a user creates an object based on a relevant object class/object class stereotype, a statutory language selector will open in which the language can be selected. In the object editor that opens, the user can capture all attributes that support automated translation in the statutory language. If the XML attribute `EnableStatutoryLanguage` is not set, the object class stereotype will inherit the definition of the **Enable Statutory Language** attribute specified for the object class. Specifying a Statutory Language for the Enterprise

4) Click the **Save** button to save your changes.

Configuring Release Status Definitions for the Contract Management Object Classes

Release status definitions must be created for the following object classes:

- `Contract`
- `ContractItem`
- `ContractDeliverable`
- `ContractPayment`

The release status definition includes the complete set of release statuses implemented in the `Contract-Management` functionality. Each object class can have a different release status definition, as needed. If contract stereotypes are implemented, each contract stereotype may have a different release status. The technical name "`Contract`:<StereotypeName>" must be specified in the attribute `ClassNames`.

The release statuses must be defined in the XML object ***ReleaseStatusDefs*** available in the **XML Objects** folder in the **Presentation** tab. For detailed information about how to define the XML object ***ReleaseStatusDefs***, see the section Configuring Release Status Definitions for Object Classes in the chapter Configuring the Class Model.

Configuring the Protected Enumerations Implemented in Contract Management

The following protected enumerations are implemented in the **Contract Management** functionality (`ContractManagement`) and can be edited, as needed:

- `ContractDeliverableUnit`: This enumeration allows you to specify the values that can selected by users for the **Unit** field in the **Contract Deliverable** editor. Typical units for contract deliverables could be, for example, Licenses, Floating Licenses, Support Hours, Man/Days, etc. For more information about the **Unit** field for a contract deliverable, see the section *Contract Deliverables Page View* in the Alfabet online help.
- `ContractDependencyType`: This enumeration allows you to specify the values that can selected by users for the **Dependency Type** field in the **Contract Dependency** editor in the context of the `ContractManagement` functionality. Typical dependency types for contracts could be, for example,

Contract Successors or Contract Amendments. For more information, see the *Contract Dependencies Page View* in the Alfabet online help.

- `ContractPaymentType`: This enumeration allows you to specify the values that can selected by users for the **Type** field in the **Contract Payment** editor in the context of the `ContractManagement` functionality. Typical payment types for contracts could be, for example, Income or Expenditure. For more information, see the *Contract Payment Schedule Page View* in the Alfabet online help.

The protected enumerations are modified in the **Enums** folder in the **Meta-Model** tab. For detailed information about how to edit protected enumerations, see the section Defining Protected and Custom Enumerations in the chapter Configuring the Class Model.

Configuring Monitors or Workflows for Maintaining Contracts

You may consider implementing monitors in order to alert users responsible for contracts with impending review dates or configuring workflows that initiate contract reviews in order to ensure that contracts are maintained, renewed, or cancelled in a timely manner.

Possible relevant use cases for the configuration of monitors or workflows in the context of the **Contract Management** functionality (`ContractManagement`) include monitoring or managing any of the following:

- The object class property `EndDate` of the object class `Contract` or a configured contract stereotype. The object class property `EndDate` specifies the date that a contract is concluded.
- The object class properties `StartState` or `End Date` of the object class `ContractItem`. These object class properties are optional for the object class `ContractItem`.
- The object class property `ReviewDate` of the object class `Contract` that has been associated with a via the **Contract Schedule** view. The object class property `ReviewDate` specifies the date that the contract should be reviewed in the context of the master contract that it is assigned to.
- The object class property `DeliveryDate` of the object class `ContractDeliverable`. The object class property `DeliveryDate` specifies the date that the contract deliverable is due.
- The object class property `ExecutionDate` of the object class `ContractPayment`. The object class property `ExecutionDate` specifies the date that a contract payment should be made.

Monitors are configured in the **Monitors** functionality accessible in Alfabet via the `Admin` user profile, see the section *Configuring Monitors to Track Objects in Alfabet* in the reference manual *User and Solution Administration* as well as the section Configuring Monitors in this manual.

For more information about the configuration of workflows, see the chapter Configuring Workflows.

Configuring Class Settings for Contracts/Contract Stereotypes and Contract Items/Contract Item Stereotypes

If you want to define class settings for a object class stereotypes defined for the object classes `Contract` and `ContractItem`, you must manually add the stereotypes configured by your enterprise to the **Class Settings** folder in the **Presentation** tab. To add a project stereotype to the **Class Settings** folder, navigate to and right-click the relevant class setting for the object classes `Contract` and `ContractItem` and select either:

- **New Class Settings as Copy** to create a class setting as copy for the selected class, or
- **New Class Settings for Stereotype as Copy** to create a class setting for a stereotype of the selected class. In the dialog that opens, select the contract/contract item stereotype and click **OK**. You must use this function if there are no class settings already defined for the new stereotype.

The contract/contract item stereotype will be added with the naming convention `Contract`:<ContractStereotype> or `ContractItem`:<ContractItemStereotype>. The class settings can be edited, as needed. The class settings for the contract/contract item stereotypes can be configured in the same manner as class settings for other object classes.

For more information about configuring class settings, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.

Configuring Wizards for the Contract Management Capability

Wizards can be implemented in the context of the **Contract Management** functionality (`ContractManagement`). Similar to class settings for other object settings, each class setting defined for a contract stereotype may have a different wizard assigned to.

Please note, however, that when a wizard is defined, it is first defined independent of the contract stereotype that it is relevant for. Thus, when defining the wizard, you must select the value `Contract` for the **Class Name** attribute. The wizard is then assigned to a class setting for the relevant contract stereotype. A wizard may be reused for multiple contract stereotypes.

For more information about configuring wizards, see the chapter Configuring Wizards.

Configuring the Resource Management Capability

Alfabet provides a Resource Management capability to support the planning of the availability of requested resources. The Resource Management capability is highly configurable and typically relies on the specification of object class stereotypes for the object class **Resource Request** (`ITResource`).

A resource request is any entity that is needed as a source of support for another entity in the IT enterprise. A resource request can be provided by an application, component, device, deployment, service product, standard platform, and organization (or any of their object class stereotypes). A resource request is requested by a resource requester which is typically a project, application, component, device, deployment, service product, standard platform, and organization. For example, the project stereotype Project Step may request the resource Development Organization which specifies that an organization based on the object class stereotype Business Unit may be requested. The project stereotype Project Step may also request the resource Project Development which specifies that a service product based on the object class stereotype System Hosting may be requested. Resource requests can then be planned in detail in the context of resource planning for projects, for example.

A resource request is associated with a cost type and includes the specification of the amount needed for the resource as well as the total cost of the use of the requested resource. An object requesting a resource may request multiple resources and an object that is requested as a resource may be requested by multiple objects.

The Resource Management capability is available in the workspace **Resource Management** (`RSRC_ResourceManagement`), which per default contains the *Required Resources Page View* (`RSRC_RequestorResources`), *Requested Resources Time Schedule Page View* (`RSRC_RequestorGantt`), and *Provided Resources Time Schedule Page View* (`RSRC_ProviderGantt`).

The following information is available:

- Configuring Object Class Stereotypes for Resource Management
- Specifying the Object Classes That Can Request a Resource in the XML Object ResourceManager
- Configuring Custom Selectors for Resource Management
- Configuring Custom Object Views and User Profiles for Resource Management

Configuring Object Class Stereotypes for Resource Management

Before you begin to configure the object class stereotypes needed for the Release Management capability, you should carefully consider which object class stereotypes are required for the object class `ITResource` as well as the object classes/object class stereotypes that may provide and request the resources. For example, typical object class stereotypes for the object class `ITResource` might be Organizational Resource, Service Product Resource, and Person Resource.

All object class stereotypes for the object class `ITResource` that you will configure in the XML object ***ResourceManager*** must be configured in the **Stereotypes** attribute for the object class `ITResource`. This is described below. Once the enterprise begins to work with the Resource Management capability, you should not remove existing object class stereotypes. If you do so, any existing objects must be updated to the new configuration by means of a customized ADIF scheme. For more information about configuring an ADIF scheme, see the reference manual *Alfabet Data Integration Framework*.

To create object class stereotypes for the object class `ITResource`:

1) Go to the **Meta-Model** tab and navigate to the protected class node  for the object class `ITResource` in the **Class Model** explorer.
2) Click the relevant object class in the explorer to open the attribute window.
3) Click the **Browse**  button available for the **Stereotypes** attribute to open the XML editor. Define the following XML attributes:

An error message will be displayed upon closing the text editor if syntax errors are written in the XML definition of the **Stereotypes** attribute.

- `Stereotype Name`: Enter the technical name for each object class stereotype. You may define a name of up to 64 characters. This name should not be changed once the object class stereotypes are implemented.

  Changing the technical name after objects have been created based on the object class stereotype requires such objects to be updated by means of an ADIF scheme. For more information, see the reference manual *Alfabet Data Integration Framework*.

- `Caption`: Enter the caption for the object class stereotype. This is the name that users will see in the Alfabet user interface.

  Please note that the following:

  - The caption of the object class stereotype should be different than the caption of its object class. This is particularly important for the implementation of the search functionality for the object class stereotype. For more information, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.
  - The values defined for the XML attribute `Caption` can be translated whereas the values defined for the XML attribute `Stereotype Name` cannot be translated. However, in the table of the **Simple Search** functionality as well as other search selectors, the values defined for the XML attribute `Stereotype Name` of the object class stereotype are displayed in the **Stereotypes** column. If the values defined for the XML attribute `Caption` should be displayed, a custom selector should be defined. For more information, see the section Configuring a Custom Selector for Search Functionalities.
  - If you want to enter a caption (`Caption`) or plural form for the caption (`CaptionPlural`) that contains special characters (for example, a greater then (>) or lesser then (<) symbol), you must replace the special characters with respective XML compliant code, for example:
    - > for >
    - < for <
    - " for "

- [for [
-] for]

- The caption will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.

- `Description`: Provide a brief description about the object class stereotype. The description will be displayed in the **Stereotype Selector** that opens when the user must select which stereotype a new object is to be based on. The description should help the user to understand which stereotype the new object should be based on. The description will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.
- `CaptionPlural`: Enter a plural form for the caption. This is the name in plural that users will see in the Alfabet user interface.
- `IDPrefix`: Enter no more than 5 characters for the ID prefix used to generate the unique identification number for new objects based on the object stereotype. If the XML attribute `IDPrefix` is not specified, the ID prefix of the object class that the object class stereotype is based on will be used to generate the unique ID. Please note that the integer generated for the unique ID is based on the number of instances created for the object class that the object class stereotype is based on and includes all instances of all object class stereotypes created for the object class.

  The prefix ID of existing objects based on an object class stereotype can be updated via the ADIF import scheme `SetStereotypeID` available in the `Alfabet Standard Jobs` folder. For more information, see the section *Predefined ADIF Schemes* in the reference manual *Configuring Alfabet with Alfabet Expand*.

- `Comments`: Enter text in order to provide information about the object class stereotype. This information is only for the purposes of configuration. It is not displayed in the Alfabet user interface.
- `HasMandates`: Enter "`true`" if the mandate capability should be available for the object class stereotype. Enter "`false`" if the mandate capability should not be available for the object class stereotype. If the XML attribute `HasMandates` is defined as "`false`" for an object class stereotype, objects based on the object class stereotype will be visible to users with any mandate implemented in your enterprise. For information about additional configuration requirements necessary to implement the mandate capability, see the section *Implementing the Mandate Capability for a Federated Architecture* in the chapter Configuring Access Permissions for Alfabet .
- `EnableStatutoryLanguage`: Set to `True` if the statutory language capability shall be available for the object class stereotype. The statutory language capability is available in order to address regulatory requirements that mandate that objects are captured in a specific language. When a user creates an object based on a relevant object class/object class stereotype, a statutory language selector will open in which the language can be selected. In the object editor that opens, the user can capture all attributes that support automated translation in the statutory language. If the XML attribute `EnableStatutoryLanguage` is not set, the object class stereotype will inherit the definition of the **Enable Statutory Language** attribute specified for the object class. Specifying a Statutory Language for the Enterprise

4) Click the **Save** button t to save your changes.

Specifying the Object Classes That Can Request a Resource in the XML Object ResourceManager

Once all object class stereotypes have been created for the object class `ITResource`, you can refine the definition of functionality for each resource request stereotype in the XML object ***ResourceManager***.

To edit the XML object ***ResourceManager***:

1) Go to the **Presentations** tab, expand the **XML Objects** folder, and expand the **Solution Managers** folder.
2) Right-click the XML object ***ResourceManager*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see the section Working with XML Objects.
3) Edit the XML attributes, as needed. The table below displays the XML attributes that can be edited for the XML object ***ResourceManager***:

XML Elements (bold) and Attributes	Explanation
ResourceManager	
Stereotype	Define the following attributes for each object class that may request resources.
`ClassName`	Enter the technical name of the object class that may request resources. You may enter any of the following object classes: `Application`, `Component`, `Device`, `Deployment`, `ServiceProduct`, `StandardPlatform`, `Project`, and `OrgaUnit` **NOTE**: You must spell the technical name of the object class stereotype exactly as it is spelled in the **Stereotypes** attribute of the relevant object class.
`Name`	Enter the technical name of the object class stereotype configured for the object class in the `ClassName` attribute that may request resources. Leave this attribute empty if an object class stereotype is not relevant. **NOTE**: You must spell the technical name of the object class stereotype exactly as it is spelled in the **Stereotypes** attribute of the relevant object class.
Stereotype	Define the following attributes for each object class that may provide resources.
`ClassName`	Enter the technical name of the object class that may fulfill resource requests. You may enter any of the following object class: `Application`,

XML Elements (bold) and Attributes	Explanation
	`Component`, `Device`, `Deployment`, `ServiceProduct`, `StandardPlatform`, `Project`, and `OrgaUnit` **NOTE**: You must spell the technical name of the object class stereotype exactly as it is spelled in the **Stereotypes** attribute of the relevant object class.
`Name`	Enter the technical name of the object class stereotype configured for the object class in the XML attribute `ClassName` that may provide resources. Leave this attribute empty if an object class stereotype is not relevant.
`ResourceStereotype`	Enter the technical name of the stereotype that is requested. You must spell the technical name of the object class stereotype exactly as it is spelled in the **Stereotypes** attribute of the object class `ITResource`.
`Selector`	Enter the name of the custom selector to implement to find the objects based on the object class/object class stereotype that may provide resources. For more information about configuring custom selectors, see the section Configuring a Custom Selector for Search Functionalities.
`CostType`	Enter the name of the cost type assigned to this resource request. Cost types are configured in the **Reference Data** functionality. For more information, see the chapter *Configuring Cost Types and Income Types for Cost Management Capabilities* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.
`ButtonCaption`	Enter the text that should appear in the **New** menu in the *Required Resources Page View* (`RSRC_RequestorResources`) when a user creates the resource. This text can help the user to understand the type of resource request that they are creating. For example, **Create New Application Management Resource** could be displayed in the **New** menu to find the organizations relevant for providing application management. The custom selector defined in the `Selector` attribute that would then open would ideally only display those organizations relevant for providing application management.

4) In the toolbar, click the **Save** button to save the XML definition.

Configuring Custom Selectors for Resource Management

Custom selectors may be configured as needed for any of the object classes associated with the Resource Management capability. The custom selector can be specified in the XML object ***ResourceManager*** to specify which custom selector to implement to find the objects referenced by the relevant object class stereotype based on the class `ITResource`. For more information about configuring custom selectors, see the ssection Configuring a Custom Selector for Search Functionalities.

Configuring Custom Object Views and User Profiles for Resource Management

In order to ensure that the relevant users can access the functionalities necessary to work with the Resource Management capability, you should consider which user profiles have access to the page views that allow users to create resource requests and analyze resource use. You should take into careful consideration which user profiles will need to access which object class stereotypes. This will require the configuration of custom object views as well as custom class settings. For more information, see the chapters Configuring Object Views and Configuring User Profiles for the User Community.

It is recommended that class settings are specified for each object class stereotype and that a different color is defined for each of these object class stereotypes. This will help support users working with the Gantt charts available in the context of resource management as well as resource planning in the context of projects.

The workspace **Resource Management** (`RSRC_ResourceManagement`) is available in the standard object views for the object classes `Application`, `Component`, `Device`, `Deployment`, `ServiceProduct`, `StandardPlatform`, `Project`, and `OrgaUnit`. The **Resource Management** workspace contains the *Required Resources Page View*, *Requested Resources Time Schedule Page View* (`RSRC_RequestorGantt`), and *Provided Resources Time Schedule Page View* (`RSRC_ProviderGantt`).

To make this capability available to the user community, the entire workspace or individual views must be added to a custom object view configured for the object classes `Application`, `Component`, `Device`, `Deployment`, `ServiceProduct`, `StandardPlatform`, and `OrgaUnit` and the custom object views must be available in the relevant user profiles. For more information about assigning a workspace to an object view, see the section Configuring Workspaces for an Object Profile in the chapter Configuring Object Views.

Furthermore, resource requests can be created in the context of project planning in the *Project, Skill Request and Resource Request Time Schedule Page View* (`PRJ_TimeSchedule`), *Project, Skill Request and Resource Request Time Schedule (Gantt) Page View* (`PRJ_TimeScheduleGantt`), and *Project Resource Planning Page View* (`PRJ_ResourcePlanning`). You must ensure that these views are available in the custom object of the relevant object class stereotype created for the object class `Project`.

Configuring the Service Product Portfolio Management Capability

Alfabet provides a capability that allows you to define and track the service products that are offered by an organization. This capability is highly configurable and typically relies on the specification of object class stereotypes for the object classes `ServiceProduct`, `ServiceItem`, and `SLA`.

For an overview of the methodology of Service Product Portfolio Management, see the section *Service Product Portfolio Management* in the reference manual *Portfolio Management Complete*.

In order to work with the Service Product Portfolio Management capability, the following configuration may be required:

- Typically, object class stereotypes are to be configured for the object classes `ServiceProduct`, `ServiceItem`, and `SLA` (Service Level Agreement (SLA)), these must be configured in the configuration tool Alfabet Expand. In this case, the solution designer should consider which service product stereotypes are required, what kind of service product items are required to provide each service product stereotype, whether the service product hierarchy should be recursive for a service product stereotype, and which service level agreement stereotype should be assigned to which service product stereotype. Object class stereotypes may also be necessary for the object classes `Contract` and `Market Product`. You should consider whether a service product stereotype should only be associated with a specific contract stereotype and whether a specific service product stereotype should only be associated with specific market product stereotype.
- The relationship between the object class stereotypes configured for SLAs and the object class stereotypes for service products must be configured in the XML object ***ServiceProductManager***.
- Custom properties may be configured as needed for the object classes `Service Product Group`, `ServiceProduct`, `ServiceItem`, and `SLA`. Please note that the object class `SLA` has only basic standard properties and it is typically necessary to configure custom attributes to capture relevant details about the SLA.
- Service products have a lifecycle, object state, and release status definition.
- Custom editors and custom selectors must be configured for each object class stereotype in order to capture data for the custom attributes.
- Please note the following additional configuration requirements:
  - If IT resources are to be specified for service products, then object class stereotypes may be defined for the object class Resource. You must also specify that service products may request service products in the XML object ***ResourceManager***. Details about the necessary configuration are described in the section Configuring the Resource Management Capability in the reference manual *Configuring Alfabet with Alfabet Expand*.
  - If contracts are to be specified for service products, then object class stereotypes may be defined for the object class Contract. Details about the necessary configuration are described in the section Configuring the Contract Management Capability in the reference manual *Configuring Alfabet with Alfabet Expand*.

The following information is available:

- Configuring Object Class Stereotypes for Service Product Portfolio Management
- Specifying the Relationships for the Service Product, Service Product Item, and SLA Stereotypes in the XML Object ServiceProductManager

- Configuring Lifecycle, Object State, and Release Status Definitions
- Configuring Custom Properties and Editors for Service Product Portfolio Management
- Configuring Custom Selectors for Service Product Portfolio Management
- Configuring Custom Object Views and User Profiles for Service Product Portfolio Management
- Configuring Mandates for Service Product Portfolio Management

Configuring Object Class Stereotypes for Service Product Portfolio Management

Before you begin to configure the object class stereotypes needed for the Service Product Portfolio Management capability, you should carefully consider which object class stereotypes are required for the object classes `ServiceProduct`, `ServiceItem`, and `SLA`. as well as the object classes/object class stereotypes that may provide the service product items. You should consider the following questions:

- Which service product stereotypes are required. Typical service product stereotypes could be, for example:
  - Application Business Service
  - Application Maintenance
  - Application Development
  - User Support Service
  - User License Service
  - System Hosting
  - Information Business Service
  - Terminal Service
- What kind of service level agreements (SLA) are required for each service product stereotype? In Alfabet, users may only specify one SLA per service product. Each service product stereotype is configured to have one permissible SLA stereotype that the SLA is based on. An SLA stereotype is typically configured for a specific service product stereotype in order to capture details including the units of measurements relevant to the service product stereotype. Each SLA stereotype typically requires the configuration of custom attributes and a custom editor.

  For example, an SLA stereotype User Support Service SLA for the service product stereotype User Support Service could have the custom attributes Service Desk Support Hours, 2nd Level Support Hours, 3rd Level Support Hours, Target Resolution Time, Support Availability. The Application Development SLA for the service product stereotype Application Development could have the custom attributes, Target Resolution Time, Defect Backlog Size Limit, Maximum Number of Test Failures.

- What kind of service product items are required to provide each service product stereotype? In Alfabet, multiple service product items of various object class stereotypes may provide a service product. Users associate each service product item with an object from an object class/object class stereotype that is permissible for the service product item. The base object classes include `Application`, `Component`, `Device`, `Deployment`, `StandardPlatform`, `OrgaUnit`, and

`ServiceProduct`. The association of object classes with the service product item is configured in the XML object ***ServiceProductManager***.

For example, the service product stereotype User Support Service could have the service product stereotypes. Application User Support Service which would be associated with a specified application or application stereotype, Deployment User Support Service which would be associated with a specified deployment or deployment stereotype, and Device User Support Service, which would be associated with a specified device or device stereotype.

- Should the service product hierarchy for a service product stereotype be recursive? In other words, should the service product hierarchy allow service product stereotypes to be subordinate to other service product stereotypes? If so, then the subordinate service product stereotypes must be specified as service product items in the XML object ***ServiceProductManager***. All service product stereotypes, whether on a parent level or child level, must be configured in the **Stereotypes** attribute for the object class ServiceProduct.

  For example, the service product stereotype Application Business Service is configured to be the parent service product stereotype in the service product hierarchy. The service product items that provide the service product Application Business Service are other service products. In other words, the service product items are associated with the relevant object class stereotypes for the object class `ServiceProduct` such as System Hosting, Application Maintenance, Application Development, User License Services, and User Support Services.

  Therefore, in addition to the service product stereotype Application Business Service, the service product stereotypes System Hosting, Application Maintenance, Application Development, User License Services, and User Support Services must be configured in the **Stereotypes** attribute for the object class ServiceProduct. Their relationship to the parent service product stereotype must be configured in the XML object ***ServiceProductManager***.

- Are object class stereotypes required for other object classes that are relevant to service products. For example, should the ownership of service product based on a service product stereotype be assigned to a specific organization stereotype? Should a service product based on a service product stereotype be defined for a specific contract stereotype? Should a service product based on a service product stereotype be associated with specific market product stereotype?

All object class stereotypes for the object classes `ServiceProduct`, `ServiceItem`, and `SLA` that you will configure in the XML object ***ServiceProductManager*** must be configured in the **Stereotypes** attribute for that object class. This is described below. **Once the enterprise begins to work with the Service Product Portfolio Management capability, you should not remove existing object class stereotypes. If you do so, any existing objects must be updated to the new configuration by means of a customized ADIF scheme.** For more information about configuring an ADIF scheme, see the reference manual *Alfabet Data Integration Framework*.

Example of the XML definition in the ***Stereotype*** element for the object class ServiceProduct:

```
<ClassStereotypes>

   <Stereotype Name="ApplicationBusinessService"
   Caption="Application Business Service" CaptionPlural="Application
   Business Services" Comments="" HasMandates="false" />

   <Stereotype Name="InformationBusinessService"
   Caption="Information/Data Business Service"
```



```
CaptionPlural="Information/Data Business Services" Comments=""
HasMandates="false" />

<Stereotype Name="SystemHosting" Caption="Server/System Hosting"
CaptionPlural="Server/System Hosting" Comments=""
HasMandates="false" />

<Stereotype Name="ApplicationMaintenance" Caption="Application
Maintenance" CaptionPlural="Application Maintenance" Comments=""
HasMandates="false" />

<Stereotype Name="ApplicationDevelopment" Caption="Application
Development" CaptionPlural="Application Development" Comments=""
HasMandates="false" />

<Stereotype Name="UserLicenseService" Caption="User/License
Management Service" CaptionPlural="User/License Management
Services" Comments="" HasMandates="false" />

<Stereotype Name="UserSupportService" Caption="User Support
Service" CaptionPlural="User Support Services" Comments=""
HasMandates="false" />

<Stereotype Name="DataService" Caption="Data Service"
CaptionPlural="Data Services" Comments="" HasMandates="false" />

<Stereotype Name="TerminalService" Caption="Terminal Service"
CaptionPlural="Terminal Services" Comments="" HasMandates="false"
/>

</ClassStereotypes>
```

To create object class stereotypes for the object classes `ServiceProduct`, `ServiceItem` and `SLA`:

1) Go to the **Meta-Model** tab and navigate to the protected class node  for the object classes `ServiceProduct`, `ServiceItem` and `SLA` in the **Class Model** explorer.
2) Click the relevant object class in the explorer to open the attribute window.
3) lick the **Browse**  button available for the **Stereotypes** attribute to open the XML editor. Define the following XML attributes:

An error message will be displayed upon closing the text editor if syntax errors are written in the XML definition of the **Stereotypes** attribute.

- `Stereotype Name`: Enter the technical name for each object class stereotype. You may define a name of up to 64 characters. This name should not be changed once the object class stereotypes are implemented.

  Changing the technical name after objects have been created based on the object class stereotype requires such objects to be updated by means of an ADIF scheme. For more information, see the reference manual *Alfabet Data Integration Framework*.

- `Caption`: Enter the caption for the object class stereotype. This is the name that users will see in the Alfabet user interface.

  Please note that the following:

- The caption of the object class stereotype should be different than the caption of its object class. This is particularly important for the implementation of the search functionality for the object class stereotype. For more information, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.
- The values defined for the XML attribute `Caption` can be translated whereas the values defined for the XML attribute `Stereotype Name` cannot be translated. However, in the table of the **Simple Search** functionality as well as other search selectors, the values defined for the XML attribute `Stereotype Name` of the object class stereotype are displayed in the **Stereotypes** column. If the values defined for the XML attribute `Caption` should be displayed, a custom selector should be defined. For more information, see the section Configuring a Custom Selector for Search Functionalities.
- If you want to enter a caption (`Caption`) or plural form for the caption (`CaptionPlural`) that contains special characters (for example, a greater then (>) or lesser then (<) symbol), you must replace the special characters with respective XML compliant code, for example:
  - > for >
  - < for <
  - " for "
  - [for [
  -] for]
- The caption will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.

- `Description`: Provide a brief description about the object class stereotype. The description will be displayed in the **Stereotype Selector** that opens when the user must select which stereotype a new object is to be based on. The description should help the user to understand which stereotype the new object should be based on. The description will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.
- `CaptionPlural`: Enter a plural form for the caption. This is the name in plural that users will see in the Alfabet user interface.
- `IDPrefix`: Enter no more than 5 characters for the ID prefix used to generate the unique identification number for new objects based on the object stereotype. If the XML attribute `IDPrefix` is not specified, the ID prefix of the object class that the object class stereotype is based on will be used to generate the unique ID. Please note that the integer generated for the unique ID is based on the number of instances created for the object class that the object class stereotype is based on and includes all instances of all object class stereotypes created for the object class.

The prefix ID of existing objects based on an object class stereotype can be updated via the ADIF import scheme `SetStereotypeID` available in the `Alfabet`

`Standard Jobs` folder. For more information, see the section *Predefined ADIF Schemes* in the reference manual *Configuring Alfabet with Alfabet Expand*.

- `Comments`: Enter text in order to provide information about the object class stereotype. This information is only for the purposes of configuration. It is not displayed in the Alfabet user interface.
- `HasMandates`: Enter "`true`" if the mandate capability should be available for the object class stereotype. Enter "`false`" if the mandate capability should not be available for the object class stereotype. If the XML attribute `HasMandates` is defined as "`false`" for an object class stereotype, objects based on the object class stereotype will be visible to users with any mandate implemented in your enterprise. For information about additional configuration requirements necessary to implement the mandate capability, see the section *Implementing the Mandate Capability for a Federated Architecture* in the chapter Configuring Access Permissions for Alfabet .
- `EnableStatutoryLanguage`: Set to `True` if the statutory language capability shall be available for the object class stereotype. The statutory language capability is available in order to address regulatory requirements that mandate that objects are captured in a specific language. When a user creates an object based on a relevant object class/object class stereotype, a statutory language selector will open in which the language can be selected. In the object editor that opens, the user can capture all attributes that support automated translation in the statutory language. If the XML attribute `EnableStatutoryLanguage` is not set, the object class stereotype will inherit the definition of the **Enable Statutory Language** attribute specified for the object class. Specifying a Statutory Language for the Enterprise

4) Click the **Save** button to save your changes.

Specifying the Relationships for the Service Product, Service Product Item, and SLA Stereotypes in the XML Object ServiceProductManager

Once all object class stereotypes have been created for the object classes `ServiceProduct`, `ServiceItem` and `SLA`, you can refine the definition of functionality for each service product stereotype in the XML object ***ServiceProductManager***.

To edit the XML object ***ServiceProductManager***:

1) Go to the **Presentations** tab, expand the **XML Objects** folder, and expand the **SolutionManagers** folder.
2) Right-click the XML object ***ServiceProductManager*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see the section Working with XML Objects.
3) Edit the XML attributes, as needed. The table below displays the XML attributes that can be edited for the XML object ***ServiceProductManager***:

XML Elements (bold) and Attributes	Explanation
ServiceProductManager	
Stereotype	Define the following attributes for each service product stereotype.
`Name`	Enter the technical name of the object class stereotype defined for the object class `ServiceProduct`. This is the name defined in the XML attribute `Name` in the **Stereotypes** attribute for the object class `ServiceProduct`. You must spell the technical name of the service product stereotype exactly as it is spelled in the **Stereotypes** attribute of the object class `ServiceProduct`.
`SLA_Stereotype`	Enter the technical name of the object class stereotype defined for the object class `SLA` that is available for the service product stereotype. Only one SLA stereotype may be specified per service product stereotype. Enter the name defined in the `Name` parameter in the **Stereotypes** attribute for the object class `SLA`. You must spell the technical name of the SLA stereotype exactly as it is spelled in the **Stereotypes** attribute of the object class `SLA`.
Stereotype	Define the following attributes for each service product item stereotype.
`ClassName`	The object class or object class stereotype that is referenced by the service product item stereotype relevant for the service product. You must spell the technical name of the object class stereotype exactly as it is spelled in the **Stereotypes** attribute of the relevant object class.
`Name`	Allows you to specify a recursive definition in the service product hierarchy. Enter the name of the child service product stereotype that is referenced by the parent service product stereotype identified in the ascendant ***Stereotype*** `Name` element.
`Selector`	The selector to implement to find the objects referenced by the service product item stereotype. For more information about configuring custom selectors, see the section Configuring a Custom Selector for Search Functionalities.

XML Elements (bold) and Attributes	Explanation
ServiceItemStereotype	The technical name of the object class stereotype of the service product item that will be used when creating a new service item object. The new service product item object will be based on the specified object class stereotype. The caption configured for the object class stereotype will be displayed in the menu for the creation of a service product item object. If this attribute is not defined, the object class/object class stereotype specified in the ClassName attribute will automatically be referenced. You must spell the technical name of the stereotype exactly as it is spelled in the **Stereotypes** attribute of the object class ServiceItem.
AllowDuplications	Enter "true" if the same object may be used to create multiple independent service product items assigned to the same service product. For example, if one application should be used to provide three service products Product Environment, Text Environment, and Development Environment. Enter "false" if the same object may not be used to create multiple independent service product items assigned to the same service product.

4) In the toolbar, click the **Save** button to save the XML definition.

Configuring Lifecycle, Object State, and Release Status Definitions

The object class ServiceProduct supports the configuration of lifecycles, object states, and release statuses. These must be configured in the relevant XML object as described in the following sections in the chapter Configuring the Class Model:

- Configuring Release Status Definitions for Object Classes
- Configuring Object State Definitions for Object Classes
- Configuring Lifecycle Definitions for Object Classes

Configuring Custom Properties and Editors for Service Product Portfolio Management

Custom properties may be configured as needed for the object classes ServiceProduct, ServiceItem, and SLA. Please note that the object class SLA in particular typically requires the configuration of custom properties in order to capture details regarding the units of measurements relevant to the service product stereotype.

All custom properties needed for the configured object class stereotypes should be configured for the base object class. In other words, all custom properties needed by all SLA stereotypes should be defined for the object class SLA. This is described in detail in the chapter Configuring Custom Properties for Protected or Public Object Classes.

Once the custom properties have been defined, you can configure custom editors in order to capture the custom properties relevant for the stereotype. This is described in detail in the chapter Configuring Custom Editors.

Configuring Custom Selectors for Service Product Portfolio Management

Custom selectors may be configured as needed for any of the object classes associated with the Service Product Portfolio Management capability. A selector definition is available in the XML object ***ServiceProductManager*** to specify which custom selector to implement to find the objects referenced by a service product item stereotype. For more information about configuring custom selectors, see the section Configuring a Custom Selector for Search Functionalities.

Configuring Custom Object Views and User Profiles for Service Product Portfolio Management

In order to ensure that the relevant users can access the functionalities necessary to work with the Service Product Portfolio Management capability, you should consider which user profiles have access to the following business functions (explorers), object views, and page views.

This capability typically includes a high level of configuration, in particular because object class stereotypes can be configured for the object classes `ServiceProduct`, `ServiceItem`, `SLA`, `OrgaUnit`, as well as the object classes that can be configured to provide service product items. Therefore, you should take into careful consideration which user profiles will need to access which object class stereotypes. This will require the configuration of custom object views as well as custom class settings. For more information, see the chapters Configuring Object Views and Configuring User Profiles for the User Community.

The following business functions are relevant for the Service Product Portfolio Management capability:

- **Service Product Groups** explorer (`SRVPRDG_Explorer`): Provides access to the object classes `ServiceProductGroup`, `ServiceProduct`, `ServiceItem`, and `SLA`. The following object views and page views are relevant for the **Service Product Groups** explorer:
  - `SRVPPRDG_ObjectView`/custom object views should include any necessary **Basic Data** page views and the following views:
    - *Sub-Groups Page View* (`SRVPRDG_SubGroups`)
    - *Service Products Page View* (`SRVPRDG_Products`)
    - *Service Product Portfolio Page View* (`SRVPRDG_ServiceProductPortfolio`)
  - `SRVPRD_ObjectView`/custom object views should include any necessary **Basic Data** page views and the following views:
    - *Service Product Items Page View* (`SRVPRD_Items`)
    - *Subordinate Value Stream Groups Page View* (`SRVPRD_Lifecycle`)

- *Service Product Overlap Report Page View* (`SRVPRD_OverlapReport`)
- *Service Product Usage Gantt Page View* (`SRVPRD_UsageGantt`)
- *Existing Contracts Page View* (`SRVPRD_Contracts`)

- `SRVITM_ObjectView`/custom object views (standard object view contains only the *Attachments Page View*)
- `SLA_ObjectView`/custom object views (standard object view contains only the *Attachments Page View*)

- **Enterprise Service Products Catalog** (`SRVPRD_Explorer`): Provides access to the object class `OrgaUnit`. The `ORG_ObjectView`/custom object views should include the following views:
  - *Service Product Portfolio Page View* (`ORG_ServiceProductPortfolio`)
  - *Owned Service Products Page View* (`ORG_ServiceProducts`)
- The relevant object views for the object class `MarketProduct` (`PROD_ObjectView`/custom object views) should include the *Service Products Page View* (`PROD_ServiceProducts`)
- The relevant object views for object classes/object class stereotypes referenced as providers of a service product item should include the *Existing Service Product Contracts Page View* (`SRVPRD_ArtifactContracts`).

Configuring Mandates for Service Product Portfolio Management

The mandate functionality is typically configured when the service product stereotypes are created. However, the mandate assignment can be edited, as needed. The mandate capability can be activated or deactivated for the entire service product capability as well as explicitly activated or deactivated for specific service product stereotypes.

The mandate configuration may differ for the service product stereotypes defined. Per default, the XMl attribute `HasMandates` is defined as "`false`" for all service product stereotypes. This means that the default setting is that service products based on a service product stereotype will be visible to users with any mandate implemented in your enterprise. Therefore, if visibility should be controlled for the service products based on a service product stereotype, you must explicitly define the mandate capability for each service product stereotype.

For more information about the mandates capability as well as how to create mandates for your enterprise, see the section *Implementing the Mandate Capability for a Federated Architecture* in the chapter Configuring Access Permissions for Alfabet.

To configure the mandate capability for your service product hierarchy:

1) Go to the **Meta-Model** tab and navigate to the protected class node  for the object class `ServiceProduct` in the **Class Model** explorer.
2) Click the object class `ServiceProduct` to display the attribute window.
3) Set the **Can Have Mandates** attribute to `True` if the mandate capability should be available for the service product capability as a whole. Select `False` if the mandate capability should not be available.

4) Next, you must explicitly define the mandates capability for each service product stereotype. Click the **Browse** button available for the **Stereotypes** attribute.

5) To implement the mandate capability for a service product stereotype, the XML attribute `HasMandates` must be defined as "`true`". If you do not want to implement the mandate capability for a service product stereotype, the XML attribute `HasMandates` must be defined as "`false`". If the XML attribute `HasMandates` is defined as "`false`" for a service product stereotype, service products based on the stereotype will be visible to permissible users with any mandate implemented in your enterprise.

6) Close the XML definition of the **Stereotypes** attribute by clicking **OK** in the editor.

7) In the toolbar, click the **Save** button to save your changes.

Configuring the ICT Object Hierarchy

You can create multiple ICT object stereotypes, as needed. The ICT objects based on an ICT object stereotype are instances of the ICT object stereotype.

You must first configure the ICT object stereotypes in the **Stereotypes** attribute of the object class `ICTObject`. Once the ICT object stereotypes have been created, you can specify which object classes, or their object class stereotypes may be owned by an ICT object stereotypes. Possible object classes and their configured stereotypes include `Application`, `Component`, `Device`, `LocalComponent`, `StandardPlatform`, `SystemBuildingBlock`, and `VendorProduct`. This is configured in the XML object ***ICTObjectManager***.

For a methodological overview of ICT objects, see the section *Application Architecture Definition* in the reference manual *Enterprise Architecture Management*.

Once defined, an ICT object stereotype is similar to a standard object class. Correspondingly, you can configure the mandate capability to control user access to each ICT object stereotype. Additionally, you can configure custom editors, wizards, workflows, and reports for each ICT object stereotype, as needed.

You can configure a class setting for each individual ICT object stereotype, if needed. If class settings are not configured for a stereotype, the class setting specified for the class `ICTObject` will be used. Please note the following logic of which editor/wizard opens when the **Create ICT Object** option is configured is as follows:

1) The editor/wizard defined in the default class setting (**Default** attribute set to `True`) specified for the relevant view scheme associated with the user's user profile.
2) The editor/wizard defined in the default class setting (**Default** attribute set to `True`) specified for the ICT object stereotype
3) The editor/wizard defined in the default class setting (**Default** attribute set to `True`) specified for the class **ICT Object**.

The following information is available:

- Creating Object Class Stereotypes for ICT Objects
- Configuring the Attributes and Functionalities Available for an ICT Object Stereotype

Creating Object Class Stereotypes for ICT Objects

To create ICT object stereotypes:

1) Go to the **Meta-Model** tab and navigate to the protected class node for the object class ICT object in the **Class Model** explorer.
2) Click the object class `OrgaUnit` in the explorer to open the attribute window.
3) Click the **Browse** button available for the **Stereotypes** attribute to open the XML editor. Define the following XML attributes:

An error message will be displayed upon closing the text editor if syntax errors are written in the XML definition of the **Stereotypes** attribute.

- `Stereotype Name`: Enter the technical name for each object class stereotype. You may define a name of up to 64 characters. This name should not be changed once the object class stereotypes are implemented.

  Changing the technical name after objects have been created based on the object class stereotype requires such objects to be updated by means of an ADIF scheme. For more information, see the reference manual *Alfabet Data Integration Framework*.

- `Caption`: Enter the caption for the object class stereotype. This is the name that users will see in the Alfabet user interface.

  Please note that the following:

  - The caption of the object class stereotype should be different than the caption of its object class. This is particularly important for the implementation of the search functionality for the object class stereotype. For more information, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.
  - The values defined for the XML attribute `Caption` can be translated whereas the values defined for the XML attribute `Stereotype Name` cannot be translated. However, in the table of the **Simple Search** functionality as well as other search selectors, the values defined for the XML attribute `Stereotype Name` of the object class stereotype are displayed in the **Stereotypes** column. If the values defined for the XML attribute `Caption` should be displayed, a custom selector should be defined. For more information, see the section Configuring a Custom Selector for Search Functionalities.
  - If you want to enter a caption `(Caption)` or plural form for the caption (`CaptionPlural`) that contains special characters (for example, a greater then (>) or lesser then (<) symbol), you must replace the special characters with respective XML compliant code, for example:
    - > for >
    - < for <
    - " for "
    - [for [
    -] for]
  - The caption will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.

- `Description`: Provide a brief description about the object class stereotype. The description will be displayed in the **Stereotype Selector** that opens when the user must select which

stereotype a new object is to be based on. The description should help the user to understand which stereotype the new object should be based on. The description will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.

- `CaptionPlural`: Enter a plural form for the caption. This is the name in plural that users will see in the Alfabet user interface.
- `IDPrefix`: Enter no more than 5 characters for the ID prefix used to generate the unique identification number for new objects based on the object stereotype. If the XML attribute `IDPrefix` is not specified, the ID prefix of the object class that the object class stereotype is based on will be used to generate the unique ID. Please note that the integer generated for the unique ID is based on the number of instances created for the object class that the object class stereotype is based on and includes all instances of all object class stereotypes created for the object class.

  The prefix ID of existing objects based on an object class stereotype can be updated via the ADIF import scheme `SetStereotypeID` available in the `Alfabet Standard Jobs` folder. For more information, see the section *Predefined ADIF Schemes* in the reference manual *Configuring Alfabet with Alfabet Expand*.

- `Comments`: Enter text in order to provide information about the object class stereotype. This information is only for the purposes of configuration. It is not displayed in the Alfabet user interface.
- `HasMandates`: Enter "`true`" if the mandate capability should be available for the object class stereotype. Enter "`false`" if the mandate capability should not be available for the object class stereotype. If the XML attribute `HasMandates` is defined as "`false`" for an object class stereotype, objects based on the object class stereotype will be visible to users with any mandate implemented in your enterprise. For information about additional configuration requirements necessary to implement the mandate capability, see the section *Implementing the Mandate Capability for a Federated Architecture* in the chapter Configuring Access Permissions for Alfabet .
- `EnableStatutoryLanguage`: Set to `True` if the statutory language capability shall be available for the object class stereotype. The statutory language capability is available in order to address regulatory requirements that mandate that objects are captured in a specific language. When a user creates an object based on a relevant object class/object class stereotype, a statutory language selector will open in which the language can be selected. In the object editor that opens, the user can capture all attributes that support automated translation in the statutory language. If the XML attribute `EnableStatutoryLanguage` is not set, the object class stereotype will inherit the definition of the **Enable Statutory Language** attribute specified for the object class. Specifying a Statutory Language for the Enterprise

4) Click the **Save** button to save your changes.

Configuring the Attributes and Functionalities Available for an ICT Object Stereotype

Once the ICT object stereotypes have been created for the object class `ICTObject`, you can refine the definition of functionality for each ICT object stereotype in the XML object ***ICTObjectManager***.

To edit the XML object ***ICTObjectManager***:

1) Go to the **Presentations** tab, expand the **XML Objects** folder, and expand the **SolutionManagers** folder.
2) Right-click the XML object ***ICTObjectManager*** and select **Edit XML**. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) Edit the XML attributes, as needed. The table below displays the XML attributes that can be edited for the XML object ***ICTObjectManager***:

XML Elements (bold) and Attributes	Explanation
ICT ICTObjectManager	
Stereotype	Each ICT object stereotype begins with the element ***Stereotype*** in the XML object. Define the following attributes for each ICT object stereotype.
`Name`	Enter the technical name of the ICT object stereotype. This is the name defined in the `Name` attribute in the **Stereotypes** attribute for the object class `ICTObject`. The stereotype name is passed on to the objects that users create for the selected ICT object stereotype.

`AllowedArtifacts`	Enter the names of the object classes or object class stereotypes that may be created for the ICT object stereotype.
	Possible object classes and their configured stereotypes include `Application`, `Component`, `Device`, `LocalComponent`, `StandardPlatform`, `SystemBuildingBlock`, and `VendorProduct`.
	FOR EXAMPLE:
	`<ICTObjectManager`
	`<Stereotype Name="TechnicalICTO"`
	`AllowedArtifacts="Application:TechnicalApplication"`
	`CreateClasses="Application" />`
	`<Stereotype Name="BusinessICTO"`

XML Elements (bold) and Attributes	Explanation

	<pre>AllowedArtifacts="Application:BusinessApplication"
	CreateClasses="Application" />
	<Stereotype Name="ComponentICTO"
	AllowedArtifacts="Component"
	CreateClasses="Component" />
	</ICTObjectManager></pre>
	NOTE: to define an object class, use the value of the Name attribute of the object class. To define an object class stereotype, use the value of the **Name** attribute of the object class and the XML attribute `Name` of the XML definition of the stereotype in the **Stereotype** attribute of the object class. Object class name and object class stereotype name are separated with a colon.

4) In the toolbar, click the **Save**  button to save the XML definition.

Configuring the Demand Hierarchy

A branching hierarchy of demand stereotypes may be configured. You can create multiple demand stereotypes and configure multiple demand hierarchies, including which demand stereotypes are permissible for a demand group stereotype. A demand stereotype may have multiple subordinate demand stereotypes. Furthermore, a demand stereotype may be configured to be recursive, thereby allowing subordinate demands to be defined that are based on the same demand stereotype. Thus, a demand stereotype may be configured to allow subordinate demands based on the same demand stereotype to be defined as well as a subordinate demands based on a permissible subordinate demand stereotype to be defined. The name, number, and hierarchical ordering of the demand stereotypes are configurable.

You must first configure the demand stereotypes in the **Stereotypes** attribute of the object class `Demand`. Once the demand stereotypes have been created, you can order the hierarchy of demand stereotypes in the XML object ***DemandManager***

Once defined, a demand stereotype is similar to a standard object class. Correspondingly, you can configure the mandate capability to control user access to each demand stereotype. All demand stereotypes can be configured to be searchable via class settings and thus selected in the list of searchable object classes displayed in the **Search for** field in the **Simple Search** and **Browse** functionalities. Additionally, you can configure custom editors, wizards, workflows, and reports for each demand stereotype, as needed.

The following information is available:

- Creating Object Class Stereotypes for Demands
- Configuring the Attributes for the XML Object DemandManager

Creating Object Class Stereotypes for Demands

To create demand stereotypes:

1) Go to the **Meta-Model** tab and navigate to the protected class node  for the object class `Demand` in the **Class Model** explorer.
2) Click the object class `Demand` in the explorer to open the attribute window.
3) Click the **Browse**  button available for the **Stereotypes** attribute to open the XML editor. Define the following XML attributes:

An error message will be displayed upon closing the text editor if syntax errors are written in the XML definition of the **Stereotypes** attribute.

- `Stereotype Name`: Enter the technical name for each object class stereotype. You may define a name of up to 64 characters. This name should not be changed once the object class stereotypes are implemented.

Changing the technical name after objects have been created based on the object class stereotype requires such objects to be updated by means of an ADIF scheme. For more information, see the reference manual *Alfabet Data Integration Framework*.

- `Caption`: Enter the caption for the object class stereotype. This is the name that users will see in the Alfabet user interface.

  Please note that the following:

  - The caption of the object class stereotype should be different than the caption of its object class. This is particularly important for the implementation of the search functionality for the object class stereotype. For more information, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.
  - The values defined for the XML attribute `Caption` can be translated whereas the values defined for the XML attribute `Stereotype Name` cannot be translated. However, in the table of the **Simple Search** functionality as well as other search selectors, the values defined for the XML attribute `Stereotype Name` of the object class stereotype are displayed in the **Stereotypes** column. If the values defined for the XML attribute `Caption` should be displayed, a custom selector should be defined. For more information, see the section Configuring a Custom Selector for Search Functionalities.
  - If you want to enter a caption `(Caption)` or plural form for the caption (`CaptionPlural`) that contains special characters (for example, a greater then (>) or lesser then (<) symbol), you must replace the special characters with respective XML compliant code, for example:
    - > for >
    - < for <
    - " for "
    - [for [
    -] for]
  - The caption will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.

- `Description`: Provide a brief description about the object class stereotype. The description will be displayed in the **Stereotype Selector** that opens when the user must select which stereotype a new object is to be based on. The description should help the user to understand which stereotype the new object should be based on. The description will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.
- `CaptionPlural`: Enter a plural form for the caption. This is the name in plural that users will see in the Alfabet user interface.
- `IDPrefix`: Enter no more than 5 characters for the ID prefix used to generate the unique identification number for new objects based on the object stereotype. If the XML attribute `IDPrefix` is not specified, the ID prefix of the object class that the object class stereotype is based on will be used to generate the unique ID. Please note that the integer generated for the unique ID is based on the number of instances created for the object class that the object

class stereotype is based on and includes all instances of all object class stereotypes created for the object class.

The prefix ID of existing objects based on an object class stereotype can be updated via the ADIF import scheme `SetStereotypeID` available in the `Alfabet Standard Jobs` folder. For more information, see the section *Predefined ADIF Schemes* in the reference manual *Configuring Alfabet with Alfabet Expand*.

- `Comments`: Enter text in order to provide information about the object class stereotype. This information is only for the purposes of configuration. It is not displayed in the Alfabet user interface.
- `HasMandates`: Enter "`true`" if the mandate capability should be available for the object class stereotype. Enter "`false`" if the mandate capability should not be available for the object class stereotype. If the XML attribute `HasMandates` is defined as "`false`" for an object class stereotype, objects based on the object class stereotype will be visible to users with any mandate implemented in your enterprise. For information about additional configuration requirements necessary to implement the mandate capability, see the section *Implementing the Mandate Capability for a Federated Architecture* in the chapter Configuring Access Permissions for Alfabet .
- `EnableStatutoryLanguage`: Set to `True` if the statutory language capability shall be available for the object class stereotype. The statutory language capability is available in order to address regulatory requirements that mandate that objects are captured in a specific language. When a user creates an object based on a relevant object class/object class stereotype, a statutory language selector will open in which the language can be selected. In the object editor that opens, the user can capture all attributes that support automated translation in the statutory language. If the XML attribute `EnableStatutoryLanguage` is not set, the object class stereotype will inherit the definition of the **Enable Statutory Language** attribute specified for the object class. Specifying a Statutory Language for the Enterprise

4) Click the **Save** button to save your changes.

Configuring the Attributes for the XML Object DemandManager

Once the demand stereotypes have been created for the object class `Demand`, you can specify one or more demand stereotype hierarchies in the XML object ***DemandManager***. A demand stereotype may have multiple subordinate demand stereotypes. A demand stereotype may be configured to be recursive, thereby allowing subordinate demands to be defined that are based on the same demand stereotype. The order of the demand stereotypes listed in the XML object determines the order of the levels in the demand hierarchy. Furthermore, you can specify which demand stereotypes are permissible for which demand groups.

When a user creates or redefines a demand in the Alfabet user interface, the stereotype selector will open if the new/redefined demand may be based on more than one demand stereotype. The user must select the relevant stereotype that the new/redefined demand shall be based on and then the relevant **Demand** editor will open. If only one demand stereotype is permissible, the new/redefined demand will automatically be based on the demand stereotype and the relevant **Demand** editor will open. Only the demands configured at the top-level of the demand hierarchy may be created in the *Demand Management Functionality*, *Capture Demands Functionality*, and *Demand Groups Functionality*.

To edit the XML object ***DemandManager***:

1) Go to the **Presentations** tab, expand the **XML Objects** folder, and expand the **Solution Managers** folder
2) Right-click the XML object ***DemandManager*** and select **Edit XML**. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) As a child of the XML element `DemandManager`, create an XML element `Stereotype` for each object class stereotype on the first hierarchy level. Define the following XML attributes for each XML element `Stereotype`:
   - `Name`: Enter the technical name of the demand stereotype. This is the name defined in the `Name` attribute in the **Stereotypes** attribute for the object class `Demand`. The stereotype name is passed on to the objects that users create for the selected demand stereotype.
   - `IsRecursive`: Enter "`true`" if a sub-demand of this demand stereotype may be created. Enter "`false`" if a sub-demand of the same demand stereotype may not be created for this demand stereotype.
   - `ArchitectureClasses`: Enter a comma-separated list of object classes and objects class stereotypes that may be added as affected architecture for the demand stereotype. A menu entry with the caption **Add <Object Class>** or **Add <Object Class Stereotype>** will be added to the *Affected Architecture Page View*. If you are defining an object class stereotype in the XML attribute `ArchitectureClasses`, you must use the syntax ObjectClass:ObjectClassStereotype. If you are defining an object class stereotype in the XML attribute `ArchitectureClasses`, you must use the syntax ObjectClass:ObjectClassStereotype. If the XML attribute `ArchitectureClasses` is not defined, the standard object classes preconfigured by Software AG will be displayed.

     Please note the following:
     - The order of the demand stereotypes listed in the XML object determines the order of the levels in the demand hierarchy. Each demand stereotype begins with the element ***Stereotype*** in the XML object. The first demand stereotype in the XML object is the demand stereotype that represents the most ascendant level in the hierarchy.
     - To change the order of the demand stereotypes, select the entire demand stereotype and cut and paste it to the correct position in the order. Be sure to include the syntactic constructs (< and />), otherwise you will see an error message at the bottom of the XML editor.
4) If further subordinate hierarchy levels are needed for a stereotype, create the XML element `Stereotype` as child of the parent XML element `Stereotype` and define the XML attributes `Name` and `IsRecursive` as described above.
5) As a child of the XML element `DemandManager`, create an XML element `ClassAccess` for each demand group you want to define and specify the following XML attributes:
   - `ClassName`: Specify `DemandGroup`:<DemandGroupStereotype>, whereby <DemandGroupStereotype> is the `Name` attribute of the XML definition of the demand group stereotype.
   - `Stereotypes`:Enter a a comma-separated list of the permissible demand stereotypes that have been defined at the root node of the XML element `Stereotype`.

6) In the toolbar, click the **Save**  button to save the XML definition.

Configuring the Feature Capability

You can create multiple feature stereotypes and configure multiple feature hierarchies, as needed. The features based on a feature stereotype are instances of the feature stereotype. A feature stereotype may have multiple subordinate feature stereotypes. Furthermore, a feature stereotype may be configured to be recursive, thereby allowing subordinate features to be defined that are based on the same feature stereotype. Thus, a feature stereotype may be configured to allow subordinate features based on the same feature stereotype to be defined as well as subordinate features based on a permissible subordinate feature stereotype to be defined. The name, number, and hierarchical ordering of the feature stereotypes are configurable.

You must first configure the feature stereotypes in the **Stereotypes** attribute of the object class `Feature`. Once the feature stereotypes have been created, you can order the hierarchy of feature stereotypes in the XML object ***FeatureManager***.

You can specify a subset of object classes (`Component`, `Application`, `ICTObject`, `StandardPlatform`, `LocalComponent`, or `PlatformElement`) that the feature may target independent of the affected architecture elements currently defined for the demand. When creating a feature for one those objects, it will automatically be added to the demand's affected architecture.

Once defined, a feature stereotype is similar to a standard object class. Correspondingly, you can configure the mandate capability to control user access to each feature stereotype. All feature stereotypes can be configured to be searchable via class settings and thus selected in the list of searchable object classes displayed in the **Search for** field in the **Simple Search** and **Browse** functionalities. Additionally, you can configure custom editors, wizards, workflows, and reports for each feature stereotype, as needed.

The following information is available:

- Creating Object Class Stereotypes for Features
- Configuring the Attributes and Functionalities Available for a Feature Stereotype

Creating Object Class Stereotypes for Features

To create feature stereotypes:

1) Go to the **Meta-Model** tab and navigate to the protected class node  for the object class `Feature` in the **Class Model** explorer.
2) Click the object class `Feature` in the explorer to open the attribute window.
3) Click the **Browse**  button available for the **Stereotypes** attribute to open the XML editor. Define the following XML attributes:

An error message will be displayed upon closing the text editor if syntax errors are written in the XML definition of the **Stereotypes** attribute.

- `Stereotype Name`: Enter the technical name for each object class stereotype. You may define a name of up to 64 characters. This name should not be changed once the object class stereotypes are implemented.

  Changing the technical name after objects have been created based on the object class stereotype requires such objects to be updated by means of an ADIF scheme. For more information, see the reference manual *Alfabet Data Integration Framework*.

- `Caption`: Enter the caption for the object class stereotype. This is the name that users will see in the Alfabet user interface.

  Please note that the following:

  - The caption of the object class stereotype should be different than the caption of its object class. This is particularly important for the implementation of the search functionality for the object class stereotype. For more information, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.
  - The values defined for the XML attribute `Caption` can be translated whereas the values defined for the XML attribute `Stereotype Name` cannot be translated. However, in the table of the **Simple Search** functionality as well as other search selectors, the values defined for the XML attribute `Stereotype Name` of the object class stereotype are displayed in the **Stereotypes** column. If the values defined for the XML attribute `Caption` should be displayed, a custom selector should be defined. For more information, see the section Configuring a Custom Selector for Search Functionalities.
  - If you want to enter a caption `(Caption)` or plural form for the caption (`CaptionPlural`) that contains special characters (for example, a greater then (>) or lesser then (<) symbol), you must replace the special characters with respective XML compliant code, for example:
    - > for >
    - < for <
    - " for "
    - [for [
    -] for]
  - The caption will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.

- `Description`: Provide a brief description about the object class stereotype. The description will be displayed in the **Stereotype Selector** that opens when the user must select which stereotype a new object is to be based on. The description should help the user to understand which stereotype the new object should be based on. The description will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.

- `CaptionPlural`: Enter a plural form for the caption. This is the name in plural that users will see in the Alfabet user interface.
- `IDPrefix`: Enter no more than 5 characters for the ID prefix used to generate the unique identification number for new objects based on the object stereotype. If the XML attribute `IDPrefix` is not specified, the ID prefix of the object class that the object class stereotype is based on will be used to generate the unique ID. Please note that the integer generated for the unique ID is based on the number of instances created for the object class that the object class stereotype is based on and includes all instances of all object class stereotypes created for the object class.

  The prefix ID of existing objects based on an object class stereotype can be updated via the ADIF import scheme `SetStereotypeID` available in the `Alfabet Standard Jobs` folder. For more information, see the section *Predefined ADIF Schemes* in the reference manual *Configuring Alfabet with Alfabet Expand*.

- `Comments`: Enter text in order to provide information about the object class stereotype. This information is only for the purposes of configuration. It is not displayed in the Alfabet user interface.
- `HasMandates`: Enter "`true`" if the mandate capability should be available for the object class stereotype. Enter "`false`" if the mandate capability should not be available for the object class stereotype. If the XML attribute `HasMandates` is defined as "`false`" for an object class stereotype, objects based on the object class stereotype will be visible to users with any mandate implemented in your enterprise. For information about additional configuration requirements necessary to implement the mandate capability, see the section *Implementing the Mandate Capability for a Federated Architecture* in the chapter Configuring Access Permissions for Alfabet .
- `EnableStatutoryLanguage`: Set to `True` if the statutory language capability shall be available for the object class stereotype. The statutory language capability is available in order to address regulatory requirements that mandate that objects are captured in a specific language. When a user creates an object based on a relevant object class/object class stereotype, a statutory language selector will open in which the language can be selected. In the object editor that opens, the user can capture all attributes that support automated translation in the statutory language. If the XML attribute `EnableStatutoryLanguage` is not set, the object class stereotype will inherit the definition of the **Enable Statutory Language** attribute specified for the object class. Specifying a Statutory Language for the Enterprise

4) Click the **Save** button to save your changes.

Configuring the Attributes and Functionalities Available for a Feature Stereotype

Once the feature stereotypes have been created for the object class `Feature`, you can specify one or more feature stereotype hierarchies in the XML object ***FeatureManager***. A feature stereotype may have multiple subordinate feature stereotypes. Furthermore, a feature stereotype may be configured to be recursive, thereby allowing subordinate features to be defined that are based on the same feature stereotype. The order of the feature stereotypes listed in the XML object determines the order of the levels in the feature hierarchy.

When a user creates a feature in the user interface, the stereotype selector will open if the new feature may be based on more than one feature stereotype. The user must select the relevant stereotype that the new feature shall be based on and then the relevant **Feature** editor will open. If only one feature stereotype is permissible, the new feature will automatically be based on the feature stereotype and the relevant **Feature** editor will open.

Additionally, you can specify the relevant object classes for which features may be defined. The *Features Page View* (`OBJ_Features`) should be added to the custom object views of these classes in order to allow features to be defined for the relevant objects.

To edit the XML object ***FeatureManager***:

1) Go to the **Presentations** tab, expand the **XML Objects** folder, and expand the **Solution Managers** folder
2) Right-click the XML object ***FeatureManager*** and select **Edit XML**. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) For the root XML element `FeatureManager`, create an XML attribute `AvailableClasses` and specify comma-separated list of classes that constitutes the subset of object classes for which features may be defined. You may enter any of the following: `Component`, `Application`, `ICTObject`, `StandardPlatform`, `LocalComponent`, or `PlatformElement`.
4) As a child of the XML element `FeatureManager`, create an XML element `Stereotype` for each object class stereotype on the first hierarchy level. Define the following XML attributes for each XML element `Stereotype`:
   - `Name`: Enter the technical name of the feature stereotype. This is the name defined in the `Name` attribute in the **Stereotypes** attribute for the object class `Feature`. The stereotype name is passed on to the objects that users create for the selected feature stereotype.
   - `IsRecursive`: Enter "`true`" if a sub-feature of this feature stereotype may be created. Enter "`false`" if a sub-feature of the same feature stereotype may not be created for this feature stereotype.

   Please note the following:
   - The order of the feature stereotypes listed in the XML object determines the order of the levels in the feature hierarchy. Each feature stereotype begins with the element ***Stereotype*** in the XML object. The first feature stereotype in the XML object is the feature stereotype that represents the most ascendant level in the hierarchy.
   - To change the order of the feature stereotypes, select the entire feature stereotype and cut and paste it to the correct position in the order. Be sure to include the syntactic constructs (< and />), otherwise you will see an error message at the bottom of the XML editor.
5) If further subordinate hierarchy levels are needed for a stereotype, create the XML element `Stereotype` as child of the parent XML element `Stereotype` and define the XML attributes `Name` and `IsRecursive` as described above.
6) In the toolbar, click the **Save** button to save the XML definition.

Configuring the Policy Hierarchy

You can create multiple policy stereotypes as well as well as specify which policy stereotypes are permissible for a policy group stereotype. The policies based on a policy stereotype are instances of the policy stereotype. Policy stereotypes are not structured as a policy hierarchy with a parent policy and multiple level of subordinate policies. Instead, the policy stereotype concept is a flat list of potential stereotypes. However, you can specify which policy stereotypes are permissible for a policy when a policy is redefined in the *Redefining Policies Page View* (`ITPLC_RedefiningPolicies`.

For a methodological overview of managing the enterprise's policies, see the section *Business IT Synchronization* in the reference manual *IT Planning Basic*.

You must first configure the policy stereotypes in the **Stereotypes** attribute of the object class `ITPolicy`. Once the policy stereotypes have been created, you can specify which policy stereotypes are permissible for redefined policies in the XML object ***ITPolicyManager***.

Once defined, a policy stereotype is similar to a standard object class. Correspondingly, you can configure the mandate capability to control user access to each policy stereotype. All policy stereotypes can be configured to be searchable via class settings and thus selected in the list of searchable object classes displayed in the **Search for** field in the **Simple Search** and **Browse** functionalities. Additionally, you can configure custom editors, wizards, workflows, and reports for each policy stereotype, as needed.

The following information is available:

- Creating Object Class Stereotypes for Policies and Policy Groups
- Configuring Affected Architecture and Redefined Policies for Policy Stereotypes

Creating Object Class Stereotypes for Policies and Policy Groups

Object class stereotypes may be configured for the object classes **Policy** (`ITPolicy`) and **Policy Group** (`ITPolicyGroup`). To create policy stereotypes:

1) Go to the **Meta-Model** tab and navigate to the protected class node  for the object class `ITPolicy` or `ITPolicyGroup` in the **Class Model** explorer.
2) Click the object class `ITPolicy` in the explorer to open the attribute window.
3) Click the **Browse**  button available for the **Stereotypes** attribute to open the XML editor. Define the following XML attributes:

An error message will be displayed upon closing the text editor if syntax errors are written in the XML definition of the **Stereotypes** attribute.

- `Stereotype Name`: Enter the technical name for each object class stereotype. You may define a name of up to 64 characters. This name should not be changed once the object class stereotypes are implemented.

Changing the technical name after objects have been created based on the object class stereotype requires such objects to be updated by means of an ADIF scheme.

For more information, see the reference manual *Alfabet Data Integration Framework*.

- `Caption`: Enter the caption for the object class stereotype. This is the name that users will see in the Alfabet user interface.

  Please note that the following:

  - The caption of the object class stereotype should be different than the caption of its object class. This is particularly important for the implementation of the search functionality for the object class stereotype. For more information, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.
  - The values defined for the XML attribute `Caption` can be translated whereas the values defined for the XML attribute `Stereotype Name` cannot be translated. However, in the table of the **Simple Search** functionality as well as other search selectors, the values defined for the XML attribute `Stereotype Name` of the object class stereotype are displayed in the **Stereotypes** column. If the values defined for the XML attribute `Caption` should be displayed, a custom selector should be defined. For more information, see the section Configuring a Custom Selector for Search Functionalities.
  - If you want to enter a caption (`Caption`) or plural form for the caption (`CaptionPlural`) that contains special characters (for example, a greater then (>) or lesser then (<) symbol), you must replace the special characters with respective XML compliant code, for example:
    - > for >
    - < for <
    - " for "
    - [for [
    -] for]
  - The caption will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.

- `Description`: Provide a brief description about the object class stereotype. The description will be displayed in the **Stereotype Selector** that opens when the user must select which stereotype a new object is to be based on. The description should help the user to understand which stereotype the new object should be based on. The description will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.
- `CaptionPlural`: Enter a plural form for the caption. This is the name in plural that users will see in the Alfabet user interface.
- `IDPrefix`: Enter no more than 5 characters for the ID prefix used to generate the unique identification number for new objects based on the object stereotype. If the XML attribute `IDPrefix` is not specified, the ID prefix of the object class that the object class stereotype is

based on will be used to generate the unique ID. Please note that the integer generated for the unique ID is based on the number of instances created for the object class that the object class stereotype is based on and includes all instances of all object class stereotypes created for the object class.

The prefix ID of existing objects based on an object class stereotype can be updated via the ADIF import scheme `SetStereotypeID` available in the `Alfabet Standard Jobs` folder. For more information, see the section *Predefined ADIF Schemes* in the reference manual *Configuring Alfabet with Alfabet Expand*.

- `Comments`: Enter text in order to provide information about the object class stereotype. This information is only for the purposes of configuration. It is not displayed in the Alfabet user interface.
- `HasMandates`: Enter "`true`" if the mandate capability should be available for the object class stereotype. Enter "`false`" if the mandate capability should not be available for the object class stereotype. If the XML attribute `HasMandates` is defined as "`false`" for an object class stereotype, objects based on the object class stereotype will be visible to users with any mandate implemented in your enterprise. For information about additional configuration requirements necessary to implement the mandate capability, see the section *Implementing the Mandate Capability for a Federated Architecture* in the chapter Configuring Access Permissions for Alfabet .
- `EnableStatutoryLanguage`: Set to `True` if the statutory language capability shall be available for the object class stereotype. The statutory language capability is available in order to address regulatory requirements that mandate that objects are captured in a specific language. When a user creates an object based on a relevant object class/object class stereotype, a statutory language selector will open in which the language can be selected. In the object editor that opens, the user can capture all attributes that support automated translation in the statutory language. If the XML attribute `EnableStatutoryLanguage` is not set, the object class stereotype will inherit the definition of the **Enable Statutory Language** attribute specified for the object class. Specifying a Statutory Language for the Enterprise

4) Click the **Save** button to save your changes.

Configuring Affected Architecture and Redefined Policies for Policy Stereotypes

Once the policy stereotypes have been created for the object class `ITPolicy`, you can specify the object class stereotypes for the class `ITPolicy` that are permissible when redefining policies in the *Redefining Policies Page View* (`ITPLC_RedefiningPolicies`). When a user redefines a policy in the user interface, the stereotype selector will open if the redefined policy may be based on more than one policy stereotype. The user must select the relevant stereotype that the redefined policy shall be based on and then the relevant **Policy** editor will open. If only one policy stereotype is permissible, the new policy will automatically be based on the policy stereotype and the relevant **Policy** editor will open.

Furthermore, you can specify which policy stereotypes are permissible for which policy groups.

Only the policy stereotypes specified on the parent level of the policy hierarchy in the XML object ***ITPolicyManager***will be displayed in the *Policies Page View* (`ITPLCG_Policies`) for a policy group. In other words, if a policy stereotype is only available as a child stereotype and thus only available for purposes of redefining its parent stereotype, the policy based on the child stereotype will not be displayed in the view.

To edit the XML object ***ITPolicyManager***:

1) Go to the **Presentations** tab, expand the **XML Objects** folder, and expand the **Solution Managers** folder

2) Right-click the XML object ***ITPolicyManager*** and select **Edit XML**. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.

3) As a child of the XML element `ITPolicyManager`, create an XML element `Stereotype` for each object class stereotype on the first hierarchy level. Define the following XML attributes for each XML element `Stereotype`:

   - `Name`: Enter the technical name of the policy stereotype. This is the name defined in the `Name` attribute in the **Stereotypes** attribute for the object class `ITPolicy`. The stereotype name is passed on to the objects that users create for the selected policy stereotype.
   - `IsRecursive`: Enter "`true`" if a sub-policy of this policy stereotype may be created. Enter "`false`" if a sub-policy of the same policy stereotype may not be created for this feature stereotype.
   - `ArchitectureClasses`: Enter a comma-separated list of object classes and objects class stereotypes that may be added as affected architecture for the policy stereotype. A menu entry with the caption **Add <Object Class>** or **Add <Object Class Stereotype>** will be added to the *Affected Architecture Elements Page View* and *Implementing Architecture Elements Page View*. If you are defining an object class stereotype in the XML attribute `ArchitectureClasses`, you must use the syntax ObjectClass:ObjectClassStereotype. If the XML attribute `ArchitectureClasses` is not defined, the standard object classes preconfigured by Software AG will be displayed.

     Please note the following:

     - The order of the policy stereotypes listed in the XML object determines the order of the levels in the policy hierarchy. Each policy stereotype begins with the element ***Stereotype*** in the XML object. The first policy stereotype in the XML object is the policy stereotype that represents the most ascendant level in the hierarchy.
     - To change the order of the policy stereotypes, select the entire policy stereotype and cut and paste it to the correct position in the order. Be sure to include the syntactic constructs (< and />), otherwise you will see an error message at the bottom of the XML editor.

4) If further subordinate hierarchy levels are needed for a stereotype, create the XML element `Stereotype` as child of the parent XML element `Stereotype` and define the XML attributes `Name` and `IsRecursive` as described above.

5) As a child of the XML element `ITPolicyManager`, create an XML element `ClassAccess` for each policy group you want to define and specify the following XML attributes:

   - `ClassName`: Specify `ITPolicyGroup`:<PolicyGroupStereotype>, whereby <PolicyGroupStereotype> is the `Name` attribute of the XML definition of the policy group stereotype.
   - `Stereotypes`:Enter a comma-separated list of the permissible policy stereotypes that have been defined at the root node of the XML element `Stereotype`.

6) In the toolbar, click the **Save** button to save the XML definition.

Configuring the Organizational Hierarchy

You can create multiple organization stereotypes and configure multiple organization hierarchies, as needed. The organizations based on an organization stereotype are instances of the organization stereotype. An organization stereotype may have only one ascendant organization stereotype but may have multiple subordinate organization stereotypes. Furthermore, an organization stereotype may be configured to be recursive, thereby allowing subordinate organizations to be defined that are based on the same organization stereotype. Thus, an organization stereotype may be configured to allow subordinate organizations based on the same organization stereotype to be defined as well as a subordinate organizations based on a permissible subordinate organization stereotype to be defined. The name, number, and hierarchical ordering of the organization stereotypes are configurable.

For a methodological overview of working with organizations in the context of the enterprise architecture, see the section *Organization Definition* in the reference manual *Configuring Alfabet with Alfabet Expand*.

Organization stereotypes will be available in the **Organizations** explorer (`ORG_Explorer`), **Organization Groups** explorer (`ORG_Explorer`), **Applications by Consumer** explorer (`ORG_APP_Consumer_Explorer`), **Applications by Owner** explorer (`ORG_APP_Owner_Explorer`), **Business Processes by Organizations** explorer (`ORG_PROC_Explorer`), and **Organization Admin Desktop** functionality (`ORG_AdminDesktop`).

You must first configure the organization stereotypes in the **Stereotypes** attribute of the object class `OrgaUnit`. Once the organization stereotypes have been created, you can order the hierarchy of organization stereotypes in the XML object ***OrganizationManager***.

Once defined, an organization stereotype is similar to a standard object class. Correspondingly, you can configure the mandate capability to control user access to each organization stereotype. All organization stereotypes can be configured to be searchable via class settings and thus selected in the list of searchable object classes displayed in the **Search for** field in the **Simple Search** and **Browse** functionalities. Additionally, you can configure custom editors, wizards, workflows, and reports for each organization stereotype, as needed.

The following information is available:

- Creating Object Class Stereotypes for Organizations
- Configuring the Attributes and Functionalities Available for an Organization Stereotype
- Configuring Alfabet to Capture Legal Ownership of Organizations

Creating Object Class Stereotypes for Organizations

To create organization stereotypes:

1) Go to the **Meta-Model** tab and navigate to the protected class node  for the object class organization in the **Class Model** explorer.
2) Click the object class `OrgaUnit` in the explorer to open the attribute window.
3) Click the **Browse**  button available for the **Stereotypes** attribute to open the XML editor. Define the following XML attributes:

An error message will be displayed upon closing the text editor if syntax errors are written in the XML definition of the **Stereotypes** attribute.

- `Stereotype Name`: Enter the technical name for each object class stereotype. You may define a name of up to 64 characters. This name should not be changed once the object class stereotypes are implemented.

  Changing the technical name after objects have been created based on the object class stereotype requires such objects to be updated by means of an ADIF scheme. For more information, see the reference manual *Alfabet Data Integration Framework*.

- `Caption`: Enter the caption for the object class stereotype. This is the name that users will see in the Alfabet user interface.

  Please note that the following:

  - The caption of the object class stereotype should be different than the caption of its object class. This is particularly important for the implementation of the search functionality for the object class stereotype. For more information, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.
  - The values defined for the XML attribute `Caption` can be translated whereas the values defined for the XML attribute `Stereotype Name` cannot be translated. However, in the table of the **Simple Search** functionality as well as other search selectors, the values defined for the XML attribute `Stereotype Name` of the object class stereotype are displayed in the **Stereotypes** column. If the values defined for the XML attribute `Caption` should be displayed, a custom selector should be defined. For more information, see the section Configuring a Custom Selector for Search Functionalities.
  - If you want to enter a caption `(Caption)` or plural form for the caption (`CaptionPlural`) that contains special characters (for example, a greater then (>) or lesser then (<) symbol), you must replace the special characters with respective XML compliant code, for example:
    - > for >
    - < for <
    - " for "
    - [for [
    -] for]
  - The caption will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.

- `Description`: Provide a brief description about the object class stereotype. The description will be displayed in the **Stereotype Selector** that opens when the user must select which

stereotype a new object is to be based on. The description should help the user to understand which stereotype the new object should be based on. The description will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.

- `CaptionPlural`: Enter a plural form for the caption. This is the name in plural that users will see in the Alfabet user interface.
- `IDPrefix`: Enter no more than 5 characters for the ID prefix used to generate the unique identification number for new objects based on the object stereotype. If the XML attribute `IDPrefix` is not specified, the ID prefix of the object class that the object class stereotype is based on will be used to generate the unique ID. Please note that the integer generated for the unique ID is based on the number of instances created for the object class that the object class stereotype is based on and includes all instances of all object class stereotypes created for the object class.

  The prefix ID of existing objects based on an object class stereotype can be updated via the ADIF import scheme `SetStereotypeID` available in the `Alfabet Standard Jobs` folder. For more information, see the section *Predefined ADIF Schemes* in the reference manual *Configuring Alfabet with Alfabet Expand*.

- `Comments`: Enter text in order to provide information about the object class stereotype. This information is only for the purposes of configuration. It is not displayed in the Alfabet user interface.
- `HasMandates`: Enter "`true`" if the mandate capability should be available for the object class stereotype. Enter "`false`" if the mandate capability should not be available for the object class stereotype. If the XML attribute `HasMandates` is defined as "`false`" for an object class stereotype, objects based on the object class stereotype will be visible to users with any mandate implemented in your enterprise. For information about additional configuration requirements necessary to implement the mandate capability, see the section *Implementing the Mandate Capability for a Federated Architecture* in the chapter Configuring Access Permissions for Alfabet .
- `EnableStatutoryLanguage`: Set to `True` if the statutory language capability shall be available for the object class stereotype. The statutory language capability is available in order to address regulatory requirements that mandate that objects are captured in a specific language. When a user creates an object based on a relevant object class/object class stereotype, a statutory language selector will open in which the language can be selected. In the object editor that opens, the user can capture all attributes that support automated translation in the statutory language. If the XML attribute `EnableStatutoryLanguage` is not set, the object class stereotype will inherit the definition of the **Enable Statutory Language** attribute specified for the object class. Specifying a Statutory Language for the Enterprise

4) Click the **Save** button to save your changes.

Configuring the Attributes and Functionalities Available for an Organization Stereotype

Once the organization stereotypes have been created for the object class `OrgaUnit`, you can specify one or more organization stereotype hierarchies in the XML object ***OrganizationManager***. An organization stereotype may have multiple subordinate organization stereotypes. Furthermore, an organization stereotype may be configued to be recursive, thereby allowing subordinate organizations to be defined that are based on the same organization stereotype. The order of the organization stereotypes listed in the XML object determines the order of the levels in the organization hierarchy.

When a user creates an organization in the user interface, the stereotype selector will open if the new organization may be based on more than one organization stereotype. The user must select the relevant stereotype that the new organization shall be based on and then the relevant **Organization** editor will open. If only one organization stereotype is permissible, the new organization will automatically be based on the organization stereotype and the relevant **Organization** editor will open.

To edit the XML object ***OrganizationManager***:

1) Go to the **Presentations** tab, expand the **XML Objects** folder, and expand the **Solution Managers** folder
2) Right-click the XML object ***OrganizationManager*** and select **Edit XML**. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) Edit the XML attributes, as needed. The table below displays the XML attributes that can be edited for the XML object ***OrganizationManager***:

XML Elements (bold) and Attributes	Explanation
OrganizationManager	
Stereotype	The order of the organization stereotypes listed in the XML object determines the order of the levels in the organization hierarchy. Each organization stereotype begins with the element ***Stereotype*** in the XML object. The first organization stereotype in the XML object is the organization stereotype that represents the most ascendant level in the hierarchy. To change the order of the organization stereotypes, select the entire organization stereotype and cut and paste it to the correct position in the order. Be sure to include the syntactic constructs (< and />), otherwise you will see an error message at the bottom of the XML editor. Define the following attributes for each organization stereotype.
`Name`	Enter the technical name of the organization stereotype. This is the name defined in the `Name` attribute in the **Stereotypes** attribute for the object class `OrgaUnit`. The stereotype name is passed on to the objects that users create for the selected organization stereotype.

XML Elements (bold) and Attributes	Explanation
	NOTE: You must spell the technical name correctly.
`IsRecursive`	Enter "`true`" if a sub-organization of this organization stereotype may be created. Enter "`false`" if a sub-organization of the same organization stereotype may not be created for this organization stereotype.

4) In the toolbar, click the **Save** button to save the XML definition.

Configuring Alfabet to Capture Legal Ownership of Organizations

Alfabet provides a functionality that allows the legal ownership of organizations to be captured. Please note the following configuration required to implement this functionality:

- Custom properties may be captured for the object class `LegalOwnership` in order to capture relevant data regarding the legal ownership relationships in your enterprise. For more information, see the section Configuring Custom Properties for Protected or Public Object Classes.
- The **Legal Owners** page view (`ORG_LegalOwnerships`) and the **Legally-Owned Organizations** page view (`ORG_LegalSubordinations`) must be assigned to the custom object views configured for the relevant object class stereotypes configured for the object class `OrgaUnit`. The **Legal Owners** page view (`ORG_LegalOwnerships`) should be available for the organization stereotype that owns other legal entities and the **Legally-Owned Organizations** page view (`ORG_LegalSubordinations`) should be available for the organization stereotype that represents organizations owned by other legal entities. The percentage that various organizations own a given organization is defined for the owned organization in the **Legal Owners** page view (`ORG_LegalOwnerships`). For more information, see the sections Configuring Object Class Stereotypes for Object Classes and Configuring Object Views.
- The values available for the **Ownership Type** attribute in the **Legal Ownership** editor (`LEGOWN_Editor`) may be configured via the enumeration ***OwnershipType***. For more information, see the section Defining Protected and Custom Enumerations.

Configuring Affected Architecture for Risk Mitigation Templates

You can specify the object classes and objects class stereotypes that may be added as affected architecture for risk mitigation templates.

1) Go to the **Presentations** tab, expand the **XML Objects** folder, and expand the **Solution Managers** folder
2) Right-click the XML object ***AffectedArchManager*** and select **Edit XML**. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) Create an XML element `ClassEntry` and define the following XML attributes:
   - `ClassName`: Enter the technical name `RiskMitigationTemplate`.
   - `ArchitectureClasses`: Enter a comma-separated list of object classes and objects class stereotypes that may be added as affected architecture for risk mitigation templates. A menu entry with the caption **Add <Object Class>** or **Add <Object Class Stereotype>** will be added to the *Affected Architecture Page View*. If you are defining an object class stereotype in the XML attribute `ArchitectureClasses`, you must use the syntax ObjectClass:ObjectClassStereotype. If the XML attribute `ArchitectureClasses` is not defined, the standard object classes preconfigured by Software AG will be displayed.
4) In the toolbar, click the **Save** button to save the XML definition.

Configuring Object Class Stereotypes for Technical Services

A technical service is a service provided by a component or local component in order to fulfill technical needs that are necessary to support business service requests. Technical services can be defined only for components and local components for which the **Component Type** attribute is set to `Service`. For each technical service, technical service operations can be defined that describe how the technical service is to be provided by a component. Technical service methods provide the details about the implementation of the technical service operation (for example, `GET`, `POST`, `DELETE`, etc.).

For a methodological overview of working with technical services in the context of the technical architecture, see the section *Technology Architecture Definition* in the reference manual *Enterprise Architecture Management*.

You can create multiple object class stereotypes for the object classes **Technical Service** (`Service`), **Technical Service Operation** (`ServiceOperation`), and **Technical Service Operation Method** (`ServiceOperationMethod`). If your enterprise configures object class stereotypes for technical services, then typically object class stereotypes will also be configured for technical service operations and technical service operation methods. In this case, only certain technical service operations will be relevant for a particular technical service, and only certain technical service operation methods will be relevant for a particular technical service operation. The XML object ***TechServiceManager*** allows you to map the following:

- For each technical service stereotype, specify which technical service operation stereotypes are permissible.
- For each technical service operation stereotype, specify which technical service operation method stereotypes are permissible.

- If your enterprise supports solution planning and technical services and technical service operations are included as part of the to-be architecture, then you must create an identical configuration for the corresponding solution object classes `SolutionService` and `SolutionServiceOperation`.

Technical services may be created explicitly in Alfabet or may be created based on imported assets from another repository such as CentraSite®, webMethods® API Gateway, or webMethods® API Portal as well as based on WSDL and OpenAPI Specificaiton Swagger (JSON) files. Therefore, you should consider your source for technical services when configuring the object class stereotypes for the object classes **Technical Service** (`Service`), **Technical Service Operation** (`ServiceOperation`), and **Technical Service Operation Method** (`ServiceOperationMethod`).

The following information is available:

- Creating Object Class Stereotypes for Technical Services, Technical Service Operations, and Technical Service Operation Methods
- Mapping Object Class Stereotypes with Technical Services and Technical Service Operations

Creating Object Class Stereotypes for Technical Services, Technical Service Operations, and Technical Service Operation Methods

You can create multiple object class stereotypes for the object classes **Technical Service** (`Service`), **Technical Service Operation** (`ServiceOperation`), and **Technical Service Operation Method** (`ServiceOperationMethod`). Technical services may be created explicitly in Alfabet or may be created based on imported assets from another repository such as CentraSite®, webMethods® API Gateway, or webMethods® API Portal. Therefore, you should consider your source for technical services when configuring the object class stereotypes for the object classes **Technical Service** (`Service`), **Technical Service Operation** (`ServiceOperation`), and **Technical Service Operation Method** (`ServiceOperationMethod`).

If your enterprise supports solution planning and technical services and technical service operations are included as part of the to-be architecture, then you must create an identical configuration for the corresponding solution object classes `SolutionService`, `SolutionServiceOperation`, and `SolutionServiceMethod`. For example, the configuration defined in the **Stereotypes** attribute for the object class `Service` should be copied and pasted to the **Stereotypes** attribute for the object class `SolutionService`. If the configuration is not the same, errors may occur when solution objects are checked in to the Alfabet inventory in the context of project and solution planning. For more about configuring the solution planning capability, see the section Making the As-Is Architecture and To-Be Architecture Capabilities Available.

To create object class stereotypes for the object classes `Service`, `ServiceOperation`, and `ServiceOperationMethod` (as well as the corresponding solution object classes `SolutionService`, `SolutionServiceOperation`, and `SolutionServiceMethod`):

1) Go to the **Meta-Model** tab and navigate to the protected class node  for the relevant object class in the **Class Model** explorer.
2) Click the object class in the explorer to open the attribute window.
3) Click the **Browse**  button available for the **Stereotypes** attribute to open the XML editor. Define the following XML attributes:

An error message will be displayed upon closing the text editor if syntax errors are written in the XML definition of the **Stereotypes** attribute.

- `Stereotype Name`: Enter the technical name for each object class stereotype. You may define a name of up to 64 characters. This name should not be changed once the object class stereotypes are implemented.

  Changing the technical name after objects have been created based on the object class stereotype requires such objects to be updated by means of an ADIF scheme. For more information, see the reference manual *Alfabet Data Integration Framework*.

- `Caption`: Enter the caption for the object class stereotype. This is the name that users will see in the Alfabet user interface.

  Please note that the following:

  - The caption of the object class stereotype should be different than the caption of its object class. This is particularly important for the implementation of the search functionality for the object class stereotype. For more information, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.
  - The values defined for the XML attribute `Caption` can be translated whereas the values defined for the XML attribute `Stereotype Name` cannot be translated. However, in the table of the **Simple Search** functionality as well as other search selectors, the values defined for the XML attribute `Stereotype Name` of the object class stereotype are displayed in the **Stereotypes** column. If the values defined for the XML attribute `Caption` should be displayed, a custom selector should be defined. For more information, see the section Configuring a Custom Selector for Search Functionalities.
  - If you want to enter a caption `(Caption)` or plural form for the caption (`CaptionPlural`) that contains special characters (for example, a greater then (>) or lesser then (<) symbol), you must replace the special characters with respective XML compliant code, for example:
    - > for >
    - < for <
    - " for "
    - [for [
    -] for]
  - The caption will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.

- `Description`: Provide a brief description about the object class stereotype. The description will be displayed in the **Stereotype Selector** that opens when the user must select which

stereotype a new object is to be based on. The description should help the user to understand which stereotype the new object should be based on. The description will be available in the vocabularies and can be translated to the secondary languages implemented in your enterprise.

- `CaptionPlural`: Enter a plural form for the caption. This is the name in plural that users will see in the Alfabet user interface.
- `IDPrefix`: Enter no more than 5 characters for the ID prefix used to generate the unique identification number for new objects based on the object stereotype. If the XML attribute `IDPrefix` is not specified, the ID prefix of the object class that the object class stereotype is based on will be used to generate the unique ID. Please note that the integer generated for the unique ID is based on the number of instances created for the object class that the object class stereotype is based on, and includes all instances of all object class stereotypes created for the object class.

  The prefix ID of existing objects based on an object class stereotype can be updated via the ADIF import scheme `SetStereotypeID` available in the `Alfabet Standard Jobs` folder. For more information, see the section *Predefined ADIF Schemes* in the reference manual *Configuring Alfabet with Alfabet Expand*.

- `Comments`: Enter text in order to provide information about the object class stereotype. This information is only for the purposes of configuration. It is not displayed in the Alfabet user interface.
- `HasMandates`: Enter "`true`" if the mandate capability should be available for the object class stereotype. Enter "`false`" if the mandate capability should not be available for the object class stereotype. If the XML attribute `HasMandates` is defined as "`false`" for an object class stereotype, objects based on the object class stereotype will be visible to users with any mandate implemented in your enterprise. For information about additional configuration requirements necessary to implement the mandate capability, see the section *Implementing the Mandate Capability for a Federated Architecture* in the chapter Configuring Access Permissions for Alfabet .
- `EnableStatutoryLanguage`: Set to `True` if the statutory language capability shall be available for the object class stereotype. The statutory language capability is available in order to address regulatory requirements that mandate that objects are captured in a specific language. When a user creates an object based on a relevant object class/object class stereotype, a statutory language selector will open in which the language can be selected. In the object editor that opens, the user can capture all attributes that support automated translation in the statutory language. If the XML attribute `EnableStatutoryLanguage` is not set, the object class stereotype will inherit the definition of the **Enable Statutory Language** attribute specified for the object class. Specifying a Statutory Language for the Enterprise

4) Click the **Save** button to save your changes.

Mapping Object Class Stereotypes with Technical Services and Technical Service Operations

A technical service operation is a detailed description about how a technical service is to be provided by a component in order to support a component in providing a business service. If your enterprise configures object class stereotypes for technical services, then typically object class stereotypes will also be configured for technical service operations and technical service operation methods. In this case, only certain technical service operations will be relevant for a particular technical service, and only certain technical service operation methods will be relevant for a particular technical service operation.

Once all object class stereotypes have been created for the object classes **Technical Service** (`Service`), **Technical Service Operation** (`ServiceOperation`), and **Technical Service Operation Method** (`ServiceOperationMethod`), you should map the object class stereotypes in the XML object ***TechServiceManager***:

- For each technical service stereotype, specify which technical service operation stereotypes are permissible.
- For each technical service operation stereotype, specify which technical service operation method stereotypes are permissible.
- If your enterprise supports solution planning and technical services and technical service operations are included as part of the to-be architecture, then you must create an identical configuration for the corresponding solution object classes `SolutionService`, `SolutionServiceOperation`, and `SolutionServiceMethod`.

The following example displays the mapping for the stereotype `SOAPService` configured for the object class `Service`:

```
<TechServiceManager>
   <Stereotype ClassName="Service" Name="SOAPService">
      <Stereotype ClassName="ServiceOperation"
      Name="SOAPOperation">
         <Stereotype ClassName="ServiceOperationMethod"
         Name="AHTTPPOST" MethodType="HTTPPOST"
         <Stereotype ClassName="ServiceOperationMethod"
         Name="AHTTPGET" MethodType="HTTPGET" />
         <Stereotype ClassName="ServiceOperationMethod"
         Name="ASOAP12" MethodType="SOAP12"/>
         <Stereotype ClassName="ServiceOperationMethod"
         Name="ASOAP" MethodType="SOAP"/>
      </Stereotype>
   </Stereotype
</TechServiceManager>
```

If your enterprise supports solution planning, you must create an identical configuration for the corresponding solution object class and ensure that the mapping defined for the solution object classes `SolutionService`, `SolutionServiceOperation`, and `SolutionServiceMethod` is identical to the mapping defined the respective object classes `Service`, `ServiceOperation`, and `SolutionOperationMethod`. In other words, you should also map the object class stereotypes specified for the object class

`SolutionService` and to the relevant object class stereotypes specified for the object class `Solution-ServiceOperation`.

To edit the XML object ***TechServiceManager***:

1) Go to the **Presentations** tab, expand the **XML Objects** folder, and expand the **SolutionManagers** folder.
2) Right-click the XML object ***TechServiceManager*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see the section Working with XML Objects.
3) Below the root XML element ***TechServiceManager***, create a child XML element `Stereotype` and specify the following XML attributes:
   - `ClassName`: Enter `Service` (or `SolutionService`) to create the parallel mapping for solution planning):
   - `Name`: Enter the technical name of the object class stereotype defined for the object class `Service` (or `SolutionService`). This is the name defined in the XML attribute `Name` in the **Stereotypes** attribute for the object class.

     ⚠ You must spell the technical name of the technical service stereotype exactly as it is spelled in the **Stereotypes** attribute of the relevant object class.

   - `FileExtension`: Enter `wsdl` or `json` if a technical service based on this stereotype may be based on the respective file type (WSDLor JSON). For more inforamtion about the requirements to work with WSDL and JSON files in the context of technical services, see the section *Configuring the Creation/Export of Technical Services Based on WSDL and OpenAPI Specification Files* in the reference manual *API Integration with Third-Party Components*.
4) Below the XML element `Stereotype` for the class `Service`, create another XML element `Stereotype` and specify the following XML attributes:
   - `ClassName`: Enter `ServiceOperation` (or `SolutionServiceOperation`) to create the parallel mapping for solution planning):
   - `Name`: Enter the technical name of the object class stereotype defined for the object class `ServiceOperation` (or `SolutionServiceOperation`). This is the name defined in the XML attribute `Name` in the **Stereotypes** attribute for the object class.
5) Below the XML element `Stereotype` for the class `ServiceOperation`, create another XML element `Stereotype` and specify the following XML attributes:
   - `ClassName`: Enter `ServiceOperationMethod` (or `SolutionServiceMethod`) to create the parallel mapping for solution planning):
   - `Name`: Enter the technical name of the object class stereotype defined for the object class `ServiceOperationMethod` (or `SolutionServiceMethod`). This is the name defined in the XML attribute `Name` in the **Stereotypes** attribute for the object class.
   - `MethodType`: Enter the method type used for the technical service operation method.
6) In the toolbar, click the **Save** button to save the XML definition.

Configuring Standard Data Capture Environments

The following functionalities are standard data capture environments. Each must be specifically configured for the relevant user profiles that will be working with the functionality:

- **Capture Applications** (`APP_CaptureApplication`)
- **Document Applications** (`APP_UserApplications`)
- **Capture Components** (`COM_CaptureComponents`)
- **Document Components** (`COM_UserComponents`)
- **Capture Demands** (`DEM_CaptureDemands`)
- **Capture Devices** (`DVC_CaptureDevices`)
- **Document Devices** (`DVC_UserDevices`)
- **Capture ICT Objects** (`ICTO_CaptureICTObjects`)
- **Document ICT Objects** (`ICTO_UserICTObjects`)
- **Capture Master Plan Maps** (`ITMPM_CaptureMaps`)
- **Capture Peripherals** (`PRF_CapturePeripherals`)
- **Document Peripherals** (`PRF_UserPeripherals`)
- **Capture Projects** (`PRJ_CaptureProjects`)
- **Sub-Projects** page view (`PRJ_SubProjects`)

Before the data capture environment can be configured, you must first configure the following:

- Ensure that the functionality is assigned to the relevant user profile. For more information, see Making Functionalities Accessible to a User Profile in the chapter Configuring User Profiles for the User Community.
- Configure the class settings for the relevant object class for which data will be captured in the functionality. For example, to implement the `APP_CaptureApplication` functionality, you must configure the `Application` class settings. For more information, see Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.
- Create a view scheme for the user profile and assign the class setting to the view scheme. Assign the view scheme to the user profile. For more information, see Configuring a View Scheme for a User Profile in the chapter Configuring User Profiles for the User Community.

Editors can be configured for the functionalities `APP_UserApplications`, `COM_UserComponents`, `DVC_UserDevices`, `ICTO_UserICTObjects`, and `PRF_UserPeripherals` that allow multiple objects to be maintained and updated in one editor. For more information about the configuration of mass-update editors, see the section Configuring Editors for the Mass Update of Data Capture Objects and Information Flows in the chapter Configuring Custom Editors.

To configure the data capture environment for a user profile:

1) Go to the **Admin** tab and expand the **User Profiles** node.
2) Right-click the user profile  that is associated with the data capture environment and select **Configure User Profile**. The Alfabet solution opens.
3) Navigate through Alfabet to the relevant functionality.
4) On the far right of the toolbar, click the **Configure**  button to open the **View Configuration** editor.
5) In the **Excluded** column, set an X for each button that should be hidden from the display.

   Any button name appended with the word `Simple` indicates that the button represents a standard editor. If the word `Simple` is not in the button name, the button represents a standard wizard.

6) Click **OK** to save the view configuration.
7) To update the interface and review the view configuration, click your browser's back button and then reopen the functionality. The configuration of the data capture functionality will be updated to include the view configuration definition.

Configuring Text Templates for Email Notifications

A text template is the predefined text in an email that is automatically generated as the result of an action triggered by a user. Text templates for email messages are available for the following contexts:

The text template defines the email text in English, the relevant references to objects and object class properties in Alfabet, and hyperlinks to the relevant objects in Alfabet. The content displayed in the English-language text template can be edited to suit the needs of your enterprise. Locale text templates can be created in order to translate the content of the text template to the non-English languages implemented in your solution interface.

The following are examples of the use of text templates in the generation of emails in Alfabet:

- A new assignment is created. The assignee is immediately sent an email informing him/her about the new assignment.
- A user responsible for a workflow step does not carry out the task by the deadline. The responsible user, the original workflow owner, and the current workflow owner will all be sent emails informing them that the workflow step has not been completed in time.
- A discussion has been initiated about an object. All members in the discussion group will be notified that a new discussion has been launched about the object.
- A notepad entry is made about a strategic business support in the context of a business support map. The user responsible for the associated IT strategy is sent an email information him/her that information has been communicated about the strategic business support.

Protected text templates  are available in Alfabet Expand for all functionalities supporting email functionality. The protected text templates are predefined and used as the template for the relevant email, per

default. Protected text templates can be edited as needed. For an overview of all available protected text templates and the meaning of the variables available for them, see the chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

In the case of workflows, consistency monitors, notification monitors, and discussion groups, public text templates  can be created based on protected text templates. This allows you, for example, to create text templates for different types of workflows where different text or information is required depending on the purpose of the workflow or for different notification monitors which will target different aspects of the architecture. The public text template must then be explicitly assigned to the relevant configuration available for the workflows, consistency monitors, notification monitors, and discussion groups that they have been created for.

Please note that text templates for notification monitors and consistency monitors must be located in their respective folder. The `NotificationMonitorDefault` text template or any copy thereof must be located in the text template folder ***M_NOTE*** and the `ConsistencyMonitorDefault` and `ConsistencyMonitorMail` text templates or any copy thereof must be located in the text template folder ***M_CON***. The names of the text template folders must not be changed. Furthermore, all existing protected text templates as well as any new text templates for workflows must be stored in the ***WF*** folder.

For additional information regarding additional configuration required to implement email notifications in the context of workflows, see the section Configuring Email Notifications for Workflows.

The following configuration requirements must be completed by the system administrator in the tool Alfabet Administrator:

- All Alfabet functionalities for which the email capability is to be implemented require the setup of a connection to an SMTP server for outgoing email. For more information, see the section *Activating the Dispatch of Email Notifications in Alfabet* in the reference manual *System Administration*.
- The user profile used to open the Alfabet views targeted by hyperlinks in the email is, by default, the user profile that the sender was logged in with when the view was sent/triggered. However, your system administrator may configure that Alfabet views in email notifications are to be opened using the recipient's user profile. For more information, see the section *Configuring the Setup To Open the Alfabet Interface via Links in Email Notifications* in the reference manual *System Administration*.

The language displayed for a user's email notifications is specified by a user administrator in the **Email Notification Language** attribute for the relevant user in the **User Administration** functionality accessible via an administrative user profile. For more information, see the section *Defining and Managing Users* in the reference manual *User and Solution Administration*.

The following information is available:

- Editing a Protected Text Template
- Creating a Custom Text Template
- Defining a Query for a Text Template

- Sending a Test Email Based on a Text Template
- Creating a Translation of a Text Template
- Specifying Custom Text Templates for Password Generation
- Deleting a Public Text Template

Editing a Protected Text Template

The text templates define the email text, relevant references to objects and their object class properties, and hyperlinks to the relevant objects in the solution interface. You can change or replace the text in the email although you must be careful not to change the expressions in the predefined standard variables available in the text templates. The text templates may either be specified in ASCII or HTML format, whereby HTML must be XML-conform HTML, compliant with HTML 5, and use standard HTML tags. Locale texts can be created for each text template in order to provide the email in the relevant secondary languages supported by your enterprise.

The text templates define the email text, relevant references to objects and their object class properties, and hyperlinks to the relevant objects in the solution interface. You can make the following changes to any text template available in Alfabet Expand:

- The **Subject** field of the email can be specified via the **Caption** attribute of the text template.
- The text template can be either formatted as ASCII or HTML. Please note that the HTML must be XML-conform HTML, compliant with HTML 5, and use standard HTML tags.
- There are three different types of variables that can be included in a text template:
  - Predefined standard variables: These are variables that are predefined by Software AG and serve the purpose of referencing the relevant data in the Alfabet database. For example, the variable `{Object:RefImage}` typically allows the base object referenced by the text template to be included in the text template. You can add, remove, or move the entire variable via copy and paste. If the expression in the curly brackets `{XXX}` is changed, the text template may fail to work correctly.
  - Object variables: A specified set of variables is available that allows information such as the object targeted by the notification or a user referenced by the object. The syntax of these references are {<ObjectClass>:<ObjectClassProperty>}, whereby the set of object classes that can be specified in <ObjectClass> are predefined. Any scalar property defined for the <ObjectClass> expression can be referenced in the <ObjectClassProperty>. For example, if an application is the base object referenced by the email, you would enter `{Object:ID}` to include the ID of an application or `{Object:Description}` to include the description of an application. You can add, remove, or move the entire variable via copy and paste.
  - Query variables: It is possible to specify an Alfabet query or SQL query to add more complex references to the text template that cannot be captured via scalar properties (for example, additional referenced information or a document link). If you plan to specify an Alfabet query or native SQL query to the text template, you must enter the variable {`Query:` <QueryName>} to the text template, whereby the <QueryName> expression is determined by the value of the **Name** attribute specified for the query text configuration object.
- The text displayed in the templates can be edited, as needed.

- Locale texts can be created for each text template in order to provide the email in the relevant secondary languages supported by your enterprise.

For an overview of the predefined variables as well as the object variables available for each protected text template, see the chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

All protected text templates  may be edited as needed. To edit a protected text template:

1) Go to the **Presentation** tab and expand the **Text Templates** folder by clicking the + symbol. The following folders are available containing predefined text templates that may be edited.

   - ***AE***: The ***AE*** folder contains text templates for email notifications associated with the activation of a user password. For an overview of the predefined variables as well as the object variables available for each protected text template, see the section *Text Templates for Activation of User Passwords* chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*. For more information about specifying the message in the emails or configuring custom text templates to use in place of the standard text templates, see the section Specifying Custom Text Templates for Password Generation.

   - ***ASMT***: The ***ASMT*** folder contains text templates for email notifications sent in the context of the *Assignment Functionalities*. For an overview of the predefined variables as well as the object variables available for each protected text template, see the section *Text Templates for Assignments* chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*. For an overview of the configuration required to implement the assignments capability, see the section Configuring the Assignment Capability in the reference manual *Configuring Alfabet with Alfabet Expand*.

   - ***DSC*** and ***DSC_Activation***: The ***DSC*** and ***DSC_ActivationStandard*** folders contain text templates for email notifications associated with the *Managing and Contributing to the Discussion*. Email notifications may be sent when a discussion is initiated about an object or a contribution is made to the discussion by a member of a discussion group. For an overview of the predefined variables as well as the object variables available for each protected text template, see the section *Text Templates for Discussions* chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*. For more information about the configuration of the discussion functionality, see the section *Defining Discussion Groups for Collaborative Discussions* in the reference manual *User and Solution Administration*.

   - ***ITMAP***: The ***ITMapFolder*** contains a protected text template that is relevant when users create notes in the notepad capability when working with business support maps. For an overview of the predefined variables as well as the object variables available for each protected text template, see the section *Text Templates for Business Support Maps* chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*. For more information, see the section *Communicating about Business Supports via the Notepad Functionality* in the reference manual *IT Planning Basic*.

   - ***M_CON***: The ***M_CON*** folder contains protected text templates that can be used as to send email notifications about the assignments triggered by the execution of consistency monitors. For an overview of the predefined variables as well as the object variables available for each protected text template, see the section *Text Templates for Consistency Monitors* chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference

manual *Configuring Alfabet with Alfabet Expand - Appendix*. For more information about the configuration of consistency monitors, see *Defining Consistency Monitors* in the reference manual *User and Solution Administration*.

- ***M_NOTE***: The ***M_NOTE*** folder contains a text template that can be used for email notifications that are triggered for notification monitors. Notification monitors are based on configured queries that determine when a monitor should be triggered for an object and to whom email notifications should be sent. For an overview of the predefined variables as well as the object variables available for each protected text template, see the section *Text Templates for Notification Monitors* chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*. For detailed information about the configuration of notification monitors and their text templates, see the section Configuring Notification Monitors in the reference manual *Configuring Alfabet with Alfabet Expand*.
- ***MON***: The ***MON*** folder contains text templates for email notifications associated with the execution of activity, inactivity, and date monitors. A monitor is triggered whenever a defined property is either changed (activity monitor), not changed (inactivity monitor), or reaches a specified date. For an overview of the predefined variables as well as the object variables available for each protected text template, see the section *Text Templates for Activity, Inactivity, and Date Monitors* chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*. For more information about monitors, see the section *Keeping Track of Objects via Monitors* in the reference manual *User and Solution Administration*.
- ***ORG***: The ***ORG*** folder contains text templates used for the email notifications that are automatically generated in the context of an organizational change made in the *Organizational Changes Page View*. For an overview of the predefined variables as well as the object variables available for each protected text template, see the section *Text Templates for Organizational Changes* chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.
- ***PROC***: The ***PROC*** folder contains text templates used for the email notifications that are automatically generated in the context of a business process change in the *Business Process Changes Page View*. For an overview of the predefined variables as well as the object variables available for each protected text template, see the section *Text Templates for Business Process Changes* chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.
- ***WF***: The ***WF*** folder contains text templates for the email notifications sent in the context of the workflows capability. The ***WF*** folder contains all text templates for email notifications associated with workflow activities. A text template is available for each action involved in a workflow that requires notification to the relevant responsible users. For detailed information about configuring email notifications for workflows, see the section Configuring Email Notifications for Workflows. For an overview of the predefined variables as well as the object variables available for each protected text template, see the section *Text Templates for Workflows* chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

2) Expand the relevant text template folder  and click the relevant protected text template  to display its attribute window. Define the following attributes, as needed:

- **Caption**: Enter text for the subject line for the email message. The caption is displayed in the **Subject** field of the email. You can define text as well as any variables for the subject line that can be added to the body of the text template.
- **Group**: Displays the text template folder where the text template is stored. Please note that text templates must be stored in their respective folders. Otherwise, errors may occur with the email notification.
- **Is HTML**: Select `True` if the email should have HTML formatting instead of ASCII format. If you set this attribute to `True`, the HTML code must be entered in the **Text** attribute. Select `False` if the email should have ASCII format.

  Please note that if email notifications are consolidated for workflow templates and the **Is HTML** attribute is set to `True` for the associated text template, the consolidated email will also have HTML formatting. For additional information regarding additional configuration required to implement email notifications in the context of workflows, see the section Configuring Email Notifications for Workflows.

  Please note that you should not set the **Is HTML** attribute to `True` in the context of the text templates available in the ***ORG*** or ***PROC*** folder.

- **Text**: Revise the existing text and variables, as needed, to write the text in the email. Please note the following:
  - If you set the **Is HTML** attribute to `True`, enter the HTML code.
    - The HTML must be XML-conform HTML, compliant with HTML 5, and use standard HTML tags.
    - Variables in curly brackets `{XXX}` can be used but must be placed in a `<span>` tag.
    - The formatting of the HTML can either be written explicitly in the `<body>` element or can be specified via a stylesheet that is stored in the **Internal Document Selector** . In this case, the HTML must refer to the target CSS file in a `<link>` element. For more information about uploading documents to the functionality. See the chapter *Uploading Documents and Managing User Permissions to Document Folders in the Internal Document Selector* in the reference manual *User and Solution Administration*.
  - If you set the **Is HTML** attribute to `False`, edit ASCII text as needed.
  - Regardless of whether you are defining HTML formatting or ASCII format, please keep the following in mind:
    - The text can be edited as needed. You can add, delete or revise text in the text template.

3) To specify an Alfabet query or SQL query in order to add a query variable {`Query:` <QueryName>} to the text template, right-click the text template and select **Create Text Query**. The text query  is added below the text template node. Click the text query  to open the attribute grid and define the following, as needed:

- Define a name for the text query in the **Name** attribute. The name must be included in the <QueryName> expression of the variable {`Query:` <QueryName>} that you include in the **Text** attribute of the text template.
- Define either an Alfabet query in the **Alfabet Query** attribute or a native SQL query in the **Native SQL** attribute. For more information about configuring Alfabet queries and native SQL queries, see the chapter Defining Queries.

The following native SQL query returns a document link for the application that is the current object of a workflow step:

```
SELECT URL.LINK FROM ALFA_URI URL
JOIN APPLICATION APP ON URL.OBJECT=APP.REFSTR
WHERE APP.REFSTR=@OBJECT
AND URL.CATEGORY='Manual'
```

The following native SQL query returns all outgoing information flows of a workflow's base application:

```
SELECT NULL, INFORMATIONFLOW.ID, INFORMATIONFLOW.NAME FROM
INFORMATIONFLOW, APPLICATION, WORKFLOW
WHERE INFORMATIONFLOW.A_FROM = APPLICATION.REFSTR AND
WORKFLOW.ARTIFACT = APPLICATION.REFSTR
AND WORKFLOW.REFSTR = @WORKFLOW
```

- For Alfabet queries, parameters must be defined with the prefix: (colon symbol).
- For native SQL queries, parameters must be specified with the @ (at sign)
- For native SQL query, `REFSTR` is required as the first column.
- Parameters in queries return the `REFSTR` of the object they refer to.
- You can use the following parameters in the query:
  - `SENDER` to refer to the user who is triggering the notification
  - `PERSON` to refer to the recipient of the notification.
  - `OBJECT` to refer to the object targeted by the workflow step
  - All objects returned by the query will be added to the text using one line per item.
  - For text templates sent in the context of workflows only: `WORKFLOW` to refer to the current workflow. Please note that this parameter cannot be used for a text template that is based on a query if the text template is assigned to the first workflow step of a workflow.
  - For text templates sent in the context of workflows only: `WORKFLOWSTEP` to refer to the current workflow step. Please note that this parameter cannot be used for a text template that is based on a query if the text template is assigned to the first workflow step of a workflow.
  - All object class properties defined as show properties in the **Alfabet Query** or in the `SELECT` statement of the native SQL query will be displayed concatenated in the email.

- The results of the query can be formatted in HTML. Please note the following:
  - The **Is HTML** attribute must be set to `True` for the text template.
  - The **Returns HTML** attribute must be set to `True` for query object defined for the text template.
  - The first column of the text template query must return the `REFSTR` and the column with the HTML content must have a column name.
  - Special characters in the formatted string must be correctly escaped.
- It is possible to create more than one query per text template.

4) In the toolbar, click the **Save** button to save your changes.

Creating a Custom Text Template

It may be that your enterprise needs to create custom text templates in order to provide customized information in the context of workflows, consistency monitors, notification monitors, and discussion groups.

It is possible to create a custom text template for workflows, consistency monitors, notification monitors, and discussion groups based on any of the existing protected templates provided by Software AG.

Public text templates should not be created for assignments, activity monitors, inactivity monitors, date monitors, IT map notepad entries, and organization or business process changes. In these contexts, only the protected text templates are implemented and all public text templates will be ignored.

In order to create a custom text template, you must create a new text template and then copy an existing text template to the new text template and then modify the text or placement of variables, as needed. For example, to create a custom text template `StageGateExpired` sent in the context of a workflow targeting enterprise release management, you would copy the protected text template `WorkflowStepEscalated`, which is typically sent when a responsible user does not perform a workflow step by a targeted deadline and therefore, the status of the workflow step is escalated. In the custom text template, you could specify a subject line, include other users in the CC field to ensure that they are informed about the expired stage gate deadline, add additional explanatory text that clarifies the issue, and add additional information via the inclusion of the targeted object's scalar properties. The custom text template would then be assigned to the notification action configured for the relevant workflow step of the workflow template. All custom templates will be included in the batch processing of email notifications if batch processing is specified for the workflow template.

Although it is technically possible to create custom text templates for other contexts such as assignments or notepad entries, only the protected text templates are implemented, and all public text templates will be ignored by the system. Only public text templates for workflows, consistency monitors, notification monitors, and discussion groups will be implemented.

All existing protected text templates as well as custom text templates must be stored in the appropriate text template folder . The following is required in order to implement custom text templates:

- Custom text templates for user password generation must be located in the `AE` folder. For information about specifying custom text templates for email notifications for password generation, see the section Specifying Custom Text Templates for Password Generation.
- Custom text templates for workflows must be located in the `WF` folder. For information about specifying email notifications for workflows, see the section Configuring Email Notifications for Workflows.
- Custom text templates for consistency monitors must be located in the `M_CON` folder. For information about specifying email notifications for consistency notifications, see the section *Defining Consistency Monitors* in the reference manual *User and Solution Administration*.
- Custom text templates for notification monitors must be located in the `M_NOTE` folder. For information about specifying email notifications for notification monitors, see the section *Defining Notification Monitors* in the reference manual *User and Solution Administration*.
- Custom text templates for discussion groups must be located in the `DSC_Activation` folder. For information about specifying email notifications for discussion groups, see the chapter *Defining Discussion Groups for Collaborative Discussions* in the reference manual *User and Solution Administration*.
- For more information about the predefined standard variables and configurable object variables allowed for each relevant text template, see the chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

To create a new text template:

1) Go to the **Presentation** tab and expand the **Text Templates** folder by clicking the + symbol.
2) Right-click the relevant text template folder  and select **New Text Template**.
3) Click the new public text template  to display its attribute window. Define the technical name for the text template in the **Name** attribute.

   A validation mechanism checks for correct syntax when defining a technical name. The technical name of a configuration object may not begin with an empty space nor include any of the following special characters: **\ / * ? " > < | ':**

   Furthermore, please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.

   Furthermore, if the technical name of a configuration object is changed, the name will be correctly updated in other configuration objects referencing the changed object during design time. Please note that the name of a changed configuration object will not be updated in guide pages nor if the guide pages reference a configuration object listed in the **Show Usage** functionality. If you plan to change the name of a configuration object, the reference in the guide page should be changed prior to changing the configuration object name.

4) In the **Group** attribute, enter the name of the folder in which the text template should be stored. Please note that text templates must be stored in their respective folders. Otherwise, errors may occur with the email notification. Please note the following:

- Specify `AE` to store text templates for password generation. See the section Specifying Custom Text Templates for Password Generation for further configuration requirements to implement custom templates for password generation.
- Specify `M_CON` to store text templates for consistency monitors.
- Specify `M_NOTE` to store text templates for notification monitors.
- Specify `WF` to store text templates for workflow notifications.
- Specify `DSC_Activation` to store text templates for discussion groups.

5) You now need to create the email message for the new text template. To do so, you should copy an existing text template including its text and variables and paste it in the new text template. You can then modify the text and cut and paste the variables, as needed.
   - To copy the text of a standard text template for workflows, select the relevant text template and open the text editor by clicking the **Browse** button available for the **Text** attribute. Select the text and right-click and select **Copy**.
   - To paste the text to the new text template, select the new text template and open the text editor by clicking the **Browse** button available for the **Text** attribute. Right-click in the editor and select **Paste**.
6) Define the attributes in the attribute grid for the public text template as described in the section Editing a Protected Text Template. For more information about the predefined standard variables and configured object variables that are allowed for each text template, see the chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.
7) In the toolbar, click the **Save** button to save your changes.

Sending a Test Email Based on a Text Template

A **Send Test Mail** functionality has been added to text templates configured in the configuration tool. If selected, the email will be registered in the `ALFA_EMAIL_BUS` database table. Once the Alfabet Server is running, the email will be sent to the user configured for the test email in the server alias settings or the **Alfabet Configuration Overrides** functionality.

To test the text template, right-click the text template and select **Send Test Mail**. Confirm the info message by clicking **OK**. The sent email can be tracked in the **Email Message Log** functionality, described in the section *Tracking the Email Messages Sent in the Context of AlfabetFunctionalities* of the reference manual *User and Solution Administration*.

The **Alfabet Configuration Overrides** functionality allows solution designers that do not have access to the Alfabet Administrator and need to test a configuration in which emails are triggered to override the test email account and sender email address definitions made in the server alias configuration of the Alfabet Web Application. The email settings defined via the user interface will override settings made in the server alias configuration of the Alfabet Web Application. For more information about the **Alfabet Configuration Overrides** functionality, see the section *Overriding Server Configurations for Testing Purposes* in the reference manual *User and Solution Administration*. For more information about the configuration of the email functionality, see

the section *Activating the Dispatch of Email Notifications in Alfabet* in the reference manual *System Administration*.

Creating a Translation of a Text Template

A locale text template is a localized version of a standard or custom text template in which the text can be translated to another language. Text templates can be generated in multiple languages, allowing the translated email notifications generated within Alfabet to be displayed for a selected culture.

Please note that in the context of email notifications, regardless of whether or not a locale text template is created, any translated captions of object classes, object class stereotypes, object class properties, etc. will be displayed in the email if the user triggering the email has the relevant language selected for the Alfabet interface.

The language displayed for a user's email notifications is specified by a user administrator in the **Email Notification Language** attribute for the relevant user in the **User Administration** functionality accessible via the `Admin` user profile. For more information, see the section *Defining and Managing Users* in the reference manual *User and Solution Administration*.

To create and translate a locale text template:

1) Go to the **Presentation** tab and expand the **Text Templates** node. Expand the relevant text template folder.
2) Right-click the text template that you want to create a language version for and select **Create Locale Text**. You will see a copy of the text template  appear in the explorer tree below the original English-language text template.

   The locale text template must be stored in the appropriate folder with its parent text template. For example, all text templates for workflows must be stored in the `WF` folder, all text templates for consistency monitors must be stored in the `M_CON` folder, etc.

3) Click the new locale text template to open the attribute window and define the following attributes:
   - **Locale Name**: Select the base culture specified in the culture that this local text template is translated for.
   - **Text**: Edit the text of the email message as you would for any protected text template. Please note the following:
     - The text displayed in the templates can be translated, as needed.
     - Predefined variables as well as configurable object variables can be moved via copy and paste within a text template.
     - The expression in curly brackets `{XXX}` in predefined variables may not be edited.
     - For detailed information about specifying a text template, see the section *Editing a Protected Text Template*.

4) In the toolbar, click the **Save**  button to save your changes.

Specifying Custom Text Templates for Password Generation

The standard protected text templates `ActivationEmailNewPassword` is used when a new user password is generated for a user and `ActivationEmailURLLink` is used when a new user password is generated for a user. However, you can create custom templates to replace these protected text templates. To create the custom text templates, following the instructions as described in the section Creating a Custom Text Template. Please note that the custom text templates for user password generation must be located in the `AE` folder.

For an overview, of the variables available for text templates in the context of password generation, the section *Text Templates for Activation of User Passwords* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*. For more information about the password regeneration functionality, see the section *Defining, Clearing, and Resetting a User's Password* in the reference manual *User and Solution Administration*.

Once the custom text templates have been created, you must specify which custom text templates to implement instead of the default text templates `ActivationEmailNewPassword` and `ActivationEmailURLLink`:

1) Go to the **Presentation** tab and expand the **XML Objects** folder.
2) Right-click the XML object ***SolutionOptions*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) In the XML attribute `GeneratePassWordMailTemplate_Pass`, enter specify the name of the custom text template that will replace the default text template `ActivationEmailNewPassword`.
4) In the XML attribute `GeneratePassWordMailTemplate_Link`, enter specify the name of the custom text template that will replace the default text template `ActivationEmailURLLink`.
5) In the toolbar, click the **Save**  button to save the XML definition.

Deleting a Public Text Template

Before a public text template is deleted, you should check to see if the text template is implemented in other configuration objects that are configured in Alfabet Expand such as workflow templates.

Public text templates implemented in notification monitors, consistency monitors, and discussion groups will not be reported in the **Show Usage** functionality because they are not configured in Alfabet Expand but rather in the context of the Solution Configuration functionality in the Alfabet interface that can be accessed via the `Admin` user profile. You must manually check the configuration of these objects before you delete them. For more information, see the sections *Defining Consistency Monitors*, *Defining Notification Monitors*, and *Defining Discussion Groups for Collaborative Discussions* in the reference manual *User and Solution Administration*.

To do so, right-click the text template and click **Show Usage**. A window will be displayed showing the configuration objects using the selected text template. For more information regarding the **Show Usage**

functionality, see the section Using the Show Usage Functionality in the chapter Getting Started with Alfabet Expand.

1) In the **Presentation** tab, click the + symbol next to the relevant text template folder.
2) Right-click the text template that you want to delete and select **Delete**.
3) Confirm the warning by clicking **Yes**. The text template is deleted from the explorer.

Configuring Monitors

Alfabet provides a number of different monitors that help your enterprise maintain accurate and up-to-date data. Monitoring allows you to keep track of specified objects for activity (the object has been edited), inactivity (the object has not been edited), or an approaching date (such as an end date) for a specific object or all objects in an object class, and consistency of data. When a monitor is activated, all users defined as listeners to the monitor are automatically sent an email notification. The users defined will be able to access the object that the monitor is about via a hyperlink in the email notification by means of a specified user profile.

Alfabet provides a variety of monitors to alert Alfabet users about changes that have occurred to specific objects which may require further activity, reviews that did not occur as intended, or transactions that were expected to occur. The following monitor types are available in Alfabet:

- **Activity Monitor**: An activity monitor alerts specified users about changes that have been made to specified objects in an object class. The monitor owner must specify a set of attributes that are to be monitored and may specify a set of users that are to be alerted if the monitor is triggered. An email will be sent to the monitor owner and all defined users when the monitor is triggered.

  The monitor owner and specified users should be alerted about changes made to the start and end dates of information flows defined for a specific set of applications. In this case, you would define an activity monitor of the type **Application** with the monitored context **Application - Information Flows**. You would then define the attributes **Start Date** and **End Date** to track.

The following configuration is required:

- The activity monitor is created in the **Monitors** functionality that may be accessed via an administrative user profile. For more information about defining activity monitors, see the section *Defining Activity Monitors for Specific Objects* in the reference manual *Getting Started with Alfabet*.
- Configure the following in Alfabet Expand:
  - Set `True` for the **Audit** attribute for the object classes that are to be monitored.
  - Set `True` for the **Enabled** attribute for the monitor template.
  - Set `True` for the **Enabled** attribute for the monitor contexts.
  - Configure the text template `MonitorObjectChanged` in the ***MON*** folder. This is used for email notifications about an object that has been changed.

- **Inactivity Monitor**: An inactivity monitor alerts the monitor owner and specified users about the absence of activity occurring to specified objects in an object class. The monitor owner must specify

a set of objects that are to be monitored and may specify a set of users that are to be alerted if the monitor is triggered. An email will be sent to the monitor owner and all defined users if a specified object is not changed or reviewed within a specified period of time. The monitor owner can specify the monitor object as reviewed via the **Mark as Reviewed** button in the object profile of the targeted object.

The monitor owner and specified users should be alerted if no changes are made to an application. Specified applications should be reviewed on a monthly basis. In this case, you would define an inactivity monitor of the type **Application** with a monthly frequency.

The following configuration is required:

- An inactivity monitor is created in the **Monitors** functionality that may be accessed via an administrative user profile. For more information about defining inactivity monitors, see the section *Defining Inactivity Monitors for Specific Objects* in the reference manual *Getting Started with Alfabet*.
- Configure the following in Alfabet Expand:
  - Set `True` for the **Audit** attribute for the object classes that are to be monitored.
  - Set `True` for the **Enabled** attribute for the monitor template.
  - Set `True` for the **Enabled** attribute for the monitor contexts.
  - Configure the text template `MonitorObjectChanged` in the ***MON*** folder. This is used for email notifications about an object that has been changed.

- **Date Monitor**: A date monitor allows you to keep track of the approach of a date (start date, end date, target date, etc.) for specified objects or the date of related objects. The monitor owner must specify a set of objects that are to be monitored and may specify a set of users that are to be alerted if the monitor is triggered. An email will be sent to the monitor owner and all defined users when the monitor is triggered.

The monitor owner and specified users should be alerted about the approaching end date of the business supports provided by specific applications. In this case, you would define a date monitor of the type **Application** with the monitored context **Application Business Support**. You would then specify that the end date should be tracked.

The following configuration is required:

- A date monitor is created in the **Monitors** functionality that may be accessed via an administrative user profile. For more information about defining inactivity monitors, see the section *Defining a Date Monitor for Specific Objects* in the reference manual *Getting Started with Alfabet*.
- Configure the following in Alfabet Expand:
  - Set `True` for the **Enabled** attribute for the monitor template.
  - Set `True` for the **Enabled** attribute for the monitor contexts.
  - Configure the text template `MonitorObjectDateAlert` in the ***MON*** folder. This is used for email notifications about an object's specified date falls in the review period.

- **Notification Monitor**: A notification monitor allows email notifications to be automatically triggered based on configured Alfabet queries or native SQL queries. The queries specify the objects and attributes that are targeted as well as the users who shall be notified about the objects found by the queries.

  Users who have a specified role for applications should be alerted about any new information flows that are created for the applications that they are associated with.

The following configuration is required:

- A notification monitor is created in the **Notification Monitors** functionality that may be accessed via an administrative user profile. For more information about defining notification monitors, see the section *Defining Notification Monitors* in the reference manual *User and Solution Administration*.
- Configure the following in Alfabet Expand:
  - Configure the text template `NotificationMonitorDefault` located in the text template folder ***M_NOTE***. This is used for email notifications about notification monitors. Please note that text templates for notification monitors must be located in the text template folder ***M_NOTE***. The name of the text template folders must not be changed.

- **System Date Monitor**: A system date monitor is a time-triggered monitor for an object class on a system-wide basis. A system date monitor allows you to keep track of the approach of a date for all objects in a specified object class and their related objects (such as an object's start date, end date, target date, etc.). When a specified date approaches for an object in the object class, an assignment will be generated for the object's authorized user. The authorized user will also receive an email information them of their new assignment.

  All users responsible for any application should be alerted about business supports that have an end date later than the applications providing the business support.

The following configuration is required:

- A system date monitor is created in the **System Date Monitors** functionality that may be accessed via an administrative user profile. For more information about defining system date monitors, see the section *Defining System Date Monitors* in the reference manual *User and Solution Administration*.
- Set `True` for the **Enabled** attribute for the monitor template.
- Set `True` for the **Enabled** attribute for the monitored contexts.
- Configure the XML object ***SystemMonitorDefaultDef*** to configure the assignments generated in the context of system date monitors
- Configure the text template `AssignmentNew` in the `ASMT` folder. This is used for email notifications about the assignments generated for the monitor.
- Configured the text template `MonitorObjectCountdownReview` in the ***MON*** folder. This is used to fill the **Description** attribute of the assignments that are created.

- **Consistency Monitor**: A consistency monitor supports the system-wide maintenance of objects in the Alfabet database. The consistency monitor is specified to periodically search for inconsistencies

among objects. Each consistency monitor is based on an Alfabet query or native SQL query that defines the object classes targeted by the query as well as the inconsistent attributes to be detected. If an inconsistency is found by the query, an assignment will be generated for the object's authorized user. The authorized user will also receive an email information them of their new assignment. The timely completion of the assignment triggered by a consistency monitor can be tracked by the solution administrator.

Users responsible for applications should be alerted about business supports that have an end date later than the applications providing the business support.

The following configuration is required:

- Consistency monitors are created in the **Consistency Monitors** functionality that may be accessed via an administrative user profile. For more information about defining consistency monitors, see the section *Defining Consistency Monitors* in the reference manual *User and Solution Administration*.
- The following text templates should be configured in the configuration tool Alfabet Expand for the email notifications sent when an assignment is created if a consistency monitor is triggered. For more information, see the section Configuring Text Templates for Email Notifications in the reference manual *Configuring Alfabet with Alfabet Expand*. See the chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix* for an overview of the available text templates for notification monitors and their permissible variables.
  - The text template `ConsistencyMonitorDefault` located in the text template folder ***M_CON*** is used for the assignments generated in the context of the consistency monitors. The relevant text template must be selected for a consistency monitor in the **Text Template** attribute in the **Consistency Monitor** editor. The text template is used to fill the **Description** attribute of the assignment. The name of the assignment is copied from the name of the consistency monitor. Please note that text templates for consistency monitors must be located in the text template folder ***M_CON***. The name of the text template folders must not be changed.
  - The text template `ConsistencyMonitorMail` ocated in the text template folder ***M_CON*** is used when an assignment is generated via batch process for an inconsistent object found by a consistency monitor. If a batch process is configured, the email notification will be sent to the authorized user of an object when a specified consistency is detected. For more information about setting up a batch process, see the section *Batch Processing for Monitors and Change Management with AlfaBatchExecutor.exe* in the reference manual *System Administration*.

Please note that text templates for consistency monitors must be located in the text template folder ***M_CON***. The name of the text template folders must not be changed.

The following must be configured in order to work with monitors in Alfabet:

- In order for any monitor to be executed and email notifications to be sent to relevant users, a batch process must be configured by your system administrator. For more information about setting up a batch process, see the section *Batch Processing for Monitors and Change Management with AlfaBatchExecutor.exe* in the reference manual *System Administration*.

- All Alfabet functionalities for which the email capability is to be implemented require the setup of a connection to an SMTP server for outgoing email in the tool Alfabet Administrator. For more information, see the section *Activating the Dispatch of Email Notifications in Alfabet* in the reference manual *System Administration*.
- Text templates for email notifications may be customized for all monitors in the configuration tool Alfabet Expand. For more information, see Configuring Monitors in the reference manual *Configuring Alfabet with Alfabet Expand*.
- For each monitor created, the monitor owner must define the **User Profile to Access Object in Notification** field in the monitor's editor in the relevant monitor functionality in the **Solution Admin** tab that may be accessed via an administrative user profile. This is described for each monitor type in the chapter *Configuring Monitors to Track Objects in Alfabet* in the reference manual *User and Solution Administration*.
- Access permissions must be available for the user profile so that relevant users receiving the email notification can access the relevant objects that are targeted by the hyperlink in the email notification. For more information about the configuration of access permissions, see the section Configuring Access Permissions for Alfabet in the reference manual *Configuring Alfabet with Alfabet Expand*.
- All object classes that are to be monitored via activity monitors and inactivity monitors have history tracking enabled in the configuration tool Alfabet Expand and therefore, the **Audit** attribute must be set to `True` for these object classes. Please note that this is not necessary for date, system date, consistency, and notification monitors. These monitors will be implemented even if the **Audit** attribute is set to `False` for the relevant object class. For more information, see the section Specifying History Tracking for an Object Class in the chapter Configuring the Class Model in the reference manual *Configuring Alfabet with Alfabet Expand*.

The following information is available:

Enabling the Monitoring Capability for Activity and Inactivity Monitors

In order to implement activity or inactivity monitors for an object class, you must carry out two configuration steps:

- The **Audit** attribute must be set to `True` for the object classes that are to be monitored by activity and inactivity monitors. For more information about the auditing functionality, see the section Specifying History Tracking for an Object Class in the chapter Configuring the Class Model.

- Monitor templates and monitored contexts must be enabled for activity monitors in the **Monitor Templates** node in the **Presentations** tab.
  - Monitor templates allows you to specify and enable the object classes for which activity and inactivity monitors can be created. All object classes that are configured for monitoring will be displayed in the **Monitor Type** field in the **Activity Monitor** editor and **Inactivity Monitor** editor.
  - Monitored contexts are available for activity monitors and allow users to monitor a specific context, such as the lifecycle or information flows associated with an object. All monitored contexts that are enabled will be displayed in the **Monitored Context** field in the **Activity Monitor** editor. Monitored contexts are preconfigured by Software AG.

To configure an object class for monitoring:

1) First, you should configure the relevant object class for the auditing capability. In the **Classes** folder in the **Meta Model** tab, click the protected class or public class that you want to implement the auditing functionality for.
2) In the attribute window, set the **Audit** attribute to `True`. If you select `False`, activity and inactivity monitors cannot be created for the selected object class. This does not impact date, system date, notification, or consistency monitors.
3) Next, you must enable the relevant monitor template for monitoring. For example, if you have set the **Audit** attribute to `True` for the object class `Application`, then you must now configure the monitor template `Application`. In the **Presentation** tab, expand the **Monitor Templates** node in the explorer, and click the relevant monitor template to open its attribute window.
4) In the attribute window, set the **Enabled** attribute to `True` to allow monitors to be created for the associated object class. If you select `False`, activity and inactivity monitors cannot be created for the selected object class.
5) Expand the monitor template to display its monitored contexts. For each monitored context that should be displayed in the **Monitored Context** field in the **Activity Monitor** editor, click the relevant monitored context to open its attribute window. Set the **Enabled** attribute to `True` to enable the selected monitored context for the monitor template or select `False` to disable the selected monitored context for the monitor template.
6) In the toolbar, click the **Save** button to save your changes.

Enabling the Monitoring Capability for Date Monitors

Monitor templates and monitored contexts must be enabled for date monitors in the **Monitor Templates** node in the **Presentations** tab. All object classes that are configured for monitoring will be displayed in the **Monitor Type** field in the **Date Monitor** editor. Monitor templates allows you to specify and enable the object classes for which date monitors can be created. Monitored contexts are

available for date monitors and allow users to monitor a specific context, such as the lifecycle or information flows associated with an object. Monitored contexts are preconfigured by Software AG.

To configure an object class for monitoring:

1) Go to **Presentation** tab, expand the **Monitor Templates** node in the explorer, and click the relevant monitor template to display its attribute window.
2) Set the **Enabled** attribute to `True` to allow monitors to be created for the associated object class. If you select `False`, date monitors cannot be created for the selected object class.
3) Enable the relevant monitored contexts nested below a monitor template. Expand the relevant monitor template to display its monitored contexts.
4) Expand the monitor template to display its monitored contexts. For each monitored context that should be displayed in the **Monitored Context** field in the **Date Monitor** editor, click the relevant monitored context to open its attribute window. Set the **Enabled** attribute to `True` to enable the selected monitored context for the monitor template or select `False` to disable the selected monitored context for the monitor template.
5) In the toolbar, click the **Save** button to save your changes.

Configuring Notification Monitors

A notification monitor is a type of monitor that allows email notifications to be automatically triggered based on configured Alfabet queries or native SQL queries. The queries specify the targeted objects and their object class properties as well as the users who shall be notified about the objects found by the queries.

Notification monitors are primarily configured in the **Notification Monitors** functionality that may be accessed via an administrative user profile. For more information about the creation and configuration of a notification monitor, see the section *Defining Notification Monitors* in the reference manual *User and Solution Administration*.

There are several aspects that must be configured in Alfabet Expand:

- The text templates used for the email notifications sent out when the monitor is triggered may need to be configured via the **Text Templates** node in Alfabet Expand.
- The queries used to find the relevant objects to monitor as well as the users to whom a notification is sent can be defined as an Alfabet query or a native SQL query. All AQL queries defined for notification monitors will be checked for correct syntax upon execution of the **Check All AQL Queries** functionality available in the **Meta-Model** menu in Alfabet Expand.
- Please note that the consolidation of emails for notification monitors must be specified in the XML attribute `ConsolidatedNotificationMail` in the XML object `SystemMonitorDefaultDef`. If set to `"true"`, one email will be sent to the relevant person regarding all objects triggering a notification. If set to `"false"`, a separate email will be sent for each object triggering the notification. The attribute is set to `"true"` per default.

The following information is available:

Creating and Defining a Text Template for Notification Monitors

In the **Notification Monitors** functionality in Alfabet, the creator of the notification monitor can either select a configured text template in the **Notification Monitors** editor or write an ad-hoc text for the notification directly in the **Notification Monitor** editor. If the creator of the notification monitor defines both a preconfigured text template and creates an ad-hoc text for the notification in the editor, then the ad-hoc text template definition will have precedence over the preconfigured text template selected in the editor.

Please note however that an ad-hoc text cannot be translated to other languages. If the Alfabet solution implemented in your enterprise is used for more than one language, then it is recommended that a text template is configured in Alfabet Expand so that additional language versions of the template can also be defined.

The default text template `NotificationMonitorDefault` is available in the `MON` folder for notification monitors. The default text template will be used for the generation of email notifications if no custom text template is defined and therefor no text template selected in the **Notification Monitor** editor, and if no ad-hoc text is written in the **Notification Monitor** editor. If the default text template `NotificationMonitorDefault` is used, the subject line of the email notification will be automatically generated and display Notification Monitor <NameOfMonitor>, whereby the name of the monitor is taken from the object class property `Name` defined for the notification monitor in the **Notification Monitor** editor in Alfabet. For more information, see the section *Defining Notification Monitors* in the reference manual *User and Solution Administration*.

All Alfabet functionalities for which the email capability is to be implemented require the setup of a connection to an SMTP server for outgoing emails. For more information, see the section *Activating the Dispatch of Email Notifications in Alfabet* in the reference manual *System Administration*. Furthermore, in order for email notifications to be sent in the context of the monitor functionality, your system administrator must configure a batch process. For more information about setting up a batch process, see the section *Batch Processing for Monitors and Change Management with AlfaBatchExecutor.exe* in the reference manual *System Administration*.

To create a custom text template for the notification monitors:

1) Go to the **Presentation** tab and expand the **Text Templates** folder by clicking the + symbol.
2) Right-click either the `MON` node and select **New Text Template**.
3) Click the new text template to display its attribute window. In the attribute window, define the technical name for the text template in the **Name** attribute.
4) Set the **Group** attribute to `M_NOTE`. This is the folder in which you will store text templates for notification monitors.
5) For the **Caption** attribute, enter text for the subject line for the email message.
6) You now need to create the email message for the new text template. To do so, you should copy an existing text template including its text and the variables and paste it in the new text template. You can then modify the text and cut and paste the variables, as needed.

- To copy the text of a standard text template for workflows, select the relevant text template and click the **Browse** button available for the **Text** attribute. Select the text and right-click and select **Copy**.
- To paste the text to the new text template, select the new text template and click the **Browse** button available for the **Text** attribute. Right-click in the editor and select **Paste**.

7) Edit the text, as described above. See the chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix* for an overview of the available text templates for notification monitors and their permissible variables.

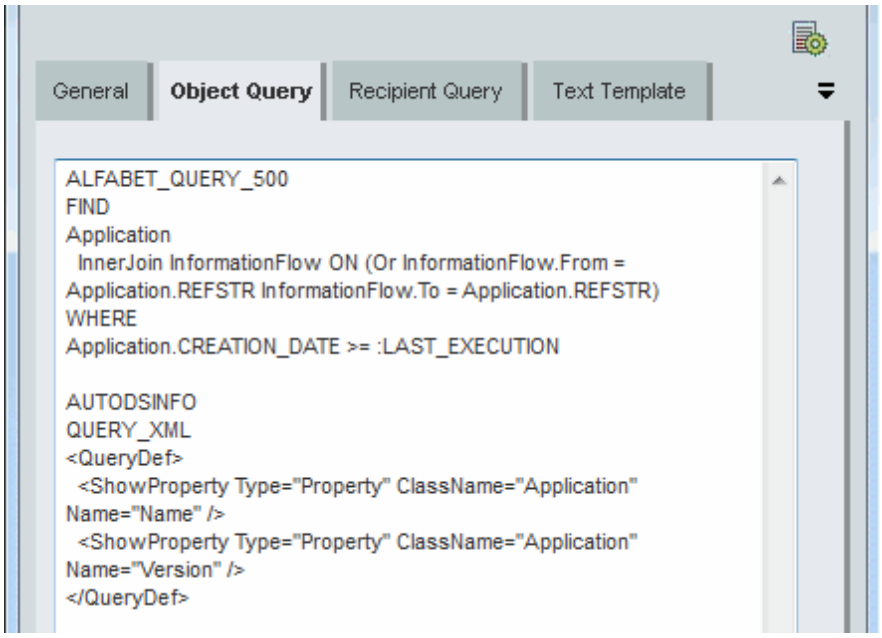
8) If other language versions need to be defined for the text template, create a local language version by right-clicking the new text template and selecting **Create Locale Text**. For more information, see the section Creating a Translation of a Text Template.

9) In the toolbar, click the **Save** button to save your changes.

Configuring the Queries Used in Notification Monitors

Notification monitors are created and defined in the **Notification Monitors** functionality in order to define objects to be monitored and notifications to be sent based to relevant users. This is achieved by defining rules to find 1) the objects that are the target of the monitor as well as 2) the persons to whom the notification email. The persons identified to receive the notification could be the authorized users of the target objects, persons having roles defined for the target objects, or persons who are not directly associated with the target object.

In the **Notification Monitor** editor in which notification monitors are created and edited, the user defining the monitor must enter a relevant Alfabet query or SQL query to find the monitored objects as well as a relevant Alfabet query or SQL query to find the persons to whom the notification should be sent.



A valid Alfabet query or native SQL query must be defined in the **Object Query** tab to find the objects for which the monitor shall be executed. Please keep the following in mind:

- The `FIND` class of the Alfabet query should be the object class that shall be monitored.
- `JOIN` and `WHERE` clauses may be used to define a subset of objects of the base class to be evaluated.
- The Show properties defined in the Alfabet query determine the information displayed about the objects in the **Objects** page view of the notification monitor.
- All Alfabet queries defined for notification monitors will be checked for correct syntax upon execution of the **Check All AQL Queries** functionality available in the **Meta-Model** menu in Alfabet Expand.
- When defining a native SQL query, the first `SELECT` property must return the values of the `REFSTR` property of the selected object. All following `SELECT` properties define the information displayed about the objects in the **Objects** page view of the notification monitor.
- Both Alfabet queries and native SQL queries may use the variable `LAST_EXECUTION` to compare date information of the objects with the last execution date for the monitor.
- For information about defining Alfabet queries and about the rules that apply to the definition of native SQL queries in the context of Alfabet, see Defining Queries. For more information about reviewing query syntax, see the section Testing Queries for Compliance with the Current Release.

A valid Alfabet query or native SQL query must be defined in the **Recipient Query** tab to find the users to whom the notification shall be sent. Please keep the following in mind:

- The `FIND` class of the Alfabet query should be the object class Person.
- The relation to the object class for which objects are to be evaluated must be defined via `JOIN`s and `WHERE` clauses.
- The query may use the parameter `BASE` to specify that the recipient query is executed for each of the objects found by the object query.
- The `REFSTR` property of the base class `PERSON` and of the object class of the objects to be monitored by the person found by the Alfabet query must be included in the show property definition of the Alfabet query.
- When defining a native SQL query, the first `SELECT` property must return the values of the `REFSTR` property of objects of the object class `PERSON`.

Configuring Assignments for System-Wide Date Monitors

System-wide date monitors are time-triggered monitors defined for an object class on a system-wide basis. When a specified date (usually a start, end, or target date) approaches, users responsible for objects in that object class will automatically be sent an assignment and notification per email asking them to review the object. System-wide date monitors can be configured for a specific object class in the **System Date Monitors** functionality that may be accessed via an administrative user profile. The XML object ***SystemMonitorDefaultDef*** allows you to configure the assignments generated in the context of system-wide date monitors.

- For more information about system-wide date monitors, see the section *Defining System Date Monitors* in the reference manual *User and Solution Administration*.

- Software AG provides a variety of monitors including another type of date monitor that can be configured to monitor specified objects in an object class. For more information about the available monitors in Alfabet, see *Keeping Track of Objects via Monitors* in the reference manual *User and Solution Administration*.

The following must also be configured before the system date monitor functionality can be implemented:

- Monitored contexts are available for date monitors and allow users to monitor a specific context, such as the lifecycle or information flows associated with an object. All monitored contexts that are enabled will be displayed in the **Monitored Context** field in the **System Date Monitor** editor. Monitored contexts are preconfigured by Software AG . To specify the monitored contexts that should be available for the system date monitors, expand the monitor template to display its monitored contexts. For each monitored context that should be displayed in the **Monitored Context** field in the **System Date Monitor** editor, click the relevant monitored context to open its attribute window. Set the **Enabled** attribute to `True` to enable the selected monitored context for the monitor template or select `False` to disable the selected monitored context for the monitor template.
- The text displayed in automatically-generated email notifications is also configured by your solution designer in the configuration tool Alfabet Expand. For more information, see the section Configuring Text Templates for Email Notifications. See the chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix* for an overview of the available text templates for date monitors and their permissible variables.
- All Alfabet functionalities for which the email capability is to be implemented require the setup of a connection to an SMTP server for outgoing emails. For more information, see the section *Activating the Dispatch of Email Notifications in Alfabet* in the reference manual *System Administration*.
- In order for email notifications to be sent in the context of the monitor functionality, your system administrator must configure a batch process. For more information about setting up a batch process, see the section *Batch Processing for Monitors and Change Management with AlfaBatchExecutor.exe* in the reference manual *System Administration*.

To edit the XML object ***SystemMonitorDefaultDef***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder.
2) Right-click the XML object ***SystemMonitorDefaultDef*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) Define the XML attributes, as needed. The table below displays the XML elements and XML attributes that can be edited for the XML object ***SystemMonitorDefaultDef***:

XML Element (bold) / XML Attribute	Explanation
SystemMonitorDefaultDef	
`Assignee`	Enter the user name of the default user that should receive assignments associated with an object if the impacted object has no authorized user defined.
`Originator`	Enter the user name of the default user that is the creator of the automatically-generated assignment. This user would have to follow-up on assignments that have not been completed by the target date and have Therefore, been returned as the result of the assignment's expiration.
`CountdownReviewTime`	Enter an integer to establish the target date when the assignee should complete the review/update of the relevant object. The target date is computed based on the date when the assignment is generated plus the value defined in the `CountdownReviewTime` attribute.  If the `CountdownReviewTime` is defined as 30, the assignee will have 30 days after the date that the assignment was generated to complete the assignment. An assignment generated on Jan. 1, 2008 will have a target date of Jan. 30, 2008.

4) In the toolbar, click the **Save**  button to save the XML definition.

Configuring the Diagram Capability

In Alfabet Expand you can configure custom diagram item templates to use for the display of diagram items in diagrams. Furthermore, you can configure custom diagrams that allow users to design and view diagrams that visualize objects and references between objects that are not addressed by the standard diagrams.

You can also configure the visualization of connection items such as information flows in standard diagrams as well as the default settings for the Alfabet Diagram Designer.

The following information is available:

Configuring Custom Diagram Item Templates

In Alfabet, diagrams are used to visualize objects in the IT landscape and their relationships. For example, an application diagram allows your enterprise to design a landscape diagram showing the application groups, applications, local components, peripherals and information flows relevant to a selected application. From a technical perspective, each application group, application, local component, peripheral and information flow displayed in the diagram is a diagram item. All diagram items are based on diagram item templates, which determine the visualization of the diagram item for the relevant object class. For example, standard diagram item templates for applications are rectangles that are colored powder blue, show the standard icon for the object class Application, and display a standard set of attributes. Diagram item templates are available for all object classes that are visualized in diagrams, but your enterprise can configure custom diagram item templates to use in place of or in addition to the standard diagram item templates.

Custom diagram templates allow you to configure alternative visualizations of diagram item templates. The custom diagram item templates you create in Alfabet Expand are based on selected preconfigured diagram item templates that are relevant for a specific type of diagram such as an application diagram or platform diagram. You can then modify the custom diagram item template as needed. For example, you can change the color, shape, borders, and size of the custom diagram item template. You can alter the attributes displayed on the diagram view item or add additional text. Furthermore, you can change the icon displayed on the diagram item template and even design it so that it appears to consist of an icon only or another shape rather than a rectangle.

The custom diagram item templates you configure can also be assigned to diagram view items in the **Configuration** functionality. The custom diagram item template can be selected in the **Diagram Item Template** field in the **Diagram View Item** editor. For more information, see the section *Configuring Diagram Views for Diagrams* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.

The following information is available:

- Creating a Custom Diagram Item Template
- Configuring the Color, Border, and Static Text of the Custom Diagram Item Template
- Configuring Attributes or Text to Display on the Custom Diagram Item Template
- Configuring Icons to Display on the Custom Diagram Item Template
- Configuring a Connection Item as a Custom Diagram Item Template
- Configuring a Different Shape for the Custom Diagram Item Template

Creating a Custom Diagram Item Template

When you create a custom diagram item template, you must base it on a selected preconfigured diagram item template that is relevant for a specific type of diagram such as an application diagram or platform diagram. A selector provides information about the available diagram item templates in order to help you decide which to use. Although each diagram item template is associated with a type of diagram, not every diagram item template is necessarily associated with one specific object class. For example, the diagram item template Application always represents applications, and their object class stereotypes in application diagrams whereas the diagram item template Migration Node can represent applications, ICT objects, or solution building blocks in migration diagrams.

You can create one or more custom diagram item templates per standard diagram item template. This could be especially relevant if object class stereotypes have been configured for an object class. All custom

diagram item templates you create will be available in the **Toolbox Items** pane for the relevant diagram in the Alfabet Diagram Designer and the user designing the diagram can choose which standard or custom diagram item template to use for each individual object that is to be visualized in the diagram. In the case of application diagrams, for example, the diagram designer could add applications based on different custom diagram item templates that represent different application stereotypes to the application diagram.

When you first create a custom diagram item template, it is displayed per default as a rectangle in the center editor pane in the **Diagrams** tab of Alfabet Expand. The standard icon and attributes predefined for the selected diagram item template will be displayed per default on the custom diagram item template.

To create a custom diagram item template:

1) Go to the **Diagrams** tab and expand the **Diagram Tools** node.
2) Right-click the **Custom Diagram Item Templates** node and select **Create New Custom Diagram Item Template**. The **Diagram Item Template Selector** opens. This selector displays all standard diagram item templates  that you can base a custom diagram item template on. The **Description** column provides general information about each diagram item template in order to help you decide which diagram item template to use.
3) In the **Diagram Item Template Selector** editor, select the diagram item template that is relevant for the custom diagram item template that you want to configure and click **OK**. The new custom diagram item template is displayed in the center editing pane.
4) Right-click the new custom diagram item template node and select **Design Diagram Item Template**.
5) In the attribute window on the right, enter a technical name for the custom diagram item template in the **Name** attribute.

   Please keep the following in mind:

   - The technical name may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.
   - The technical name should only contain standard `ASCII` characters.
   - The technical name should consist of upper-case letters only.
   - The maximum length of a technical name may not exceed 30 characters.
   - The technical name may not coincide with any of the reserved key words of the relational database management system.

   A validation mechanism checks for correct syntax when defining a technical name. Please note that if the technical name of a configuration object is changed, the name will be correctly updated in other configuration objects referencing the changed object during design time. However, the name of a changed configuration object will not be updated in guide pages nor if the guide pages reference a configuration object listed in the **Show Usage** functionality. If you plan to change the name of a configuration object, the reference in the guide page should be changed prior to changing the configuration object name.

6) In the **Description** attribute, amend the standard description of the diagram item template as needed to provide information about the configuration of the custom diagram item template. The description is only visible in Alfabet Expand.

7) To define the size of the custom diagram item template, change the default value by entering an integer in the **Height** and **Width** attributes.

8) To define the visualization of the icon representing the custom diagram item template:

   - In the **Caption** attribute, enter a caption for the custom diagram item template.
   - In the **Icon** attribute, select an icon for the custom diagram item template.
   - In the **Icon Background Color** attribute, select the background color for the icon for the custom diagram item template.
   - In the **Layout Type** attribute, select `Scalable` if the background shapes should be scaled based on the sizing of the custom item. This may be necessary if the diagram item template shall be used as a custom shape used to define the background of a diagram.
   - In the **Hint** attribute, modify the text to provide information to users about the purpose of the custom diagram item template. The initial text in the **Hint** attribute is copied from the base diagram item template. A tooltip containing the hint text will be displayed when the user points to the icon of the custom diagram item template in the **Toolbox Items** pane.

9) To specify that the diagram items in the Alfabet interface are updated if changes are made to the custom diagram item template they are based on, set the **Keep Original Shape Rendering** attribute to `True`. To specify that the diagram items in the Alfabet interface are not updated if changes are made to the custom diagram item template, set the **Keep Original Shape Rendering** attribute to `False`. In this case, the user designing the diagram can manually apply the changes to the diagram item by selecting the **Other Actions** > **Reset Original Shape** option in the Alfabet Diagram Designer. Please note that if the **Keep Original Shape Rendering** attribute is changed from `False` to `True`, the update will only apply to the diagram items that have been created while the **Keep Original Shape Rendering** attribute was set to `True`. In other words, the current setting of the value of the **Keep Original Shape Rendering** attribute is passed on to the diagram item.

10) To specify that the custom diagram item template is not displayed in standard diagrams implementing the standard diagram item template that the custom diagram item template is based on, set the **Show In Toolbox of Base Item** to `False`. If the custom diagram item templates should be displayed in standard diagrams, select `True`.

11) In the toolbar, click the **Save**  button to save the custom diagram item template.

Configuring the Color, Border, and Static Text of the Custom Diagram Item Template

A new custom item template will originally be displayed with the default configuration of the diagram item template that it is based on. The color and border can be changed as needed. Optionally, static text can be added that will be displayed on all diagram items based on the custom diagram item template.

To configure the size, color, and borders of a custom diagram item template:

1) Right-click the custom diagram item template node  and select **Design Diagram Item Template**.

2) To define the color attributes for the custom diagram item template, expand the **Background Shape** section of the attribute window and define the following:

- For the **Back Color** attribute, select a color for the custom diagram item template. Please note that a different color can be assigned to any diagram item in the context of the Alfabet Diagram Designer.
- Set the **Back Brush Type** attribute to `Solid` if the custom diagram item template should display no color gradation, `AutoVerticalGradient` if the custom diagram item template should display a vertical color gradation, or `AutoHorizontalGradient` if the custom diagram item template should display a horizontal color gradation.
- If the user designing the diagram item should determine the color and border, set the **Use Background Shape** attribute to `False`. The color and borders of any diagram item based on the selected custom diagram item template will be transparent and the user designing the diagram can define the color and border of each individual diagram item. If the **Use Background Shape** attribute to `True`, the attributes defined in the **Background Shape** section of the attribute window will be applied when diagram items are added to the diagram.

3) To define the borders of the custom diagram item template:
   - Set the **Rounded Border** attribute to `False` if the custom diagram item template should have square corners. Select `True` if the custom diagram item template should have rounded corners. If the **Rounded Border** attribute is set to `True`, enter an integer in the **Round Size** attribute to specify how large the curves of the rounded corners are. A small number such as 2 draws minimally rounded corners and a large number such as 10 draws significantly rounded corners.
   - Set the **Double Border** attribute to `False` if the custom diagram item template should have a single border. Select `True` if the custom diagram item template should have a double border.
   - Expand the **Pen Attributes** section and define the color, style, and width of the borders in the **Color**, **Style** and **Width** attributes.
4) To define static text that is displayed on the custom diagram item template, enter text in the **Text** attribute. Expand the **Text Attributes** section to define the position, color, size, and style of the text displayed in the custom diagram item template. Please note that the font is determined by the configuration of the ***AlfaGUIScheme*** object. For more information, see the chapter Configuring the Visualization of the Alfabet Interface.
5) In the toolbar, click the **Save**  button to save the custom diagram item template.

Configuring Attributes or Text to Display on the Custom Diagram Item Template

Each diagram item template is preconfigured with one or more rectangle shapes that allow you to display an object's attributes or text. The diagram item template that you base the custom diagram item template on will typically have one or more rectangle shapes on it with predefined attributes. You can modify, add, or delete the existing rectangles.

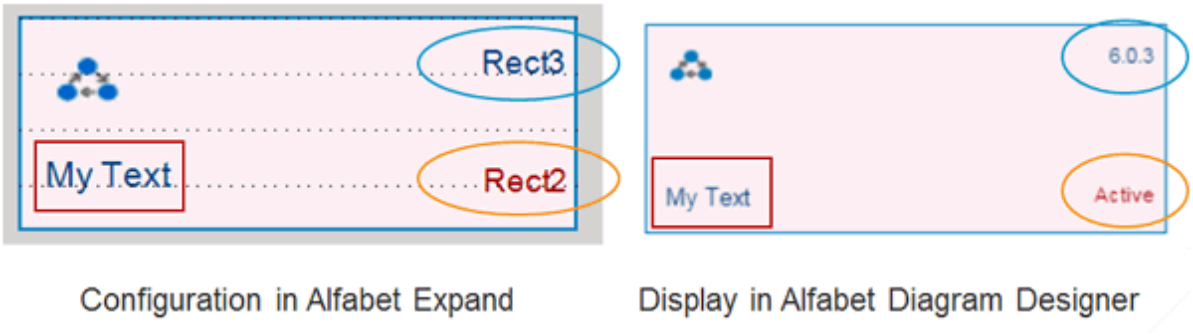


FIGURE: Custom diagram item template with a text and two attribute values

When you create a custom diagram item template, the predefined rectangles will be displayed with the labels `Rect1`, `Rect2`, `Rect3`, etc. Each rectangle with the label `Rect...` is predefined to display an attribute value. Click the rectangle to view its predefined attribute in the **Control Content Into** field. This can be changed as needed.

Per default, the predefined rectangles associated with the diagram item template are transparent and have a transparent border so that only the rectangle content (attribute or text) will be displayed. New rectangles added to the custom diagram item template will be white with borders, per default. You can change this and display the rectangles with color and borders, if necessary. Additionally, you should ensure that the size of the rectangle is large enough so that the content that will be displayed will not be truncated.

To add a rectangle to the custom diagram item template or specify the visualization or content of the custom diagram item template:

1) Right-click the custom diagram item template node and select **Design Diagram Item Template**. The menus and toolbars relevant for configuring custom diagram item templates are displayed.

   If the menus and toolbars are not displayed in Alfabet, you can either right-click the custom diagram item template and select **Design Diagram Item Template** or click the custom diagram item template displayed in the design editor in the center pane.

2) To add a new rectangle to the custom diagram item template, click the **Rectangle** button and click in the custom diagram item template. You can define the following to modify the rectangle:

   - Drag the handles to resize the rectangle. You must ensure that the size of the rectangle is large enough for the content that could be displayed in the diagram item. If the rectangle is not large enough, the content may be truncated.
   - Click-and-drag the rectangle to move it to another location in the custom diagram item template. It is advised that you click the rectangle and define the **Anchor** attribute in order to dock the rectangle to the defined borders if a user resizes the custom diagram item template in the Alfabet interface.
   - To define a color for the rectangle, click the rectangle and set **Back Color** to `Transparent` if the rectangle should have no color or select a color.
   - To define the rectangle border, click the rectangle and expand the **Pen Attributes** section. For the **Color** attribute, select a color for the border or select `Transparent` if the rectangle should have no border. If you have selected a color for the border, define the border style in the **Style** attribute and the width of the border in the **Width** attribute. To further specify the border, you can also define the attributes **Rounded Box**, **Round Size**, and **Shadow**.

3) To specify the content of the rectangles displayed on the custom diagram item template, click the rectangle and define the following in the attribute window.

   - To display an attribute value in the rectangle shape, set the **Content Type** attribute to `PropertyValue`. Enter the name of the standard or custom object class property in the **Control Content Into** attribute. You must manually enter the name of the standard or custom attribute that should be displayed. The name of the standard or custom object class property must be spelled exactly as it is defined in the **Name** attribute of the relevant property. If no attribute value has been defined for the object, then nothing will be displayed in the rectangle shape.

     Please note that because the diagram item template Migration Node can represent applications, ICT objects, or solution building blocks in migration diagrams, you must specify the **Control Content Into** attribute only with object class properties that are common to the object classes Application, ICT Object, or Solution Building Block (for example, `Status` or `Version`). Otherwise, an error may occur.

   - To display a text in the rectangle shape, set the **Content Type** attribute to `PlainText`. Enter the text to display in the **Text** attribute. You must manually enter the name of the standard or custom attribute that should be displayed. You must spell the name of the standard or custom object class property correctly.

   - To display text if no attribute has been defined n the rectangle shape, set the **Content Type** attribute to `ConditionalText`. Enter the name of the standard or custom object class property in the **Control Content Into** attribute. You must manually enter the name of the standard or custom attribute that should be displayed. You must spell the name of the standard or custom object class property correctly. Enter the text to display in the **Text** attribute if no attribute value has been defined for the object. If an attribute value is defined it will be displayed in the diagram element, but if no attribute value has been defined for the object, then the configured text will be displayed in the diagram item.

   - To display a value that is assigned to the diagram item template via a diagram view in a configured diagram view report, set the **Content Type** attribute to `QueryValue`. For information about the definition of diagram views including the use of diagram item templates in the diagram views, see Creating Custom Diagram Views for a Custom Diagram View Report.

4) Specify the opacity of the attribute value or text in the rectangle in the **Opacity** attribute by entering a number between 0 and 1 (for example, 0.5). The lower the number, the more transparent will be the attribute value or text.

5) In the toolbar, click the **Save**  button to save the configuration.

Configuring Icons to Display on the Custom Diagram Item Template

Each diagram item template is preconfigured with a standard icon representing the object class for which the custom diagram item template is typically implemented. This icon can be changed, additional icons can be added, or the entire custom diagram item template can be displayed as an icon only with no text, attribute values, or borders, etc.

Depending on your solution configuration, the icon defined for the custom diagram item template might not be displayed in the Alfabet interface:

- An icon configured for a relevant class setting may overrule the icon configured for the custom diagram item template. For more information about the configuration of class settings, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.
- An icon defined for an object in the object's editor in the Alfabet interface may overrule the icon configured for the class setting and the custom diagram item template.

To ensure that the icon defined for the custom diagram item template overrules any icon defined for the relevant class setting or for an object, the **Icon Type** attribute must be set to `Manual`.

To specify the icon displayed on the custom diagram item template, click the icon and define the following in the attribute window:

1) Right-click the custom diagram item template node and select **Design Diagram Item Template**. The menus and toolbars relevant for configuring custom diagram item templates are displayed.

   If the menus and toolbars are not displayed in Alfabet, you can either right-click the custom diagram item template and select **Design Diagram Item Template** or click the custom diagram item template displayed in the design editor in the center pane.

2) To add a new icon to the custom diagram item template, click the **Icon** button and click in the custom diagram item template, or to modify an existing icon, click the icon in the custom diagram item template to open the attribute window.
3) Specify the **Icon Gallery** attribute by selecting the icon gallery containing the icon you want to display on the custom diagram item template. The icon gallery you select here will determine which icons are available for the **Icon** attribute. `Small` are icons that are 22x22 pixel in size, `Large` are icons that are 36x36 pixel in size, and `Free` are icons of an arbitrary size. For more information about adding an icon to an icon gallery, see the section in the section Adding and Maintaining Icons for the Alfabet Interface in the chapter Configuring the Visualization of the Alfabet Interface.
4) Specify the **Icon** attribute by selecting the icon that should be displayed on the custom diagram item template in the diagram. The drop-down menu will display all icons available in the icon gallery specified in the **Icon Gallery** attribute.
5) Set the **Icon Type** attribute to `Manual` to ensure that the icon defined in the **Icon** attribute overrides any other icon definition. Select `Object` if the icon defined for the class setting or the icon defined in the object's editor in the Alfabet interface should override the icon selected in the `Icon` attribute.

   Please note that because the diagram item template **Migration Node** can represent applications, ICT objects, or solution building blocks in migration diagrams, you should set the **Icon Type** attribute to `Object` so that the icon displayed on the custom diagram item template is determined by the class setting for migration nodes.

   If the **Icon Type** attribute has been set to `Object` and the diagram item template shall display an object of the class `AlfaMMClassInfo`, you must define the following in order to display the icon of the object class that the custom diagram template shall represent in the diagram:

   - **Control Type**: Select `PropertyValue`.

- **Content Control Info**: Enter `Icon` to display the icon defined in the **Icon** attribute of the object class of the object that shall be displayed.

6) Specify the **Anchor** attribute by selecting one or more edges to lock the icon to. It is recommended that you anchor the icon to the frame of the custom diagram item template so that it is correctly positioned if the diagram item is resized in the Alfabet Diagram Designer.

7) Specify the opacity of the icon in the **Opacity** attribute by entering an integer between 0 and 1 (for example, 0.5). The lower the number, the more transparent will be the icon.

8) In the toolbar, click the **Save** button to save the configuration.

Configuring a Connection Item as a Custom Diagram Item Template

A diagrams, connection items are typically drawn as lines with arrows between two objects. A connection item could be, for example, an information flow between applications or between components in a platform, a migration rule, which describes the migration steps from one application to another, or a role, which describes the functional relationship between an object such as an application and a person or organization.

1) Go to the **Custom Diagram Item Templates** node, create the custom diagram item template based on a diagram item template representing a connection item (for example, migration rule, role, or any type of information flow) and define the basic attributes for the custom diagram item template as described in the section Creating a Custom Diagram Item Template.

2) Right-click the custom diagram item template node and select **Design Diagram Item Template** to display its attribute window.

3) Specify the **Back Color** attribute by selecting a color for the custom diagram item template. Please note that a different color can be assigned to any diagram item in the context of the Alfabet Diagram Designer.

4) Specify the opacity of the connection item in the **Opacity** attribute by entering an integer between 0 and 1 (for example, 0.5). The lower the number, the more transparent will be the connection item.

5) Expand the **Start Arrow** section of the table and **End Arrow** section and define the following for the start and end arrows:

   - **Style**: Specify the shape on the start of the connection item.
   - **Size**: Specify the size of the start arrow.

6) Expand the **Pen Attributes** section and define the color, style, and width of the borders in the **Color**, **Style** and **Width** attributes.

   - **Color**: Specify the color of the connection item.
   - **Style**: Specify the line style of the connection item.
   - **Width**: Specify the line width in pixel of the connection item.

7) In the context of the Alfabet interface, users can display attributes on the connection items. The position, color, size, and style of the text visualizing these attributes can be configured for the custom diagram item template. Expand the **Text Attributes** section to define the position, color, size, and style of text displayed on the custom diagram item template.

The attributes that can be displayed on connection items is configured in the XML object ***DiagramInformationFlowDef***. For more information about configuring the attributes to display on connection items, see the section Configuring Attributes or Text to Display on the Custom Diagram Item Template The font is determined by the configuration of the ***AlfaGUIScheme*** object. For more information, see the chapter Configuring GUI Scheme Definitions for the Alfabet Interface in the chapter Configuring the Visualization of the Alfabet Interface.

8) In the toolbar, click the **Save** button to save the custom diagram item template.

Configuring a Different Shape for the Custom Diagram Item Template

Instead of displaying the default rectangle shape that most custom item diagram templates are based on, you can use any shape displayed in the toolbar for the custom diagram item template. It is important to understand that the default custom diagram item template is configured to be transparent in the editor pane and the shape is placed within the boundaries of the transparent custom diagram item template.

Custom shapes can also be implemented in custom diagrams. The custom diagram item template will be displayed in the **Diagram Standard Shapes Selector** when a new shape is created via the **Shapes** node in the context of the **Custom Diagram Definitions** node. For more information, see the section Adding Shapes to the Custom Diagram.

1) Go to the **Custom Diagram Item Templates** node, create the custom diagram item template based on a diagram item template representing a connection item (for example, migration rule, role, or any type of information flow) and define the basic attributes for the custom diagram item template as described in the section Creating a Custom Diagram Item Template.

2) Right-click the custom diagram item template node and select **Design Diagram Item Template**. The menus and toolbars relevant for configuring custom diagram item templates are displayed.

3) Change the **Height** and **Width** attributes so that the shape will fit inside the template in the center pane.

4) Remove the color of the rectangle and border and content from the custom diagram item template displayed in the center pane:

   - Set the **Back Color** attribute to `Transparent` for the custom diagram item template.
   - Expand the **Pen Attributes** section and specify the following:
     - **Color**: Specify `Transparent` for the custom diagram item template.
     - **Style**: Specify the line style of the custom diagram item template.
     - **Width**: Specify the line width in pixel of the custom diagram item template.
   - Delete the rectangles and icon from the custom digram item template.

5) Ensure that the menus and toolbars relevant for configuring the custom diagram item template is displayed. To do so, right-click the custom diagram item template node and select **Design Diagram Item Template**.

6) To add a new shape to the custom diagram item template, click the relevant shape button in the toolbar and click in the custom diagram item template. Please note that the shape must fit inside the boundaries of the diagram item template.

7) Click the shape and configure the attributes for the color and border the shape as described in the section Configuring the Color, Border, and Static Text of the Custom Diagram Item Template. Please note that a different color can be assigned to any diagram item in the context of the Alfabet Diagram Designer.

8) Click the shape and add rectangles to configure attributes and static text as described in the section Configuring Attributes or Text to Display on the Custom Diagram Item Template.

9) Click the shape and add an icon as described in the section Configuring Icons to Display on the Custom Diagram Item Template.

10) In the toolbar, click the **Save** button to save the custom diagram item template.

Configuring Custom Diagrams

In addition to the standard diagrams available in Alfabet, you can configure custom diagrams that allow users to design and view diagrams that visualize objects and references between objects that are not addressed by the standard diagrams. For example, a custom diagram could be configured to display standard platforms and applications with connection items that represent the references between the standard platforms and applications. The complete infrastructure available for standard diagrams including filters and toolbar buttons will also be available for custom diagrams in the Alfabet user interface. Users with relevant access permissions to the Alfabet Diagram Designer can edit the custom diagrams.

The configuration of a custom diagram requires the definition of a custom diagram definition and a configured report that is specified to handle the diagram functionalities for the custom diagram definition:

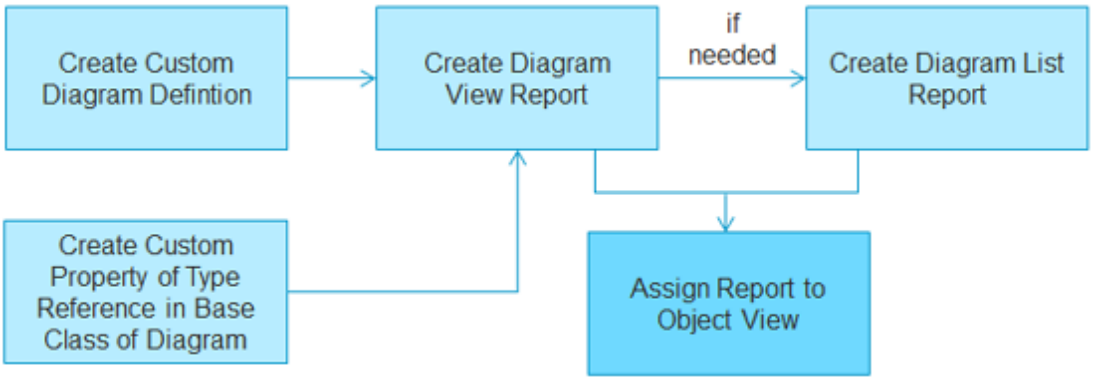


FIGURE: Sequence of configuration steps for custom diagram definition

- The custom diagram definition defines which object classes, connections, toolbox items, and graphic elements are available to the user when designing the diagram. It is the basis for the definition of the diagram that can be designed in the Alfabet Diagram Designer. A new custom diagram definition must be based on an existing standard diagram definition but can be modified as needed. The custom diagram definition includes a set of nodes and connection items that represent the object classes that may be visualized in the diagram, toolbox items that allow objects and connection items to be added to the diagram as well as buttons providing various graphic shapes to help design the

appearance of the diagram. New nodes, connections, toolbox items, and buttons can be created and those provided with the standard diagram definition may be removed as needed.

- A configured report must be configured as the framework to display the diagram in the Alfabet user interface. The report includes filters and the button to open the Alfabet Diagram Designer and displays the diagram that has been designed in the Alfabet Diagram Designer based on the custom diagram definition. When a user opens the Alfabet Diagram Designer via the configured diagram view report and creates a diagram based on a custom diagram definition, the diagram is saved as an object of the object class `ReportDiagram`. This object class stores the content of the diagram together with the information about the custom diagram definition that the diagram is based on as well as the object the diagram is created for.
  - There are two different configuration options that allow users to create diagrams in the Alfabet user interface. You can either define the configured report so that users can create the diagram as the main standard diagram for the object or you can create a configured diagram list report so that users can create multiple diagrams for the object.
  - If a diagram shall be created as the main standard diagram for the object, a custom property of the type `Reference` must be created for the base class for which the diagram shall be created in order to link to the object of the report diagram.
- The reports may be assigned to an object profile or object cockpit so that users can access the diagram view report and design and view the custom diagram. The reports will also be available via the **Configured Reports** functionality. For more information, see the section Integration of Configured Reports in the Alfabet Interface.
- The protected enumeration `CustomDiagramViewType` is available for the definition of diagram views displayed in the context of custom diagrams. The enumeration items defined for the `CustomDiagramViewType` enumeration can be selected in the **Type** field in the **Diagram View** editor. The query specified for the `DiagramView` filter field for the `DiagramViewReport` report must refer to the relevant standard or custom diagram view type. The diagram views based on the standard or custom diagram view type can selected in configured reports based on the `DiagramViewReport` report template. For more information about configuring protected enumerations, see the section *Overview of Protected Enumerations* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*. For more information about configuring diagram views, see the section *Configuring Diagram Views for Diagrams* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.

The following information is available:

- Configuring the Custom Diagram Definition
  - Creating a Custom Diagram Definition
  - Creating Nodes for the Custom Diagram
  - Creating Node Groups for the Custom Diagram
  - Creating Connections for the Custom Diagram
  - Creating a Connected Diagram Definition for the Custom Diagram
  - Creating Toolbox Items for the Custom Diagram
  - Adding Shapes to the Custom Diagram
  - Defining Rules to Automatically Add Objects to New Diagrams

Configuring the Custom Diagram Definition

A new custom diagram definition must be based on an existing standard diagram definition that is then modified. The custom diagram definition includes a set of nodes and connection items that represent the object classes that may be displayed in the diagram, toolbox items that allow objects and connection items to be added to the diagram as well as buttons providing various graphic shapes to help design the visualization of the diagram. New nodes, connections, toolbox items, and buttons can be created and those provided with the standard diagram definition may be removed as needed.

The following configuration is required for the custom diagram definition:

- A node or connection must be defined for every object class that may be visualized in the custom diagram. A node represents an object class such as application, component, information flow, etc. that may be displayed in the diagram. It is especially important to note that the definition of the node or connection does not implicate that object can be created in or explicitly added to the diagram. Some classes may be automatically included in the diagram via their relationships to other objects in the diagram (such as information flows between applications) or when functionalities such as **Add Subordinates** or **Generate Network** are executed in the Alfabet Diagram Designer. Therefore, you must create a node for all relevant object classes that may be visualized in the diagram including those specified via connections and node groups.
- A node group must be specified if a semantic relationship between the node and its referenced objects are to be displayed in the diagram. Node groups are visualized in the diagram as nested diagram items. Nested diagram items are typical, for example, when applications are visualized in the application groups that own them or local components are visualized in the diagram items of the applications that own them.
- A connection must be specified for each connection item that may be visualized in the custom diagram. The definition of the connection node may be either based on a semantic class (such as an `InformationFlow`, `ValueArc`, or `DemandArch`) that reference a source and target object, or a property of type `ReferenceArray` such as `Device.Networks` or `Network.BelongsTo`.
- A toolbox item must be specified for every node and connection item that shall be created, created as a copy, or explicitly added to the custom diagram by the Alfabet user designing the diagram in the Alfabet Diagram Designer. All toolbox items will be available in the top section of the **Toolbox Items** pane in the Alfabet Diagram Designer.
- A shape is required for each graphic element that shall be available to design the diagram. The shapes that may be included in the custom diagram are either preconfigured by Software AG or are

based on configured diagram item templates. All shapes added to the diagram item template will be available in the bottom section of the **Toolbox Items** pane in the Alfabet Diagram Designer.

The following information is available:

- Creating a Custom Diagram Definition
- Creating Nodes for the Custom Diagram
- Creating Node Groups for the Custom Diagram
- Creating Connections for the Custom Diagram
- Creating a Connected Diagram Definition for the Custom Diagram
- Creating Toolbox Items for the Custom Diagram
- Adding Shapes to the Custom Diagram
- Defining Rules to Automatically Add Objects to New Diagrams
  - Defining a Category for the Rules in the XML Object UseCaseCategories
  - Defining the Query for the Rule in a Configured Report
  - Adding the Rule to the Custom Diagram Definition

Creating a Custom Diagram Definition

A new custom diagram definition must be based on an existing standard diagram definition. You should select a standard diagram that already includes some of the object classes that you want to visualize in the diagram. The standard diagram can then be further modified.

To create a custom diagram definition:

1) Go to the **Diagrams** tab and expand the **Diagram Tools** node.
2) Right-click the **Custom Diagram Definitions** node and select **Create New Custom Diagram Definition**.
3) In the **Diagram Definition Selector**, select the standard diagram that the custom diagram definition shall be based on and click **OK**.
4) Click the custom diagram definition  and in the attribute window and define the following attributes:
   - **Name**: Enter a technical name for the custom diagram definition.
   - **Caption**: Specify the caption of the custom diagram. The caption will be displayed in the Alfabet user interface.
   - **Group**: Enter the name of a new folder to store the custom diagram definition below the **Diagram Definitions** node or select an existing folder.
   - **Name Dependency**: Specify the display of information in the diagram header:
     - `OwnName`: The diagram header displays the name of the object. This setting is recommended if multiple diagrams of the diagram type associated with the custom

diagram definition shall be available for the same master object specified in the configured report.

- `MasterName`: The diagram header displays the image properties of the master object specified in the configured report. The actual name of the diagram will be ignored. This setting is useful if there will always be at most one diagram of the diagram type associated with the custom diagram definition for any given master object.
- `PrimaryMasterName`: This is exclusively used for some standard diagram definitions in Alfabet and is not relevant in the context of custom diagram definitions.

Please note that the information displayed in the diagram header is also used as name of the diagram in diagram list reports.

- **Nodes Sequence**: Define the order that the nodes shall be displayed in the explorer in Alfabet Expand. Click the **Browse** button. In the editor, click the **Up/Down** arrows to sequence the nodes in the **Toolbox Items** pane. Click **OK** to close the editor and save the sequence of the nodes.
- **Connections Sequence**: Define the order that the connections shall be displayed in the explorer in Alfabet Expand. Click the **Browse** button. In the editor, click the **Up/Down** arrows to sequence the connections in the **Toolbox Items** pane. Click **OK** to close the editor and save the sequence of the connections.
- **Create Connections on Rescan**: Select `True` if the connections between nodes in the diagram shall be added to the diagram when the diagram is reloaded in the browser.
- **Shape Sequence**: Define the order that the shapes shall be displayed in the bottom section of the **Toolbox Items** pane in the Alfabet Diagram Designer as well as in the explorer in Alfabet Expand. Click the **Browse** button. In the editor, click the **Up/Down** arrows to sequence the shapes in the **Toolbox Items** pane. Click **OK** to close the editor and save the sequence of the shapes.
- **Toolbox Items Sequence**: Define the order that the configured toolbox items shall be displayed in top section of the **Toolbox Items** pane in the Alfabet Diagram Designer as well as in the explorer in Alfabet Expand. Click the **Browse** button. In the editor, click the **Up/Down** arrows to sequence the toolbox items in the **Toolbox Items** pane. Click **OK** to close the editor and save the sequence of the toolbox items.

5) In the toolbar, click the **Save** button to save the custom diagram definition.

Creating Nodes for the Custom Diagram

Nodes are visualized as diagram items in the context of a diagram. A node must be defined for every object class that shall be displayed in the custom diagram. You must create a node for all of the following:

- Every object class that shall be visualized in the diagram.
- Every object class that is specified for a node group.

- Every object class that is specified in a connection.

Please note that the inclusion of the node in the custom diagram definition is necessary for the visualization of the object class in the diagram. The inclusion of a node does not implicate that an object can be explicitly added to or created in the diagram. If the object class shall be added to the diagram or created in the context of the diagram, you must add the node as a toolbox item. For more information, see the section Creating Toolbox Items for the Custom Diagram.

If the node has a semantic reference to other classes and these classes shall be nested in the diagram item of the node, you must also define a node group for the node. Node groups are visualized in the diagram as nested diagram items. Nested diagram items are typical, for example, when applications are visualized in the application groups that own them or local components are visualized in the diagram items of the applications that own them. For more information, see the section Creating Node Groups for the Custom Diagram. For more information about specifying the relationships between classes, see the reference manual *Alfabet Meta-Model*.

To create a node for the custom diagram definition:

1) Expand the custom diagram definition  , right-click the **Nodes**  node and select **New Node**.
2) In the **Select Node Class** editor, select the class that the node is based on and click **OK**.
3) Click the new node and in the attribute window and define the following attributes:
   - **Class Name**: Displays the class that the node is based on.
   - **Node Group Item Only**: Select `True` if the object class may only be visualized as a node group. If set to `True`, you must specify the **Node Group Owner** attribute.
   - **Node Group Owner**: Specify the property that describes the relationship of the node to the node group. The node group must reference this property via the **Back Property** attribute of the node group.
   - **Item Operations**: Specify `True` for each operation that shall be allowed for the node in the context of the Alfabet Diagram Designer. The following operations can be specified:
     - **Cannot Add to Node Group**: The nodes cannot be moved to a node group.
     - **Cannot Remove from Node Group**: The nodes cannot be removed from a node group.
     - **Confirm Deletion**: The deletion of nodes must be confirmed by clicking an **OK** button in a message dialog.
     - **Delete**: The nodes can be deleted from the Alfabet database.
     - **Detach**: The nodes can be removed from the diagram.
     - **Edit**: The nodes can be edited.
     - **Move**: The nodes can be moved from a parent object to another parent object.
     - **Navigate**: The user can navigate to the object profile/object cockpit of the node.
     - **Resize**: The node can be resized.
   - **Diagram Item Template**: Specify a standard or custom diagram item template to visualize the node in the diagram.

- **Enable Substitute**: Select `True` if the **Substitute Object** options shall be available for the node in the **Semantic Actions** menu in the Alfabet Diagram Designer.
- **Enable Change Diagram Item Template**: Select `True` if the **Change Diagram Item Template** option shall be available for the node in the **Other Actions** menu in the Alfabet Diagram Designer.

4) In the toolbar, click the **Save** button to save the node definition.
5) Once all nodes have been defined, you can optionally click the custom diagram definition and in the attribute window and define the order that the nodes shall be displayed in the explorer in Alfabet Expand. For the **Nodes Sequence** attribute, click the **Browse** button. In the editor, click the **Up/Down** arrows to order the nodes in the **Toolbox Items** pane and click **OK** to save the sequence.
6) In the toolbar, click the **Save** button to save the custom diagram definition.

Creating Node Groups for the Custom Diagram

A node group represents the object classes that are semantically nested in the object class defined for the node. A node group must be specified if a semantic relationship between the node and its referenced objects are to be displayed in the diagram. Node groups are visualized in the diagram as nested diagram items. Nested diagram items are typical, for example, when applications are visualized in the application groups that own them or local components are visualized in the diagram items of the applications that own them. The node group is the referenced class that is nested in the node.

Node groups can be specified either via a property, reference, or an Alfabet query or native SQL query. For more information about specify the relationships between classes, see the reference manual *Alfabet Meta-Model*.

To create a node group for the custom diagram definition:

1) Expand the custom diagram definition , right-click the node and select **New Node Group**.
2) Click the node group and define the following attributes:
   - **Class Name**: Select the object class that the node group shall visualize.
   - **Caption**: Specify the caption of the node group. The caption will be displayed on the tab in the selector that opens when the **Add Subordinates for Object** option in the **Semantic Actions** menu in the Alfabet Diagram Designer is selected for a node.
   - **Node Group Type**: Select one of the following to find the objects for the node group definition:
     - `PropertyBased`: Select if the node group shall be visualized based on a property. Specify the referenced property in the **Property** attribute of the node group.

       For example, for a diagram that displays applications as node groups for application groups, the class Application would be a defined as node group for the class Application Group. The following would be defined:

- Create an Application node:
  - **Class Name**: `Application`
- Create an Application Group node group created for the Application node:
  - **Class Name**: `Application`
  - **Node Group Type**: `PropertyBased`
  - **Property**: `Application`

- `BackPropertyBased`: Select if the node group shall be visualized based on a reference. Specify the referenced property in the **Back Property** attribute of the node group. The reference property must be specified in the Node of the node that has the same class as the node group.

  For example, for a diagram that displays local components as node groups for applications, the class Local Component would be a defined as node group for the class Application. The following would be defined:

  - Create a Local Component node:
    - **Class Name**: `LocalComponent`
    - **Node Group Owner**: `Owner`
  - Create a Local Component node group created for the Application node:
    - **Class Name**: `LocalComponent`
    - **Node Group Type**: `BackPropertyBased`
    - **Back Property**: `Owner`

- `QueryBased`: Select if the reference between the node group and node shall be found via an Alfabet query. Specify the Alfabet query in the **Query as Text** attribute.
- `NativeSqlBased`: Select if the reference between the node group and node shall be found via a native SQL query. Specify the native SQL query in the **Native SQL Query** attribute.

- **Property**: If `PropertyBased` has been specified for the **Node Group Type** attribute: Specify the property of the node object class to find the objects in the node group object class.
- **Back Property**: If `BackPropertyBased` has been specified for the **Node Group Type** attribute: Specify the property of the node object class that the node group object class references. This property must be defined in the **Node Group Owner** of the node that the node group is subordinate to.
- **Information Message**: Enter the text that shall be displayed if the **Warn on Semantic Layout Changes** option is selected in the **Diagram Settings** editor in the Alfabet Diagram Designer and user then makes semantic changes in the diagram.

3) In the toolbar, click the **Save** button to save the node group definition.

Creating Connections for the Custom Diagram

A connection represents a reference between objects in a diagram (such as an information flow, business process rule, etc.). A connection must be specified for each reference that may be visualized in the custom diagram. The definition of the connection node may be either based on a semantic class (such as an `InformaionFlow`, `ValueArc`, or `DemandArch`) that reference a source and target object, or a property of type `ReferenceArray` such as `Device.Networks` or `Network.BelongsTo`. Connections representing a link object such as an information flow can be clicked and navigated (depending on the definition in the associated class setting) while connections representing simple references or reference arrays such as `Device.Networks` cannot be clicked or navigated.

This applies in particular to objects that are connected to other objects via a reference or a link object or objects that are defined through semantic groups. Please note that the inclusion of the connection in the custom diagram definition is necessary for the visualization of the connection in the diagram. The inclusion of a connection does not necessarily mean that a connection can be explicitly added to or created in the diagram.

New connections require the definition of the connection node that may be either based on a link (such as information flow, local component, or role) or a source and target relationship such as the reference Responsible Organization on the class Value Node. Connections representing a link object can be clicked and navigated (depending on the definition in the associated class setting) while connections representing simple references or reference arrays cannot be clicked or navigated.

we support only two types of connections: defined by properties of outgoing and incoming nodes, defined by properties of connection object itself.

To create a connection for the custom diagram definition:

1) Expand the custom diagram definition  , right-click the **Connections**  node and select **New Connection**.
2) In the **Select Connection from Class** editor, select the class that shall be the **From Class** of the connection and click **OK**. This class will be displayed in the **From Class** attribute.
3) In the **Select Connection to Class** editor, select the class that shall be the **To Class** of the connection and click **OK**. This class will be displayed in the **To Class** attribute.
4) Click the connection and define the following attributes:
   - **Caption**: Specify a caption of the connection. The caption will be displayed in, for example, the **Generate Network** editor available in the Alfabet Diagram Designer.
   - **Isolated**: The specification of the **Isolated** attribute will determine which attributes must be defined for the connection:
     - Select `True` if the connection represents a semantic class (such as an `InformationFlow`, `ValueArc`, or `DemandArch`) and the connection shall be established based on the **From Property** attribute which identifies the source node and the **To Property** attribute which identifies the target node and the **Link Class** attribute which references the source and target object. If the **Isolated** attribute is set to `True`, define the following:
       - **From Property**: Specify the property that represents the source node of the connection.
       - **To Property**: Specify the property that represents the target node of the connection.

- **Link Class**: Specify the semantic class (such as an `InformationFlow`, `ValueArc`, or `DemandArch`) that references the source and target object.

- Select `False` if the connection in the diagram does not represent a semantic class but rather a property of type `ReferenceArray` such as `Device.Networks` or `Network.BelongsTo`. In this case, the connection will be established based on the intersection of the **Output Property** attribute which identifies the source node and the **Input Property** which identifies the target node. If the **Isolated** attribute is set to `False`, define the following:

  - **Output Property**: Specify the property that identifies the source node of the connection. Specify the property of the source node that intersects with the property identified in the **Input Property** attribute.

  - **Input Property**: Specify the property of the target node that intersects with the property identified in the **Output Property** attribute.

- **Diagram Item Template**: Specify a standard or custom diagram item template to visualize the connection in the diagram.

- **Item Operations**: Select `True` for each operation that shall be allowed for the connection in the context of the Alfabet Diagram Designer. The following operations can be specified:

  - **Confirm Deletion**: The deletion of connections must be confirmed by clicking an **OK** button in a message dialog.

  - **Delete**: The connection can be deleted from the Alfabet database.

  - **Edit**: The connections can be edited.

  - **Move**: The connections can be moved from a parent object to another parent object.

  - **Navigate**: The user can navigate to the object profile/object cockpit of the connection.

  - **Resize**: The connection can be resized.

- **Allow Self Connection**: Select `True` if a connection can have the same source and target object. Select `False` if the source and target object must be different for the connection.

- **Available for Generate Net**: Select `True` if the connection shall be available in the **Include Connection Types to Build Network** field in the **Generate Network** editor available in the Alfabet Diagram Designer.

- **Enable Change Diagram Item Template**: Select `True` if the **Change Diagram Item Template** option shall be available for the connection in the **Other Actions** menu in the Alfabet Diagram Designer.

- **Implicit**: Select `True` if the connection should be displayed as the result of an aggregation. For example, if the **Implicit** attribute is set to `True` then:

  - If information flows are defined between applications and the diagram displays application groups but no applications, the information flows between applications can be displayed as information flows between the application groups that the applications are assigned to.

  - If information flows are defined between local components and the diagram displays application but no local components, the information flows between local components

can be displayed as information flows between the applications that the local components are assigned to.

- If connections are defined between applications and business processes and the diagram displays application groups but no applications, the connections between applications and business process can be displayed as connections between the application groups that the applications are assigned to and the business processes.

5) In the toolbar, click the **Save** button to save the connection definition.

6) Once all connections have been defined, click the custom diagram definition to open the attribute window and define the following:

- **Connections Sequence**: Define the order that the connections shall be displayed in the **Toolbox Items** pane in the Alfabet Diagram Designer. For the **Connections Sequence** attribute, click the **Browse** button. In the editor, click the **Up/Down** arrows to order the connections in the **Toolbox Items** pane and click **OK** to save the sequence.
- **Create Connections on Rescan**: Select `True` if the connections between nodes in the diagram shall be added to the diagram when the diagram is reloaded in the browser.

7) In the toolbar, click the **Save** button to save the custom diagram definition.

Creating a Connected Diagram Definition for the Custom Diagram

Objects displayed in custom diagrams may provide linkage to other diagrams. Only one diagram may be linked per diagram object and only diagram types that are permissible for the selected node may be specified. If an object in a diagram has been configured to be linked to another diagram, the user can click the dark blue **Navigate** icon on the lower-right corner of the diagram object. The name of the target diagram and its base object will be displayed in the tooltip for the **Navigate** icon. Please note that navigation to diagrams is not supported for connections.

You can specify one or more permissible diagram connections for a custom diagram definition. A configured report must be defined to find the target diagram for the navigation. For more information about configuring reports for custom diagrams, see the section Configuring Reports for Custom Diagrams.

1) Expand the custom diagram definition , right-click the **Connected Diagrams** node and select **New Connected Diagram**.
2) In the **Diagram Definition** attribute, select the relevant custom diagram definition that shall be used for the target diagram in the navigation.
3) In the **View Link** attribute, specify the configured report that specifies the target diagram of the navigation.

Creating Toolbox Items for the Custom Diagram

A toolbox item is required for every object class that can be created, created as a copy, or added to the custom diagram. If a connection shall be created via a linked class, you must also create a toolbox item for the classes specified in the **From Class**, **To Class**, and **Link Class** attributes of the connection.

To create a toolbox item for the custom diagram definition:

1) Expand the custom diagram definition , right-click the **Toolbox Items** node and select **New Toolbox Item**.
2) In the **Select Class** editor, select the class that shall be created for or added to the diagram. This class will be displayed in the **Class Name** attribute.
3) Click the toolbox item and in the attribute window and define the following attributes:
   - **Name**: Define a technical name for the toolbox item.
   - **Caption**: Specify a caption of the toolbox item. The caption will be displayed in, for example, the **Generate Network** editor.
   - **Operation**: Specify the operation that may be executed via the toolbox item:
     - `Add`: Select if an object of the relevant object class may be added to the diagram. If `Add` has been specified for the **Operation** attribute, define the following as needed:
       - **Object Selector**: Specify the custom selector that allows users to select the object to add to the diagram. If the **Object Selector** attribute is defined, the **Selected Class** attribute should be empty. If a custom selector is not specified, the selector specification in the **Selector Definition** attribute of the relevant class setting associated with the user profile that the user is logged in with will be used. The custom selectors will replace the standard selector except when a hard-coded selector is required. For more information about configuring a class setting, see the section Configuring a Custom Selector for Search Functionalities.
       - **Multi-Select**: Select `True` if multiple objects may be selected in the object selector and added simultaneously to the diagram via the object selector. Select `False` if only a single object may be selected in the object selector and added to the diagram via the object selector.
     - `Create`: Select if an object of the relevant object class may be created in the context of the diagram. If `Create` or `CreateAsCopy` has been specified for the **Operation** attribute, define the following as needed:
       - **Edit After Create**: Select `True` if the object that has been created in the context of the diagram may be edited in the context of the diagram after it has been created. Select `False` if the object that has been created in the context of the diagram may not be edited after it has been created.
     - `CreateAsCopy`: Select if an object of the relevant object class may be created as a copy of an existing object in the context of the diagram.
   - **Diagram Item Template**: Select the standard or custom diagram item template that shall be displayed in the toolbox pane.

- **Icon**: Specify the icon that shall override the icon defined for the diagram item template selected in the **Diagram Item Template** attribute. Please note the following:
  - The definition of the **Icon** attribute for the toolbox item has precedence over the **Icon** attribute defined in the relevant class settings as specified via the **Selected Class** attribute for the toolbox item.
  - The **Icon** attribute defined in the relevant class settings has precedence over the **Icon** attribute defined for the diagram item template specified in the **Diagram Item Template** attribute of the toolbox item.
- **Hint**: Specify a tooltip to display when the user points to the icon of the diagram item template.
- **Expand Derivatives**: Select `True` if toolbox items should be displayed in the **Toolbox Items** pane in the Alfabet Diagram Designer for all custom diagram item templates that are based on the diagram item template selected in the **Diagram Item Template** attribute. Please note that this may lead to duplicated items being displayed in the **Toolbox Items** pane. Select `False` to display only the diagram item template explicitly selected in the **Diagram Item Template** attribute. In this case, duplicated toolbox items will not be displayed.
- **Selected Class**: Specify the object class that the user shall select on object from when adding the toolbox item to the diagram. The object selector that opens for the selected class will be based on the specification of the **Selector Definition** attribute of the respective class setting. Please note that you can define an explicit selector by means of the **Object Selector** attribute of a toolbox item. In this case, the **Selected Class** attribute should be empty. For more information, see the section Creating Toolbox Items for the Custom Diagram.

4) In the toolbar, click the **Save** button to save the toolbox item definition.

5) Once all toolbox items have been defined, click the custom diagram definition and in the attribute window and define the order that the toolbox items shall be displayed in the **Toolbox Items** pane in the Alfabet Diagram Designer. For the **Toolbox Items Sequence** attribute, click the **Browse** button. In the editor, click the **Up/Down** arrows to order the toolbox items in the **Toolbox Items** pane and click **OK** to save the sequence.

6) In the toolbar, click the **Save** button to save the custom diagram definition.

Adding Shapes to the Custom Diagram

In addition to the diagram item templates that allow object classes to be added to the **Toolbox Items** pane in the Alfabet Diagram Designer, you can specify diagram item templates that allow shapes, lines, and images to be added as graphic elements to the diagram. The shapes that may be added to the diagram definition are predefined by Software AG.

You can also specify diagram item templates as custom shapes to add to the diagram. Such complex shapes are typically used to define the background of a diagram. Please note that for background shapes that should be scaled based on the sizing of the custom item, the value `Scalable` should be selected for the **Layout Type** attribute of the diagram item template.

To create a button for the custom diagram definition:

1) Expand the custom diagram definition , right-click the **Shapes** node and select **New Shape**.
2) In the editor that opens, select a standard shape or configured diagram item template (lsited under the **Custom Shape** section) to add to the diagram. The shape will be added below the **Buttons** node.
3) In the toolbar, click the **Save** button to save the shape definition.
4) Once all nodes have been defined, click the custom diagram definition and in the attribute window and define the order that the shapes shall be displayed in the bottom section of the **Toolbox Items** pane in the Alfabet Diagram Designer. For the **Shape Sequence** attribute, click the **Browse** button. In the editor, click the **Up/Down** arrows to order the shapes in the bottom section of the **Toolbox Items** pane and click **OK** to save the sequence.
5) In the toolbar, click the **Save** button to save the custom diagram definition.

Defining Rules to Automatically Add Objects to New Diagrams

By default, a new diagram based on a custom diagram definition will be empty and must be defined from scratch by the user opening the diagram in the Alfabet Diagram Designer. Optionally, rules can be added to the custom diagram definition that automatically add default objects found by the rule to a diagram on creation of the diagram.

For diagrams created via a diagram list report, the default objects will be added to the diagram on creation of the diagram via the list and will be available when the user accesses the diagram via the navigate button. For diagrams created via a diagram view report that has been added directly to an object profile or object cockpit, the objects will be added the first time that the diagram is opened in the Alfabet Diagram Designer.

Default objects can only be added automatically to diagrams when the diagram is first created. Once a diagram has been created, all updates to the number and kind of default objects in the Alfabet database must be added by a user designing the diagram in the Alfabet Diagram Designer. A menu option **Add Default Objects** has been added to the **Semantic Actions** button to update the diagram with default objects. The existing design will be maintained when the default objects are added. Default objects already added to the diagram will not be added again and will remain in the position and design already defined for them in the diagram.

The following is required to add rules for the automatic assignment of objects to diagrams to the custom diagram definition:

- Defining a Category for the Rules in the XML Object UseCaseCategories
- Defining the Query for the Rule in a Configured Report
- Adding the Rule to the Custom Diagram Definition

Defining a Category for the Rules in the XML Object UseCaseCategories

Reports can only be used for the definition of rules for custom diagram definitions if they are assigned to a category related with the functionality. Category names are assigned to functionalities in the XML object ***UseCaseCategories***. After having assigned a name for the category in the XML object ***UseCaseCategories***, the category must then be assigned to the configured report via the attribute **Category** of the configured report.

To set the category name in the XML object ***UseCaseCategories***:

1) In the **Presentation** tab of Alfabet Expand, expand the explorer node **XML Objects**.
2) Right-click on the XML object ***UseCaseCategories*** in the explorer and select **Edit XML** from the context menu. The XML object opens in the middle pane.
3) Add the following code as child XML element to the XML root element `UseCaseCategories`:

```
<UseCaseInfo UseCase="Diagrams">
<ScopeInfo Scope="Report" Categories="CommaSeparatedListOfCategories" />
</UseCaseInfo>
```

4) Substitute *CommaSeparatedListOfCategories* with a category name. If you enter more than one name comma separated, all names will be valid as category name for the functionality.
5) In the toolbar, click the **Save**  button.

Defining the Query for the Rule in a Configured Report

The rule that finds the default objects to be automatically added to the diagram references a configured report. The following configuration is required for the configured report:

For general information about configured reports, see Configuring Reports. For general information about defining queries for Alfabet configurations, see Defining Queries.

- The **Type** attribute of the configured report must be `Query` or `NativeSQL`.
- The `Category` attribute of the configured report must be set to one of the categories defined for the use case `Diagram` in the XML object ***UseCaseCategories***.
- The rule reads the information about which objects are found via the query. For Alfabet queries, these are the objects defined as `FIND` class. For native SQL queries, the first argument in the `SELECT` statement must return the `REFSTR` of the objects to be added to the diagram. All other data returned via the `SELECT` statement of native SQL queries or the **Show Properties** of Alfabet Queries will be ignored except for the special use case of hierarchically structured objects described below.
- The Alfabet query language parameter `BASE` can be used in the queries to refer to the object the diagram is created for.

For example, the following Alfabet query will add all application groups to the diagram to which the application the diagram is created for is assigned:

```
ALFABET_QUERY_500
FIND ApplicationGroup
```



```
WHERE ApplicationGroup.Applications CONTAINSOR:BASE
```

- If objects should be added in a hierarchical structure, which means a parent object containing a group of subordinate objects, the query must return a grouped data set based on a `GroupBy_Ex` instruction. The structure can have multiple levels.

  For example, the following native SQL query will add for a diagram defined for business data all relevant applications and will also add the local components assigned to the applications as object group subordinate to the applications:

  ```
  SELECT app.REFSTR, app.REFSTR As 'app.REFSTR', lc.REFSTR As
  'lc.REFSTR'

  FROM APPLICATION app, LOCALCOMPONENT lc, BusinessDataUsage
  bdu

  WHERE lc.OWNER = app.REFSTR

  AND bdu.OBJECT = app.REFSTR

  AND bdu.DATA = @BASE

  /* Alfabet Instructions */

  GroupBy_Ex("gr.REFSTR","sub.REFSTR","gr",0);
  ```

- The configured report must have the **State** set to `Active` to be executed for new diagrams.

Adding the Rule to the Custom Diagram Definition

The following configurations are required in an existing custom diagram definition to add a rule for automatic placement of default objects on creation of the diagram. Multiple rules can be defined for a custom diagram definition to automatically add default objects found by multiple queries.

1) Expand the custom diagram definition  , right-click the **Default Objects Queries**  node and select **New Default Objects Query**. The new default objects query will be added below the **Default Objects Query** node.
2) Click the new defaults objects query in the explorer to open the attribute window.
3) In the **Report** attribute, select the configured report providing the query.
4) Make sure that for each object found by the default object query a node is defined in the custom diagram definition. The attribute settings of the node must match the following rules:
   - For nodes that are added via a non-grouped dataset or on the first level of the hierarchy in a grouped dataset, the **Node Group Item** only attribute must be set to `False`.
   - For nodes that are added on a subordinate level of a grouped dataset, the **Node Group Item Only** attribute can be set to `True`. If it is set to `True`, the **Node Group Owner** attribute must be set to the object class property defining the relation to the parent object class.
   - For nodes that are added on a subordinate level of a grouped dataset, a node group must be defined subordinate to the parent object class node.
5) In the toolbar, click the **Save**  button to save the custom diagram definition.

Configuring Reports for Custom Diagrams

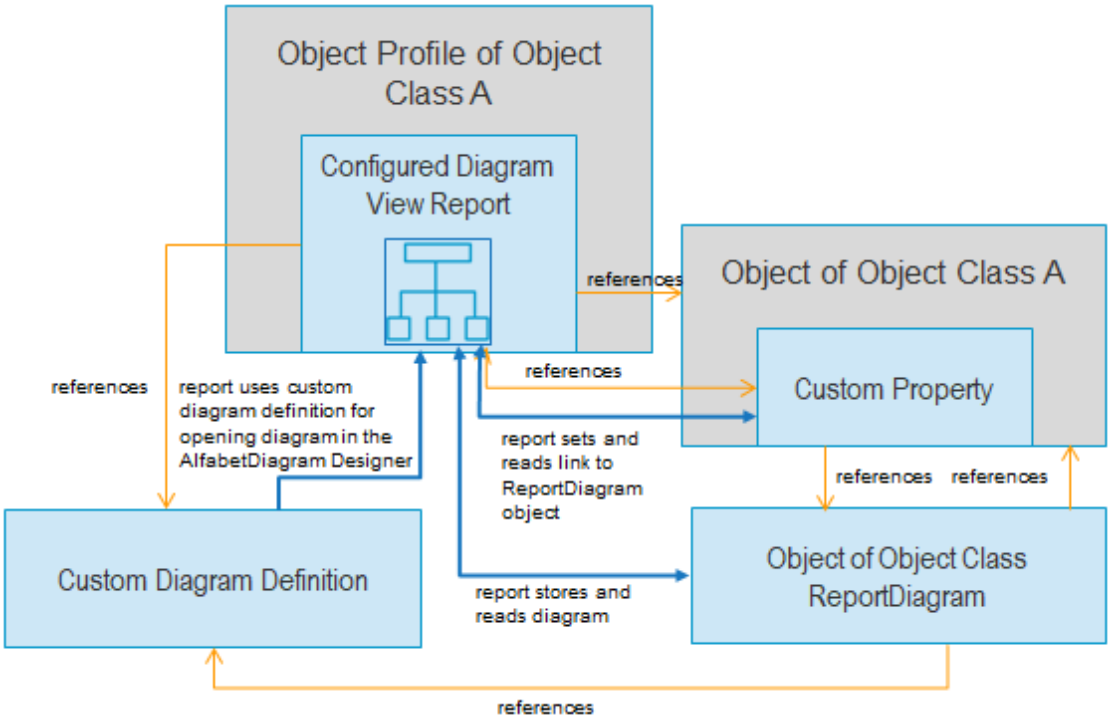
A configured report must be configured to handle the diagram functionalities for the custom diagram definition. A configured report must be created that displays the diagram to the user and provides the button for opening the Alfabet Diagram Designer and editing the diagram. The report must be of the type `Custom` and based on the template `DiagramViewReport`. Upon creation of the configured report view, standard filters and buttons for diagrams will be automatically added to the report. The filters must then be adapted to the requirements of the custom diagram definition. The custom diagram definition that the diagram shall be based on must be selected in the report assistant.

The protected enumeration `CustomDiagramViewType` is available for the definition of diagram views displayed in the context of custom diagrams. The enumeration items defined for the `CustomDiagramViewType` enumeration can be selected in the **Type** field in the **Diagram View** editor. The query specified for the `DiagramView` filter field for the `DiagramViewReport` report must refer to the relevant standard or custom diagram view type. The diagram views based on the standard or custom diagram view type can selected in configured reports based on the `DiagramViewReport` report template. For more information about configuring protected enumerations, see the section *Overview of Protected Enumerations* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*. For more information about configuring diagram views, see the section *Configuring Diagram Views for Diagrams* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.

There are two different configuration options that allow users to create diagrams in the Alfabet user interface. You can either define the configured report so that users can create the diagram as the main standard diagram for the object or so that users can create multiple diagrams for the object:

- **The user can create the diagram as the main standard diagram for the object.** The diagram is available in a configured diagram view report added to the object profile or object cockpit of the object class. The configured diagram view report is designed to show and provide editing for the standard diagram only.

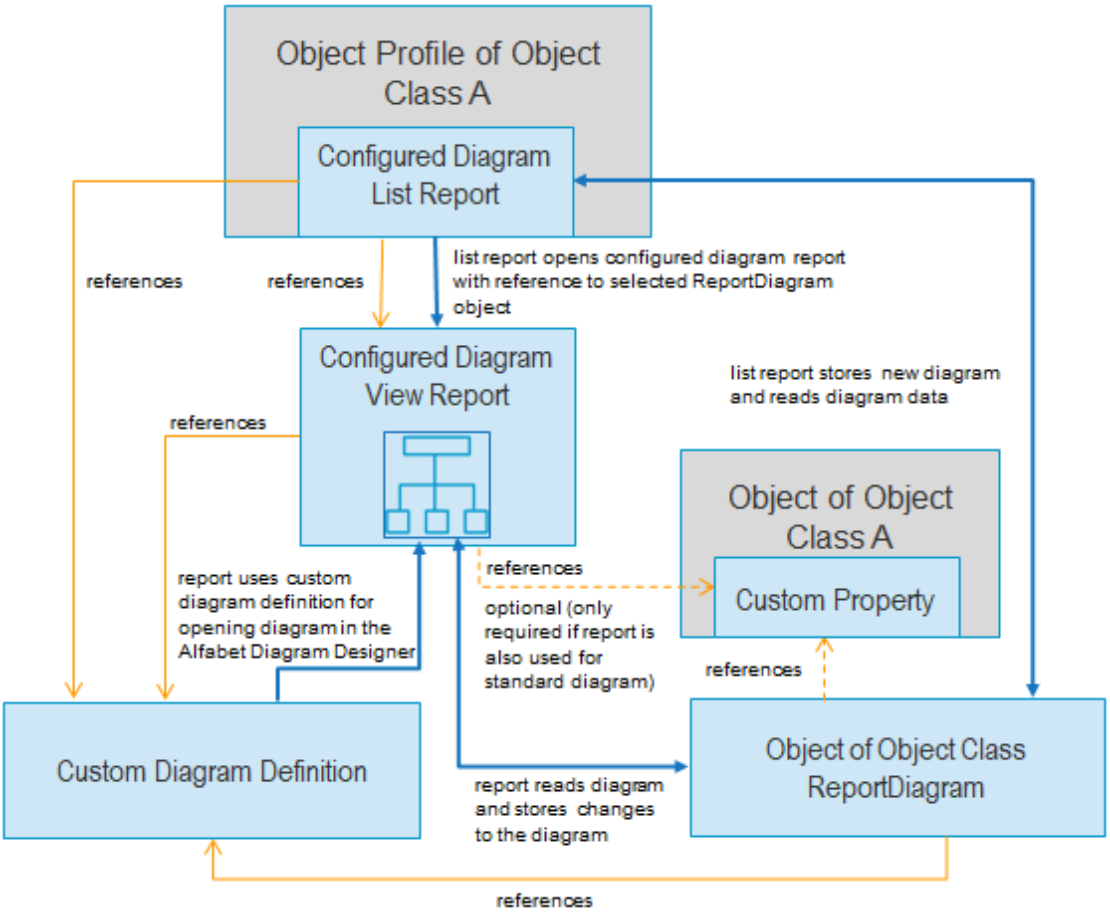
A custom property of the type `Reference` referencing the object class `ReportDiagram` must be defined for the object class. When a user opens the configured diagram view report for an object and creates a diagram, the configured diagram view report will set a reference to the defined `ReportDiagram` object in the custom property. The next time any user opens the configured diagram view report for the object, the configured diagram view report will open the diagram referenced in the custom property.



The following configuration steps are required:

- Creating a Custom Diagram Definition
- Creating a Custom Property with a Reference to the Standard Diagram
- Creating a Configured Diagram View Report
- Adding the configured diagram view report to the object profile or object cockpit of the object class as described in the chapter Configuring Object Views.

- **The user can create multiple diagrams for the object.** The diagrams are then available via a configured diagram list report listing all diagrams in a table. The user can create diagrams via the **New** button in the toolbar of the configured report. The user selects the custom diagram definition he/she want to use for the new diagram in the editor for creating the new diagram. The new diagram is added to the table. The user can open and edit the diagram by double-clicking the diagram in the table. The diagram opens in a configured diagram view report that is defined to handle diagrams based on the custom diagram definition the selected diagram is based on. The configured diagram view report is designed to display and provide editing for any diagram based on the selected custom diagram definition.



The following configuration steps are required:

- Creating a Custom Diagram Definition
- Creating a Configured Diagram View Report
- Creating a Configured Report Listing All Available Diagrams for an Object
- Adding the configured diagram list report to the object profile or object cockpit of the object class as described in the chapter Configuring Object Views.

  This step is optional. The configured list report can also be used to create a diagram independent of a base object. The report should then be made available independent of an object view or object cockpit (for example, in the **Configured Reports** functionality).

The reports may be assigned to an object profile or object cockpit so that users can access the diagram view report and design and view the custom diagram. The reports will also be available via the **Configured Reports** functionality. For more information, see the section Integration of Configured Reports in the Alfabet Interface.

The following information is available:

- Creating a Custom Property with a Reference to the Standard Diagram
- Creating a Configured Diagram View Report

- Creating Custom Diagram Views for a Custom Diagram View Report

- Creating a Configured Report Listing All Available Diagrams for an Object

Creating a Custom Property with a Reference to the Standard Diagram

The object class for which the diagram shall be created must be configured to link to the report diagram object that represents the standard diagram based on a custom diagram definition.

To define the custom object class property:

1) Go to the **Meta-Model** tab and expand the **Classes** node.

2) Right-click the relevant object class  and select **Add New Property**. The **Create New Property** editor opens. Define the following fields:

   - **Property Name**: Enter a unique name for the custom object class property. This value is displayed in the solution interface if the **Caption** attribute is not defined.

   - **Tech Name**: Enter a unique technical name for the custom object class property that will be used in the Alfabet database table. Click **OK** to save the definition.

     Please note that the `Tech Name` is the name of the column that will be created in the database table of the object class for storing values for the custom property. The following rules apply:

     - The technical name may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.

     - The technical name should only contain standard `ASCII` characters.

     - The technical name should consist of upper-case letters only.

     - The maximum length of a technical name may not exceed 30 characters.

     - The technical name may not coincide with any of the reserved key words of the relational database management system.

     A validation mechanism checks for correct syntax when defining a technical name.

3) In the attribute window, define the following attributes:

   - **Property Type**: Select `Reference`.

   - **Comments**: Provide information relevant to solution designers regarding the maintenance of the custom object class property.

   - **Type Info**: Click the **Browse**  button on the right of the field to open the object class selector. Set the checkbox **Semantic Classes** and select the class `ReportDiagram` from the drop-down list. Click **OK**.

4) In the toolbar, click the **Save**  button to save the new custom object class property.

Creating a Configured Diagram View Report

The configured diagram view report displays the diagram and provides the button for opening the Alfabet Diagram Designer and editing the diagram.

You must create at least one configured diagram view report for each custom diagram definition.

A configured diagram view report can be used for the definition of a standard diagram as well as diagrams available via a configured diagram list report. The configured diagram view report will only store the diagram as the standard diagram of an object class if the custom object class property is selected in the report assistant.

To create a configured diagram view report:

1) Go to the **Reports** tab, right-click the **Reports** folder or a sub-folder thereof and select **Create New Report**. You will see the new configured report listed below the selected folder.

As an alternative to defining all attributes for a new configured report, you can copy and paste an existing configured report and make modifications, as needed. To do so, right-click the existing configured report that you want to copy and select **Copy**. Right-click the new configured report and select **Paste**. Modify the necessary attributes described below.

2) In the attribute window, define the following attributes for the configured report.

- **Type**: Select `Custom`.
- **Name**: Optionally, change the technical name of the configured report. The technical name of the configured report must be unique.

The technical name is used for identification of the configured report in the context of technical processes such as merging the report configuration with a report configuration saved as an XML file. Therefore, the following is required when defining the name of a configured report:

- The name must be unique. The name must be unique regardless of the report folder it is structured in.
- The name may not be changed once a configured report has been created.
- A validation mechanism checks for correct syntax when defining a technical name. Please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.
- If the technical name of the configured report is changed, the name will be correctly updated in other configuration objects referencing the changed report during design time.

Please note that the name of a changed configuration object will not be updated in guide pages or guide views. If you plan to change the name of a configuration object, the reference in the navigation page should be changed prior to changing the configuration object name. You can use the **Show Usage** functionality in the context menu of the configured report to evaluate whether the configured report has been added to a guide view or guide page.

- **Caption**: Enter a caption for the configured report that is displayed on the Alfabet user interfaced for the configured report. The caption of the configured report may exceed the conventional 64 character limit.
- **Description**: Provide a short information about the configured report that is useful to Alfabet users. This comment will be displayed in the object profile under the page view caption as a short description.
- **Technical Comment**: Provide information that is useful for solution designers working with Alfabet Expand in order to better understand the configured report from a technical perspective. The **Technical Comment** will not be displayed in the user interface.
- **Template**: Select `DiagramViewReport`.
- **Apply to Class**: Leave this attribute empty. Diagram view report must be assigned to an object class via the attribute **Base Class** in the **Report Assistant** and not via the **Apply to Class** attribute. If you define htis attribute, navigation For example, via the AlfaBot will not work correctly.
- **Help Index**: Specify the location of the external Help file using the following syntax:(For example:). The external help file will be displayed when the user clicks the context-sensitive Help link in the standardHelp menu.

  Server variables can be used in the Help links. Server variables allow you to define all or part of the URL definition in the server alias configuration of the Alfabet Server instead of directly defining it in the configuration in Alfabet Expand. Use of server variables is useful, for example, in order to change a Web server name defined in multiple URL definitions in Alfabet Expand. Instead of changing all URL definitions, only the variable value in the server alias configuration must be updated. The use of server variables is described in the section Using Server Variables in Web Link and Database Server-Related Specifications in the chapter Configuring Reports.

- **Execute on Enter**: Set to `True` to execute the configured report with empty filter settings when the user opens the configured report in the Alfabet interface. Set to `False` to open the configured report without execution of the Alfabet query with empty filter settings. By default, this attribute is set to `True`. It should only be set to `False` if the configured report would result in an exceedingly big dataset when executed with empty filter settings.

  When you set the **Execute on Enter** attribute to `False`, you must make sure that the **Submit** button is available in the custom report view of the configured report. If the **Submit** button is deleted from the custom report view, the configured report cannot be displayed. The setting and execution of the **Execute on Enter** attribute is independent of the definition of filters for a configured report. A configured report without filters must nevertheless have a **Submit** button when the **Execute on Enter** attribute is set to `False`.

- **Selector Behavior**: This attribute must be set to `NotVisible`. The attribute is set to `Visible` per default and cannot be edited directly in the attribute window. Right-click the configured report and select **Set Report Selector Behavior to 'Not Visible'** from the context menu.

  If the attribute is set to `NotVisible`, the configured report will be excluded from the selector for adding configured reports to the **Configured Reports** functionality. The diagram view report is exclusively for use in object cockpits/object profiles and in configured diagram list reports and should not be accessible anywhere else.

Please note however that this setting does not exclude the configured report from any standard views or customer configured views and selectors but excludes it only from the standard selector for configured reports implemented in Alfabet.

- **Can Create Express View**: Select `True` if an express view may be created for the report. Select `False` if an express view may not be created for the report. Please note that for reports that have a custom report view, the setting must be consistent with the setting of the attribute **Can Create Express View** of the custom report view.

An express view is a link to a specific Alfabet view displaying data that allows individuals inside or outside of the user community to view information in Alfabet. When the express view is created, an email notification is automatically mailed to a specified recipient. The recipient receives a URL that allows him/her to access the current page view in Alfabet. For more information, see the section *Sending and Receiving Express Views* in the reference manual *Getting Started with Alfabet*.

3) In the explorer, right-click the configured report and select **Create Configured Report View**. The view editor opens. In the explorer, the custom report view is displayed as a subordinate object of the configured report. The attributes for the custom report view are displayed in the attribute window. You can optionally edit the following attributes:

- **Name**: Optionally you can change the name for the custom report view. The **Name** must be unique for custom report views.
- **Can Create Bookmark**: Select `True` if a bookmark may be created for the configured report. Select `False` if a bookmark may not be created for the configured report.
- **Can Create Express View**: Select `True` if an express view may be created for the configured report. Select `False` if an express view may not be created for the configured report. Please note that the setting must be consistent with the setting of the attribute **Can Create Express View** of the configured report.
- **Custom Context-Sensitive Help URL**: Enter the URL or server variable that targets the custom context-sensitive help.
- **Standard Context-Sensitive Help Index**: This field will be empty for a configured report.

The custom report view initiates the **Report Assistant**. A custom report view should never be deleted from the configured report after the **Report Assistant** has been used to configure the configured report. When you delete the custom report view of an existing template-based configured report and create a new custom report view, the **Report Assistant** will be reset to the example specified per default and your configuration will be lost.

4) In the view editor, the filter panel shows all filter fields and buttons that are available in the standard page view to display diagrams. You can alter the filter fields of the diagram as follows:

- You can remove any of the filter fields.
- Per default, the `Status` (Object State) filter field limits the applications displayed in the diagram according to the **Object State** attribute. If objects of another object class shall be filtered according to object state values, change the **Parameter** attribute of the `Status` filter to the name of the object class for which the object state shall be filtered in the diagram.
- `LinkProperty` (**Connection Attributes**) filter field: This filter field is a multi-select combobox that allows object class properties of the connection object class and its referenced

object classes to be selected for display in the decoration boxes of the connections in the custom diagram. The content of the filter field depends on the settings in the XML object ***DiagramInformationflowDef***. For more information, see Configuring the Visualization of Connection Items and Subordinate Object in Diagrams.

- The protected enumeration `CustomDiagramViewType` is available for the definition of diagram views displayed in the context of custom diagrams. The enumeration items defined for the `CustomDiagramViewType` enumeration can be selected in the **Type** field in the **Diagram View** editor. The query specified for the `DiagramView` filter field for the `DiagramViewReport` report must refer to the relevant standard or custom diagram view type. The diagram views based on the standard or custom diagram view type can selected in configured reports based on the `DiagramViewReport` report template.

To display the relevant custom diagram views specified for the custom diagram definition in the drop-down list of the `DiagramView` filter field, change the query defined for the filter field in the **Range as Alfabet Query** attribute of the filter field. The `WHERE` condition of the query must be changed to return the diagram views specified with the standard or custom diagram view type:

```
WHERE
    (AND
        Type = 'StandardOrCustomDiagramViewType'
        (OR
            UserProfiles IS NULL
            UserProfiles CONTAINSOR:CURRENT_PROFILE
        )
    )
```

For more information about configuring protected enumerations, see the section *Overview of Protected Enumerations* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*. For more information about configuring diagram views, see the section *Configuring Diagram Views for Diagrams* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.

- You can change the **Caption** and **Hint** attributes of all filter fields as needed.

5) In the explorer, right-click the configured report and select **Start Alfabet Report Assistant**. The **Report Assistant** opens.

6) Specify the following attributes in the **Report Assistant**:

- **Diagram Def**: Select the custom diagram definition that the diagram shall be based on.
- **Base Class**: Select the object class that the diagram shall be created for.
- **Base Property**: This attribute is only required if the configured diagram view report is intended to be used as to create standard diagram of that diagram template for the objects belonging to the base class. Select the custom object class property that you created for the object class defined in the **Base Class** attribute to store the link to the report diagram object created as standard diagram.

7) Click **OK** to save the configuration.

8) In the toolbar, click the **Save** button to save your changes.

9) In the explorer, right-click the configured report and select **Review Report**. The configured report opens in a web browser. If the results are not as expected, you can edit the configured report until the output of the configured report is adequate.

10) Right-click the configured report and select **Set Report State to 'Active'** from the drop-down menu.

Creating Custom Diagram Views for a Custom Diagram View Report

A diagram view is a configuration that is associated with a diagram. It allows users to superimpose qualitative information - such as aggregated indicators or attribute values - associated with these architectural elements. For standard as well as custom diagrams, diagram views can be defined in the *Configuring Diagram Views for Diagrams* functionality on the Alfabet user interface.

For custom diagrams, additional diagram views can be configured as a functionality of the configured diagram view report used for opening the diagram. These report specific diagram views combine the functionalities available for standard diagram views with report specific functionality. The following functionality can be added to a custom diagram via a diagram view defined in the configured diagram view report used to open the diagram:

- Additional filters can be added to the configured diagram view report to filter the content of the diagram. For example, for a diagram displaying applications, display might be limited to applications with a start date after a defined value or for applications that have local components assigned. Objects not matching the filter specification are either displayed in pale grey or completely removed from the dataset.
- The diagram view items used in the diagram to display objects can be exchanged with diagram view items providing additional information for all or a subset of objects displayed in the diagram.
- Icons can be displayed in the diagram view items via the definition of indicator rules. Icons defined via indicator rules are displayed within the diagram view items following the rules that are implemented for all types of configured reports.

To define a diagram view for an existing custom diagram view report:

1) If you would like to use custom diagram items showing addition information specified via a query within the diagram view, define custom diagram item templates that contain rectangles with the Content Type attribute set to `QueryValue`. For information about defining custom diagram view item templates, see Configuring Custom Diagram Item Templates.

2) In the explorer, right-click the configured report and select **Set Report State to 'Plan'**.

3) To add filter fields to the configured diagram view, expand the node of the configured report in the explorer and double-click the child node. The configured report view opens in the middle pane. Add filter fields to the view according to requirements. Please note that **Edit** and **Edit Search** filter fields are not supported. For information about the definition of filter fields, see Defining Filters for Configured Reports and Selectors.

4) In the explorer, right-click the configured report and select **Start Alfabet Report Assistant**. The **Report Assistant** opens.

5) Right-click the root node of the explorer in the **Report Assistant** and select **Add New Diagram View**.

6) Expand the new diagram view node in the explorer of the **Report Assistant**.

7) To add filters and / or display custom diagram view items with rectangles with a content type `QueryValue`, define a query for each object class that you would like to filter or display with additional information. To add a query, right-click the **Queries** node and select **Add New Query**. Set the following attributes of the new query:

   - **Name**: Define a name that is displayed in the explorer to identify the query.
   - **Alfabet Query / Native SQL / Query as Text**: Define an Alfabet Query or native SQL query returning the following information:
     - The row reference class must be the object class displayed in the diagram that shall be filtered or for that additional information shall be provided. The row reference class is by default the `FIND` class of an Alfabet query or for native SQL queries the object class for that the `REFSTR` is returned in the first argument of the `SELECT` statement. The row reference can be changed via a `SetRowReference` instruction.
     - Make sure that the query returns all relevant objects of the row reference object class that may be displayed in the diagram.
     - For each filter that should be applied to the object class a column must be added that returns the value that shall be compared with the value returned in the current filter settings. The filter definition will refer to the column name. If you would like a filter to be applied to multiple object classes, you must add a column with an identical name returning the same value type to each of the queries.
     - For each rectangle of the type `QueryValue` in the custom diagram view item template that shall be used to display the objects in the diagram a column must be added that returns the value that shall be displayed for the object. The column name must be identical to the name of the rectangle in the diagram view item template.

8) For each object class that shall be displayed with another than the diagram specific diagram view item, a custom diagram view item definition must be added to the diagram view. To add a custom diagram view item definition, right-click the **Custom Diagram View Items** node and select **Add New Diagram View Item**. Set the following attributes of the new diagram view item:

   - **Name**: Define a unique name for the diagram view item.
   - **Class Name**: Define the object class name or object class stereotype that the diagram view item shall be displayed for.
   - **Diagram Item Template**: Define the diagram view item template that the diagram view item is based on.

9) To add filters to the custom diagram view, define a filter rule for each filter added to your filter panel. For each object in the diagram, the value to be filtered is read from the query for the corresponding object class defined as sub-node of the **Queries** node. The value returned by the query is compared to the value in the filter field using an operator. If the value returned by the query and the filter settings do not match for an object, the object is hidden. A query rule defines the filter field and query data as well as the operator for the filtering.

   To add a filter rule, right-click the **Diagram Filters** node and select **Add New Filter Rule**. Set the following attributes of the new filter rule:

   - **Name**: Define a unique name that will be used for the rule in the explorer of the **Report Assistant**.

- **Class Name**: Select the object class or object class stereotype for the object that shall be filtered. The data to be compared with the current value of the filter field will be read from the query defined for the object class as sub-node of the **Queries** node.
- **View Control**: Select the filter field the rule should apply to.
- **Comparison Operator**: Select the operator to be used to compare the current filter field value with the value returned in the dataset column defined in the **Data Set Column** attribute.
- **Data Set Column**: Enter the name of the dataset column in the query defined for the object class within the **Queries** node that returns the value to be compared with the filter field using the operator.

10) Indicator rules can be defined for the custom diagram view. The definition of indicator rules is identical to the definition of indicator rules for all configured reports. Indicators will be displayed for the objects both on the diagram view items that are standard for the diagram or the diagram view items that are defined for the custom diagram view. The indicator rule determines which objects will display indicator icons independent from all other configurations in the diagram view.

    To add an indicator rule, right-click the **Indicator Rules** node and select **Add New Indicator Rule**. For information about how to define indicator rules, see Defining Indicator Rules.

Creating a Configured Report Listing All Available Diagrams for an Object

A diagram list report lists the diagrams defined for an object. The diagrams can be based on multiple custom diagram definitions. For each custom diagram definition, a configured diagram view report must first be created to open the diagram. The diagram list report specifies which custom diagram view report shall be used to open diagrams defined for each available custom diagram definition.

The `NAME` property of the object class `ReportDiagram` stores the name that is defined by the user when creating a diagram in the configured diagram list report. If a custom diagram definition is added to the configured diagram list report that is also used to create a standard diagram for the object, the standard diagram will also be added to the listed diagrams. The naming convention for the standard diagram is "<Caption of the custom diagram definition> for <object>", where the information about the object is defined by the image properties defined in the relevant class settings.

To create a diagram list report:

1) Go to the **Reports** tab, right-click the **Reports** folder or a sub-folder thereof and select **Create New Report**. You will see the new configured report listed below the selected folder.

    As an alternative to defining all attributes for a new configured report, you can copy and paste an existing configured report and make modifications, as needed. To do so, right-click the existing configured report that you want to copy and select **Copy**. Right-click the new configured report and select **Paste**. Modify the necessary attributes described below.

2) In the attribute window, define the following attributes for the configured report.
    - **Type**: Select `Custom`.
    - **Name**: Optionally, change the technical name of the configured report. The technical name of the configured report must be unique.

The technical name is used for identification of the configured report in the context of technical processes such as merging the report configuration with a report configuration saved as an XML file. Therefore, the following is required when defining the name of a configured report:

- The name must be unique. The name must be unique regardless of the report folder it is structured in.
- The name may not be changed once a configured report has been created.
- A validation mechanism checks for correct syntax when defining a technical name. Please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.
- If the technical name of the configured report is changed, the name will be correctly updated in other configuration objects referencing the changed report during design time.

Please note that the name of a changed configuration object will not be updated in guide pages or guide views. If you plan to change the name of a configuration object, the reference in the navigation page should be changed prior to changing the configuration object name. You can use the **Show Usage** functionality in the context menu of the configured report to evaluate whether the configured report has been added to a guide view or guide page.

- **Caption**: Enter a caption for the configured report that is displayed on the Alfabet user interface for the configured report. The caption of the configured report may exceed the conventional 64 character limit.
- **Description**: Provide a short information about the configured report that is useful to Alfabet users. This comment will be displayed in the object profile under the page view caption as short description.
- **Technical Comment**: Provide information that is useful for solution designers working with Alfabet Expand in order to better understand the configured report from a technical perspective. The **Technical Comment** will not be displayed in the user interface.
- **Template**: Select `DiagramListReport`.
- **Help Index**: Specify the location of the external Help file using the following syntax:(For example:). The external help file will be displayed when the user clicks the context-sensitive Help link in the standardHelp menu.

Server variables can be used in the Help links. Server variables allow you to define all or part of the URL definition in the server alias configuration of the Alfabet Server instead of directly defining it in the configuration in Alfabet Expand. Use of server variables is useful, for example, in order to change a Web server name defined in multiple URL definitions in Alfabet Expand. Instead of changing all URL definitions, only the variable value in the server alias configuration must be updated. The use of server variables is described in the section Using Server Variables in Web Link and Database Server-Related Specifications in the chapter Configuring Reports.

- **Execute on Enter**: Set to `True` to execute the configured report with empty filter settings when the user opens the configured report on the Alfabet interface. Set to `False` to open the

configured report without executing the Alfabet query with empty filter settings. By default, this attribute is set to `True`. It should only be set to `False` when a configured report would result in an exceedingly big dataset when executed with empty filter settings.

When you set the **Execute on Enter** attribute to `False`, you must make sure that the **Submit** button is available in the custom report view of the configured report. If the **Submit** button is deleted from the custom report view, the configured report cannot be displayed. The setting and execution of the **Execute on Enter** attribute is independent of the definition of filters for a configured report. A configured report without filters must nevertheless have a **Submit** button when the **Execute on Enter** attribute is set to `False`.

- **Selector Behavior**: The attribute can be used to exclude configured reports that are defined for special purposes from being selectable in the **Configured Reports** table. By default, the attribute is set to `Visible` and the configured report is visible in the selector for adding configured reports to the **Configured Reports** table if all other conditions for access to the configured report apply. If the attribute is set to `NotVisible`, the configured report is excluded from the selector for adding configured reports to the **Configured Reports** functionality. Please note however, that this setting does not exclude the configured report from any standard views or configured views and selectors, but exclusively from the standard selector for configured reports implemented in Alfabet. The attribute cannot be edited directly in the attribute window. To change the setting, right-click the configured report and select **Set Report Selector Behavior to 'Not Visible'**.
- **Can Create Express View**: Select `True` if an express view may be created for the report. Select `False` if an express view may not be created for the report. Please note that for reports that have a custom report view, the setting must be consistent with the setting of the attribute **Can Create Express View** of the custom report view.

An express view is a link to a specific Alfabet view displaying data that allows individuals inside or outside of the user community to view information in Alfabet. When the express view is created, an email notification is automatically mailed to a specified recipient. The recipient receives a URL that allows him/her to access the current page view in Alfabet. For more information, see the section *Sending and Receiving Express Views* in the reference manual *Getting Started with Alfabet*.

3) In the explorer, right-click the configured report and select **Create Configured Report View**. The view editor opens. In the explorer, the custom report view is displayed as a subordinate object of the configured report. The attributes for the custom report view are displayed in the attribute window. You can optionally edit the following attributes:

- **Name**: Optionally you can change the name for the custom report view. The **Name** must be unique for custom report views.
- **Can Create Bookmark**: Select `True` if a bookmark may be created for the configured report. Select `False` if a bookmark may not be created for the configured report.
- **Can Create Express View**: Select `True` if an express view may be created for the configured report. Select `False` if an express view may not be created for the configured report. Please note that the setting must be consistent with the setting of the **Can Create Express View** attribute of the configured report.

The custom report view initiates the **Report Assistant**. A custom report view shall never be deleted from the configured report after the **Report Assistant** has already been used to configure the configured report. When you delete the custom report view of an

existing template-based configured report and create a new one, the **Report Assistant** will be reset to the example specified by default and your configuration is lost.

4) In the view editor, the filter panel displays a filter field to select a custom diagram definition. The name of the filter field is `@DIAGRAMDEF`. If you do not want users to be able to filter the list according to custom diagram definitions, you can remove the filter panel.
5) In the explorer, right-click the configured report and select **Start Alfabet Report Assistant**. The **Report Assistant** opens.
6) For each custom diagram definition that users shall be able to create diagrams for, right-click the **Definitions** node and select **Add New Definition**. A new definition is added as sub-node to the **Definitions** node.
7) Set the following attributes of each added definition node:
   - **Diagram Def**: Select the custom diagram definition that the diagram shall be based on.
   - **Navigation Report**: Select the configured diagram view report that shall be used to display diagrams based on the selected custom diagram definition.
8) Right-click the **Queries** node and select **Add New Query**. A **Query** node is added as sub-node to the **Queries** node.
9) Define the tabular dataset displayed in the configured diagram list report either with a native SQL query in either of the **Native SQL** or **Query as Text** attributes or with an Alfabet query in either the **Alfabet Query** or **Query as Text** attributes.

   The **Query as Text** attribute opens a simple text editor, whereas the **Alfabet Query** attribute opens the Alfabet query builder and the **Native SQL** attribute opens a native SQL editor with a special tab for the definition of Alfabet instructions.

   The query should match the following criteria:

   - If you define an Alfabet query, the `FIND` class must be specified as `ReportDiagram`. If you define a native SQL query, the first argument of the `SELECT` statement must return the `REFSTR` of the object class `ReportDiagram`.
   - If you did not remove the filter panel, the query should contain a `WHERE` statement comparing the setting of the filter field `@DIAGRAMDEF` with the `DIAGRAMDEF` property of the object class `ReportDiagram`. Please note that the parameter definition in the Alfabet query builder results in a parameter:`DIAGRAMDEF` instead of `@DIAGRAMDEF`. This setting must then be changed manually via the **Query as Text** attribute to a parameter name starting with `@`.
   - The tabular dataset returned by the query should be limited to display diagrams for the current object only. A `WHERE` statement should be added that compares the `MASTEROBJECT` property of the object class `ReportDiagram` with the Alfabet query language parameter `@BASE`.

     For example:

```
SELECT REFSTR, DIAGRAMDEF_CAPTION AS 'Diagram Definition',
NAME As 'Diagram Name'

FROM REPORTDIAGRAM

WHERE REPORTDIAGRAM.MASTEROBJECT = @BASE

AND REPORTDIAGRAM.DIAGRAMDEF = @DIAGRAMDEF
```


10) Click **OK** to save the configuration.

11) In the toolbar, click the **Save**  button to save your changes.

12) In the explorer, right-click the configured report and select **Review Report**. The configured report opens in a web browser. If the results are not as expected, you can edit the configured report until the output of the configured report meets your expectations.

13) Right-click the configured report and select **Set Report State to 'Active'** in the drop-down menu.

Configuring the Sizes of Diagram Items in Automatically Generated Diagrams

Automatically-generated diagrams such as the *Information Flow Diagram Page View*, *Interface System Diagram Page View*, or the *Business Data Usage Diagram* typically display an **Item Size** field that allow users to select which size to use to display the diagram items (which typically represent applications) in the diagram. Because some diagrams may be highly complex and display a significant number of objects, you can improve the layout of the diagram by configuring one or more diagram sizes. For example, you can configure that the sizes small (30x13 mm), medium (40x15 mm), and large (50x20 mm). In additional, you can also specify that the icon, object name, and attribute information will not be displayed in the diagram. In this case, the user can point to the diagram item to display a tooltip with the name or click-and-hold to open the preview.

The diagram item sizes are configured in the XML element ***UserItemSizeDef*** in the XML object ***DiagramInformationFlowDef***.

```
<DiagramInformationFlowDef LinkCountDecoration = "true" >
    <InformationFlowClassDef
        ClassName = "InformationFlow"
        Properties = "ID,Name,StartDate,EndDate,ObjectState,
                      Data.Name,InterfaceSystems.Name,
                      TargetService.Operations.Name,ConnectionType.Name,
                      ConnectionMethod.Name,ConnectionFrequency.Name"
        Position = "PureCenter"
        BackColor = "White"
    />

    <UserItemSizeDef
        Name = "Small"
        Caption = "Small"
        Width = "30"
        Height = "13"
        Tiny = "true"
    />
    <UserItemSizeDef
        Name = "Medium"
        Caption = "Medium"
        Width = "40"
        Height = "15"
        Tiny = "true"
    />
    <UserItemSizeDef
        Name = "Large"
        Caption = "Large"
        Width = "50"
        Height = "20"
        Tiny = "false"
    />
</DiagramInformationFlowDef>
```

FIGURE: Configuration of XML element UserItemSizeDef

Please note that the **Item Size** field is only displayed in automatically-generated diagrams and is not relevant for diagrams that are designed in the Alfabet Diagram Designer. The size of diagram

items in diagrams is controlled either by means of the size configured for a custom diagram item template or by the design of the diagram item in the context of the Alfabet Diagram Designer.

To configure the sizes available in the **Item Size** field:

1) Go to the **Presentation** tab and expand the **XML Objects** folder.
2) Right-click the XML object ***DiagramInformationFlowDef*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) Below the XML element ***DiagramInformationFlowDef***, create an XML element ***UserItemSizeDef*** for each option you want to define in the **Item Size** field. Configure the following XML attributes for each XML element ***UserItemSizeDef***, as needed:
   - `Name`: Define the technical name of the diagram item size. The name is not visible in the Alfabet interface.
   - `Caption`: Define a caption for the diagram item size. Captions will be visible in a drop-down field in the Alfabet interface.
   - `Width`: Enter the width of the diagram item size in mm.
   - `Height`: Enter the height of the diagram item size in mm.
   - `Tiny`: Enter "`false`" to specify that information such as name, icon, or attributes IS displayed if the diagram item size is selected. Enter "`true`" to specify that information such as name, icon, or attributes is NOT displayed if the diagram item size is selected. The user can point to the diagram item to display a tooltip with the object's name or click-and-hold to open the preview of the object.
4) In the toolbar, click the **Save**  button to save the XML definition.

Configuring the Visualization of Connection Items and Subordinate Object in Diagrams

Various aspects can be configured in terms of the default visualization of connection items in diagrams in the *Information Flow Diagram Page View*, *Business Data Usage Diagram*, *Interface System Diagram Page View*, *Standard Application Diagram Page View*, *Business Processes Diagram*, *Platform Diagrams Page View*, *Network Diagrams Page View* and the *Migration Diagram Page View* as well as in diagram view reports displaying diagrams based on a custom diagram definition. These views will typically display a filter field such as the **Information Flow Attribute**, **Migration Rule Attribute**, **Network Route Attribute,** or **Connection Attributes** filter field that allows the user to select one or more attributes to display on connection items in the diagram. In the figure below, the **Connection Type** attribute is displayed on the information flows in an application diagram.

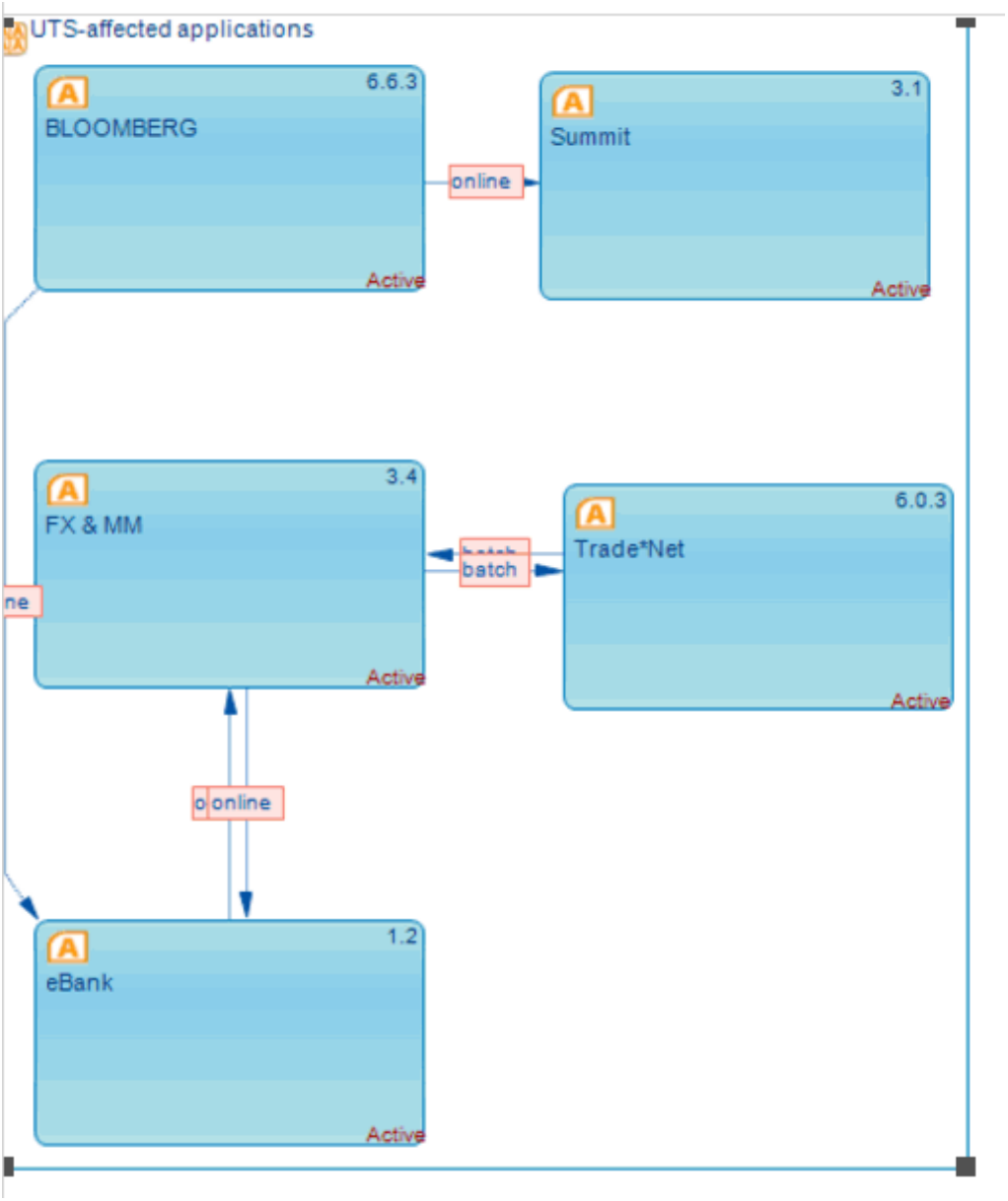


FIGURE: Information flows displaying Connection Type attributes in decoration boxes

The XML object ***DiagramInformationFlowDef*** allows you to specify which standard or custom attributes may be displayed on the connection items. Connection items in standard diagrams can be information flows, business process information flows, solution information flows, platform information flows, and migration rules. In diagram based on a custom diagram definition, other object classes may be used as connection items. In addition to the attributes to be selected, the back color and border color of the decoration boxes displaying the attributes in the diagram, and the maximum number of diagram items and connection items that may be displayed in a diagram.

The XML object ***DiagramInformationFlowDef*** also allows you to specify the sizes that can be selected for the diagram items in the **Diagram Item Size** field in diagrams. This is described in the section Configuring the Sizes of Diagram Items in Automatically Generated Diagrams.

The following image displays the configuration of the class Information Flow in the XML object ***DiagramInformationFlowDef***.


```
<DiagramInformationFlowDef LinkCountDecoration = "true"
NodeNumberThreshold = "10" LinkNumberThreshold = "30">

    <InformationFlowClassDef
        ClassName = "InformationFlow"
        Properties = "ID,Name,StartDate,EndDate,ObjectState,
                    Data.Name,InterfaceSystems.Name,
                    TargetService.Operations.Name,ConnectionType.Name,
                    ConnectionMethod.Name,ConnectionFrequency.Name"
        Position = "PureCenter"
        BackColor = "MistyRose"
        BorderColor = "Salmon"
    />
    />
    <InformationFlowClassDef
        ClassName = "PlatformInformationFlow"
        Properties = "Name,ConnectionType.Name,ConnectionMethod.Name,
                    ConnectionDataFormat.Name,ConnectionFrequency.Name,
                    Operation.Name"
        Position = "PureCenter"
        BackColor = "White"
    />
    />
    <UserItemSizeDef
        Name = "Small"
        Caption = "Small"
        Width = "30"
        Height = "13"
        Tiny = "true"
    />
</DiagramInformationFlowDef >
```

FIGURE: Configuration of XML elements DiagramInformationFlowDef and InformationFlowClassDef

To edit the XML object ***DiagramInformationFlowDef***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder.
2) Right-click the XML object ***DiagramInformationFlowDef*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) In the XML element ***DiagramInformationFlowDef***, configure the following XML attributes as needed:

- `LinkCountDecoration`: Enter "`true`" to specify that the number of connection items between the nodes (source/target objects) is displayed if the **Layout** attribute is set to **Condensed** and no attributes are displayed on the connection items. Enter "`false`" if no numbers should be displayed between connection items if the **Layout** attribute is set to **Condensed** and no attributes are displayed on the connection items
- `NodeNumberThreshold`: Enter a numerical value as the limit for the number of nodes (typically diagram items representing applications) for which connection items should be generated. Remaining nodes exceeding the threshold value will not be displayed. The reduction of the number of objects in the diagram will improve performance for diagrams with an excessive number of application interfaces. The default value is "500".
- `ChildNodeThreshhold`: Some objects displayed in the diagram have referenced objects that can be added to the diagram. The subordinate objects may be automatically added to the object it references via the **Add Subordinates for Object** or the **Update Subordinates...** functionalities. Enter a numerical value as the limit for the number of child objects that may be generated for a diagram. Remaining nodes exceeding the threshold value will not be displayed. The reduction of the number of objects in the diagram will improve performance for diagrams with an excessive number of application interfaces. The default value is -1 which equates to no limit in order to provide backward compatibility.

- `LinkNumberThreshold`: Enter a numerical value as the limit for the number of connection items that should be generated. Remaining connection items exceeding the threshold value will not be displayed. The default value is "500".

  The XML attributes `NodeNumberThreshold` and `NodeNumberThreshold` are relevant for the following views:*Information Flow Diagram Page View* (`APP_DiagramReport`, `APP_Connections_Diagram`, `DVC_DiagramReport`), *Business Data Usage Diagram* (`BO_DataUsageDiagram`, `ICTO_DataUsageDiagram`), and the *Interface System Diagram Page View*, (`COM_InterfaceSystemDiagram`).

4) Below the XML element ***DiagramInformationFlowDef***, create an XML element ***InformationFlowClassDef*** for each object class representing a connection item that you want to configure. Configure the following XML attributes for each XML element ***InformationFlowClassDef***, as needed:

- `ClassName`: Enter the name of the object class representing the connection items that you want to configure. Permissible values for Alfabet standard diagrams include `InformationFlow`, `BPInformationFlow`, `SolutionInformationFlow`, `PlatformInformationFlow`, `NetworkRoute`, and `MigrationLink`. For diagrams based on a custom diagram definition, all classes that are specified as a connecting class in **Connection** elements of the custom diagram definition can be defined. If no class is defined, then all possible object classes are included by default. Class names that are misspelled will be ignored. The value of the **Name** attribute of the object class must be used to specify the object class.

- `Properties`: Enter a comma-separated list of standard and custom object class properties of the relevant object classes that may be displayed for the connection item. These properties will be available in the attribute filter available in the relevant diagram page view. The user can select one or more of these properties to display its values on the connection item. If multiple properties are selected for a custom diagram, they will be separated by a linebreak in the decoration box. If no property is defined in the XML attribute `Properties`, then all possible properties will be included per default. The properties will be displayed lexicographically for the selected interface language. You can define any of the following:

  - To add a property of the object class defined with the XML attribute `ClassName`, enter the name of the object class property.

    For example, if `ClassName` is `InformationFlow` and you would like the object state of the information flow to be displayed in the filter field, enter `ObjectState`.

  - If the object class defined with the XML attribute `ClassName` has object class properties of the type `Reference` or `ReferenceArray`, you can add any custom or standard property of the object class or object classes that are the reference target to the list. Enter the name of the object class property of the type `Reference` or `ReferenceArray` that establishes the reference and the name of the object class property of the target class that shall be displayed separated with a dot.

    For example, if `ClassName` is `InformationFlow` and you would like to add the name of the target object referenced via the property `From` of the information flow to the filter field, enter `From.Name`.

Please note the following:

- The attribute **Type Info** of the object class property of the type Reference or ReferenceArray displays the allowed target classes for the reference. If `Artifact` is defined as **Type Info**, any object class that represents instances of the IT architecture can be referenced.

  For example, for the object class property `From` for the object class `InformationFlow`, the following object classes are referenced by means of the **Type Info** attribute: `Application`, `Component`, `LocalComponent`, `Peripheral`, `Device`. Any of the classes may be a source object of the class `InformationFlow`. To specify the attributes of the source applications of information flows, you would then list the standard and custom object class properties of the class `Application` in the XML attribute `Properties`. If you wanted to also include the attributes of the source peripherals, you would also include the standard and custom object class properties of the class `Peripheral` in the XML attribute `Properties`.

- If multiple target object classes have object class properties with the same name, like For example, the property `Name`, the definition is valid for all target object classes. If you add a target object class property specific for one of multiple target object classes to the list, the object class property will only be displayed if the connection references the relevant object class.

- `Position`: Define the position of the attribute on the connection items in the diagram. Permissible values include:
  - `PureCenter`: Attribute is placed on the middle segment of the information flow
  - `Start`: Attribute is placed along the first segment of the connection above or left to the line
  - `End`: Attribute is placed along the last segment of the connection above or left to the line
  - `Center`:Attribute is placed along the middle segment of the connection above or left to the line
- `BackColor`: Define the background color for the caption of the attribute on the connection items. The value entered should be a Windows, Web, hexadecimal, or RGB color value.
- `BorderColor`: The attribute information is displayed in a rectangle called a decoration. Define the border color for the decorations associated with connection items. If the diagram designer designs a line to connect the decoration box to the connection item, the connecting line will also have this color. The specified color will be applied to the decorations that are automatically generated as well as those defined by the user. If no color is specified, the border will be light grey per default. The value entered should be a Windows, Web, hexadecimal, or RGB color value. For more information about the specification of decoration boxes within the context of the Alfabet Diagram Designer, see the section *Modifying the Decoration Boxes Displayed for Connection Items* in the reference manual *Designing IT Landscape Diagrams in Alfabet*.

5) In the toolbar, click the **Save**  button to save the XML definition.

Configuring the Default Settings for the Alfabet Diagram Designer

The XML object ***DiagramOptions*** allows you to configure the default settings for the attribute window in the Alfabet Diagram Designer. The XML object ***DiagramOptions*** allows you to configure such aspects as the default display of connection items (straight or elbowed lines), ruler and grid settings, and the default settings for the print preview and print functionalities. A user working in the Alfabet Diagram Designer can change the default setting as needed via the menu option **Actions** > **Diagram Settings**. For more information about defining the diagram settings in the context of the Alfabet Diagram Designer, see the section *Setting Up the Diagram* in the reference manual *Designing IT Landscape Diagrams in Alfabet*.

Designing diagrams is an involved process and is described in detail in the reference manual *Designing IT Landscape Diagrams in Alfabet*.

To edit the XML object ***DiagramOptions***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder.
2) Right-click the XML object ***DiagramOptions*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) Define the XML attributes, as needed. The table below displays the XML elements and XML attributes that can be edited for the XML object ***DiagramOptions***:

XML Element (bold) / XML Attribute	Explanation
DiagramOptions	
`DrawGrid`	Enter "`true`" if a grid should be displayed in the Alfabet Diagram Designer. Enter "`false`" if a grid should not be displayed. The grid version can be defined in the XML attribute `GridMode`. This attribute is only applicable to the display in the Alfabet Diagram Designer and not to the resulting diagram in the diagram page view.
`Rulers`	Enter "`true`" if a rulers should be displayed in the Alfabet Diagram Designer. Enter "`false`" if rulers should not be displayed. This XML attribute is only applicable to the display in the Alfabet Diagram Designer and not to the resulting diagram in the diagram page view.
`HorGridSize`	Enter an integer to determine the horizontal grid size used in millimeters in the Alfabet Diagram Designer. This XML attribute is only applicable to the display in the Alfabet Diagram Designer and not to the resulting diagram in the diagram page view.
`VerGridSize`	Enter an integer to determine the vertical grid size used in millimeters in the Alfabet Diagram Designer. This XML attribute is only applicable to the

XML Element (bold) / XML Attribute	Explanation
	display in the Alfabet Diagram Designer and not to the resulting diagram in the diagram page view.
PhysicalMode	Enter "true" to apply the margin definition to the diagram in the Alfabet Diagram Designer. The printed image will include all areas that do not belong to the margin definition. Enter "false" if the margin definition should not be applied to the diagram. The layout is typically distributed to four pages, depending on the paper size and margin definition. The pages are printed individually.
ViewPrintLayout	Enter "true" to apply the print layout to the diagram. The margins as well as header/footer are greyed out. Enter "false" if the print layout should not be applied to the diagram.
LeftMargin	Enter an integer to determine the applicable left margin in millimeters in the Alfabet Diagram Designer. This XML attribute is only relevant when displaying the diagram in the Alfabet Diagram Designer in print layout or when printing the diagram.
RightMargin	Enter an integer to determine the applicable right margin in millimeters in the Alfabet Diagram Designer. This XML attribute is only relevant when displaying the diagram in the Alfabet Diagram Designer in print layout or when printing the diagram.
TopMargin	Enter an integer to determine the applicable top margin in millimeters in the Alfabet Diagram Designer. This XML attribute is only relevant when displaying the diagram in the Alfabet Diagram Designer in print layout or when printing the diagram.
BottomMargin	Enter an integer to determine the applicable bottom margin in millimeters in the Alfabet Diagram Designer. This XML attribute is only relevant when displaying the diagram in the Alfabet Diagram Designer in print layout or when printing the diagram.
HeaderMargin	Enter an integer to determine the applicable header margin (for example, the margin between the header and the main diagram) in millimeters in the Alfabet Diagram Designer. This XML attribute is only relevant when displaying the diagram in the Alfabet Diagram Designer in print layout or when printing the diagram.
FooterMargin	Enter an integer to determine the applicable footer margin (for example, the margin between the footer and the main diagram) in millimeters in the Alfabet Diagram Designer. This XML attribute is only relevant when

XML Element (bold) / XML Attribute	Explanation
	displaying the diagram in the Alfabet Diagram Designer in print layout or when printing the diagram.
`RepeatHeader`	Enter one of the following XML attributes to determine how the header should be applied to the various pages of the printed diagram. Options include: • `EveryPage`: Header is applied to every page of the printed output. • `ButFirstPage`: Header is applied to every but the first page of the printed output. • `FirstPage`: Header is only applied to the first page of the printed output. • `None`: Header does not apply to the printed output.
`RepeatFooter`	Enter one of the following XML attributes to determine how the footer should be applied to the various pages of the printed diagram. Options include: • `EveryPage`: Footer is applied to every page of the printed output. • `ButFirstPage`: Footer is applied to every but the first page of the printed output. • `FirstPage`: Footer is only applied to the first page of the printed output. • `None`: Footer does not apply to the printed output.
`ElbowedConnection`	Enter "`true`" if the connection items (information flows, migration rules, etc.) that are automatically added to the diagram should be drawn as an elbow line (with 90° angle). Enter "`false`" if the connection items should be created as a straight line.
`CenterConnection`	Enter "`true`" if the end points of connection items should be determined by the user via the drag-and-drop action. Enter "`false`" if the end points of connection items should always be drawn from the center of one diagram object to the center of the other diagram object.
`PrintScale`	Enter an integer to determine the scaling factor that should be applied to the diagram when printing.

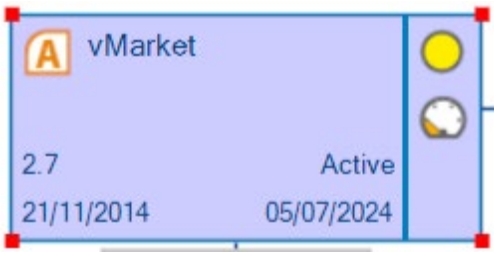
XML Element (bold) / XML Attribute	Explanation
`Landscape`	Enter "`true`" if the diagram should use the specified paper size in landscape mode. Enter "`false`" if the diagram should use the specified paper size in portrait mode.
`GridMode`	Enter "`true`" if the grid should be displayed as grid lines in the Alfabet Diagram Designer. Enter "`false`" if the grid should be displayed as grid points.
`LockToolBoxItem`	Enter "`true`" if the mouse pointer icon should continue to represent the item selected in the **Toolbox Items** pane in the Alfabet Diagram Designer (for example, Application) until a different item is selected in the **Toolbox Items** pane. Enter "`false`" if the mouse pointer icon should revert to the pointer tool upon completion of every interaction.
`Format`	Enter one of the following XML attributes to determine the paper format to be used as the default for diagrams. Permissible options are: • `A4 210x297`: Standard A4 paper size • `A3 297x420`: Standard A3 paper size • `A2 420x594`: Standard A2 paper size • `A1 594x840`: Standard A1 paper size • `A0 840x1186`: Standard A0 paper size • `Letter`: US Letter paper size (216x279 cm or 8.5x11 in) • `Legal`: US Legal paper size (216x356 cm or 8.5x14 in) • `11x17`: US 11x17 paper size (279x432 cm)
`WarnOnSemanticLayoutChange`	Enter "`true`" if a warning message should be displayed if changes made in a diagram result in a semantic change.
`AutomaticallyRescanDiagramOnLoad`	Enter "`true`" to automatically display semantic changes made to an object in a diagram when the diagram is reloaded in the Alfabet user interface or Alfabet Diagram Designer. Enter "`false`" if semantic changes shall only be updated to the diagram when the **Update** button is triggered in the diagram's view in the Alfabet user interface or when the **Refresh Diagram** option in the **Semantic Actions** menu is triggered in the Alfabet Diagram Designer. Please note that the XML attribute `AutomaticallyRescanDiagramOnLoad` i must be set to "`false`" if the **Automatically Rescan on Load** checkbox is selected in the **Diagram Settings** editor in the Alfabet Diagram Designer. If the XML attribute `AutomaticallyRescanDiagramOnLoad` is set

XML Element (bold) / XML Attribute	Explanation
	to "`true`", the semantic changes will not be explicitly updated to the diagram regardless of the definition of the **Automatically Rescan on Load** attribute.
`SaveWorkinProgress`	Enter "`true`" if the diagram shall be automatically saved even if a session time-out occurs. If a diagram is opened after a session has timed out, the user will be asked if the diagram that was in progress when the session expired should be opened and replace the most recently saved diagram. A standard message is displayed for semantic groupings defined in Alfabet standard diagrams.
`SaveFrequency`	Enter the number of minutes in order to define how frequently the diagram shall be automatically saved.

4) In the toolbar, click the **Save** button to save the XML definition.

Configuring the Color, Opacity, and Size of the Selection Handles of Diagram Items

The attributes available in the **Visual Selection Layout** section of the GUI scheme configuration allow the color, opacity, and size of the handles surrounding selected objects in diagrams and matrices to be specified.



To specify the handles, expand the **GUI Schemes** node in the **Presentation** tab and select the relevant GUI scheme you want to define. In the **Application** section of the attribute window, expand the **Visual Selection Layout** and specify the **Color**, **Opacity**, and **Size** attributes as needed. In the toolbar, click the **Save** button to save the definition.

For more information, see the section Configuring GUI Scheme Definitions for the Alfabet Interface in the chapter Configuring the Visualization of the Alfabet Interface as well as a detailed documentation of all GUI scheme attributes in the chapter *Overview of GUI Scheme Attributes* in reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

Configuring the Assignment Capability

Assignments allow users to collaborate with one another about objects in the enterprise architecture.

An assignment is a task that is defined for a selected object and assigned to a specific user. The assignee is expected to provide the required input for the object by a specified due date. Assignments can be defined to be optional or mandatory. Email notifications may be configured to be sent in the context of the assignment capability.

The assignment capability is available for all object classes that support the use of assignments.

- For an overview of the object classes for which assignments can be created, see the chapter *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.
- For general information about the use of assignments in Alfabet, see the section *Sending and Receiving Assignments for Alfabet Objects* in the reference manual *Getting Started with Alfabet*.

A number of configurations are required in order to implement the full range of functionality available with the assignment capability. The following configurations must be completed by the solution designer. These configurations are carried out in the configuration tool Alfabet Expand and are explained in the section below:

- If emails are to be sent in the context of the assignment capability, the email capability must be activated and the content of the text templates used for the emails may need to be modified. For more information about the configuration required for email notifications in the context of the assignment capability, see the section Configuring Email Notifications in the Context of Assignments.
- The statuses to be used in the context of your enterprise's assignment workflow may be configured in the XML object ***ReleaseStatusDefs***.
- A **Notify Authorized User** button may be displayed in the toolbar of an object profile in order to allow an assignment to be quickly created and sent to the authorized user of a selected object. The implementation of this button must be configured for all relevant object classes for which it is to be implemented. For more information about configuring the display of the **Notify Authorized User** button, see the section Configuring the Display of the Notify Authorized User Button for Assignment Creation.
- The views needed for the assignment capability are available in the relevant user profiles. Please note the following regarding the relevant views required to work with the assignment capability:
  - The **My Assignments** functionality (`Home_Assignments`) allows a user to view all assignments that he/she is responsible for. The **Sent Assignments** functionality (`Home_SentAssignments`) allows the user who has created assignments to keep track of all assignments that he/she is the originator of. Both functionalities allow users to set reminders for themselves about upcoming assignment deadlines. As the solution designer, you must ensure that the functions **My Assignments** (`Home_Assignments`) and **Sent Assignments** (`Home_SentAssignments`) are available in the relevant user profiles you configure for your Alfabet community. For more information about configuring user profiles, see the section Making Functionalities Accessible to a User Profile in the chapter Configuring User Profiles for the User Community.
  - The **Assignments** page view (`ObjectAssignments`) allows users to create new assignments for a selected object. The view also features a notepad functionality that allows you to

communicate about the assignment or the object targeted by the assignment. The **Assignments** page view (`ObjectAssignments`) should be available in all object views for which assignments should be created in your enterprise. For more information about configuring object views, see the section Configuring Object Views.

The language displayed for a user's email notifications is specified by a user administrator in the **Email Notification Language** attribute for the relevant user in the **User Administration** functionality accessible via the `Admin` user profile. For more information, see the section *Defining and Managing Users* in the reference manual *User and Solution Administration*.

The following configuration requirements must be completed by the system administrator in the tool Alfabet Administrator:

- All Alfabet functionalities for which the email capability is to be implemented require the setup of a connection to an SMTP server for outgoing email. For more information, see the section *Activating the Dispatch of Email Notifications in Alfabet* in the reference manual *System Administration*.
- The user profile used to open the Alfabet views targeted by hyperlinks in the email is, by default, the user profile that the sender was logged in with when the view was sent/triggered. However, your system administrator may configure that Alfabet views in email notifications are to be opened using the recipient's user profile. For more information, see the section *Configuring the Setup To Open the Alfabet Interface via Links in Email Notifications* in the reference manual *System Administration*.

In order to trigger functionalities associated with target dates, a batch job must be configured and executed by your system administrator. For more information about configuring a batch job for the review of assignment target dates, see the section *Triggering Target Date Control for the Assignments Capability* in the reference manual *System Administration*.

The following information is available

Configuring Email Notifications in the Context of Assignments

The assignment capability allows for email notifications to be automatically sent in various contexts in the assignment workflow. Email notifications may be sent automatically in the following situations in the context of the assignment capability:

- to the assignee when a new assignment is created
- to a new assignee when an assignment is reassigned
- to the originator of the assignment when the assignment's status is changed

- to either the assignee and/or the originator when a notepad entry is made about the assignment
- to the assignee when the assignment nears its impending target date
- to the originator if the assignment is not completed by its target date

In order to send the email notifications in the contexts of assignments, the email capability must be explicitly activated for the assignments. Furthermore, default text templates that email notifications are based on are provided by Software AG. These text templates can be used as is or they can be edited according to the needs of your enterprise.

The following configuration requirements must be completed by the system administrator in the tool Alfabet Administrator:

- All Alfabet functionalities for which the email capability is to be implemented require the setup of a connection to an SMTP server for outgoing email. For more information, see the section *Activating the Dispatch of Email Notifications in Alfabet* in the reference manual *System Administration*.
- The user profile used to open the Alfabet views targeted by hyperlinks in the email is, by default, the user profile that the sender was logged in with when the view was sent/triggered. However, your system administrator may configure that Alfabet views in email notifications are to be opened using the recipient's user profile. For more information, see the section *Configuring the Setup To Open the Alfabet Interface via Links in Email Notifications* in the reference manual *System Administration*.

For more information about the configuration of email notifications in the context of the assignment capability, see the following sections below:

- Implementing the Email Capability for the Assignment Functionality
- Defining the Text Templates Used for the Assignment Capability

Implementing the Email Capability for the Assignment Functionality

The XML object ***SolutionOptions*** allows you to define whether email notifications should be sent in the context of assignments. Email notifications may be sent automatically in the following situations in the context of the assignment capability:

- to the assignee when a new assignment is created
- to a new assignee when an assignment is reassigned
- to the originator of the assignment when the assignment's status is changed
- to either the assignee and/or the originator when a notepad entry is made about the assignment
- to the originator if the assignment is not completed by its target date
- to the assignee when the assignment nears its impending target date

The configuration of the email capability via the XML attribute `SendAssignmentsMails` in the XML object ***SolutionOptions*** is a prerequisite for the implementation of the reminder email notifications. However, setting the XML attribute `SendAssignmentsMails` to "`true`" does not result

in reminder emails being automatically sent per default. Email notifications will only be sent to the assignee if he/she selects the **Send Reminder Notifications** option when defining assignment reminders in the **My Assignments** functionality.

The following configuration requirements must be completed by the system administrator in the tool Alfabet Administrator:

- All Alfabet functionalities for which the email capability is to be implemented require the setup of a connection to an SMTP server for outgoing email. For more information, see the section *Activating the Dispatch of Email Notifications in Alfabet* in the reference manual *System Administration*.
- The user profile used to open the Alfabet views targeted by hyperlinks in the email is, by default, the user profile that the sender was logged in with when the view was sent/triggered. However, your system administrator may configure that Alfabet views in email notifications are to be opened using the recipient's user profile. For more information, see the section *Configuring the Setup To Open the Alfabet Interface via Links in Email Notifications* in the reference manual *System Administration*.

To activate the email capability in the context of assignments in the XML object ***SolutionOptions***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder.
2) Right-click the XML object ***SolutionOptions*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) In the XML attribute `SendAssignmentsMails`, enter "`true`" if an email notification should be automatically sent when an assignment is created (email sent to assignee) and the assignment status is changed (email sent to assignment originator). Enter "`false`" if email notifications should not be automatically generated.

   Setting the XML attribute `SendAssignmentsMails` to "`true`" also enables the sending of emails when notepad entries are made in the context of the strategy planning and master planning capabilities.

4) In the toolbar, click the **Save**  button to save the XML definition.

Defining the Text Templates Used for the Assignment Capability

Software AG provides several default text templates that are used for the email notifications sent in the context of the assignments capability. All preconfigured text templates that are available for the assignments capability can be found in the ***ASMT*** folder  . To access the folder and available text templates, click the **Presentation** tab, expand the **Text Templates**  node in the explorer, and expand the ***ASMT*** folder  .

The text templates define the email text, relevant references to objects and their object class properties, and hyperlinks to the relevant objects in the solution interface. You can make the following changes to any text template available in Alfabet Expand:

- The **Subject** field of the email can be specified via the **Caption** attribute of the text template.
- The text template can be either formatted as ASCII or HTML. Please note that the HTML must be XML-conform HTML, compliant with HTML 5, and use standard HTML tags.
- There are three different types of variables that can be included in a text template:
    - Predefined standard variables: These are variables that are predefined by Software AG and serve the purpose of referencing the relevant data in the Alfabet database. For example, the variable `{Object:RefImage}` typically allows the base object referenced by the text template to be included in the text template. You can add, remove, or move the entire variable via copy and paste. If the expression in the curly brackets `{XXX}` is changed, the text template may fail to work correctly.
    - Object variables: A specified set of variables is available that allows information such as the object targeted by the notification or a user referenced by the object. The syntax of these references are {<ObjectClass>:<ObjectClassProperty>}, whereby the set of object classes that can be specified in <ObjectClass> are predefined. Any scalar property defined for the <ObjectClass> expression can be referenced in the <ObjectClassProperty>. For example, if an application is the base object referenced by the email, you would enter `{Object:ID}` to include the ID of an application or `{Object:Description}` to include the description of an application. You can add, remove, or move the entire variable via copy and paste.
    - Query variables: It is possible to specify an Alfabet query or SQL query to add more complex references to the text template that cannot be captured via scalar properties (for example, additional referenced information or a document link). If you plan to specify an Alfabet query or native SQL query to the text template, you must enter the variable {`Query:` <QueryName>} to the text template, whereby the <QueryName> expression is determined by the value of the **Name** attribute specified for the query text configuration object.
- The text displayed in the templates can be edited, as needed.
- Locale texts can be created for each text template in order to provide the email in the relevant secondary languages supported by your enterprise.

- For detailed information about the configuration of text templates as well as creating various language versions of the text templates for your enterprise, see the section Configuring Text Templates for Email Notifications.
- For an overview of all available text templates for the assignments capability and an explanation of the purpose of each variable, see the chapter *Overview of Preconfigured Text Templates and Their Variables* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

Defining the Statuses Used for the Assignment Capability

The XML object ***ReleaseStatusDefs*** allows you to configure the statuses available for the assignment capability as well as the transition from one status to the next. A default configuration is available in the XML object. However, the status definitions can be edited and adapted to the needs of your enterprise.

This XML object should be defined as part of the initial configuration of the Alfabet solution. These definitions should not be changed once the Alfabet database is in use.

To edit the XML object ***ReleaseStatusDefs*** for assignments

1) Go to the **Presentation** tab and expand the **XML Objects** node.
2) Right-click the XML object ***ReleaseStatusDefs*** and select **Edit XML**. The XML editor is displayed in the center pane.

```
<ReleaseStatusDef
        ClassNames = "Assignment"

        StatusSet = "Created,Accepted,In Progress,Work Completed,Returned,
                Re-Assigned,Closed"
        RetiredStatusSet = "Created,Closed"
        EditableStatusSet = "Created,Accepted,In Progress,Returned,Re-Assigned,
                Work Completed"
        PrivateStatus = "Returned"
        RedefinedStatus = "Re-Assigned"
        ApprovedStatus = "Work Completed"
        ClosedStatus = "Closed"
        DefaultStatus = "Created">

        <StatusTransition ToStatus="Created" FromStatuses=""/>
        <StatusTransition ToStatus="Accepted" FromStatuses="Created,Re-Assigned"/>
        <StatusTransition ToStatus="In Progress"  FromStatuses="Created,Accepted,
                Work Completed,Re-Assigned"/>
        <StatusTransition ToStatus="Work Completed"  FromStatuses="Created,Accepted,
                In Progress,Re-Assigned"/>
        <StatusTransition ToStatus="Returned" FromStatuses=""/>
        <StatusTransition ToStatus="Re-Assigned" FromStatuses="Created,Accepted,
                In Progress,Returned"/>
        <StatusTransition ToStatus="Closed" FromStatuses=""/>

</ReleaseStatusDef>
```

3) Edit the XML attributes in the status definition for the object class `Assignment`. The table below explains the XML attributes that may be defined.

XML Element (bold) / XML Attribute	Description
ReleaseStatusDef	
`ClassNames`	Enter `Assignment`.
`StatusSet`	Enter a comma-separated list of all statuses that are to be implemented in the assignment workflow. `"Created,Accepted,In Progress,Work Completed,Returned,Re-Assigned,Closed"`

XML Element (bold) / XML Attribute	Description
	 Only ASCII characters made be used in the name of an enumeration item. Please note that NO error will be displayed if you use characters that are not conform to ASCII.
`RetiredStatusSet`	Enter a comma-separated list of statuses defined in the XML attribute `StatusSet` in order to indicate that an assignment is retired and may be deleted. If an object with the release status specified in the XML attribute `RetiredStatusSet` may not be edited by users, it should not be listed in the XML attribute `EditableStatusSet`. Users with relevant access permissions may delete an assignment with a status defined in the XML attribute `RetiredStatusSet`. If no status is defined in the XML attribute `RetiredStatusSet`, the deletion of assignments will be forbidden for all users other than a user with administrative permissions.
`EditableStatusSet`	Enter a comma-separated list of statuses defined in the XML attribute `StatusSet` that may be edited by users with relevant access permissions. If an object with the release status specified in the XML attribute `RetiredStatusSet` may not be edited by users, it should not be listed in the XML attribute `EditableStatusSet`.
`PrivateStatus`	Enter one status defined in the XML attribute `StatusSet` that should be assigned to expired mandatory assignments. If the XML attribute `PrivateStatus` is not included in the XML object ***ReleaseStatusDefs*** of it is left unspecified, the value will be defined as `Private` per default.  Expired mandatory assignments are automatically reassigned to the assignment's originator: Only the originator will have edit permissions for the reassigned assignment. In contrast, expired optional assignment will be automatically assigned the XML attribute `ClosedStatus`.
`RedefinedStatus`	Enter one status defined in the XML attribute `StatusSet` to indicate that an assignment has been reassigned to a different user. The status value of the assignment will automatically be changed to the valued specified in the XML attribute `RedefinedStatus` if the assignment is reassigned to another user. If the XML attribute `RedefinedStatus` is not included in the XML object ***ReleaseStatusDefs*** of it is left unspecified, the value will be defined as `Redefined` per default.

XML Element (bold) / XML Attribute	Description
`ApprovedStatus`	Enter one status defined in the XML attribute `StatusSet` that indicates that the assignment is complete. Once a user assigns the status specified in the XML attribute `ApprovedStatus` to the assignment, the assignment will no longer be automatically reassigned to the originator when the target date is reached. If the XML attribute `ApprovedStatus` is not included in the XML object ***ReleaseStatusDefs*** of it is left unspecified, the value will be defined as `Approved` per default.
`ClosedStatus`	Enter one status defined in the XML attribute `StatusSet` that should be assigned to an assignment that has been completed by the assignee and has been confirmed as completed by the originator. If the XML attribute `ClosedStatus` is not included in the XML object ***ReleaseStatusDefs*** of it is left unspecified, the value will be defined as `Closed` per default. Expired optional assignments will be automatically assigned the status specified in the XML attribute `ClosedStatus`. In contrast, expired mandatory assignments will automatically be assigned the status specified in the XML attribute `PrivateStatus`.
`DefaultStatus`	Enter one status that is the default status for assignments. The default status is automatically assigned to a new assignment upon its creation.

4) The XML element ***StatusTransition*** allows the sequence of statuses to be defined. You should create an XML element ***StatusTransition*** for each status listed in the XML attribute `StatusSet`. The definition for status transitions is determined by the configuration of the XML attributes `FromStatuses` and `ToStatus`. The status defined for the XML attribute `ToStatus` can be selected by the user subsequent to the status(es) defined in the XML attribute `FromStatuses`. The table below explains the XML attributes that may be defined.

For example, an object must have the statuses `Created` or `Re-Assigned` in order to allow a transition to the status `Accepted`.

```
<StatusTransition ToStatus="Accepted"
FromStatuses="Created,Re-Assigned"/>
```

If you do not enter a value in the quotation marks, no statuses can be selected in order to transition for a status by the user.

XML Element (bold) / XML Attribute	Description
StatusTransition	
`ToStatus`	Enter one status that may follow the statuses described in the XML attribute `FromStatuses`. An XML attribute `ToStatus` definition should be made for each status in the status set.
`FromStatuses`	Enter one or more statuses in a comma-separated list that may precede the status described in the XML attribute `ToStatus`. It is recommended that a XML attribute `FromStatuses` definition should be made for each XML attribute `ToStatus` definition in the status set. Enter the word "any" in the quotation marks if all statuses may be defined for either the XML attribute `ToStatus` or the XML attribute `FromStatuses`. For example: `<StatusTransition ToStatus="Planned" FromStatuses="any"/>`

5) The XML element `Status` definitions allow tooltip texts to be defined for the statuses available for assignments.

XML Element (bold) / XML Attribute	Description
Status	Allows you to provide custom tooltips for each status. Users will see the tooltips for the statuses in the relevant editor. The tooltip text you define in the context of the status configuration will be added to the tooltip hover box for the object class property `Status`. The text defined in the XML attribute `Hint` for the statuses will be line-separated and appended below the text for the standard tooltip.
`Name`	Enter the name of the status.
`Hint`	Enter a Help text that will help users to understand the meaning of the status. It is recommended that the text is not excessively long. The value defined in the XML attribute `Hint` will be displayed as custom Help in editors, filters, and legends. The texts for the individual statuses will be line-separated. The texts defined for custom Help can be translated in the **Translation Editor**. For more information about the translation of custom terminology, see the chapter Localization and Multi-Language Support for the Alfabet Interface.

6) In the toolbar, click the **Save** button to save the XML definition.

Configuring the Display of the Notify Authorized User Button for Assignment Creation

The XML object ***AssignmentConfigDef*** allows you to determine which object classes allow for the **Notify Authorized User** button to be displayed in the toolbar of the object profile.

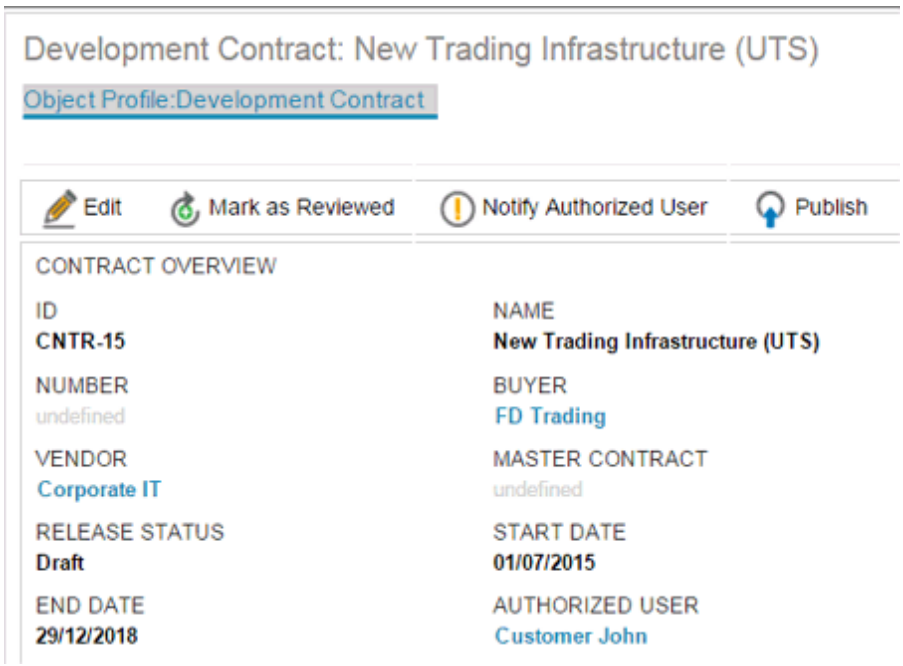


FIGURE: Notify Authorized User button in an object profile

By clicking the **Notify Authorized User** button, the user can quickly create an assignment that is assigned to the authorized user of the selected object. The assignment will be sent to the **My Assignments** functionality of the object's authorized user and, if the email capability has been configured, the authorized user will also receive an email informing him/her that a new assignment has been created.

- An assignment can be created via the **Notify Authorized User** button only for objects for which an authorized user has been defined.
- The email capability must be explicitly activated for assignments in order for email notifications to be sent when an assignment is created via the **Notify Authorized User** button. For more information, see the section Configuring Email Notifications in the Context of Assignments.
- Documents and Web links (URLs) can be attached to the assignment in the **Notification Assignment** editor available via the **Notify Authorized User** button. For more information about making documents available for attachment, see the section *Uploading Documents and Managing User Permissions to Document Folders in the Internal Document Selector* in the reference manual *User and Solution Administration*. For more

information about defining a default prefix for a Web link, see the section Configuring Default URL Prefixes for the Attachments Page View in this chapter.

To edit the XML object ***AssignmentConfigDef***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder, and expand the **Administration** folder.

2) Right-click the XML object `AssignmentConfigDef` and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.

3) Define the XML attributes, as needed. The table below displays the XML elements and XML attributes that can be edited for the XML object ***AssignmentConfigDef***:

XML Element (bold) / XML Attribute	Explanation
AssignmentConfigDef	
`NotifyResponsibleClasses`	Enter a comma-separated list of object classes for which the **Notify Authorized User** button should be available. You must enter the value of the **Name** attribute of the object class. The **Notify Authorized User** button will be added to the toolbar of the object profiles for all cobject lasses defined in this XML attribute. If no class is specified in the XML attribute `NotifyResponsibleClasses`, the **Notify Authorized User** capability will be available for all object classes for which the concept of authorized user/user group is relevant.
`NotifyResponsiblePeriod`	Enter an integer to establish the default target date for the assignment. The target date is computed based on the date when the assignment is generated plus the value defined in the XML attribute `NotifyResponsiblePeriod`. The user creating the assignment can edit the target date, as needed. **EXAMPLE**: If the XML attribute `NotifyResponsiblePeriod` is defined as 30, the authorized user will have 30 days after the date that the assignment was generated to complete the assignment. An assignment generated on Jan. 1, 2008 will have a target date of Jan. 30, 2008.

4) In the toolbar, click the **Save**  button to save the XML definition.

Configuring Standard Business Support Matrices

The XML object ***ITMapDef*** allows you to configure the layout and behavior of the standard business support matrices in Alfabet. Configurable aspects of Alfabet 's standard business support matrices include:

- the visualization of matrix objects and axis objects including, for example, their color and caption information
- the default classes displayed on the X- and Y-axes of business support matrices as well as the order of the axis objects
- the visualization of blueprint map objects in the business support matrices
- the object classes that may be defined as providers of operational, strategic, and tactical business supports
- the object classes representing operational aspects that can be displayed on business supports in business support matrices
- relevance values that allow Alfabet users to filter information based on its relevance for the current analysis of the IT landscape
- the indicator displayed on business appraisal objects in the business support matrices

The specification of the XML object ***ITMapDef*** is relevant for the configuration of views displaying business support matrices available for the following object classes:

- `Application`
- `ICTObject`
- `BusinessFunction`
- `BusinessProcess`
- `Domain`
- `MarketProduct`
- `OrgaUnit`
- `SystemBuildingBlock`
- `ITStrategyMap`
- `MasterPlanMap`
- `MasterPlanMapView`
- `SolutionMap`

For a methodological overview of planning business supports in the context of master planning and conceptualizing the enterprise's target architecture, see the reference manual *IT Planning Basic*.

Business support matrices in Alfabet can convey a wealth of information about the enterprise's IT support. For information about functionalities common to most business support matrices, see the section *Appendix:Working with Business Support Maps* in the reference manual *IT Planning Basic*.

In addition to configuring the standard business support matrices in Alfabet, you can also define a configured report to display a customized matrix that allows business supports to be viewed and edited. For more information, see Configuring Matrices for Multi-Editing of Object Relations or Business Supports in the chapter Configuring Reports.

The attributes available in the **Visual Selection Layout** section of the GUI scheme configuration allow the color, opacity, and size of the handles surrounding selected objects in diagrams and

matrices to be specified. For more information, see the section Configuring GUI Scheme Definitions for the Alfabet Interface in the chapter Configuring the Visualization of the Alfabet Interface as well as a detailed documentation of all GUI scheme attributes in the chapter *Overview of GUI Scheme Attributes* in reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

To edit the XML object ***ITMapDef***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder by clicking the + symbol. You will see all XML objects that can be edited.
2) Right-click the XML object ***ITMapDef*** and select **Edit XML**. The XML editoris displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) Specify the following XML attributes below the root node ***ITMap*** to configure standard business support matrices, as needed.

- `XClass`: Enter the object class to display per default on the X-axis when a user accesses the matrix for the first time. Choices include either `Process` (for the object class `BusinessProcess`) or `Domain`. The default is "`Process`". Users can modify the X-axis in the context of a business support map.
- `YClass`: Enter the object class to display per default on the Y-axis when a user accesses the matrix for the first time. Choices include either `Organization` or `Product` (for the object class `MarketProduct`). The default is "`Organization`". Users can modify the Y-axis in the context of a business support map.
- `OBSProviders`: Enter the names of the object classes that are permissible to be the provider of operational business supports. Possible values are the object classes `Application` and `OrgaUnit`.The default value is `Application`.
- `TBSProviders`: Enter the names of the object classes that are permissible to be the provider of tactical business supports. Possible values are the object classes `ICTObjectVirtual`, `ICTObject`, `Application`, `SystemBuildingBlock`, and `OrgaUnit`.The default value is `ICTObjectVirtual`, `ICTObject`, and `Application`.
- `SBSProviders`: Enter the names of the object classes that are permissible to be the provider of strategic business supports. Possible values are the object classes `ICTObjectVirtual`, `ICTObject`, `SystemBuildingBlock`, and `OrgaUnit`.The default value is `ICTObjectVirtual` and `ICTObject`.
- `UseAxisClassColor`: Set to `True` to configure that the **Foreground Color** and **Background Color** attributes of the relevant class setting are displayed for the object classes that are displayed as axis objects in matrix views.
- `UseSupportClassColor`: Set to `True` to configure that the **Foreground Color** and **Background Color** attributes defined for the relevant class settings are displayed for the object classes that are displayed as matrix objects in matrix views.
- `ItemFont`: Specify the font to display for texts on the matrix objects. It is recommended that you enter "`Arial,12`" to correctly display the text. Font names with empty spaces may be defined (for example, `Times New Roman,6`).
- `AttributeFont`: Specify the font to display for texts in the attribute boxes available on the matrix objects. The attribute boxes display an abbreviate for the object states of the object providing the business support as well as the business support itself and the relevant operational aspects. It is recommended that you enter "`Arial,6`" to correctly display the

attribute information. Font names with empty spaces may be defined (for example, `Times New Roman,6`).

- `CopyResponsibilities`: Specify who the authorized user of a new operational business support based on an existing strategic or tactical business support will be. Enter "`true`" if the authorized user of the new operational business support is copied from the source business support. Enter "`false`" if the currently logged in user creating the new business support will be the authorized user. The default is set to "`false`".

  This attribute is relevant for the creation of operational business supports based on existing strategic or tactical business supports in the *Business Support Map Page View* for a master plan map.

- `ShowShortName`: Enter "`true`" if a concatenation of the short name and version of the object providing the business support should be displayed on the matrix objects. If no short name is defined, the full name will be displayed. Enter "`false`" if the full name of the business support should be displayed.

- `AspectClasses`: Enter the names of the object classes that can be defined as operational aspects for the object classes on the matrix axes as well as for operational business supports, tactical business supports, strategic business supports, and solution business supports. You can specify one or more of the standard operational aspect object classes (`Brand`, `CustomerSegment`, `Market`, and `SalesChannel`). However, any other object class or object class stereotype may be specified in the XML element `AspectClasses` as long as the object class property `ShortName` is available for the object class. The object classes `Brand`, `CustomerSegment`, `Market`, and `SalesChannel` are specified per default.

  Please note the following:

  - The object class property `ShortName` is required in order to visualize the operational aspects in a very small attributes box in cells in business support matrices. Please note that the short name should consist of only 2 or 3 letters.

  - Custom selectors may be configured for operational aspect classes that will be defined in the *Measure Types Page View*. For each <Class:Stereotype> defined in the XML element `AspectClasses` in the XML object ***ITMapDef***, a menu option will be generated in the **New** menu in the *Measure Types Page View*. The custom selector specified in the class setting of the respective object class or object class stereotype will open when the user clicks the menu option.

  - Depending on the object classes that may be specified as operational aspects, the following functionalities must be available to the relevant user profiles for which creating operational aspects shall be permissible:

    - *Capture Brands Functionality* (`BRND_CaptureBrands`)
    - *Capture Customer Segments Functionality* (`CSGM_CaptureCustomerSegments`)
    - *Capture Markets Functionality* (`MRKT_CaptureMarkets`)
    - *Capture Sales Channels Functionality* (`SCHNL_CaptureSalesChannels`)

- For a methodological overview of implementing operational aspects, see the section *Business Model Definition* in the reference manual *IT Planning Basic*.

- `BluePrintBColor`: Define the background color for the matrix objects in a blueprint map displayed on the business support matrices. The value entered should be a Windows, Web, hexadecimal, or RGB color value. The default value is "`Blue`".

- `BluePrintFColor`: Define the font color for the matrix objects in a blueprint map displayed on the business support matrices. The value entered should be a Windows, Web, hexadecimal, or RGB color value. The default value is "`Gold`".

4) Create an XML element `ClassEntry` for each object class that may be displayed on the X-axis or Y-axis (`Process`, `Domain`, `OrgaUnit`, or `MarketProduct`) of a business support matrix as well as for each object class that may be business support provider (`Application`, `OrgaUnit`, `ICTObject`, or `VirtualICTObject` displayed in a business support matrix. Please note that the class `OrgaUnit` should only be specified once. Specify the following XML attributes below the XML element `ClassEntry` as needed:

- `ClassName`: Mandatory: Specify the class name of the object class that shall be displayed in standard business support matrices. You may create an XML attribute `ClassName` for the following object classes: `Process`, `Domain`, `OrgaUnit`, `MarketProduct`, `Application`, `ICTObject`, or `VirtualICTObject`.

  The XML attribute `ClassName` may not be edited!

- `ShowProps`: This attribute can only be configured for the object classes `BusinessProcess`, `Domain`, `OrgaUnit`, and `MarketProduct`. Enter a comma-separated list of object class properties that should be displayed on the axis objects for the respective object class. The default settings for the object class properties for business processes or domains include the name (`Name`) of the object as well as the level ID number (`LevelIDNum`) that indicates its position in the respective process/domain hierarchy.

  The definition in the XML attribute `ShowProps` will also be applied to the *Business Process/Domain-Based Schedule Report Page View* (`ITMPM_Process-Based_ScheduleReport`) and *Organization/Market Product-Based Schedule Report Page View* (`ITMPM_OrganizationBased_ScheduleReport`).

- `SortProps`: This attribute can only be configured for the object classes `BusinessProcess`, `Domain`, `OrgaUnit`, and `MarketProduct`. Enter a comma-separated list of one or more object class properties that should be used to order the axis objects in the matrix. The default settings for the object class properties orders organizations and market products by name and orders business processes or domains by the level ID number (`LevelIDNum`).

  Please note that if the XML attribute `SortProps` is defined for an object class, the sorting will occur automatically, and manual sorting by the user will not be possible. If users shall be allowed to manually reorder the axis objects for an object class via the **Axes Quick Editor** available in the **Business Support Map** page views, you must ensure that NO values have been defined for the XML attribute `SortProps` for the relevant object class. If the XML attribute `SortProps` is defined for all matrix object classes, the **Axes Quick Editor** will be disabled in all relevant views.

- `Picture`: Specify the icon to display on the matrix objects for the object class specified in the XML attribute `ClassName`. Please note the order of priority for determining the icon displayed on matrix objects:

  1) Object class definition
  2) Class setting definition
  3) XML object ***ITMapDef*** definition
  4) Default specification of presentation object.

5) Create an XML element `Relevance` to define the values available for selection in the **Relevance Group** filter available in the **Business Support Map Options** editor in the business support matrices. The relevance values serve to filter out information that does not have enough relevance to be included in the business support maps. Alfabet users can select the minimum value that should be displayed in the business support matrix. All elements with a relevance value equal to or higher than the selected value will be displayed in the matrix. Specify the following XML attributes below the XML element `Relevance` as needed:

   - `Values`: In a comma-separated list, enter the possible values that can be selected in the **Relevance Group** filter.
   - `IsActive`: Enter "`true`" if the **Relevance Group** filter should be enabled. Enter "`false`" if the **Relevance Group** filter should NOT be enabled. If the filter is not enabled, a relevance group cannot be defined standard business support matrices. The default is "`true`".

6) Create an XML element `BusinessAppraisalDef` to define an indicator to visualize on the business appraisal object in business support matrices. Specify the following XML attributes below the XML element `BusinessAppraisalDef` as needed:

   - Enter the name of one indicator type that shall be visualized on the business appraisal matrix object.

     The indicator type must be assigned to the object class **Business Appraisal** in the **Class Configuration** functionality. An icon gallery must be assigned to the indicator type in order to visualize a value. For more information, see the section *Configuring Evaluation Types* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.

7) In the toolbar, click the **Save** button to save the XML definition.

Configuring Gantt Charts

The XML object ***SolutionOptions*** allows you to define whether the current date is considered in Gantt charts when the view is opened.

To edit the XML object ***SolutionOptions***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder.
2) Right-click the XML object ***SolutionOptions*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) In the XML attribute `GanttTodayScroll`, enter "`true`" the current date should be positioned so that it is quite visible in the Gantt chart when the view is opened.). Enter "`false`" if the Gantt chart should be displayed with no consideration to the current date. For the sake of backward capability, the default value "`false`" will be set for the XML attribute `GanttTodayScroll`.
4) In the XML attribute `UseFiscalYearCalendarInGantt`, specify "`true`" to display the quarters and years based on the configured fiscal calendar.
5) In the toolbar, click the **Save** button to save the XML definition.

Configuring Cost Management Capabilities

The XML object ***CostManagerDef*** allows you to define the visibility and editability of cost definition types associated with cost centers, projects, and architecture objects. Furthermore, you can specify parameters for the fiscal year for your enterprise. The XML object ***CostManagerDef*** must be configured in order to work with costs in the following Alfabet page views:

Object Class	Page View
Application	*Consolidated Operational Expenses Page View* *Operational Expenses Page View*
Deployment	*Operational Expenses Page View*
ICT Object	*Operational Expenses Page View*
ICT Object Group	*Operational Expenses Page View* *Cost Aggregation Page View*
ICT Object Category	*Operational Expenses Page View* *Cost Aggregation Page View*

Object Class	Page View
Organization	*Operating Expenses Page View* *Business Support Costs Hierarchy Page View*
Business Process	*Business Support Costs Hierarchy Page View* *Business Support Costs Page View* *Business Service Costs Page View* *Cost Benchmarking Page View*
Business Support	*Cost Benchmarking Page View*
Business Capability Map	*Business Capability ICT Cost Report Page View*
Project	*Cost Report Page View* *Cost Accrual Page View* *Business Case Page View* *Business Case Comparison Page View* *Project, Skill Request and Resource Request Time Schedule Page View* *Cash Out Planning Page View*
Bucket	*Cost Accrual Page View*
Cost Center	*Cost Accruel Page View*
Cost Center Group	*Cost Aggregation Page View*

For an overview of various methodologies to manage costs for architecture objects in Alfabet, see the sections *OPEX Optimization* and *Cost Driver Analysis* in the reference manual *Portfolio Management Advanced*.

The currency and currency units displayed on all page views and editors in which costs are captured and visualized is configured in the **Reference Data** functionality. For more information about the configuration of currencies, see the chapter *Configuring Currencies and Currency Exchange Rates for Cost Management Capabilities* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*. The cost types required for the capture and analysis of costs are configured in the **Configuration** functionality. For more information, see the chapter

Configuring Cost Types and Income Types for Cost Management Capabilities reference manual *Configuring Evaluation and Reference Data in Alfabet*.

For more information about defining various cost aspects in the XML object ***CostManagerDef***:

- Configuring the Calculation of Business Support Costs
- Configuring the Editability of Costs in Cost Centers
- Configuring the Editability of Costs for Architecture Objects
- Specifying Quarterly or Monthly Budgeting for Cashout Planning
- Configuring the Default Values for Business Case Definitions
- Configuring the Fiscal Year for Cost Reporting in Your Enterprise

Configuring the Calculation of Business Support Costs

In the XML object ***CostManagerDef***, you can specify the means to calculate business support costs for a business process. You can specify whether the cost calculation includes the costs of the business services that the business support uses or if the calculation of the business support costs for a business process is based only on the costs of the business support.

To edit the XML object ***CostManagerDef***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder.
2) Right-click the XML object ***CostManagerDef*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) Specify the XML attribute `BusinessSupportServiceCosts`. Enter "`true`" if the calculation of business support costs for a business process includes the costs of the business services that the business support uses. Enter "`false`" if the calculation of the business support costs for a business process is based only on the costs of the business support.
4) In the toolbar, click the **Save**  button to save the XML definition.

Configuring the Editability of Costs in Cost Centers

Cost centers allow costs for applications, deployments, ICT objects, networks, service products, and projects to be grouped and the costs assigned to the cost center to be distributed to the objects in the group according to a defined allocation scheme. The costs are then visible in the standard Alfabet views for viewing or editing the object's costs.

In Alfabet, cost planning is based on user-defined cost types (for example, maintenance costs, license fees, man hours). Cost centers and cost types are configured in the **Configuration** functionality. For more information about configuring cost centers and cost types, see sections *Configuring Cost Centers for Cost Management Capabilities* and *Configuring Cost Types and Income Types for Cost Management Capabilities* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.

Costs are definable for each cost type per year. For cost planning within a time period, different cost definition types can be specified. The possible cost definition types available in Alfabet are:

- `Request`: The amount of money foreseen to be spent on the project or architecture object for the given cost type and time period. This information is relevant for budget planning to evaluate demand.
- `Budget`: The amount of money assigned to the project or architecture object for the given cost type and time period in budget planning. Please note that the value `Commited` is written to the `MonetaryID` column in the `BudgetValue` database table for the `Budget` cost definition type. This is particularly relevant for the import of costs via the data capture template functionality. For more information, see the section *Configuring a Cost-Based Data Capture Template* in the reference manual *User and Solution Administration*.
- `Current`: The costs already spent on the project or architecture object for the given cost type and time period.
- `Obligation`: The costs that are committed for the object (for example, money to be spent on contracts already signed or bills received but not paid for yet). This cost definition type is only relevant if the cash-out planning capability (*Cash Out Planning Page View*) is implemented for projects.
- `Forecast`: Costs that are not yet committed for the period but are already known to be due at the time period. This cost definition type is only relevant if the cash-out planning capability (*Cash Out Planning Page View*) is implemented for projects.

By default, only the current costs will be editable in a cost center. Cost requests and budgeting is done outside of cost centers. The default setting can be altered in the XML object ***CostManagerDef***. The visibility and editability of the different cost definition types can be defined separately. You can define, for example, that cost requests cannot be edited on the **Cost Accrual** page view of a cost center but that they are nevertheless visible in the view.

If you change the settings for cost center costs in the XML object ***CostManagerDef***, you should also adapt the settings for the editability and visibility of cost definition types in the cost views available for the architecture objects. Please note that if you define a cost definition type as editable in a cost center, it should NOT be editable in the cost views available for the architecture objects. **Any values that are edited in the cost views available for the architecture objects will overwrite the values assigned to the architecture object via the cost allocation defined for the cost center.**

To edit the XML object ***CostManagerDef***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder.
2) Right-click the XML object ***CostManagerDef*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) The default settings of the XML object ***CostManagerDef*** are displayed in the XML editor. The definition of costs for cost centers is defined in the XML element ***CostCentreCosts***, which contains one XML element ***CostColumn*** for each kind of cost definition (`Request`, `Budget`, `Current`, and `Obligation`).

   Define the visibility and editability of the cost definition types, as needed. The table below displays the attributes that can be edited for the XML element ***CostColumn***:

XML Element	Explanation
CostColumn	Specify one XML element `CostColumn` for each kind of cost definition (`Request`, `Budget`, `Current`, and `Obligation`) that should be available for cost centers. The cost definition `Forecast` is only relevant if the cash-out planning capability (*Cash Out Planning Page View*) is implemented for projects.
Type	The cost column types that are available for cost centers are `Request`, `Budget`, `Current`, and `Obligation`. The XML attribute `Type` defines which cost column type is configured with the XML element ***CostColumn***.
Visible	Enter "`true`" for a cost definition type that should be displayed in Alfabet views for cost centers. Enter "`false`" if the cost definition type should not be displayed in the views.
Editable	Enter "`true`" if the costs for a cost definition type should be editable in the *Cost Accrual Page View* of cost centers. Enter "`false`" if the costs in the column should not be editable in the relevant Alfabet page views.

4) In the toolbar, click the **Save** button to save the XML definition.

Configuring the Editability of Costs for Architecture Objects

In Alfabet, cost planning is based on user-defined cost types (for example, maintenance costs, license fees, man hours). Costs can be captured for applications, deployments, ICT objects, networks, service products, and projects per year. In the case of projects and buckets, costs can be captured on a monthly or quarterly basis via the *Cash Out Planning Page View* (`PRJ_Cashout`).

Cost types are configured in the **Configuration** functionality. For more information about configuring cost types, see chapter *Configuring Cost Types and Income Types for Cost Management Capabilities* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.

Costs are definable for each cost type per year. For cost planning within a time period, different types of cost definitions can be specified. The visibility and editability can be defined for the following cost definitions:

- `Request`: The amount of money foreseen to be spent on the project or architecture object for the given cost type and time period. This information is relevant for budget planning to evaluate demand.
- `Budget`: The amount of money assigned to the project or architecture object for the given cost type and time period in budget planning. Please note that the value `Commited` is written to the `MonetaryID` column in the `BudgetValue` database table for the `Budget` cost definition type. This is particularly relevant for the import of costs via the data capture template functionality. For more information, see the section *Configuring a Cost-Based Data Capture Template* in the reference manual *User and Solution Administration*

- `Current`: The costs already spent on the project or architecture object for the given cost type and time period.
- `Obligation`: The costs that are committed for the object (for example, money to be spent on contracts already signed or bills received but not payed for yet. This cost definition type is only relevant if the cash-out planning capability (*Cash Out Planning Page View*) is implemented for projects.

  Please note that the `Obligation` cost definition is a core concept for cash-out planning in projects. Therefore, the `Obligation` cost definition will be displayed in the context of the *Cash Out Planning Page View* regardless of whether it is configured in the XML object ***CostManagerDef***. In other words, if you specify that the `Obligation` cost definition should not be visible in the XML object ***CostManagerDef***, it will nevertheless be visible in the *Cash Out Planning Page View*.

- `Forecast`: Costs that are not yet committed for the period but are already known to be due at the time period. This cost definition type is only relevant if the cash-out planning capability (*Cash Out Planning Page View*) is implemented for projects.

By default, cost requests and budget costs are editable directly in the Alfabet views available for object costs. However, please note that per default, the current costs are not editable for object costs but are allocated to the object via a cost center. This default setting can be altered in the XML object ***CostManagerDef***.

If you change the settings for costs for architecture objects in the XML object ***CostManagerDef***, you should also adapt the settings for the editability and visibility of cost definition types in the Alfabet views available for cost centers.

Please note that if you define a cost definition type as editable in a cost center, it should NOT be editable in the cost views of the architecture objects. **Values edited in the cost views of the architecture objects overwrite the values assigned to the architecture object via the cost center cost allocation.**

You can explicitly configure the visibility and the editability of the different cost definition types in the XML object ***CostManagerDef***. For example, the current costs assigned to the object via cost center cost allocation might be configured to be visible in the Alfabet views available for object costs, but to not be editable in those views.

Please note that individual menu options (**Edit Current**, **Edit Budget**, **Edit Request**, etc) will be available for each visible cost definition type in the **Costs** menu in the *Operational Expenses Page View* (`ObjectOperatingCosts`) available in the object profiles for the object class **Application**, **Deployment**, **ICT Object**, **Network**, and **Service Product** as well as the *Cost Accrual Page View* (`PRJ_Costs`), *Cash Out Planning Page View* (`PRJ_Cashout`) available in the object profile for the class **Project**.

These menu options can be hidden for a user profile, thus allowing you to specify which user profiles may edit current costs, request costs, etc. Please note that if the XML attribute `Forecast` is set to `true` for the XML element `Editable`, for example, you should ensure that the menu option **Edit Forecast** is not visible in the *Operational Expenses Page View* (`ObjectOperatingCosts`) available in the object profiles for the object class **Application**, **Deployment**, **ICT Object**, **Network**, and **Service Product** or the *Cost Accrual Page View* (`PRJ_Costs`). For more information about hiding functionalities for a user profile, see the section Hiding Functionalities in a Page View or Configured Report.

To edit the XML object cost column type:

1) Go to the **Presentation** tab and expand the **XML Objects** folder.
2) Right-click the XML object ***CostManagerDef*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) The default settings of the ***CostManagerDef*** XML object are displayed in the XML editor.
4) Cashout planning may be executed on a quarterly or monthly basis. To specify quarterly or monthly planning for cashout planning, add the XML attribute `DetailedFinancialPlanningPeriod` as a child of the root element ***CostManagerDef***. Enter "`Month`" if the cashout planning shall occur on a monthly basis. Enter "`Quarter`" if the cashout planning shall occur on a quarterly basis.
5) The definition of costs for applications, deployments, ICT objects, networks, service products, and projects is defined in the XML element ***ArchitectureCosts***, which contains one element ***CostColumn*** for each kind of cost definition (`Request`, `Budget`, `Current`, `Obligation`, and `Forecast`).

   Define the visibility and editability of the cost definition types, as needed. The table below displays the attributes that can be edited for the elements ***CostColumn***:

XML Element	Explanation
`Type`	Specify one XML element `CostColumn` for each kind of cost definition (`Request`, `Budget`, `Current`, `Obligation` and `Forecast`) that should be available for architecture objects. The cost definition `Forecast` is only relevant if the cash-out planning capability (*Cash Out Planning Page View*) is implemented for projects.
`Visible`	Enter "`true`" for a cost definition type that should be displayed in the relevant Alfabet views for architecture object costs. Enter "`false`" if the cost definition type should not be displayed in the relevant Alfabet views for architecture object costs.
`Editable`	Enter "`true`" if the costs for a cost definition type should be editable in the relevant Alfabet views for architecture object costs. Enter "`false`" if the costs in the column should not be editable in the relevant Alfabet views for architecture object costs. **NOTE**: If you enter "`true`" for a cost definition, the relevant menu option (**Edit Current**, **Edit Budget**, **Edit Request**, etc) will be available for each visible cost definition type in the **Costs** menu in the *Operational Expenses Page View* (`ObjectOperatingCosts`) available in the object profiles for the object class **Application**, **Deployment**, **ICT Object**, **Network**, and **Service Product** as well as the *Cost Accrual Page View* (`PRJ_Costs`), *Cash Out Planning Page View* (`PRJ_Cashout`) available in the object profile for the class **Project**. This can be hidden for relevant user profiles. For more information about hiding functionalities for a user profile, see the section Hiding Functionalities in a Page View or Configured Report. **NOTE**: If cost accrual is to be captured in the object class `CostCentre`, the current costs for architecture objects should not be editable. In this case, enter "`false`" for the cost column type representing current costs.

6) In the toolbar, click the **Save** button to save the XML definition.

Specifying Quarterly or Monthly Budgeting for Cashout Planning

n the case of projects and buckets, cashout planning may be executed on a quarterly or monthly basis *Cash Out Planning Page View* (`PRJ_Cashout`). Cashout planning may be executed on a quarterly or monthly basis.

Please note that the decision about whether to specify the values `Month` or `Quarter` must occur before cashout planning is carried out in the Alfabet user interface. It is not possible to switch between monthly and quarterly planning and aggregate or divide existing costs.

Please note the following configuration requirements:

- Cost types must be configured in the **Reference Data** functionality. For more information, see the chapter *Configuring Cost Types and Income Types for Cost Management Capabilities* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.
- Currencies must be configured in the **Reference Data** functionality. For more information, see the section *Configuring Currencies and Currency Exchange Rates for Cost Management Capabilities* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.
- The relevant cost definition types (**Request**, **Budget**, **Current**, etc), default values for business cases, and the specification of the fiscal year must be configured by your solution designer in the configuration tool Alfabet Expand. For more information, see the section Configuring the Editability of Costs for Architecture Objects in the reference manual *Configuring Alfabet with Alfabet Expand*.
- The start and end of the fiscal year. Thus, if the fiscal year specified for an enterprise starts on April 1st, the monthly calendar for 2015 will start with April 2015 and finish with March 2016. For more information about configuring the fiscal year, see the section Configuring the Fiscal Year for Cost Reporting in Your Enterprise in the reference manual *Configuring Alfabet with Alfabet Expand*.

To edit the XML object ***CostManagerDef***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder.
2) Right-click the XML object ***CostManagerDef*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) Add the XML attribute `DetailedFinancialPlanningPeriod` as a child of the root element ***CostManagerDef***. Enter "`Month`" if the cashout planning shall occur on a monthly basis. Enter "`Quarter`" if the cashout planning shall occur on a quarterly basis.
4) In the toolbar, click the **Save** button to save the XML definition.

Configuring the Default Values for Business Case Definitions

The following formulas are used to calculate business cases in Alfabet:

Formula for Net Cash Flow for year i (NCFi)

$$NCF_i = (INCOME_i - COST_i) * (1 - TaxRate)$$

Formula for Discounted Cash Flow (DCF), whereby i = year and COC = Cost of Capital

$$DCF = \sum_{i=0}^{n} \frac{NCF_i}{(1+COC)^i}$$

Formula for Overall Budget (OB)

$$OB = \sum_{i=0}^{n} COST_i$$

Formula for Relative Discounted Cash Flow (RDCF)

$$RDCF = \frac{DCF * 100}{OB}$$

The XML object ***CostManagerDef*** allows the default values in business cases to be defined. The default values defined here are displayed in the *Business Case Page View* (`PRJ_BusinessCase`) available for all project stereotypes configured for the project planning capability as well as for example, the **Spend Limit** editor available for buckets.

A **Business Case** page view (`PRJ_BusinessCaseSimple`) is available as an alternative to the standard **Business Case** page view (`PRJ_BusinessCase`) that allows project costs and income to be calculated without the inclusion of a tax rate, interest rate, discounted cash flow, or amortization time.

Please not that the period of the business case should not only focus on the cost incurred during the project execution, but should also factor in cost incurred for operation, maintenance and management of forthcoming projects/project scenarios or changes to existing projects/project scenarios. Typically the business case period should be at least 3 years, although more mature organizations might define a period of 5 or 7 years. Please note that individual projects can still define business cases with a shorter period. In this case, users would simply leave the periods beyond the project scope undefined.

For more information about configuring the project planning capability, see the section Configuring the Project Management Capability.

To edit the XML object ***CostManagerDef***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder.
2) Right-click the XML object ***CostManagerDef*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) Define the XML attributes, as needed. The table below displays the XML elements and XML attributes that can be edited for the XML object ***CostManagerDef***:

XML Element (bold) / XML Attribute	Explanation
BusinessCaseDef	
`Period`	**NOTE**: Changes made to the `Period` attribute will only be applied to business cases created after the modification to the `Period` attribute. Existing business cases will retain their definitions. Starting with the current year, enter an integer to define the number of years for which the cost types and income types are to be calculated in the business case.
`TaxRate`	Enter an integer as the default value for the tax rate. The value is a percentage. For example, enter 10.5 for 10.5%.
`CostOfCapital`	Enter an integer as the default value for the capital interest rate. The value is a percentage. For example, enter 30 for 30%.

4) In the toolbar, click the **Save**  button to save the XML definition.

Configuring the Fiscal Year for Cost Reporting in Your Enterprise

The XML object ***CostManagerDef*** allows you to define the fiscal year as determined by your enterprise. The fiscal year is based on the start year of the project.

To edit the XML object ***CostManagerDef***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder.
2) Right-click the XML object ***CostManagerDef*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) Define the XML attributes, as needed. The table below displays the XML elements and XML attributes that can be edited for the XML object ***CostManagerDef***:

XML Element (bold) / XML Attribute	Explanation
FiscalYearDef	
`Day`	Enter an integer for the first day of your company's fiscal year. For example, if the first day is the first day of a specific month, enter 1.

XML Element (bold) / XML Attribute	Explanation
`Month`	Enter an integer for the first month of your company's fiscal year. For example, if the first month is September, enter 9.
`YearOffset`	You must enter either `0` or `1` to define the year for the fiscal year. The fiscal year is based on the start year of the project. For example, If the start year of the project is `2020`, the XML attribute `Day` is 1, and the XML attribute `Month` is 9: • If you define `0` for the XML attribute `YearOffset`, the fiscal year `2020` begins in September `2020` and concludes by the end of August `2021`. • If you define `1` for the XML attribute `YearOffset`, the fiscal year `2020` begins in September `2019` and concludes by the end of August `2020`.  For example, if the project start date is 05.06.2018 and end date is 07.10.2019: • If `Day` = "1", `Month` = "9" and `YearOffset` = "0", then the years 2017, 2018 and 2019 will be listed in the **Fiscal Year** filter field in the **Cash Out Planning** page view for the project. • If `Day` = "1", `Month` = "9" and `YearOffset` = "1", then the years 2018, 2019 and 2020 will be listed in the **Fiscal Year** filter field in the **Cash Out Planning** page view for the project.

4) In the toolbar, click the **Save**  button to save the XML definition.

Configuring the Permissibility of Files and Web Links in Alfabet

You can specify the file extensions that may be uploaded and downloaded from the **Internal Document Selector** as well as the linkage of documents stored in an external file system as well as specify the default prefix for URLs can be specified in order to direct the navigation. Documents can be downloaded and attached from the **Internal Document Selector** as well as an external file system by users via the *Attachments Page View*. Web links (URLs) can also be attached to an object in the *Attachments Page View*.

Apart from the configured permissability, there is a general file content size restriction for ZIP file upload. ZIP files are checked during update for the size of content after decompression. The file will not be uploaded if the decompressed size is more than 100 percent of the compressed size or if storing the file content on the local drive would result in less than 1 GByte free space remaining or if any deviations from normal compression mechanisms are detected.

The following information is available:

- Specifying the Permissible File Extensions for Uploading/Downloading Files
- Configuring Default URL Prefixes for the Attachments Page View
- Configuring Files on a Network Drive as Attachments

Specifying the Permissible File Extensions for Uploading/Downloading Files

For security reasons, a blacklist and whitelist concept is available to restrict the uploading and downloading of files with permissible file extensions. This is relevant for the **Internal Document Selector** as well as the linkage of documents stored in an external file system. File extensions can be specified in a blacklist as not permissible for uploading and downloading. Alternatively, a whitelist can be configured that explicitly and exclusively allows specified file extensions for upload and download. The whitelist is optional and will be disabled per default. A file extension specified in the blacklist will supersede the same file extension specified in the whitelist so that if a file extension is listed in both the whitelist and in the blacklist, it will not be possible to upload or download files with the specified file extension.

A default blacklist is specified in the XML attribute `Blacklist` in the XML object ***FileExtensionLists***. For example, the default blacklist includes the file formats SVG, HTML, and WSDL. It is recommended that SVG files are available only if they have been generated in the Alfabet Web application. If your enterprise should prefer to use any of these file formats, you should remove them from the XML attribute `Blacklist`.

Please note that

Please note that the following file types cannot be opened directly in Alfabet and will be automatically saved and downloaded as a ZIP file even if not defined in the blacklist or additionally defined in the whitelist: .com, .bat, .exe,.ad, .adprototype, .asax, .ascx,.ashx, .asmx, .asp, .aspx, .axd, .browser, .cd, .compiled, .config, .cs, .csproj, .dd, .exclude, .java, .jsl, .ldb, .ldd, .lddprototype, .ldf, .licx, .master, .mdb, .mdf, .msgx, .refresh, .rem, .resources, .resx, .sd, .sdm, .sdmDocument, .sitemap, .skin, .soap, .svc, .vb, .vbproj, .vjsproj, .vsdisco, and .webinfo.

The file extensions for the blacklist and the whitelist are configured in the XML object ***FileExtensionLists*** that is available both in Alfabet Expand and in the Alfabet Administrator. The XML object has the following XML structure:

```
<FileExtensionLists>
```



```
<ExtensionLists>
    <WhiteList IsEnabled="false">
        <Extension>.xls</Extension>
        <Extension>.xlsx</Extension>
        [...]
    </WhiteList>
    <BlackList>
        <Extension>.exe</Extension>
        <Extension>.bat</Extension>
        [...]
    </BlackList>
</ExtensionLists>
</FileExtensionLists>
```

To configure the XML object ***FileExtensionLists***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder and the **Administration** folder.
2) Right-click the XML object ***FileExtensionLists*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects. The following is possible:
   - To enable the whitelist, the XML attribute `IsEnabled` of the XML element `WhiteList` must be set to `true`.
   - To add a file extension to the whitelist, a new child XML element `Extension` with the definition of the file extension starting with a period (.) has to be added to the `WhiteList` or `BlackList` element respectively.
   - To add a file extension to the blacklist or to the whitelist, a new child XML element `Extension` with the definition of the file extension starting with a period (.) has to be added to the `WhiteList` or `BlackList` element respectively.
3) Click the **Save** button to save the definition.

Configuring Default URL Prefixes for the Attachments Page View

The *Attachments Page View* as well as the **Notification Assignment** editor available via the **Notify Authorized User** button allows users to attach documents or Web links (URLs) to an object. When users define a Web link, an editor opens in which users can enter the link address in a **URL** field. Per default, the prefix displayed in this field is: `http://www.`

For more information about the general configuration required to implement the **Notify Authorized User** button, see the section Configuring the Display of the Notify Authorized User Button for Assignment Creation.

The default prefix can be changed in Alfabet Expand if, for example, you want the URLs to navigate to a corporate content management solution such as Microsoft® SharePoint®. The prefix will be automatically

displayed in the field and users can enter the remaining Web address. The prefix can be changed in the **URL** field by the user, if necessary.

To define a prefix for the URL editor:

1) Go to the **Meta-Model** tab, expand the **Class Model** node, expand the **Classes** node, and click the `ALFA_URI` class node.
2) Expand the `PROPERTIES` node and click the object class property `LINK`. In the **Default Value** attribute in the attribute window, enter the URL prefix
3) Click the **Save** button to save the definition.

Configuring Files on a Network Drive as Attachments

The file selector `ALFA_URI_OLD` may be configured for the *Attachments Page View* that allows the target file to be selected from the network drive. The file selector `ALFA_URI_OLD` must be specified for the **Edit View** attribute of the class setting `ALFA_URI`. The local directory path must be enabled as a trusted site for the relevant browser and must run under HTTPS protocol.

The following browsers support opening the files via the Web link definition in Alfabet: Microsoft® Internet Explorer® 11.0 and Microsoft® Edge® in conjunction with Windows® 10. Mozilla® Firefox® 24.0 or higher including Mozilla® Firefox® Quantum are also supported but require additional configuration in the browser settings.

Configuring Dynamic Web Links That Users Can Access

The XML object ***WebViewManager*** allows you to configure dynamic web links that users can access in the *Dynamic Web Links Page View* of the relevant object classes.

Dynamic web links reference applications that are accessible via a URL. The dynamic web link references a specific location in the application, allowing you to access information in the application that is relevant to the object that the user is currently working with in Alfabet.

For example, a dynamic web link could be used in the following instances:

- To retrieve information about a selected organization in a human resources management tool.
- To retrieve cost information about a selected ICT object in an application like SAP Business Warehouse that documents cost information about the selected object.
- To retrieve information about a selected application in your enterprises's central help desk system.
- To retrieve data about a selected information flow in a document describing interface contacts in a document management system.

Next to being available in the *Dynamic Web Links Page View* page view of the relevant object class, dynamic web links can also be added to the following views:

- Dynamic web links can also be specified in value controls in object cockpits. To embed the dynamic web link in the object cockpit, a value control must be created, whereby the

Subtype attribute is set to the new `DynamicWebLink` option. All dynamic web links defined for the object class or object class stereotype of the object cockpit that have been configured in the XML object ***WebViewManager*** will be available for selection in the **Web Link** attribute of the object cockpit. At runtime, the dynamic web link will only be displayed if the object class and object class stereotype match the definition in the XML object ***WebViewManager***. For more information about configuring dynamic web links in object cockpits, see the section Adding a Dynamic Web Link to the Object Cockpit.

- Dynamic web links can be added to configured reports via an Alfabet instruction. The instruction will add an **Open Web Link** button to the report that opens the dynamic web link for the object selected in the configured report via a sub-menu option. In addition to the button interaction, the link can be opened via the **Operations** button in the object preview window. For more information about configuring dynamic web links in configured reports, see the section Adding Dynamic Web Links to a Configured Report.

To edit the XML object ***WebViewManager***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder.
2) Right-click the XML object ***WebViewManager*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) Define the XML attributes, as needed. The table below displays the XML elements and XML attributes that can be edited for the XML object ***WebViewManager***:

XML Element (bold) / XML Attribute	Explanation
WebViewDef	
`Caption`	Enter text to define the caption to be displayed as a header for the group of dynamic web links. For example, Business Intelligence Reports. The values defined for the XML elements `Caption` in the XML object ***WebViewManager*** will be extracted to the vocabularies so that dynamic web links can be rendered in the language specified for the Alfabet user interface.
`Class Names`	Enter a comma-separated list of object class names for which the dynamic web link(s) should be available. Each dynamic web link defined in the XML element ***WebViewDef*** will be available in the *Dynamic Web Links Page View* for all objects belonging to the object classes defined here. You must enter the technical name of the object class.

XML Element (bold) / XML Attribute	Explanation
	Dynamic web links can be limited to a specific object class stereotype. To this end, object class stereotypes can be specified in the XML element `ClassNames` in the XML object ***WebViewManager***. (For example: `ClassNames="Device:Cumulocity Device`).
WebLinkDef	
`Caption`	Enter text to define the caption to be displayed for the dynamic web link.

`HRef`	Enter the URL that should be invoked when a user clicks the dynamic web link. The construction rule for the URL is:
	`http://(AddressToApplication)/[NameOfReport]?Target={PropertyNames}`
	where `{PropertyName}` is a placeholder for the current object's value of the object class property with the defined name. Please note that this object class property must be defined in the XML attribute PropNames of the dynamic web link to use it as a placeholder in the XML attribute `HRef`. Each placeholder can only return one object class property. Multiple placeholders can be used in the URL definition.
	The dynamic web link will not be displayed in the *Dynamic Web Links Page View* of objects if the object class property specified for the placeholder `{PropertyNames}` has no value defined.
	f the Web link contains a special character according to XML standards (for example: &, %, ;,<,>), these characters must be replaced by their XML conformant variant (for example: & for &)
	Below is an example of a typical URL for a dynamic web link in Alfabet. In the `HRef` attribute, the placeholders {`Name`} and {`Version`} are used. When a user will open the dynamic web link for an object, the placeholders will be replaced with the name and version of the current object that the dynamic web link is associated with.
	`HRef=http://localhost/Reports/ApplicationsReport.aspx?Target={Name}&Version={Version}`
	It is possible to specify part or all of the `HRef` definition in the server alias configuration as a server variable and reference the appropriate server variable in the `HRef` definition.

XML Element (bold) / XML Attribute	Explanation
PropNames	Enter a comma-separated list of object class properties used in the specification of the reference URL defined above. You must enter the name of the object class property. In the example HRef=http://localhost/Reports/OrganizationReport.aspx?Target={Name}&Version={Version}, you would enter Name,Version for the XML attribute PropNames.
Target	Enter text to define the target browser window that is opened when the URL is invoked. If the Target attribute is left empty, then each dynamic web link will open in a new window.

4) In the toolbar, click the **Save** button to save the XML definition.

Using Server Variables in Dynamic Web Links

In the definition of the dynamic Web link in the XML object ***WebViewManager***, you can include the server variable definitions in the definition of the XML attribute HRef.

For example, a dynamic Web link to a report that is to be hosted on the company Web server is first tested on a test Web server. Therefore, the following Web server variables would be defined in the alias configuration of the Alfabet Server:

- WEBSERVER, specifying the Web server used
- APPLICATION, specifying the virtual directory for the referenced report functionality on the Web server

The URL definition in the configuration of the dynamic Web link contains the variables instead of the current Web server name and virtual directory:

```
HRef=http://$WEBSERVER/$APPLICATION/OrganizationReport.aspx?Target={
REFSTR}
```

The definition is automatically converted to a valid link containing the values of the server variables. For example:

```
HRef=http://localhost/Reports/OrganizationReport.aspx?Target={REFSTR
}
```

The definition of server variables allows you to store information about the targets of dynamic Web links in the alias configuration of the Alfabet Server. Storing the information about the URL in the alias

configuration of the Alfabet Server instead of directly defining it in the external source configuration eases the propagation of changes.

For example, if the configuration of Alfabet is done first in a test environment and the test environment is an exact copy of the production environment except that the components are installed on different servers, then all URLs defined will be identical in the test and the production environment except for the Web server name. When migrating to the production environment, the Web server name must be changed in all URL definitions in Alfabet Expand. But if the Web server name is defined as a server variable in the alias configuration of the Alfabet Server, the configurations specified in Alfabet Expand reference the Web server name as variable in the `HRef` definitions and can be reused without changes in the production environment. Only the variable definition in the alias configuration of the Alfabet Server of the production environment must be set to the Web server definition for the production environment.

Either all or part of the URL definition for dynamic Web links can be defined in a server variable. It is also possible to build the URL from a number of concatenated server variables.

Server variables are defined in the Alfabet Administrator:

1) In the Alfabet Administrator, click the **Alfabet Aliases** node in the **Administrator** explorer.
2) In the table on the right, select the server alias that you want to define a server variable for and click the **Edit**  button. The alias editor opens.
3) Go to the **Variables** tab and click the **New** button. A dialog box opens.
4) In the **Variable Name** field, enter a unique name for the server variable.

   The server variable name may only contain letters (English alphabet), numbers, and the underscore symbol.

5) In the **Variable Value** field, enter all or part of the URL that should be the target of the Web link.

   If the Web link contains a special character according to XML standards (for example: &, %, ;,<,>), these characters must be replaced by their XML conformant variant (for example: `&` for &)

6) Click **OK** to save your changes. The server variable definition appears in the list of server variables.

   To edit or delete the server variable, select the server variable in the table and click the **Edit** or **Delete** button below the table.

7) Click **OK** to save your changes and close the editor. The server variable definition is now available in the server alias configuration and can be used for the URL definition specified in Alfabet Expand.

Configuring the Export of Views in the Alfabet Interface to HTML Files

In Alfabet Expand you can configure the styles that determine the layout of views as well as the header and footer text in the HTML files that are generated when a user exports a page view in Alfabet.

The export options available in Alfabet are described in the section *Exporting Data* in the reference manual *Getting Started with Alfabet*.

The following information is available:

- Specifying the Page Sizes Available to Export DOC and PDF Files
- Configuring the Style Sheet for the Export to HTML
- Configuring Header and Footer Text for Exported HTML Files

Specifying the Page Sizes Available to Export DOC and PDF Files

The XML object ***ExportDocumentPageSizeManager*** allows you to specify the paper size formats available in the **Export Page Setup** editor that opens when **Export** > **Microsoft Word** and **Export** > **Adobe PDF** is clicked in the toolbar available in object profiles/object cockpits and page views.

```
<ExportDocumentPageSizeManager>
   <AllPageSizes>
      <PageSizeDef Name="A4 210x297" PageWidth="210" PageHeight="297"
      Unit="Millimeter" Enabled="true" PaperKind="A4" />
      <PageSizeDef Name="A3 297x420" PageWidth="297" PageHeight="420"
      Unit="Millimeter" Enabled="true" PaperKind="A3" />
      <PageSizeDef Name="A2 420x594" PageWidth="420" PageHeight="594"
      Unit="Millimeter" Enabled="true" PaperKind="A2" />
   </AllPageSizes>
</ExportDocumentPageSizeManager>
```

To edit the XML object ***ExportDocumentPageSizeManager***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder by clicking the + symbol. You will see all XML objects that can be edited.
2) Right-click the XML object ***ExportDocumentPageSizeManager*** and select **Edit XML**. The XML editor is displayed in the center pane. Edit the style sheet, as needed. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) For each page size option that shall be available in the **Page Size** field in the **Export Page Setup** editor, create an XML element `PageSizeDef` as a child element of the XML element `AllPageSizes`. For each XML element, specify the following XML attributes:
   - `Name`: Specify the name for the page size. This XML attribute is mandatory. The value will be displayed in the **Page Size** field in the **Export Page Setup** editor.

- `PageWidth`: Specify an integer for the page width of the page size in the unit that you specify in the XML attribute `Unit`. This XML attribute is mandatory.
- `PageHeight`: Specify an integer for the page height of the page size in the unit that you specify in the XML attribute `Unit`. This XML attribute is mandatory.
- `Unit`:Enter either `Millimeter`, `Inch`, `Pixel`, or `Point`. This XML attribute is optional. If no value is defined, the default value `Millimeter` will be used.
- `Enabled`: Enter `True` if the page size shall be displayed in the **Page Size** field in the **Export Page Setup** editor. Enter `False` if the page size shall not be displayed in the **Page Size** field in the **Export Page Setup** editor. This XML attribute is optional. If no value is defined, the default value `True` will be used.
- `PaperKind`: Specify the kind of paper associated with the page size. For an overview of all possible values that may be specified for the XML attribute `PaperKind`, see the specification available via the **XML XSD** attribute. This XML attribute is optional. If no value is defined, the default value `Custom` will be used.

4) In the toolbar, click the **Save**  button to save the XML definition.

Configuring the Style Sheet for the Export to HTML

The XML object ***DataSetExportCss1*** allows you to configure the styles that determine the layout of views exported from Alfabet to HTML files. Standard style sheet definitions apply.

```
<style>
BODY
{
    border: none;
    background-color: White;
}

.DS_Caption
{
    font-family: "Tahoma";
    font-size: 12px;
    font-style: normal;
    line-height: 12px;
    color: Black;
    padding-left: 6px;
    height: 26px;
    background-color:White;
    border-top: "solid 1 white";
    width: 100%;
}

.DS_CaptionText
{
    text-align: left;
    vertical-align: middle;
    width: 98%;
}

.DS_Table
{
```

FIGURE: Example of a section of the styles definition in the XML object `DataSetExportCss1`

To edit the XML object ***DataSetExportCss1***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder by clicking the + symbol. You will see all XML objects that can be edited.
2) Right-click the XML object ***DataSetExportCss1*** and select **Edit XML**. The XML editor is displayed in the center pane. Edit the style sheet, as needed. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) Paste an existing style sheet in the <style> elements or edit the existing style sheet, as needed.
4) In the toolbar, click the **Save** button to save the XML definition.

Configuring Header and Footer Text for Exported HTML Files

The settings in the XML objects ***HTMLExportHeader*** and ***HTMLExportFooter*** allow you to determine the header and footer text in the HTML files that are generated when a user exports a page view in Alfabet.

To edit the XML object ***HTMLExportHeader*** and ***HTMLExportFooter***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder.
2) Right-click the XML object ***HTMLExportHeader*** or ***HTMLExportFooter*** and select **Edit XML**. The XML editor is displayed in the center pane. In the ***HTMLExportHeader*** or ***HTMLExportFooter***, the text "powered by Alfabet" can be replaced as shown in the following example:`<p class='DS_Caption'>powered by Alfabet</p>`
3) In the toolbar, click the **Save** button to save the XML definition.

Configuring the Content Voting Capability via Object Associations

Alfabet provides a new content voting capability similar to that found in social media. The capability is configurable and can be implemented to allow users to rate, like, and identify expert users for all artifact object classes and the class **Report**. The button to vote or like an object or report will be available in the object profile toolbar for objects based on object classes or object class stereotypes for which the capability is configured. The new capability supports three different use cases:

- The `Rate` concept allows an object to be voted on based on a star-rating system. The rating can be expressed as an icon with a specified number of stars or via another type of icon gallery that supports rating an object. For example, users could specify their recommendation of an application based on a 3-star rating system.
- The `Like` concept allows an object to be endorsed based on a binary rating system. The rating can be expressed via a Like button or Heart icon or any other type of icon gallery that supports a true/false rating of an object. For example, users could click the Like button to specify their approval for a configured report. Please note that the Like button would be available in the object profile of that configured report and not in the configured report itself.
- The `Expert` concept allows a user to register him/herself as an expert of an object. For example, a user could specify him/herself to be an expert of a technology or technical component. The user could deregister him/herself at any time.

Each use case must be configured in the XML object ***ObjectAssociationsConfig***. Depending on the use case, the relevant value `Vote`, `Like`, or `Expert` must be specified in the XML attribute `Name`. For each use case, the solution designer can specify the object classes/object class stereotypes for which the content voting shall be implemented, the icon for the toolbar button as well as the icons for voting, and the text and tooltips to support the users to use the voting buttons. The use case may be enabled for read-only users as needed, thus allowing the enterprise to collect input from a large part of the user community.

The rating, like, and expert values capture for an object can be configured to be included in the **Attributes** section of the object profile or object cockpit or collected for a specified set of objects in a configured report.

The object class `ObjectAssociations` can be queried in configured reports to provide information about all object associations for a specific object or set of objects. To include the object association data for a specific object in the object cockpit, you could create a ValueControl interface element of the Sub-Type = ValueByQuery. The following displays a query to display the number of likes for the demand stereotype Idea in an object cockpit:

```
SELECT NULL, COUNT(p.REFSTR)AS 'Count'
FROM OBJECTASSOCIATIONS oa, DEMAND dem, PERSON p
WHERE dem.REFSTR = oa.A_OBJECT
AND dem.STEREOTYPE = 'Idea'
AND p.REFSTR = oa.A_USER
AND dem.REFSTR = @BASE
/* Alfabet Instructions */
PICTUREASSIGNMENT(Count, GreaterThan, 0, Small:LikeHeart, IconText,
"", "");
```


For more information about configuring object cockpits, see the section Configuring Object Cockpits for a Custom Object View. For more information about specifying configured reports, see the chapter Configuring Reports.

To configure the content voting capability:

1) Go to the **Presentation** tab and expand the **XML Objects** folder. The XML object ***ObjectAssociationsConfig*** opens.
2) Right-click ***ObjectAssociationsConfig*** and select **Edit XML**. The XML editor is displayed in the center pane.
3) Below the root XML element `ObjectAssociationsConfig`, specify an XML element `ObjectAssociationGroup` for the use case you will configure. The XML element `ObjectAssociationGroup` allows you to enable the read-only status for the object association as well as specify a top-level button with the subordinate object associations available as sub-buttons. This would be displayed on the user interface as a button (object association group) with options in a drop-down menu (object associations) and is useful if the captions for the object associations are too long for a toolbar button:

- Define the following XML attributes if you want to implement a toolbar button with a drop-down menu. If you do not plan to specify a toolbar button with a drop-down menu, then you can leave the following four XML attributes empty.
  - `Name`: Enter a technical name for the object association group. The name may not have empty spaces.
  - `Caption`: Enter the caption that shall be displayed on the object association group button in the object profile toolbar.
  - `Tooltip`: Specify the tooltip that shall be displayed on the object association group button in the object profile toolbar.
  - `Icon`: Specify the icon that shall be displayed on the object association group button in the object profile toolbar.

    All icons implemented in the XML object ***ObjectAssociationsConfig*** must be available in the icon gallery. For more information, see the section Adding and Maintaining Icons for the Alfabet Interface in the chapter Configuring the Visualization of the Alfabet Interface.

- If you want to enable read-only user profiles to access the object associations defined in the object association group, you must specify `"true"` for the XML attribute `EnableForReadOnly` for the XML element `ObjectAssociationGroup` as well as for the XML element `ObjectAssociation`.
- Create a child XML element `ObjectAssociation` for each use case you want to implement and define the following XML attributes. Please note that multiple object associations of the same use case (`Vote`, `Like`, or `Expert`) may be configured.
  - `Name`: Enter one of the following to specify the implementation of that use case for the object association.
    - `Vote` to allow an object to be voted on based on a rating system that supports rating an object (for example, via a star rating system)

- `Like` to allow an object to be endorsed based on a binary rating system (such as Like/Unlike)
- `Expert` to allow a user to register him/herself as an expert of an object

- To specify the object association for the use case where `Name` = `Vote`:
  - `ApplicationClasses`: Enter a comma-separated list of object classes and object class stereotypes where this object association may be implemented.
  - `ApplyToClasses`: Specify `"true"` if the object association should be available for the sub-classes specified in the `ApplicationClasses` attribute. For example, the class `TechicalService` has a sub-class `SolutionTechnicalService`. Users can rate the object of the parent class differently than the object of the sub-class.
  - `EnableForReadOnly`: Specify `"true"` if users with ReadOnly access permissions may define the object association. Specify `"false"` if users with ReadOnly access permissions may not define the object association. Please note that the XML attribute `EnableForReadOnly` must also be set to `"true"` for the XML element `ObjectAssociationGroup` if the XML attribute `EnableForReadOnly` is set to `"true"`.
  - `IconGallery`: Specify the icon group that shall be used for the voting. The icon group must contain multiple icons to represent the different values for rating (such as a 3-star or 5-star rating). Captions may be explicitly defined for the icons in an icon group. For more information, see the section Adding and Maintaining Icons for the Alfabet Interface in the chapter Configuring the Visualization of the Alfabet Interface.
  - `InteractionAddIcon`: Specify the name of the icon for the button that shall be displayed for the toolbar button.
  - `InteractionAddCaption`: Optionally, specify the caption that shall be displayed for the toolbar button.
  - `InteractionAddTooltip`: Optionally, specify the tooltip that shall be displayed for the toolbar button.
  - `EditAfterCreate`: Specify `"true"` if the user may modify the rating. Specify `"false"` if the user may not modify the rating
  - `Delete`: Specify `"true"` if the user may revoke the rating . Specify `"false"` if the user may not revoke the rating.
  - `InteractionDeleteIcon`: This XML attribute is typically not relevant for an object association of the type `Vote`.
  - `InteractionDeleteCaption`: This XML attribute is typically not relevant for an object association of the type `Vote`.
  - `InteractionDeleteTooltip`: You could provide a tooltip giving users instruction about how to revoke the rating. For example: "Click the rating to revoke your vote."
  - `ConfirmMessageDelete`: Specify a confirmation message that the user must confirm if the voting is revoked.

- To specify the object association for the use case where `Name` = `Like` or `Expert`:
  - `ApplicationClasses`: Enter a comma-separated list of object classes and object class stereotypes where this object association may be implemented.
  - `ApplyToClasses`: Specify `"true"` if the object association should be available for the sub-classes specified in the `ApplicationClasses` attribute. For example, the class `TechicalService` has a sub-class `SolutionTechnicalService`. Users can rate the object of the parent class differently than the object of the sub-class.
  - `EnableForReadOnly`: Specify `"true"` if users with ReadOnly access permissions may define the object association. Specify `"false"` if users with ReadOnly access permissions may not define the object association. Please note that the XML attribute `EnableForReadOnly` must also be set to `"true"` for the XML element `ObjectAssociationGroup` if the XML attribute `EnableForReadOnly` is set to `"true"`.
  - `InteractionAddIcon`: Specify the name of the icon for the button that shall be displayed for the toolbar button.
  - `InteractionAddCaption`: Optionally, specify the name of the icon for the button that shall be displayed for a positive vote.
  - `InteractionAddTooltip`: Optionally, specify the tooltip that shall be displayed for the button that shall be displayed for a positive vote.
  - `InteractionDeleteIcon`: Specify the name of the icon for the button that shall be displayed for a negative vote. For example, to unlike an object.
  - `InteractionDeleteCaption`: Optionally, specify the name of the icon for the button that shall be displayed for a negative vote.
  - `InteractionDeleteTooltip`: Optionally, specify the tooltip that shall be displayed for the button that shall be displayed for a negative vote.
  - `Delete`: Specify `"true"` if the user may revoke the rating . Specify `"false"` if the user may not revoke the rating.
  - `ConfirmMessageDelete`: Specify a confirmation message that the user must confirm if the voting is revoked.

4) In the toolbar, click the **Save**  button to save the XML definition.

Configuring the Questionnaire Functionality

The basic questionnaire functionality is provided via the standard functionalities. The creation, distribution, answering, and management of questionnaires provided by these functionalities is described in the reference manual *User and Solution Administration*.

The feature requires configuration in Alfabet Expand for the following:

- Defining configured reports to find the relevant objects and optionally the relevant users for the questionnaire.
- Creating questionnaire indicators for additional objects for an already activated questionnaire.
- Optional configuration of an environment for answering questionnaire indicators with advanced edit functionalities.
- Implementation of a review and approval process for the questionnaire.
- Providing an overview of questionnaire outcome via calculation of scores based on numeric values for each possible answer for the questions of the questionnaire.

The above-mentioned functionalities are mainly based on the definition of configured reports and events. In the following, a complete workflow is given for each feature including examples. Basic knowledge about the definition of configured reports and events is assumed. The descriptions link to the detailed information about configured reports and events where applicable.

The following information is available:

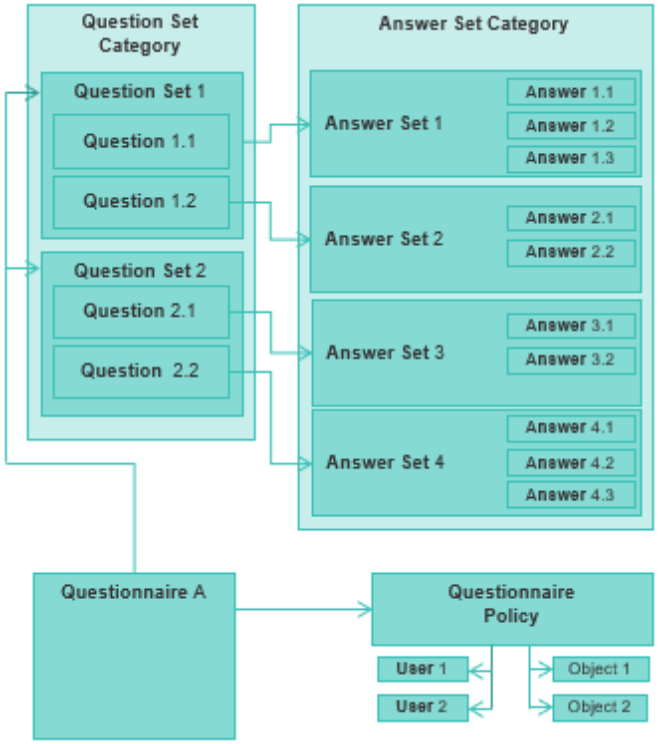
Overview of the Standard and customized Configuration of Questionnaires

A complete overview of the object classes involved in questionnaire creation, management, and answering is required to built configured reports for the feature and set the right parameters for event templates. This section provides an overview of the relevant classes for questionnaire creation. Details about the technical names of the classes and the object class properties involved are given in the following sections in the context of the individual configuration steps.

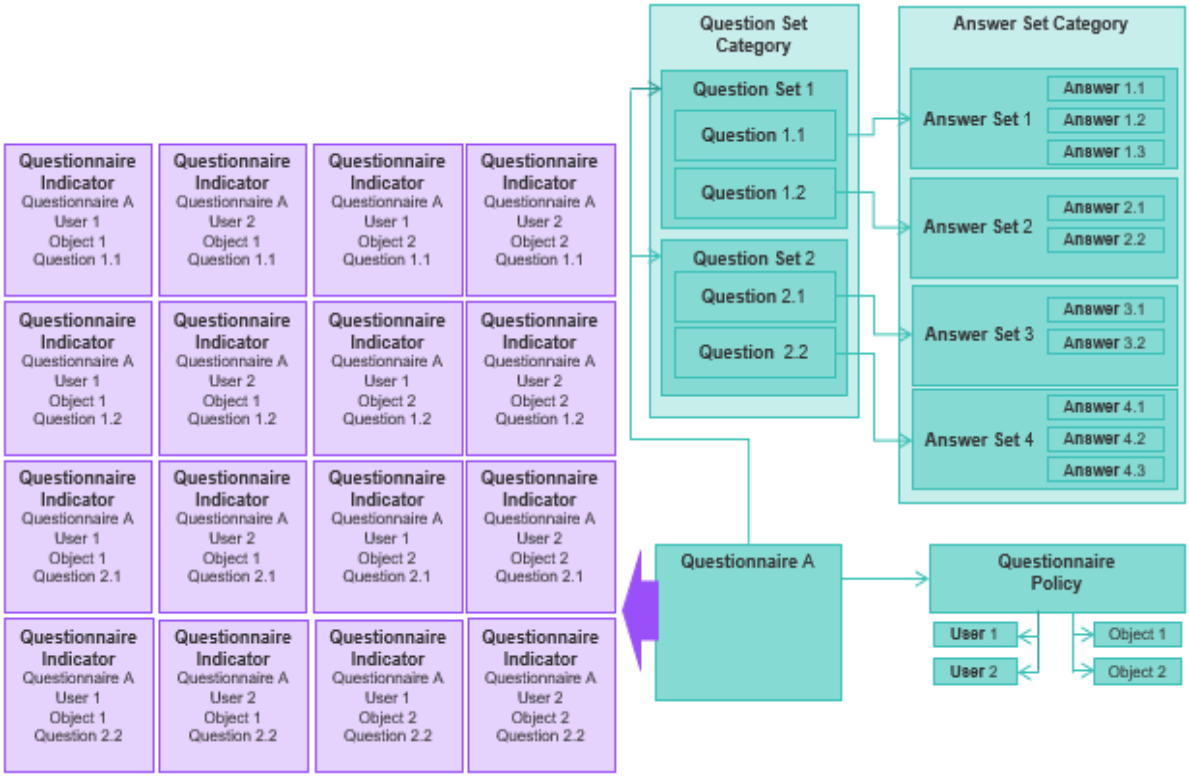
A questionnaire consists of question sets that contain the questions to be answered. Each question either links to an answer set which consists of a predefined set of possible answers that the user can select in a dropdown list, or it provides a free text field. For each question, the user can additionally provide a personal comment.

The question sets of the questionnaire are not defined directly in the questionnaire. Question sets and answer sets are created independent from the questionnaire as child elements of question set categories and answer set categories. this allows answer sets to be re-used in different questions and question ses to be added to multiple different questionnaires. Questions can only be added to questionnaires via question sets. Preconfigured answers can only be added to questions via answer sets.

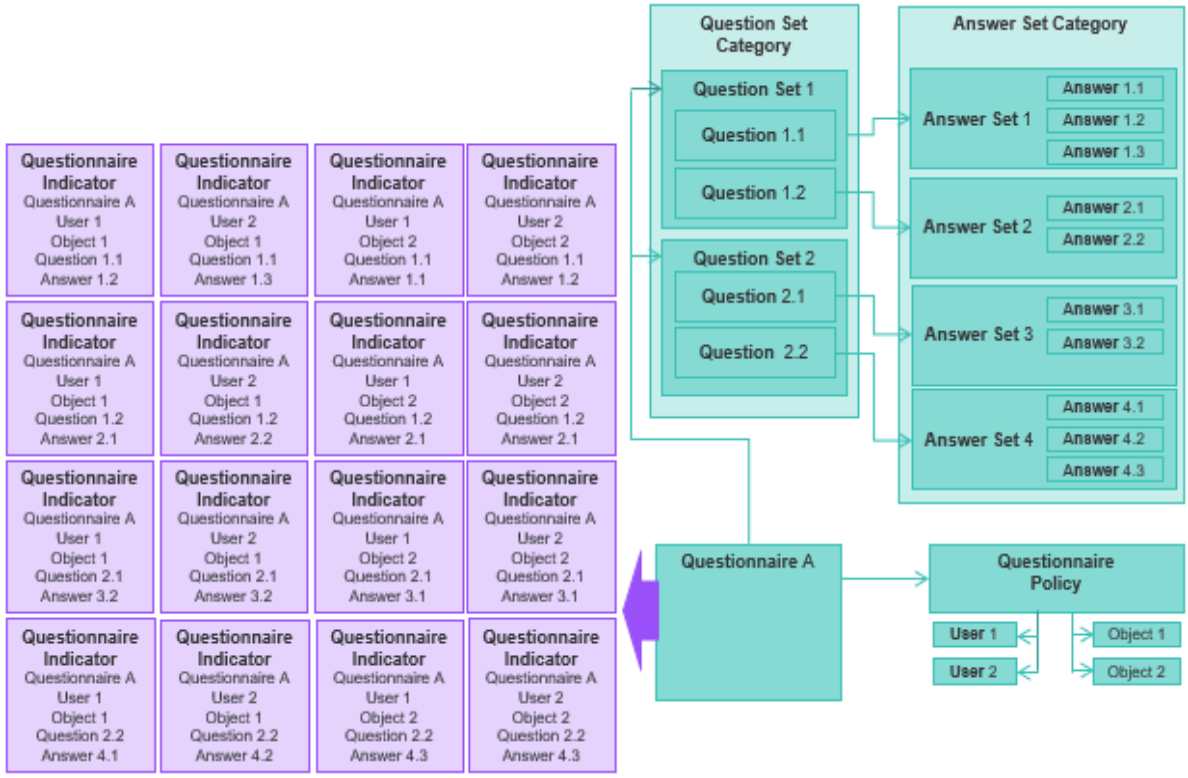
Questionnaires are defined to request information about objects in the Alfabet database from users that are responsible for the object. For example, a questionnaire can be defined to find out whether applications comply to the new company security standards. The questions shall be answered for all applications by all users with a role architect for the application. The relevant users and objects are determined via a questionnaire policy. Like question sets, the questionnaire policy is defined independent of the questionnaire and can be reused in multiple questionnaires. Questionnaire policies reference one or multiple configured reports that return the relevant objects. the relevant users can be the authorized user of each object, users with a defined role for the object, or any other users found via a configured report.



On starting the questionnaire aafter complete configuration, a questionnaire indicator is created for each question in the questionnaire for each combination of object and user derived from the questionnaire policy. questionnaire indicators reference the relevant questionnaire, question, user, and object.

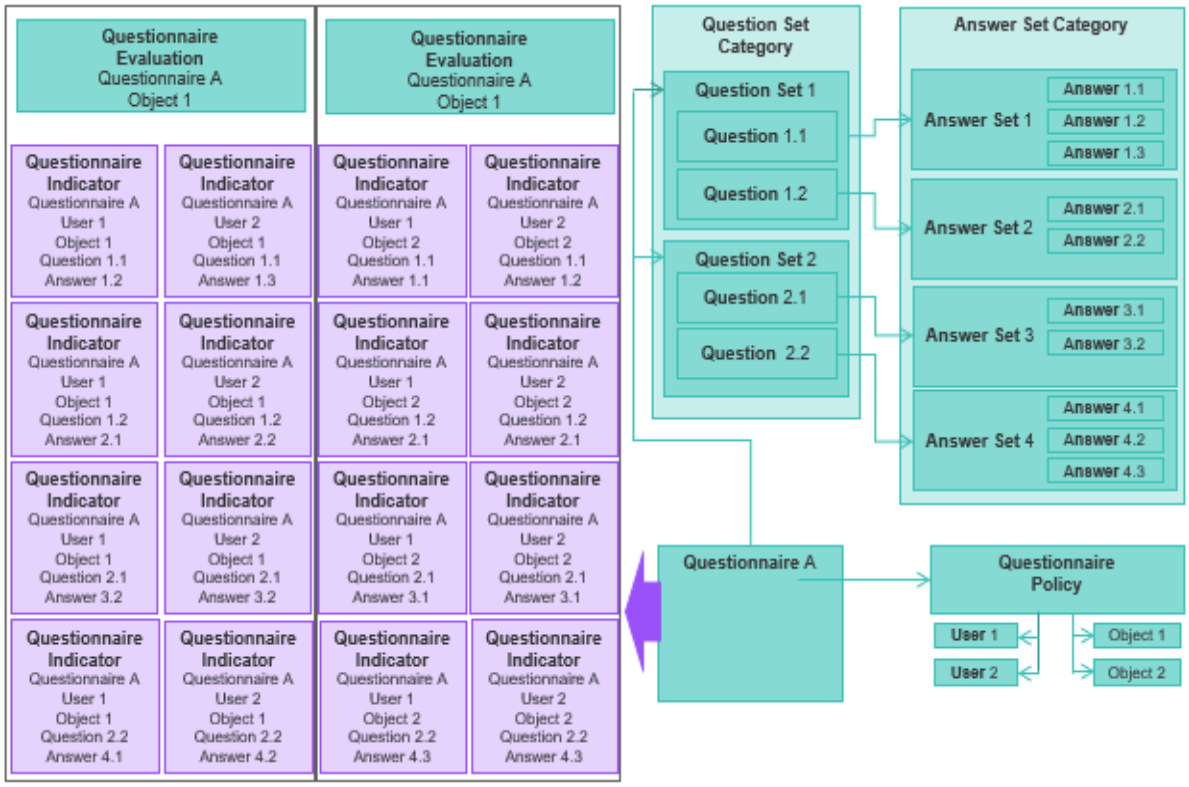


Users can edit questionnaire indicators assigned to them. The answer selected by the user or the user's free text answer will then be stored in the questionnaire indicator.



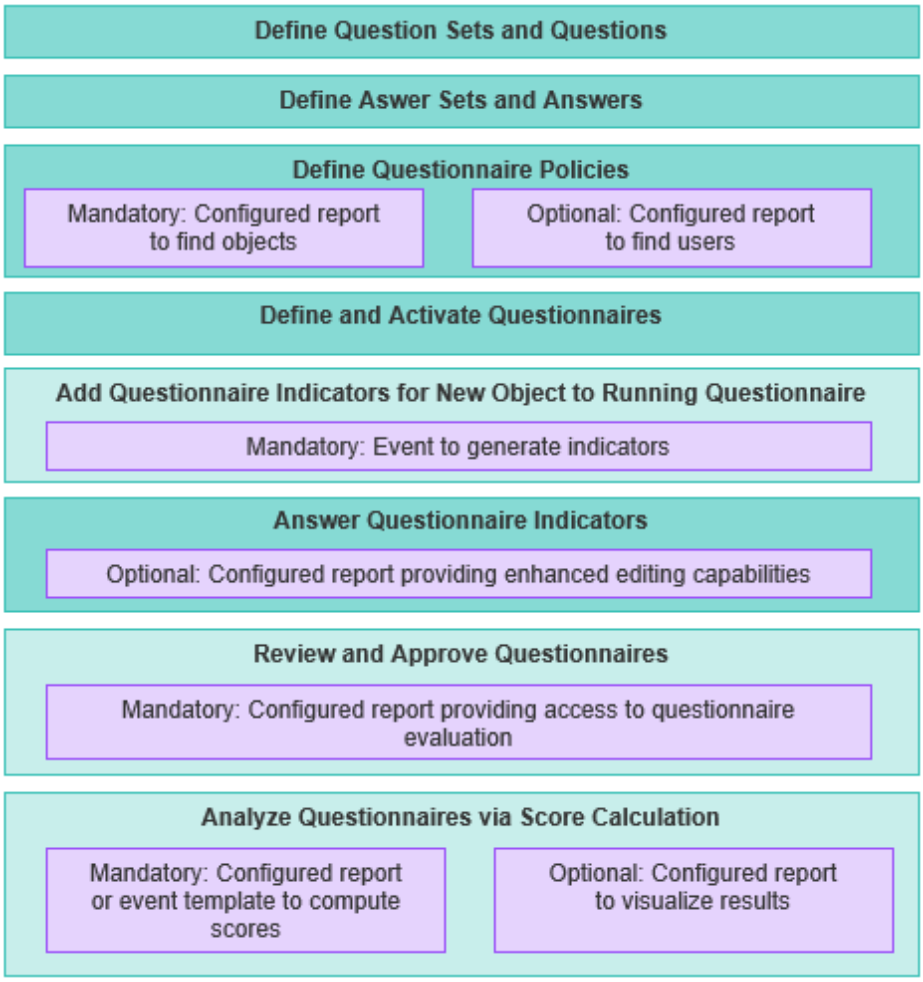
After the users have answered the questions in a questionnaire via the questionnaire indicators assigned to them, a review process can be implemented. Review of the answers can be done on the level of the individual questionnaire indicators, which can store a review comment and a status. Review can also be done on

the level of the questionnaire. For this purpose, questionnaire evaluation objects are created on activation of the questionnaire for each combination of questionnaire and object. Questionnaire evaluation objects provide the ability to review as well as to approve all answers from all involved users answering the questions in the questionnaire for the specific object. For this purpose, the questionnaire evaluation object can store a release status, a review comment and an approval comment.



In addition, the outcome of a questionnaire can be expressed by numbers assigned to each answer in the answer sets. Answers can be assigned numeric values according to their relevance for a given evaluation. For example, if the risk of applications shall be evaluated, a higher number can be given to answers indicating a high risk than to answers indicating a low risk. After the users answered the questions in the questionnaire, a score can be computed from the numeric values of the selected answers. Scores are providing information about the outcome of a questionnaire without detailed analysis the individual answers. In the example, a low overall score for the risk of applications means a low overall risk, while a high score indicators that actions might be required to reduce risk.

If you would like to use the questionnaire functionality, you can configure many of the relevant objects in the functionalities on the Alfabet user interface. Other parts of the questionnaire functionality require configurations in Alfabet Expand. The image below provides an overview of the activities that are involved in working with questionnaires. Optional features like score computation are displayed in a lighter color. If configurations in Alfabet Expand are involved or are optionally involved, this is displayed in lavender colored boxes within the feature boxes.



Configuring Reports Finding Objects Or Users for the Questionnaire Policy

A questionnaire policy defined the users that shall answer the questions in the questionnaire and which objects the users shall answer the question for.

The definition of a configured report is required to find the relevant objects.

For the specification of the relevant users, the definition of a configured report is optional and the authorized user of the object or users having a specific role assigned for the object can alternatively be specified to answer the questionnaire. The way questionnaire indicators are defined is different for the different methods.

- If the authorized user or users assigned to a role for object shall answer the questionnaire, the questionnaire indicators generated for an object are generated for the user that is authorized user of that object or assigned a role for that object. For example, if user 1 is the authorized user for object 1 and user 2 is the authorized user for object 2, only user one will have to answer questions about object 1 and only user 2 will have to answer questions about object 2.
- If users are defined via a query, a questionnaire indicator is defined for each combination of relevant object and user. For example, if the questionnaire policy finds user 1 and user 2 and finds object 1 and object 2, both user 1 and user 2 will have to answer the questions for both object 1 and object 2.

The following configuration steps are required:

- Defining a Report Category for Questionnaire Policies
- Defining a Configured Report for a Questionnaire Policy

Defining a Report Category for Questionnaire Policies

Only configured reports assigned to a report category for the use case `QuestionnairePolicies` can be used in questionnaire policies. You can define one category used for all reports, or you can define multiple categories, For example, to mark reports as relevant for finding objects or for finding users by assigning them to different categories for the use case `QuestionnairePolicies`.

To define a report category for the use case `QuestionnairePolicies`:

1) In the **Presentation** tab of Alfabet Expand, expand the explorer node **XML Objects**.
2) Right-click on the XML object ***UseCaseCategories*** in the explorer and select **Edit XML** from the context menu. The XML object opens in the middle pane.
3) Enter the following code as child element of the XML element ***UseCaseCategories***:

```
<UseCaseInfo UseCase="QuestionnairePolicies">
   <ScopeInfo Scope="Report"
   Categories="CommaSeparatedListOfCategoryNames" />
</UseCaseInfo>
```

   The XML attribute `Categories` must be set to either one category name or a comma separated list of category names. Please note that a category name is a technical name and should not contain special characters or whitespaces. All category names that are defined can be used in the **Category** attribute of a configured report to activate the report for the use in questionnaire policies.

4) In the toolbar, click the **Save**  button.

Defining a Configured Report for a Questionnaire Policy

To define a configured report for use in questionnaire policies:

1) In the **Reports** tab, right-click the **Reports** folder or a sub-folder thereof and select **Create New Report**. You will see the new configured report listed below the selected report folder.
2) In the property window, edit the following attributes of the configured report:
   - **Name**: Change the technical name of the configured report to a meaningful name. The technical name of the configured report will be displayed in the questionnaire policy editor.

     The technical name is used for identification of the configured report in the context of technical processes such as merging the report configuration with a report configuration saved as an XML file. Therefore, the following is required when defining the name of a configured report:

- The name must be unique. The name must be unique regardless of the report folder it is structured in.
- A validation mechanism checks for correct syntax when defining a technical name. Please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.

- **Category**: Select a category defined for the `QuestionnairePolicies` use case in the XML object ***UseCaseCategories***.
- **Type**: Select `Query`.
- **Alfabet Query/Query As Text**: Click the **Browse** button in the **Alfabet Query** attribute to open the **Alfabet Query Builder** or define the Alfabet query in a simple text editor in the **Query As Text** attribute. The following is required in the query definition:
  - The `FIND` class must be `Person` to find the users responsible for answering the questionnaire, or the object class of the objects for which the questionnaire shall be answered.
  - Show properties are ignored.
  - `FROM` clauses and `WHERE` clauses can be used to find the relevant objects or users.

Optionally you can set other report attributes, for example, to provide a caption and description and to exclude the configured report from being available for other functionalities. This section only describes the attributes technically required to execute the configured report in the questionnaire policy. For a complete description of available attributes, see Creating a Tabular Configured Report of the Type Query and Creating a Tabular Configured Report of the Type NativeSQL.

3) In the toolbar, click the **Save** button to save your changes.
4) Right-click the configured report and select **Set Report State to 'Active'** from the context menu.

Adding Questionnaire Indicators for New Objects to a Started Questionnaire

Starting a questionnaire will create the questionnaire indicators for all objects found by the query in the assigned questionnaire policy.

The number of relevant objects may change during the time period for answering the questionnaire. New objects may be created or the data for existing objects may be changed to match the search conditions in the questionnaire policy.

To add additional questionnaire indicators for a new object to a started questionnaire, an event can be added to a wizard or workflow. Events are based on a customer configured event template. They can be triggered via an on enter, on cancel, or on exit step action of a wizard step or via an on enter, on refuse, on expired, or on exit step action of a workflow step.

For example, an active questionnaire is currently answered by the authorized users of all applications with a criticality indicator set to 'High' or 'Very High'. The evaluation page view for the application is available as a wizard step in a wizard. If a user changes the criticality indicator and moves on to the next page of the wizard, the event will be triggered, and questionnaire indicators will be created for the application.

To trigger generation of questionnaire indicators on execution of a workflow or wizard creating or changing a relevant object, you must create an event template and add the event template to a wizard step action or a workflow step action. These steps are described below for the start of a questionnaire indicator.

Please note that the execution of event templates requires a basic configuration of event management. This basic configuration is described in the chapter Configuring Events and has to be available to perform the configurations described in the following.

- Configuring an Event Template for Generation of Questionnaire Indicators
- Adding the Event Template to a Workflow or Wizard

Configuring an Event Template for Generation of Questionnaire Indicators

This section is limited to a basic description of the settings in the event template. If you are not familiar with the definition of events, you should read the chapter Configuring Events first to understand the meaning of the different settings for event execution.

Event templates are defined in the **Events** tab of Alfabet Expand:

1) Go to the **Events** tab of Alfabet Expand.
2) Right click the node **Events** and select **Create New Questionnaire Add Indicators Event** from the context menu. The new event template is added on the root level of the event management tree.
3) Click on the new event template and edit the following attributes:
   - **Name**: Enter a unique name for the event template. The name is used to identify the event templates in configurations.
   - **Description**: Optionally enter a short text describing the functionality of the event. This information is only visible in the event template attributes in the **Events** tab to inform For example, other solution designer about the functionality implemented with the event template.
   - **Execute Immediately**: Setting this attribute to `True` will cause the Alfabet Web Application to execute the event immediately by triggering the REST API call for the event. The event is nevertheless written to the `ALFA_EVENT_BUS` table, but with the `STATE` set to `FINISHED` or, if execution was not successful, to `ERROR`. Setting this attribute to `False` will lead to standard scheduling of the event via the `ALFA_EVENT_BUS` table and execution via the Alfabet Server.
   - **Connection Type**: Select `SelfReflective`, if the event shall be executed for the same Alfabet database it is triggered from, or `Query`, if the event shall be executed on an Alfabet database that is defined via an **Alfabet Database Connection**.
   - **Connection via Query** / **Connection via Query as Text**: This attribute is only applicable and visible if the **Connection Type** is `Query`. Define a query that returns the `REFSTR` of the

Alfabet database connection to be used for execution of the event. The query must return a single `REFSTR`. It can either be defined as native SQL query or Alfabet query:

- Use the **Connection via Query** attribute to define an Alfabet query. Show properties are not required for the query.

```
ALFABET_QUERY_500
FIND Alfabet_DBConnection
WHERE Alfabet_DBConnection.Name = 'EventConnection'
```

- Use the **Connection via Query as Text** attribute to define a native SQL query with the `REFSTR` of the Alfabet database connection as only `SELECT` argument.

```
SELECT REFSTR
FROM ALFABET_DBCONNECTION
WHERE ALFABET_DBCONNECTION.NAME = 'EventConnection'
```

- **Group**: If you would like to structure your event templates in folders, enter the name of the event template folder the event template shall be located in. If event template folders have already been defined for other event templates, they will be displayed in a drop-down list and can be selected. If you want to define a new event template folder, enter a new name into the field. The folder is then created in the explorer. The event template is moved to the selected event template folder.

4) Expand the event's node in the explorer and expand the **Parameters** node. Three parameter nodes are displayed.

5) Click each of the parameter nodes and define the following in the attribute section:

- ***@Questionnaire***: This parameter is mandatory. Define a query finding the relevant questionnaire. The questionnaire must have the release status defined as `ApprovedStatus` for the object class `Questionary` in the XML object ***ReleaseStatusDef*** and the questionnaire target date must be in the future.

For more information about the definition of release status values, see Configuring Release Status Definitions for Object Classes.

You can either define an Alfabet query in the **Value for Parameter via Query** attribute or a native SQL query in the **Value for Parameter via Query as Text** attribute. The query must return the `REFSTR` of the relevant questionnaire in a dataset with one column and one row.

For example, a native SQL query to find the questionnaire may read:

```
SELECT NULL, QUESTIONARY.REFSTR
FROM QUESTIONARY
WHERE QUESTIONARY.NAME = 'Security Assessment for Cloud
Servers'
```

- ***@Object***: This parameter is mandatory. Define a query finding the relevant object only. Typically, this will be the base object of the wizard or workflow you are currently working with. You can either define an Alfabet query in the **Value for Parameter via Query** attribute or a native SQL query in the **Value for Parameter via Query as Text** attribute. The `FIND` class of

the Alfabet query must be the object class of the relevant object. The native SQL query must return the `REFSTR` of the object class of the relevant object as first `SELECT` argument. The query must contain conditions that limit the result to a single object.

For example, an Alfabet query to find the object may read:

```
ALFABET_QUERY_500
FIND Device
WHERE Device.REFSTR CONTAINS:BASE
AUTODSINFO
QUERY_XML
<QueryDef>
   <ShowProperty Type="Property" ClassName="Device"
   Name="REFSTR" />
</QueryDef>
```

- ***@UserName***: This parameter is optional. If specified, the string defined in the parameter will be written into the `CREATION_USER` object class property of the questionnaire indicators generated by the event. Please note that according to the setting of the **Update History User Name** in the server alias of the Alfabet Web Application, the `CREATION_USER` object class property stores either the user name or the technical name of users. The information returned for the parameter must match the setting in the server alias.

  For information about the setting in the server alias, see *Configuring User Information Displayed in History Tracking* in the reference manual *System Administration*.

You can either define an Alfabet query in the **Value for Parameter via Query** attribute or a native SQL query in the **Value for Parameter via Query as Text** attribute. The query must return the user name or the technical name of an Alfabet user in a tabular dataset with a single row and a single column. Instead of the defining the user via a query, a fixed user name or technical name can be defined in the **Default Value** attribute.

For example, a native SQL query to set the `CREATION_USER` to the user that is triggering the event may read:

```
SELECT PERSON.REFSTR, PERSON.USER_NAME
FROM PERSON
WHERE PERSON.REFSTR = @CURRENT_USER
```

6) Click the **Save**  button to save your changes.

7) Right-click the event template and select **Set Event State to Active** from the context menu.

Adding the Event Template to a Workflow or Wizard

Events can be triggered via an on enter, on cancel, or on exit step action of a wizard step or via an on enter, on refuse, on expired, or on exit step action of a workflow step. The implementation is the same for all wizard step actions and workflow step actions:

1) In Alfabet Expand, navigate to the wizard step or workflow step that you want the event to be triggered from.
2) Create a new wizard step action or workflow step action, as needed:
   - To create a new wizard step action for a wizard step, right-click the relevant **Action On Entered Step**, **Action On Exited Step**, or **Action On Cancelled Step** node and select the option to create the new wizard step action. For more information about configuring wizard step actions, see the section Configuring Wizard Step Actions for a Wizard Step.
   - To create a new workflow step action for a workflow step, right-click the relevant **Action On Entered Step**, **Action On Refused Step**, **Action On Expired Step**, or **Action On Exited Step** node of the workflow step and select the option to create the new workflow step action. For more information about configuring workflow step action to trigger an event, see the section Configuring Events to Be Triggered for a Workflow Step.
3) Click the new wizard step action/workflow step action in the explorer and edit the following attributes in the attribute window:
   - **Technical Name**: Enter a name for the wizard step action/workflow step action. This name is not displayed in the Alfabet interface.
   - **Action Type**: Select `TriggerEvent`.
   - **Technical Comment**: Enter any relevant information about the wizard step action/workflow step action.
   - **Event Template**: Select the relevant event template from the drop-down list.
4) Optionally, you can define a condition to execute the event depending on the availability of specific data returned by a query. For example, you could specify that the event may only be executed if a property of the base object of the wizard is set to a specific value. Edit the following attributes in the attribute window to define the condition:
   - **Check Query** / **Check Query as Text**: Define a query that will return a result only if a condition is met. The query can be defined either as an Alfabet query in the **Check Query** attribute or as a native SQL query in the **Check Query as Text** attribute.
   - **Result Type**: Select `Positive` if the event shall only be executed if the query returns a result. Select `Negative` if the event shall only be executed if the query does not return a result.

For example, an event to update an application's indicators when a wizard step is exited shall be triggered if the application's release status is set to the value "Approved". The following query will only return a result if the application that is the base application of the wizard step has the correct release status:

```
SELECT REFSTR
FROM APPLICATION
WHERE APPLICATION.STATUS = 'Approved'
```



```
AND APPLICATION.REFSTR = @WIZARDBASE
```

The **Result Type** is set to `Positive` so that that the event is only executed if the query returns a result.

5) Click the **Save**  button to save your definitions.

Configuring the AI-Enabled Data Quality Analysis Functionality

Analysis of IT infrastructure and any IT planning activity relies on up-to-date and complete data. Alfabet offers an **AI-Enabled Data Quality Analysis** functionality that scans the data about objects of an object class for similarity in attribute settings, forming clusters of objects that are structurally related based on a selected set of class properties, roles and indicators. Within these clusters, objects will only deviate in the availability of a low percentage of attributes and have high similarity in other settings. Therefore, identification of attributes that are only missing for a few of the objects in the cluster is a good way to check data completeness in a context that takes data structure into account.

The **AI-Enabled Data Quality Analysis** functionality is available for all object classes that can be structured in groups such as application groups, project groups, etc.

This feature requires configuration in Alfabet Expand for the following:

- Configuring Group Object Class for Storing Report Reference in Alfabet Expand
- Creating Parent Group Object for Cluster Storage in Alfabet User Interface
- Creating a Configured Report with AI-Enabled Data Quality Analysis functionality
  - Configuring a Report with AI-enabled Data Quality Analysis using the Report Assistant
- Enabling user access for standard functionality

Configuring Group Object Class for Storing Report Reference in Alfabet Expand

The **AI-Enabled Data Quality Analysis** functionality is available for all object classes that can be structured in groups such as application groups, project groups, etc. The results of the analysis are stored using already existing group infrastructure.

The analysis stores the results in a hierarchy of groups with a specific object class stereotype. If object class stereotypes are already configured for the group object class, it is recommended to define a separate object class stereotype for the analysis to clearly distinguish between groups created for the analysis and groups to structure objects. If object class stereotypes are not defined for the group class, you must define at least one object class stereotype for the definition of groups structuring objects in addition to the object class stereotype used to store analysis results.

A custom object class property for storing a reference to the object class `ALFA_REPORT` must also be defined for the group object class. This property is crucial in storing information about the configured report used to trigger the analysis.

Do the following to configure the group object class in Alfabet Expand:

1) Go to the **Meta-Model** tab, expand the **Classes** node and click on the relevant group object class to be configured.
2) In the attribute window, click the **Browse** button available for the **Stereotypes** attribute to open the XML text editor.
3) Define a stereotype as child elements of the XML element `ClassStereotypes`. The stereotypes are defined with the following XML syntax: `<Stereotype Name="" Caption="" CaptionPlural="" Description="" Comments="" HasMandates="" IDPrefix="" />`.

   For each object class stereotype, define the following XML attributes:

An error message will be displayed upon closing the text editor if syntax errors are written in the XML definition of the **Stereotypes** attribute.

- `Stereotype Name`: Enter the technical name for each object class stereotype. You may define a name of up to 64 characters. This name should not be changed once the object class stereotypes are implemented.

  Changing the technical name after objects have been created based on the object class stereotype requires such objects to be updated by means of an ADIF scheme. For more information, see the reference manual *Alfabet Data Integration Framework*.

- `Caption`: Enter the caption for the object class stereotype. This is the name that users will see in the Alfabet user interface.

  The caption of the object class stereotype should be different than the caption of its object class. This is particularly important for the implementation of the search functionality for the object class stereotype. For more information, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.

- `CaptionPlural`: Enter a plural form for the caption. This is the name in plural that users will see in the Alfabet user interface.
- `Description`: Provide a brief description about the object class stereotype that will be displayed to users in the **Stereotype Selector** when defining a new object of the group class. The description should inform the user that this stereotype is used for technical processes and shall not be used to create groups manually.
- `Comments`: Optionally enter text in order to provide information about the object class stereotype. This information is only for the purposes of configuration. It is not displayed in the Alfabet user interface but explains the meaning of the stereotype for solution designers. The comment can for example, inform about the role of the stereotype in data quality analysis.
- `HasMandates`: Enter "`true`" if the mandate capability should be available for the object class stereotype. Enter "`false`" if the mandate capability should not be available for the object class stereotype. If the XML attribute `HasMandates` is defined as "`false`" for an object class stereotype, objects based on the object class stereotype will be visible to users with any mandate implemented in your enterprise. For information about additional configuration requirements necessary to implement the mandate capability, see the section *Implementing the Mandate Capability for a Federated Architecture* in the chapter Configuring Access Permissions for Alfabet

Additional attributes can be set for stereotypes. However, these are not relevant for this usecase and are described in detail in the section Configuring Object Class Stereotypes for Object Classesof chapter *Configuring the Class Model*.

Example of the XML definition in the **Stereotypes** attribute for the object class `ComponentGroup`:

```
<ClassStereotypes>
   <Stereotype Name="ComponentGroup" Caption="Component
   Group" Description="This stereotype is for all ordinary
```



```
component groups." CaptionPlural="Component Groups"
Comments="" HasMandates="true" IDPrefix="" />

<Stereotype Name="ComponentDQAG" Caption="Component Data
Quality Analysis Group" Description="This stereotype is
for Component groups used to conduct data quality cluster
analysis." CaptionPlural="Component Data Quality Analysis
Groups" Comments="" HasMandates="true" IDPrefix="" />

</ClassStereotypes>
```

4) Click the **Save** button to save your changes.

5) Right-click the relevant object class node and select **Add New Property**. The **Create New Property** editor opens. Define the following fields:

   - **Property Name**: Enter a unique name for the custom object class property.
   - **Technical Name**: Enter a unique technical name for the custom object class property that will be used in the Alfabet database table as a column header.

     Please note that the `Technical Name` may not begin with an empty space nor include special characters. It should consist of upper-case letters only and must not be longer than 30 characters. For information about additional configuration requirements necessary to create a new custom property, see the section Creating a New Custom Property in the chapter Configuring the Class Model.

6) Click **OK** to save the definition.

7) In the attribute window of the custom object class, define the following attributes:

   - **Property Type**: Select `Reference`.
   - **Type Info**: Enter `ALFA_REPORT`.
   - **Caption**: Specify a caption to be displayed for the property in the Alfabet user interface.
   - **Comments**: Provide information relevant to solution designers regarding the maintenance of the custom object class property. For example: "This property is used for data quality cluster analysis and will be filled automatically."
   - **Hint**: Enter informational text about the custom object class property to be displayed in the Alfabet interface.

8) In the toolbar, click the **Save** button to save your changes.

Creating Parent Group Object for Cluster Storage in Alfabet User Interface

It is necessary to create a parent group instance specifically to store each analysis as a group with the clusters as sub-groups that contain the objects that belong to that cluster. This is done in the Alfabet User Interface for the relevant object group.

When an analysis is triggered, sub-groups representing the clusters are created and stored under this parent group. If the analysis is triggered again later, the sub-groups and the content therein will be re-created.

To create an analysis-specific parent group, navigate to the relevant object group view on the user interface and create a new group instance. A context sensitive help file is available in the standard Alfabet online help that explains the creation of object groups to end users.

Creating a Configured Report with AI-Enabled Data Quality Analysis functionality

On the Alfabet user interface, the user triggers the analysis via a configured report. Each analysis is triggered by a different configured report, therefore, one configured report must be defined for each analysis that shall be performed. The report also defined parameters changing the display of the standard views for data analysis.

When the group object class and the parent group have been configured, a configured report used for triggering the data quality analysis can then be created.

Do the following to configure the report in Alfabet Expand:

1) In the **Reports** tab, right-click the **Reports** folder or a sub-folder thereof and select **Create New Report**. You will see the new configured report listed below the selected folder.
2) In the attribute window, select `Custom` in the **Type** field.
3) In the attribute window, define the following attributes for the configured report:

- **Name**: Optionally, change the technical name of the configured report. The technical name of the configured report must be unique.

  The technical name is used for identification of the configured report in the context of technical processes such as merging the report configuration with a report configuration saved as an XML file. Therefore, the following is required when defining the name of a configured report:

  - The name must be unique. The name must be unique regardless of the report folder it is structured in.
  - The name may not be changed once a configured report has been created.
  - A validation mechanism checks for correct syntax when defining a technical name. Please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.

- **Template**: Select `DataQualityAIReport` to create a configured report that triggers the AI-enabled data quality analysis.
- **Caption**: Enter a caption for the configured report. The caption you define here will be displayed in the **Reports Administration** functionality in the **Administration** module and on top of the **Configured Reports** in the Alfabet user interface.
- **Description**: Provide short information about the configured report that is useful to Alfabet users. This comment will be displayed on the Alfabet user interface in the header of the configured report in a single line under the report caption and in the **Description** field for the configured report in the table of all views listing configured reports, like the **Configured Reports** views of the **Search** functionalities.

In the configured report, the description will be truncated if it is longer than one line on the screen. Therefore, the description should be short to ensure complete display in the configured report header.

Additional attributes are available to configure visibility, search parameter helping the user to find the configured report, and parameters relevant for filters. These advanced attribute settings are described in the section Creating a New Report for Multi-Editing of Objects in the chapter Configuring Reports.

4) In the explorer, right-click the configured report and select **Create Custom Report View** to build a view for the report.
5) In the explorer, right-click the configured report and select **Start Alfabet Report Assistant**. The **Report Assistant** opens.
6) Reports triggering data quality analysis are designed in the **Report Assistant**. Configure the report as described below in the section Configuring a Report with AI-enabled Data Quality Analysis using the Report Assistant.
7) Click **OK** to locally save the configuration.
8) In the toolbar, click the **Save** button to save your changes.

Configuring a Report with AI-enabled Data Quality Analysis using the Report Assistant

Explorer Tab Element	Required to Configure:
Data Source Definition	Definition of a query that selects the range of data from the Alfabet database to be analysed.
Default Layout	Selection of features to be included in the analysis and definition of the structural information provided for the relevant object class.
Options	Further definition of attributes for relevant properties available for the object class. Only properties of the data type `Reference` and role types are displayed here.

In the **Data Source Definition** tab, define either a native SQL query or Alfabet query that finds the objects that shall be analyzed:

An Alfabet Query can be defined in the **Alfabet Query** attribute via the Alfabet Query builder. The selection of classes as `FIND` class is limited to the object classes that can be analyzed. `FROM` and `WHERE` clauses can be used to limit the analysis to a subset of the objects available for the `FIND` class. Show Properties are not required.

A native SQL Query can be defined in the **Native SQL** attribute. The first argument of the `SELECT` statement must return the `REFSTR` of the objects that shall be analyzed. Other arguments for the `SELECT` statement are not required. `Joins` and `WHERE` clauses can be used to limit the analysis to

a subset of the objects available for the selected object class. For information about defining queries, see the chapter Defining Queries.

In the **Default Layout** tab, the table on the left displays all object class properties of the data types `string`, `Integer`, `Real`, `Boolean`, and `Reference` of the selected object class as well as indicators and role types assigned to the selected object class. These are the analyzable features. The column headers give information about the data in the column while also allowing easy editability of the features.

Clicking a column header sorts the table rows according to the values in the column. The technical name is displayed in the **Feature** column, the date type is displayed in the **Type** column, and the caption displayed in the column headers of the analysis views is displayed in the **Caption** column. The **Exclude** column allows one to set a checkmark to exclude the feature from the analysis, the **Show in Editor** column allows one to set the checkmark if the property is to be included into the editor and the **View Only** column allows one to set a checkmark if the property is only to be displayed non editable in the editor and is only considered if **Show in Editor** is also checked for the property.

The buttons above the table apply settings to all the features in the table. The **Exclude All** and **Include All** buttons set and remove checkmarks in the exclude column of the table for all the features, **Refresh Available Features** button simply refreshes the features available for analysis as generated by the query and **Number of Fixed Columns** defines which feature column should be fixed in the editor.

For the general structure of the report, the following attributes are required to be defined in the **Default Layout** tab:

Attribute	Description
Data Completion Recommendation Marker	Define a string that will be displayed in all cells of the cluster analysis views that are identified as missing information for the combination of a feature and object.
Max column Header Length	Define the maximum length of the column headers in the cluster analysis views. The name of a feature will be truncated if it is longer than the defined maximum and should be adapted to the screen size of the users performing the analysis for better readability.
Parent Group	Select the object class stereotype of the group object class that shall be used to generate the analysis group that is the parent group for the cluster groups.
Stereotype	Select the object class stereotype of the group object class that shall be used to generate the cluster groups.
Prefix	Define a prefix to be used in naming the cluster results from the data quality analysis.
Report Reference Property	Select the custom property of the group object class that shall store the reference to this configured report.

Attribute	Description
Property Completion Threshold	Define the threshold percentage of objects in a cluster that should have data filled into the object class property for the clustering analysis to regard as complete. Properties with less than this percentage across the objects in the cluster will not be considered in the analysis.
Minimum Cluster Size for Gap Detection	Define a minimum number of objects a cluster should have to be included in data quality analysis. Clusters generated with a number of objects less than the number defined in this property will not be analyzed for data gaps. The default is 10.
Max String Complexity	Define a threshold percentage of distinct string values. Object Class Properties of type String with percentage of distinct values less than or equal to this value will be included in the clustering analysis along with their values.
Background Color	Define the background color for the cells of the cluster analysis views that are identified as missing information for the combination of a feature and object.
Foreground Color	Define the text color for the cells of the cluster analysis views that are identified as missing information for the combination of a feature and object.

The **Options** tab of the **Report Assistant** offers configuration capabilities for fields in the multi-editor of the standard cluster view that take over values from a list of available results, for example, object class properties based on an enumeration or storing a reference to another object.

Only object class properties of the data type String or Reference are displayed. Object class properties of other data types and indicator types are not available in the explorer, because no attributes can be set for them.

In the attribute window of the root node, define the following for the processing of changes to the metamodel and for the behavior of the editing capabilities in the configured report:

- **Automatically Add New Class Properties**: If set to `True`, a new object class property created for the object class in Alfabet Expand is automatically added to the bottom of the dataset in the tab **Default Layout** of the configured report. If set to `False`, the new role type must be added explicitly to the dataset in the configured report by clicking the button **Refresh Available Properties** in the **Default Layout** tab. For both methods, the object class property added to the dataset will be configured to be both visible and editable in the configured report. The default is `False`.
- **Automatically Add New Indicator Types**: If set to `True`, and a new evaluation type is assigned to the object class, the indicator(s) of the evaluation type are automatically added to the bottom of the dataset in the tab **Default Layout** of the configured report. If set to `False`, the new indicator(s) must be added explicitly to the dataset in the configured report by clicking the button **Refresh Available Properties** in the **Default Layout** tab. For both methods, the indicator added to the dataset will be configured to be both visible and editable in the configured report. The default is `True`.

- **Automatically Add New Role Types**: If set to `True`, a role type that is newly assigned to the object class is automatically added to the bottom of the dataset in the tab **Default Layout** of the configured report. If set to `False`, the new role type must be added explicitly to the dataset in the configured report by clicking the button **Refresh Available Properties** in the **Default Layout** tab. For both methods, the role type added to the dataset will be configured to be both visible and editable in the configured report. The default is `True`.
- **Enable 'Set All' for Class Properties**: If set to `True`, the user can select multiple objects in the table of the configured report, select **Properties** from the drop-down list of the button **Set all** and open an editor to set identical values for properties of all selected objects. The user first selects the properties to be reset. Then he/she assigns one value to each selected property and these values are then set for all selected objects when the user closes the editor. If set to `False`, the option **Properties** is not visible in the dropdown list of the **Set all** button.
- **Enable 'Set All' for Indicators**: If set to `True`, the user can select multiple objects in the table of the configured report, select **Indicators** from the drop-down list of the button **Set all** and open an editor to set identical values for indicators of all selected objects. The user first selects the indicators to be reset. Then he/she assigns one value to each selected indicator and these values are then set for all selected objects when the user closes the editor. If set to `False`, the option **Indicators** is not visible in the dropdown list of the **Set all** button.
- **Enable 'Set All' for Role Types**: If set to `True`, the user can select multiple objects in the table of the configured report, select **Role types** from the drop-down list of the button **Set all** and open an editor to set identical values for roles of all selected objects. The user first selects the role types to be reset. Then he/she assigns one value to each selected role type and these values are then set for all selected objects when the user closes the editor. If set to `False`, the option **Role types** is not visible in the dropdown list of the **Set all** button.
- **Validate Edit Permissions**: If set to `True`, access permissions of the user are evaluated when the user opens the editor. If objects selected by the user are not editable for the user because of any of the implemented access permission concepts, the objects are not displayed in the editor. If none of the objects is editable, an empty editor might open. Therefore, it is recommended to define the query of the configured report to take access permission concepts of the user into account and display only editable objects in the dataset. If **Validate edit permissions** is set to `False`, the user can edit any object in the editor even if the access permission concepts implemented would not allow him/her to edit the object. The default is `True`.
- **Enforce Property Editability Rules**: If set to `True`, the editability of the properties in the editor will depend on the editability of the property in the editor or wizard for the base object class used in the class settings for the object profile:
  - If the **Read-Only** attribute of the editor interface control is set to `True`, the property will not be editable for all objects, even if explicitly defined as editable in the configured report.
  - If a condition defined in the **Read-Only Condition** attribute of the editor inhibits editing for a specific object, the property will not be editable for the specific object. The default is `False`.
  - If a validator is defined with the attribute Validator of an object class property, the validation is also performed in the editor of the configured report.
  - If the editor for the base object class checks whether start dates are set to a date prior to end dates, the validation of start and end dates is also performed in the editor of the configured report.

Please note the following:

- Evaluation of conditions requires a lot of computing capacity. Therefore, the number of objects that are simultaneously editable should be set to a small number with the attribute **Max Number of Records** if **Enforce Property Editability Rules** is set to `True`.
- The restrictions based on editability in editors and wizards are not applied to editing in the **Set All** functionality.
- Wizard Step conditions are not evaluated for the editor in the configured report.

- **Max Number of Records**: Defines the maximum number of objects that can be edited simultaneously in the editor. If the user selects more objects for editing, some of the selected objects are not displayed for editing and the user will be informed about that via a message on top of the editor.
- **Restrict Properties to Editor/Wizard Definition**: If set to `True`, properties not editable to the user via an editor or wizard will also be automatically set to view only in the editor of the configured report. This is independent of the editability setting in the **Default Layout** tab.

Information about additional configuration requirements for behavior of editing capabilities in a configured report is available in the section Defining Tabular Configured Reports for Multi-Editing of Object Class Properties in the chapter Configuring Reports.

Enabling user access for standard functionality

Define user access permissions for the configured report in the Reports Administration functionality of Alfabet. By default, access is possible for all users. For more information on how to assign user access rights, see the chapter *Defining and Managing User Access to Configured Reports* in the reference manual *User and Solution Administration*.

Configuring the Feedback Bot

A Feedback Bot capability is available in the slide-in toolbar that allows end users to provide feedback about any view, configured report, object cockpit, guide view, etc. implemented in their Alfabet solution. Two different concepts are configurable for the Feedback Bot:

- The Feedback Bot may provide a 5-star rating concept whereby the user rates the current view based on 1-5 stars and may optionally provide a comment but does not expect a response to the feedback.
- The Feedback Bot may provide for a more elaborate contact form concept. This feedback concept functions similar to first-level support, whereby the user provides feedback or asks a question and expects a response from a user responsible for the feedback.

The Feedback Bot may be configured to implement only one or both concepts. The start page of the Feedback Bot will only be displayed if both feedback concepts are implemented. The start page will provide the user with buttons to open the relevant Feedback Bot. When a user provides feedback, an email may be automatically sent to a person configured as the recipient of the feedback.

Please note the following configuration requirements:

- The XML object ***AlfaFeedbackBotConfig*** must be configured to implement the Feedback Bot capability. The solution designer must specify the XML element `SpecificFeedbackPage` to implement the 5- star rating system and/or the XML element `ContactUsPage` to implement a contact form for a first-level support concept for which a response from a responsible user is expected. Please note the following:
  - The texts to display in Feedback Bot in the Alfabet user interface can be configured.
  - The email address of the user responsible for the feedback as well as the email template to use for the notifications to the responsible user can be configured.
  - Screenshots may be made in association with a feedback entry. The screenshot capability may be implemented for the 5-star rating system and/or the first-level support concept. The screenshot will be made of the p of the screen where the user has the mouse pointer focused at the time that the feedback entry was made. No user information displayed in the masthead will be included in the screenshot.
  - Event templates may be configured for both Feedback Bot contexts. For example, this could be implemented to trigger events to create a ticket in a ticketing system or to consolidate feedback from all users across all instances in a single place.
- Email notifications are configurable via a text template and may contain information about the user sending the feedback, the comments they have entered in the Feedback Bot, and an express view link to the view for which the feedback was entered. Private email templates `FeedbackBot_ContactForm` and `FeedbackBot_SpecificFeedback` are available per default. Public text templates can be created to customize the email notifications. For more information about the variables that may be configured in custom text templates, see the section *Text Templates for the Feedback Bot* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.
- The types of feedback (such as Complaint, Suggestion, etc.) listed in the contact type field in the Feedback Bot may be modified in the protected enumeration ***Feedback_ContactUs_Type***. For more information about configuring the protected

enumeration ***Feedback_ContactUs_Type***, see the section *Protected Enumeration: Feedback_ContactUs_Type* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

- The following is required to enable the Feedback Bot capability:
  - The XML attribute `IsActive` must be set to "`true`" in the XML object ***AlfaFeedbackBotConfig***.
  - The **Enable Feedback Bot** attribute must be set to `True` for the relevant user profiles to use the Feedback Bot capability. For more information, see the section *Creating a User Profile* in the reference manual *User and Solution Administration*.
  - The **Enable Feedback for View** attribute must be set to `True` for the relevant user profiles to activate the **Feedback for Current View** functionality. This is relevant for user profiles responsible for reviewing and responding to feedback provided via the Feedback Bot.
  - Each user must select the **Enable Feedback Bot** checkbox in the **User Settings** editor. For more information, see the section *Defining Your User Settings in Alfabet* in the reference manual *Getting Started with Alfabet*.
- The visualization of the Feedback Bot can be specified in the GUI scheme associated with the relevant user profile. For more information about configuring the visualization of the Feedback Bot, see the section Configuring GUI Scheme Definitions for the Alfabet Interface as well as the section *Attributes Displayed in the Grid Section: Secondary Window Skins* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.
- All feedback created will be displayed in the **Feedback Review** functionality (`ADMIN_Feedback`) that must be made available to an administrative user profile. The feedback can be systematically analyzed based on various filters such as feedback types, views type, base object, user profile, etc. in order to understand areas of improvement in the configuration of the Alfabet solution. Additionally, feedback that has been provided for a view or report via the Feedback Bot can be displayed in the Alfabet user interface in a seconday view for those users responsible for reviewing and responding to the feedback. This allows the responsible users to navigate the Alfabet user interface and see the feedback for the relevant view where they currently are. A secondary view with the caption **Feedback for Current View** will be displayed with a link if feedback has been provided for the view, configured report, object cockpit, guide view, etc. Clicking the link will open the *Feedback Review Functionality* in a new browser tab which displays all feedback in detail for the view. The following is required to implement the **Feedback for Current View** capability:
  - The **Enable Feedback for View** must be set to `True` for the user profiles responsible for reviewing and responding to feedback provided via the Feedback Bot.
  - The **Enable Check Feedback for View** checkbox must be selected in the **User Settings** editor for the relevant users responsible for reviewing and responding to feedback. For more information, see the section *Reviewing the Feedback Provided by the User Community* in the reference manual *User and Solution Administration*.

To configure the Feedback Bot capability:

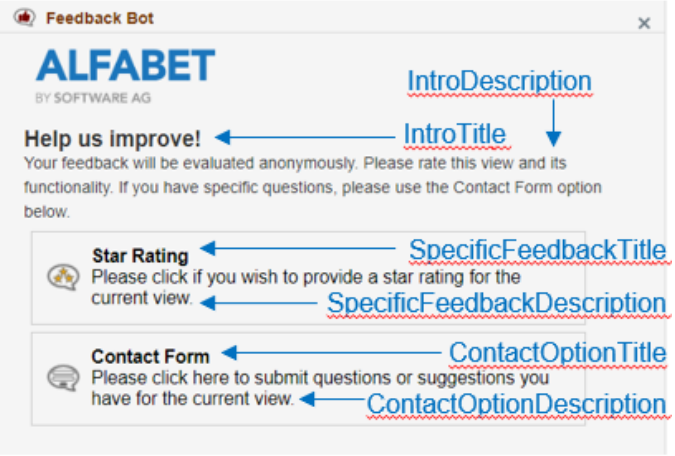
1) Go to the **Presentation** tab and expand the **XML Objects** folder. The XML object ***AlfaFeedbackBotConfig*** opens.

2) Right-click ***AlfaFeedbackBotConfig*** and select **Edit XML**. The XML editor is displayed in the center pane.

3) Specify the following XML attributes below the root XML element `FeedbackBotInfo`:

   - `IsActive`: Enter "`true`" to activate the Feedback Bot capability.
   - `UserPropertiesToSave`: Specify the properties for the object class `Person` that are stored in the `ALFA_FEEDBACK_INFO` class table in the Alfabet database.
   - `UserProfilePropertiesToSave`: Specify the properties for the object class `UserProfile` that are stored in the `ALFA_FEEDBACK_INFO` class table in the Alfabet database.
   - `Forecolor`: Specify the foreground color for the **Send** button in the Feedback Bot.
   - `Backcolor`: Specify the background color for the **Send** button in the Feedback Bot.
   - `MouseoverForecolor`: Specify the foreground color when the user mouses over the **Send** button in the Feedback Bot.
   - `MouseoverBackcolor`: Specify the background color when the user mouses over the **Send** button in the Feedback Bot.

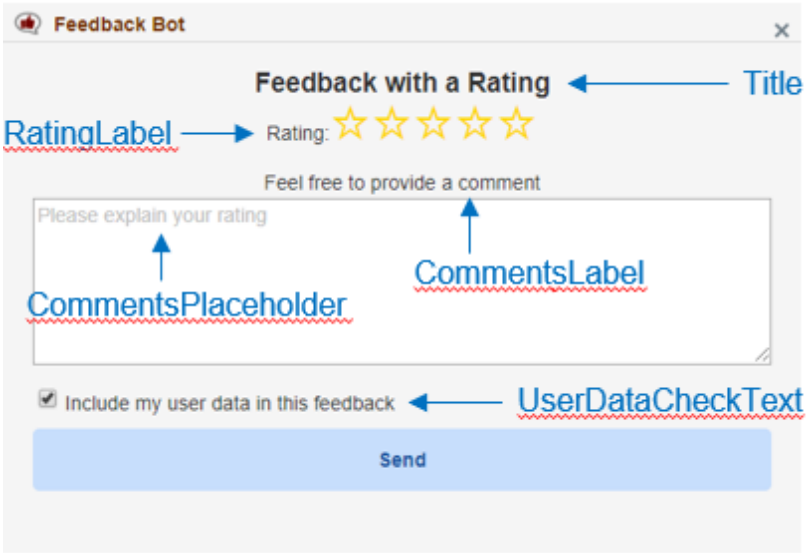
4) The XML element `FeedbackHomePage` allows you to specify the start page of the Feedback Bot if both Feedback Bot concepts are implemented (i `ShowSpecificFeedbackOption`: = "`true`" and `ShowContactOption` = "`true`" . In the start page, you can define the visibility and the text on the buttons that provide users access to the star-rating Feedback Bot and/or the contact-form Feedback Bot. If only one feedback concept is implemented, then the start page will not be displayed. Add a child XML element `FeedbackHomePage` to the root XML element `FeedbackBotInfo` and specify the following XML attributes:



   - `ShowLogo`: Specify "`true`" to display the Alfabet logo.
   - `ShowIntroText`: Specify "`true`" to display the texts defined in the XML attributes `IntroTitle` and `IntroDescription`.
   - `IntroTitle`: Specify the text to display as the title to the introduction.
   - `IntroDescription`: Specify the text to display as an introductory description to the Feedback Bot.

- `ShowSpecificFeedbackOption`: Specify "`true`" to display the button for the star-rating Feedback Bot. If set to "`false`", the values defined for the XML attributes `SpecificFeedbackTitle` and `SpecificFeedbackDescription` will not be displayed.
- `SpecificFeedbackTitle`: Specify the text to display as the title of the button for the star-rating Feedback Bot.
- `SpecificFeedbackDescription`: Specify the text to display as the explanation for the star-rating Feedback Bot.
- `ShowContactOption`: Specify "`true`" to display the button for the contact form Feedback Bot. If set to "`false`", the values defined for the XML attributes `ContactOptionTitle` and `ContactOptionDescription` will not be displayed.
- `ContactOptionTitle`: Specify the text to display as the title of the button for the contact form Feedback Bot.
- `ContactOptionDescription`: Specify the text to display as the explanation for the contact form Feedback Bot.
- `SpecificFeedbackIcon`: Specify the icon to display for the star-rating Feedback Bot.
- `ContactOptionIcon`: Specify the icon to display for the contact form Feedback Bot.

5) The XML element `SpecificFeedbackPage` allows you to implement and configure the 5- star rating system with no response is implemented. Add a child XML element `SpecificFeedbackPage` to the root XML element `FeedbackBotInfo` and specify the following XML attributes:



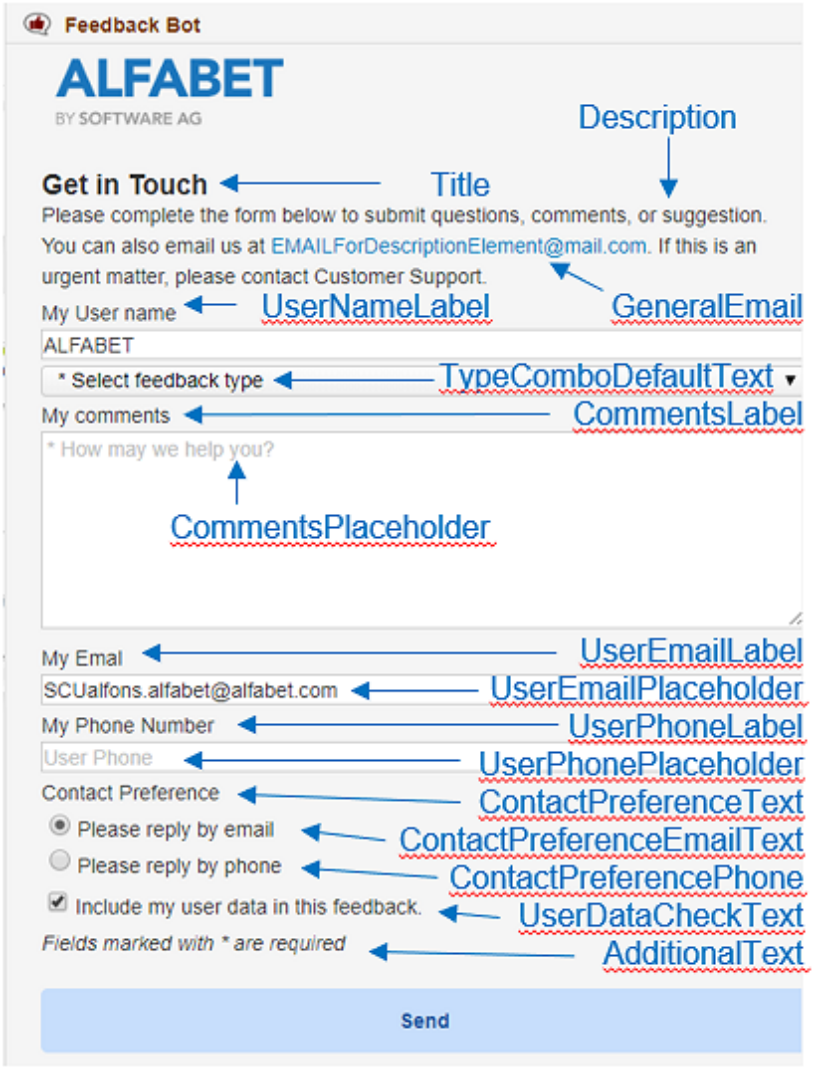
- `Title`: Specify the text to display as the title of the Feedback Bot based on the star-rating concept. The title will also be displayed in the **Feedback Type** field in the **Feedback Review** functionality.
- `ShowStars`: Enter "`true`" to display the 5 stars in the Feedback Bot. If set to "`false`", the 5 stars and the value specified in the XML attribute `RatingLabel` will not be displayed.
- `RatingLabel`: Specify the text next to the 5 stars.

- `ShowComments`: Enter "`true`" to display the comments field in the Feedback Bot. If set to "`false`", the comments field and the values defined for the XML attributes `CommentsLabel` and `CommentsPlaceHolder` will not be displayed.
- `CommentsLabel`: Specify the caption of the comments field.
- `CommentsPlaceHolder`: Specify the placeholder text displayed in the comments field.
- `ShowUserDataCheck`: Enter "`true`" to display the user data checkbox in the Feedback Bot. If set to "`false`", the checkbox and the value specified in the XML attribute `UserDataCheckText` will not be displayed.
- `UserDataCheckText`: Specify the text next to the user data checkbox.
- `SendMail`: Enter "`true`" if an email should be sent about the feedback to the email address defined in the XML attribute `ToEmail` when the **Send** button is clicked.
- `ToEmail`: Enter the email address that the email should be sent to.
- `EmailTemplate`: Specify the email template to use for the email. The private email template `FeedbackBot_SpecificFeedback` is available per default and will be used if no email template is defined. Public text templates can be created to customize the email notifications. For more information about the variables that may be configured in custom text templates, see the section *Text Templates for the Feedback Bot* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.
- `EventTemplate`: Specify an event template if needed. For example, this could be implemented to trigger events to create a ticket for the feedback in a ticketing system or to consolidate feedback from all users across all instances in a single place. For more information about implementing events, see the section Configuring Events.
- `ShowFeedbackThanks`: Enter "`true`" if the thanks page shall be displayed after the **Send** button has been clicked. The text for the thanks page can be configured in the XML element `ThanksPage` .
- `AttachmentsFolder`: Screenshots may be made in association with a feedback entry. The screenshot will be made of the aspect of the screen where the user has the mouse pointer focused at the time that the feedback entry was made. No user information displayed in the masthead will be included in the screenshot. Specify the folder in the **Internal Document Selector** to save images captured by the user of their current screen when creating the feedback entry. (For example: `AttachmentsFolder="SYSTEM\Screenshots_SpecificFeedbackPage"`). If no value is defined for the XML attribute `AttachmentsFolder`, then the screen capture capability will not be available.

  Please note that the screenshots will be saved as JPEG files. Therefore, you must ensure that the file extension.jpeg is not listed in the blacklist defined in the XML object ***FileExtensionLists***. If a whitelist is specified the XML object ***FileExtensionLists***, then the file extension.jpeg must be included as a permissible file extension. For more information, see the section Configuring the Permissibility of Files and Web Links in Alfabet .

6) The XML element `ContactUsPage` allows you to implement and configure a first-level support concept whereby a response is expected by the creator of the feedback. Add a child XML element `ContactUsPage` to the root XML element `FeedbackBotInfo` and specify the following XML attributes:



- `ShowLogo`: Specify "`true`" to display the Alfabet logo.
- `GeneralEmail`: Enter an email address that shall be displayed in the {0} variable that can be used in the XML attribute `Description`.
- `SendMail`: Enter "`true`" if an email should be sent about the feedback to the email address defined in the XML attribute `ToEmail` when the **Send** button is clicked.
- `ToEmail`: Enter the email address that the email should be sent to.
- `EmailTemplate`: Specify the email template to use for the email. The private email template `FeedbackBot_ContactForm` is available per default and will be used if no email template is defined. Public text templates can be created to customize the email notifications. For more information about the variables that may be configured in custom text templates, see the section *Text Templates for the Feedback Bot* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.
- `Title`: Specify the text to display as the title of the Feedback Bot based on the contact form concept. The title will also be displayed in the **Feedback Type** field in the **Feedback Review** functionality.
- `Description`: Specify the text to display as an introductory description to the contact form.

- `ShowUserName`: Enter "`true`" to display the current user name field in the Feedback Bot. If set to "`false`", the checkbox and the value specified in the XML attribute `UserNameLabel` will not be displayed.
- `UserNameLabel`: Specify the text to display as a caption for the current use name field.
- `ShowTypeCombo`: Enter "`true`" to display the feedback type field in the Feedback Bot. If set to "`false`", the feedback type field and the values defined for the XML attributes `TypeLabel` and `TypeComboDefaultText` will not be displayed.
- `TypeLabel`: Specify the caption of the feedback type field.
- `TypeComboDefaultText`: Enter the default placeholder text to display in the feedback type field.
- `TypeMandatory`: Enter "`true`" if the definition of the feedback type is required for the feedback. The types of feedback (such as Complaint, Suggestion, etc.) listed in the contact type field in the Feedback Bot may be modified in the protected enumeration `Feedback_ContactUs_Type`.
- `ShowComments`: Enter "`true`" to display the comments field in the Feedback Bot. If set to "`false`", the comments field and the values defined for the XML attributes `CommentsLabel` and `CommentsPlaceHolder` will not be displayed.
- `CommentsLabel`: Specify the caption of the comments field.
- `CommentsPlaceHolder`: Specify the placeholder text displayed in the comments field.
- `CommentsMandatory`: Enter "`true`" if the definition of the comments field is required for the feedback.
- `ShowUserEmail`: Enter "`true`" to display the user email field in the Feedback Bot. If set to "`false`", the user email field and the values defined for the XML attributes `UserEmailLabel` and `UserEmailPlaceHolder` will not be displayed.
- `UserEmailMandatory`: Enter "`true`" if the definition of the user email is required for the feedback.
- `UserEmailLabel`: Specify the caption of the user email field.
- `UserEmailPlaceHolder`: Specify the placeholder text displayed in the user email field.
- `ShowUserPhone`: Enter "`true`" to display the user phone number field in the Feedback Bot. If set to "`false`", the user email field and the values defined for the XML attributes `UserPhoneLabel` and `UserPhonePlaceHolder` will not be displayed.
- `UserPhoneMandatory`: Enter "`true`" if the definition of the user phone number is required for the feedback.
- `UserPhoneLabel`: Specify the caption of the user phone number field.
- `UserPhonePlaceHolder`: Specify the placeholder text displayed in the user phone number field.
- `ShowContactPreference`: Enter "`true`" to display the contact preference information in the Feedback Bot The contact preference allows the user to specify whether he/she prefers to be contacted per email or phone. If set to "`false`", the user email field and the values defined for the XML attributes `UserPhoneLabel` and `UserPhonePlaceHolder` will not be displayed.

- `ContactPreferenceText`: Specify the caption of the contact preference section of the Feedback Bot.
- `ContactPreferenceEmailText`: Specify the text for the button available for the option to be contacted per email.
- `ContactPreferencePhoneText`: Specify the text for the button available for the option to be contacted per phone.
- `ShowUserDataCheck`: Enter "`true`" to display the current user name field in the Feedback Bot. If set to "`false`", the checkbox and the value specified in the XML attribute `UserDataCheckText` will not be displayed.
- `UserDataCheckText`: Enter the text to display for the checkbox that allows the user to provide permission that his/her first and last name is included in the email sent to the user responsible for the user feedback information. (For example: Include my user data with this feedback.)
- `AdditionalText`: Enter any additional text that shall be displayed at the bottom of the Feedback Bot above the **Send** button.
- `ShowFeedbackThanks`: Enter "`true`" if the thanks page shall be displayed after the **Send** button has been clicked. The text for the thanks page can be configured in the XML element `ThanksPage` .
- `EventTemplate`: Specify an event template if needed. For example, this could be implemented to trigger events to create a ticket for the feedback in a ticketing system or to consolidate feedback from all users across all instances in a single place. For more information about implementing events, see the section Configuring Events.
- `AttachmentsFolder`: Screenshots may be made in association with a feedback entry. The screenshot will be made of the aspect of the screen where the user has the mouse pointer focused at the time that the feedback entry was made. No user information displayed in the masthead will be included in the screenshot. Specify the folder in the **Internal Document Selector** to save images captured by the user of their current screen when creating the feedback entry. (For example: `AttachmentsFolder="SYSTEM\Screenshots_ContactUsPage"`). If no value is defined for the XML attribute `AttachmentsFolder`, then the screen capture capability will not be available.

  Please note that the screenshots will be saved as JPEG files. Therefore, you must ensure that the file extension.jpeg is not listed in the blacklist defined in the XML object ***FileExtensionLists***. If a whitelist is specified the XML object ***FileExtensionLists***, then the file extension.jpeg must be included as a permissible file extension. For more information, see the section Configuring the Permissibility of Files and Web Links in Alfabet .

7) The XML element `ThanksPage` allows you to implement a thank you page after the user has hit the **Send** button.. Add a child XML element `ThanksPage` to the root XML element `FeedbackBotInfo` and specify the following XML attributes:

- `ShowLogo`: Specify "`true`" to display the Alfabet logo.
- `Title`: Enter the text that shall be displayed as the title of the thank you page.
- `Text`: Enter the text that shall be displayed for the thank you page.

8) In the toolbar, click the **Save**  button to save the XML definition.

Configuring Onboarding Emails for New Users

Alfabet can be configured to automatically send onboarding emails with an automatically generated login password and login link to new users created by a user administrator in the **Users Administration** `(ADMIN_UsersOverview)` functionality. To enable this functionality, the XML attribute `SendUserOnboardingMail` in the XML object `SolutionOptions` must be set to `"true"`. The specification of a valid email address for new users will be mandatory if the XML attribute `SendUserOnboardingMail`is set to `"true"`

For more information about creating a new user, see the chapter *Defining and Managing Users* in the reference manual *User and Solution Administration*.

To edit the XML object ***SolutionOptions***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder.
2) Right-click the XML object ***SolutionOptions*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) In the XML attribute `SendUserOnboardingMail`, enter `"true"` to automatically send onboarding emails with an automatically generated login password and login link to new users. Set to `"false"` if .
4) In the toolbar, click the **Save**  button to save the XML definition.

Enabling the User Assistance Functionality

Users can request assistance about a standard or configured view in Alfabet via a **Request User Assistance** option in the Help menu available in the main toolbar in Alfabet. If triggered, an email with a link to the view will be sent to the user in the enterprise who has been specified as responsible for user assistance.

To implement this capability, the XML attribute `EnableRequestSuperUserAssistance` in the XML object ***SolutionOptions*** must be set to `True`.

Additionally, the **Is User Assistant** checkbox in the **User** editor must be selected for the user who is responsible for answering user questions about the Alfabet solution. For more information about specifying a user as responsible for user assistance, see the section *Defining and Managing Users* in the reference manual *User and Solution Administration*.

To edit the XML object ***SolutionOptions***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder.
2) Right-click the XML object ***SolutionOptions*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.

3) In the XML attribute `EnableRequestSuperUserAssistance`, enter "`true`" if the **Request User Assistance** option in the Help menu shall be available in the main toolbar. Set to "`false`" if it should not be available.

4) In the toolbar, click the **Save**  button to save the XML definition.

Configuring Translation of Secondary and Statutory Languages to the Enterprise's Primary Language

The XML object ***SolutionOptions*** allows you to define whether data captured in a secondary language or in a statutory language shall be translated to the primary language via the automatic translation capability. The XML attribute `EnableTranslationToPrimaryLanguage` must be set to "`true`" for any of the following situations:

- If object data shall be captured in a non-primary language, the XML attribute `EnableTranslationToPrimaryLanguage` must be set to "`true`" to ensure that the object data may be automatically translated to the primary language. For more information about the configuration requirements to capture data in a non-primary language, see the section Allowing Data To Be Captured in a Non-Primary Language.
- If object data shall be captured in a statutory language, the XML attribute `EnableTranslationToPrimaryLanguage` must be set to "`true`" to ensure that the object data may be automatically translated to the primary language. For more information about the configuration requirements to capture data in a non-primary language, see the section Specifying a Statutory Language for the Enterprise.

Implementing Proposed Local Components and Proposed Information Flows

The concept of proposed local components can be implemented to perform simple technology changes that don't require extensive planning and investment. Local components discovered from other systems (such as Dynatrace© or AppDynamics©) and imported to Alfabet may be created as proposed local components and only added as real local components to the organization's IT inventory after they have been explicitly included in an approval process. Proposed local components can also be manually created in the context of the application or component they are relevant for. Application/component owners can then assess whether or not a proposed local component shall be included in the platform architecture of the application or component. Proposed local components can be approved and added as "real" local components to the inventory or dismissed and deleted from the Alfabet database. The local components will inherit the values specified for the proposed local components and can be further defined, as needed.

For an overview of the methodology of working with proposed information flows, see the section `ATO: Implementing Proposed Local Components to Manage Technolog..` and X in the reference manual *Portfolio Management Basic*.

- The class `ProposedLocalComponent` inherits the standard and custom properties of the class `LocalComponent` and the class `ProposedInformationFlow` inherits the standard and custom properties of the class `InformationFlow`. Please note that values captured

for any additional custom properties created for either `ProposedLocalComponent` or `ProposedInformationFlow` will be lost when the proposed objects are approved as "real" local components or information flows. Therefore, it is recommended that additional custom properties are not configured for the classes `ProposedLocalComponent` or `ProposedInformationFlow`.

- Components and information flows discovered in external systems (such as Dynatrace© or AppDynamics©) can be imported as objects of the classes `ProposedLocalComponent` or `ProposedInformationFlow` and mapped to the applications or components for which they are proposed. For more information about configuring an ADIF import scheme to import discovered local components and information flows from another system, see the reference manual *Alfabet Data Integration Framework*.
- Class settings (`ProposedLocalComponent` or `ProposedInformationFlow`), custom object views and object cockpits (`COMLP_ObjectView` and `IFP_ObjectView`), and custom editors (`COMLP_Editor` and `IFP_Editor`), may be configured for proposed local components and proposed information flows. Furthermore, wizards and workflows can be implemented to ease data capture.
- Imported proposed local components will be visualized in the *Platform Architecture Page View* (`PLATCOM_Matrix`) and *Components Page View* (`COM_LocalComponents`) views available for an application or component. Proposed local components can also be manually created in these views.
- Imported proposed information flows will be visualized in the *Information Flows Page View* (`APP_InformationFlows`) and (`COM_InformationFlows`) views. Proposed information flows can also be manually created in these views.
- For more information about the methodology of proposed local components, see the section `ATO: Implementing Proposed Local Components to Manage the Techn..` in the reference manual *Portfolio Management Basic*.

In order to implement the concept of proposed local components or proposed information flows in your solution configuration, the XML attribute `EnableProposedObjectHandling` must be set to "`true`" in the XML object ***SolutionOptions***. This setting ensures that imported and manually created proposed local components and proposed information flows will be visualized in the relevant views and the menu options to create, include, and edit them will be available in the Alfabet user interface.

To enable the proposed local components and proposed information flows concept in the Alfabet user interface:

1) Go to the **Presentation** tab and expand the **XML Objects** folder.
2) Right-click the XML object ***SolutionOptions*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) In the XML attribute `EnableProposedObjectHandling`, enter "`true`" if proposed local components or proposed information flows may be created and visualized in the relevant Alfabet views. In this case, existing proposed local components/proposed information flows will be displayed and relevant menu options to create and edit will be available in the Alfabet user interface. Set to "`false`" if proposed local components and proposed information flows shall not be displayed and the options to create and edit them shall not be available.
4) In the toolbar, click the **Save** button to save the XML definition.

Configuring the Creation of Business Services for Business Functions

The XML object ***SolutionOptions*** allows you to define whether each business function may have only one unique business service defined for it or whether a business function may have multiple business services defined for it.

1) Go to the **Presentation** tab and expand the **XML Objects** folder.
2) Right-click the XML object ***SolutionOptions*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) In the XML attribute `BusinessServiceUniqueness`, enter "`true`" if applications/components may provide only one business service for a specific business function. Set to "`false`" if applications/components may provide multiple business services for the same business function.

   If multiple business services may be defined, users will be able to define a unique name for each business service in the **Caption** field in the **Business Service** editor.

4) In the toolbar, click the **Save** button to save the XML definition.

Configuring the Context for the Creation of Business Data

The XML object ***SolutionOptions*** allows you to define whether business data may be created in the context of business objects and the objects that are using the business data (applications, components, etc.) or whether business data may only be created in the context of business objects.

To edit the XML object ***SolutionOptions***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder.
2) Right-click the XML object ***SolutionOptions*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) In the XML attribute `CreateBusinessDataOutsideBO`, enter "`true`" if business data may be created in the context of business objects and the objects that are using the business data (applications, components, etc.). Enter "`false`" if business data may only be created in the context of business objects.
4) In the toolbar, click the **Save** button to save the XML definition.

Making the Bookmarks Menu Available to the User Community

Alfabet provides a bookmark capability that allows you to save links that allow you to quickly navigate to the relevant object or view that you want to work with. The bookmark can point to a page view, object profile (or object cockpit), report, or object selected in an explorer. Users may access their bookmarks in the **Bookmark** > **Bookmarks** menu as well as the **Bookmark Desktop** functionality. In order to make the **Bookmarks** menu accessible to the user community, you must set the GUI scheme attribute **Show Bookmark Tree in Menu** to `True`. For more information about specifying the GUI scheme, see the section Configuring GUI Scheme Definitions for the Alfabet Interface.

Please note that the solution designer may set the **Can Create Bookmark** attribute to `False` for specific custom object views, page views, and configured reports. If this is the case, these views may not be bookmarked. For more information, see the chapter Configuring Object Views.

Implementing the AlfaBot for Navigation via a Full-Text Command

Alfabet provides an AlfaBot window that the user can access via the slide-in toolbar on the right of the user interface. The user can enter text in the AlfaBot window to do any of the following:

- Create a new object of an object class or object class stereotype.
- Edit an existing object of an object class or object class stereotype.
- Navigate to the object view or a page view of an existing object of an object class or object class stereotype.
- Navigate to a diagram.
- Navigate to a configured report.
- Start a Workflow.
- Search for information in configured reports via a full text question about object data.

User interaction in the AlfaBot window is based on natural language processing. The user enters a simple sentence and the AlfaBot searches for given structures and placeholder values based on training phrases. The training phrases are analyzed primarily by the third-party NLP processing tool Dialogflow™. Dialogflow is a third-party component but is not embedded in Alfabet. A license must be purchased by the customer directly from Google LLC to be able to use the AlfaBot.

The `Analyze` intent which enables search within the content of configured reports does not require analysis of strings via an NLP provider. At the same time, all other intents can be deactivated. The `Analyze` intent can be implemented as an additional search capability without purchasing Dialogflow.

The sentences the users type in the AlfaBot window as well as the interpretations returned by Dialogflow are stored in the Alfabet database and Alfabet will try to interpret user sentences based on the already stored and analyzed user sentences if a user uses the same wording for a similar action. If the meaning of the request can be evaluated based on stored prior requests, the sentence will not be sent to Dialogflow again. The costs for language analysis by Dialogflow can be reduced over time via this mechanism.

The AlfaBot will only be displayed in the user interface if the feature is activated and configured by a solution designer and the user interface is rendered in English language.

For end user information about accessing the AlfaBot capability in the user interface, see the section *Using the AlfaBot User Assistance* in the reference manual *Getting Started with Alfabet*.

The following configurations are required to enable navigation via the AlfaBot:

- Creating a Dialogflow Account and Configuring the Connection to the Account in Alfabet
- Configure the Alfabet Class Model to Allow Access via the AlfaBot
- Define AlfaBot Search Capabilities for Configured Reports
  - Configuring the Reports
  - Enabling Semantic Search in the Attributes of Configured Reports
    - Defining a Search Index Directory
    - Defining a Search Group for the Object Class Report
    - Updating the Search Index in Regular Intervals
    - Configuring the AlfaBot to Use the Search Group
  - Activating Search in Reports via the Analyze Intent
    - Defining the Configured Reports Included in the Search
    - Creating a Search Index for the Intent
    - Creating a Popularity Score for Reports
    - Defining a Search Result Threshold for the Semantic Search
    - Managing Automatically Generated Reports
  - Excluding Configured Reports from Results for Defined Users or User Profiles
- Activating Workflow Start via the AlfaBot for a Workflow Template
- Activating the AlfaBot in the AlfaBot Configuration Functionality
- Configuring Training Phrases for the AlfaBot
- Deactivating Intents
- Updating Dialogflow Entities With Information About Customization of the Meta-Model
- Running the AlfaBot in Offline Mode

Creating a Dialogflow Account and Configuring the Connection to the Account in Alfabet

The third-party component Dialogflow™ is not embedded in Alfabet. A license must be purchased by the customer directly from Google LLC.

Prior to setting up the connection, make sure that the web server hosting the Alfabet Web Application can access the following servers via HTTPS:

- www.googleapis.com, port 443
- dialogflow.googleapis.com, port 443

If the connection to Dialogflow fails, the error message "Name Resolution Failure" will be displayed to users trying to use the AlfaBot. The cause for this error could be connection problems with the above mentioned servers, name resolution failure, or proxy name resolution failure for the connection.

The connection to the customer's agent in the Dialogflow API must then be defined in Alfabet Expand in the XML object ***AlfaChatBotConfig***.

The description below includes information about general configuration requirements in Dialogflow to ease understanding of the required Alfabet configuration. Dialogflow handling is not described in detail. For information about how to perform the configuration steps mentioned below, consult the Dialogflow documentation.

1) Log in to Dialogflow and select **Create new agent** in the drop-down menu on the upper left of the interface. Define a name and the basic settings of the new agent and click **Create**.

   Note the following about the configuration of the new agent:

   - The default language of the agent must be English. The AlfaBot currently does not accept any other language.
   - Do not edit the intents. Intents are managed via the Alfabet user interface and will be updated in Dialogflow via the connection defined in this configuration step.
   - Do not change any settings not mentioned in the following procedure description.
   - Do not import or restore the agent from a ZIP file.
   - Do not export the agent.

2) Click on the **gear** icon next to the agent.
3) You will see the information about the **Google Project** for the account. The **Project ID** and **Service Account** properties are required as information for the configuration of access to the agent when configuring the AlfaBot in Alfabet Expand.
4) Go to the **ML Settings** tab and check the following settings:

   - Make sure that the **Match Mode** is Hybrid (**Rule-based and ML**).
   - Make sure that the **ML Classification Threshold** is 0.6.
   - It is recommended to turn the **Automatic Spell Correction** on.

5) In Dialogflow, Click on the **Service Account** . This will look like `dialogflow-xxx@<project id>.iam.gserviceaccount.com`.
6) Click **IAM** in the menu and then click the pencil shaped edit button on the right of the service account information. An editor opens on the right.
7) Add the following roles to the service account:
   - Editor
   - Dialog Flow API Admin
   - Dialog Flow API Client
8) Click **Service Accounts** in the menu to create a P12-key for your service account.
9) Click on the **Edit** icon on the top and select **Create Key**.
10) Select the **P12** checkbox and click **Create**.
11) A window opens providing a password and a file for download.
12) The password is required as information for the configuration of access to the agent when configuring the AlfaBot in Alfabet Expand.
13) Store the downloaded file in a folder that the Alfabet Web Application has access permission to.
14) Log in to Alfabet Expand.
15) and expand the nodes **XML Objects** > ***IntegrationSolutions***.
16) Double-click ***AlfaChatBotConfig***. The XML object opens in the middle pane.
17) The XML object has the following structure:

```
<AlfaChatBotConfig>
   <Proxy url="" user="" psw="" domain=""></Proxy>
   <ChatBotInfo
      Type="Dialogflow"
      IsActive="true"
      ProjectId="project id"
      AccountId="account id"
      CertificateFile="path to P12 certificate file"
      CertificateSecret="password for P12 certificate">
      <Settings MaxSelectionOptions="5" />
   </ChatBotInfo>
</AlfaChatBotConfig>
```

18) Set the XML attributes of the XML element ***ChatBotInfo*** to the following values:
   - `Type`: Enter `Dialogflow`.
   - `IsActive`: Enter `true` to activate the connection. If the XML attribute is set to `false`, the Alfabet Web Application will not connect to Dialogflow and the AlfaBot is not displayed to the users on the Alfabet user interface.
   - `ProjectId`: Enter the value of the **Project ID** property of your Dialogflow agent.
   - `AccountId`: Enter the value of the **Service Account** property of your Dialogflow agent.

- `CertificatePath`: Enter the absolute path to the location of the downloaded Dialogflow certificate file.
- `CertificateSecret`: Enter the password provided by Dialogflow for the certificate file.

Server variables can be used instead of the current values in the XML attributes `ProjectId`, `AccountId`, `CertificatePath` and `CertificateSecret`. A server variable is defined in the XML attribute as $<server variable name>. The server variables are substituted at runtime with the current value defined for the server variable in the server alias of the Alfabet Web Application. For information about the definition and use of server variables, see *Configuring Server Variables for Integration and Interoperability Solutions* in the reference manual *API Integration with Third-Party Components*.

19) Add the XML child element ***Settings*** to the XML element ***ChatBotInfo*** and define the following XML attributes, if applicable for your AlfaBot usage:

- `MaxSelectionOptions`: If multiple results are available for a user input, the AlfaBot answers with a list of available matching results. The number of matching results displayed to the user is limited. By default, the AlfaBot will provide five options, even if more results are available. To change the maximum number of results displayed in the AlfaBot, enter the number of maximum returned results in this XML attribute.

The font color and font style of the link list can be changed via the GUI scheme configuration with the attributes in the new attribute section **Application** > **AlfaBot Option Styles** > **Intent Style** attribute section of GUI schemes. For more information about the configuration of GUI schemes, see Configuring GUI Scheme Definitions for the Alfabet Interface.

- `AnalyzeIntentResultThreshold`: This XML attribute is only applicable if the `Analyze` intent for semantic analysis in configured reports shall be implemented. Optionally, you can change the maximum number of results per search. The default value is 100. The implementation of the semantic analysis is described in the section Enabling Semantic Search in the Attributes of Configured Reports below.
- `ReportSearchGroup`: This XML attribute is only applicable if the `Analyze` intent for semantic analysis in configured reports shall be implemented. A search group must be created for configured reports and the name of the search index must be entered in this attribute. The implementation of the semantic analysis is described in the section Enabling Semantic Search in the Attributes of Configured Reports below.
- `AnalyzeIntentReport`: This XML attribute is only applicable if the `Analyze` intent for semantic analysis in configured reports shall be implemented. The standard, private report `AlfabetDefaultAnalysisIntentReport` defines which reports are applicable to be added to the result dataset of the semantic analysis intent. This configuration can be overwritten. You can define a configured report returning objects of the object class `ALFA_REPORT` and enter the report name in this XML attribute to change the number and kind of configured reports considered in the semantic analysis. The implementation of the semantic analysis is described in the section Enabling Semantic Search in the Attributes of Configured Reports below.
- `AnalyzeIntentUserPermissionReport`: This XML attribute is only applicable if the `Analyze` intent for semantic analysis in configured reports shall be implemented. Optionally, you can define a configured report filtering the overall number of configured reports available for the semantic analysis on basis of user or user profile access permissions. The

implementation of the semantic analysis is described in the section Enabling Semantic Search in the Attributes of Configured Reports below.

20) Optionally, you can configure the Alfabet Web Application to direct requests to a proxy server. If you are using a proxy server, set the following XML attributes of the ***Proxy*** XML element:

- `url`: Enter the URL of the proxy server.
- `user`: If required, enter the user name for access to the proxy server. The domain name for authentication is defined separately with the domain XML attribute and must not be specified as part of the user name.
- `psw`: If required, enter the password for access to the proxy server.
- `domain`: If required, define the domain name that shall be used as part of the user name for authentication at the proxy server.

21) In the toolbar, click the **Save** button.

Configure the Alfabet Class Model to Allow Access via the AlfaBot

The ability to navigate to objects and/or to create and edit objects is restricted in the configuration of the object class and the class settings for the object class.

The following configuration is relevant on the level of the object class model:

For basic information about the configuration of the object class model, see Configuring the Class Model.

- Each object class in the meta-model has an **AlfaBot Support** attribute section with four attributes. For standard private and protected object classes, the attributes are visible but not editable. The attributes can only be set for public, customer-defined object classes. The **Creation Mode**, **Editing Mode**, and **Navigation Mode** attributes specify whether and how creation of object, editing of objects or navigation to objects is supported for the object class. The **Analysis Intent Mode** attribute specifies whether the object class is included into the search in configured reports that can be performed via the `Analyze` intent of the AlfaBot. The attributes can be set to one of the following:
  - `None`: The functionality is not supported for the AlfaBot.
  - `ContextDependent`: The functionality is supported by the AlfaBot and will take a required context into account. For example, information flows can only meaningfully be created in the context of a source or target object. If the user wants to create an information flow via the AlfaBot, the relevant view for creation of the information flow will open, like For example, the **Information Flows** page view of an application. This option is not relevant for the **Analysis Intent Mode** attribute.
  - `Full`: The functionality is supported by the AlfaBot context free. For the creation of objects with the **Creation Mode** attribute set to `Full`, the relevant editor or wizard will open.
- If the **Analysis Intent Mode** attribute of the object class is set to `Full`, a list of synonyms for the object class can be defined in the **Alias** attribute for both the object class and all object class properties thereof. If a user searches for one of the synonyms in the `Analyze` intent of the AlfaBot,

the AlfaBot will provide search results for the object class or object class property that the synonym is defined for.

The following configuration is relevant on the level of the class settings:

For information about the configuration of user profiles via view schemes and class settings, see Configuring User Profiles for the User Community.

- If at least one of the attributes **Creation Mode**, **Editing Mode**, and **Navigation Mode** in the **AlfaBot Support** attribute section of an object class is set to `Full` or `ContextDependent`, a section **Support in AlfaBot** will be visible in the class settings for the object class and all stereotypes thereof. The attributes **Support Creation**, **Support Editing**, and **Support Navigation** are displayed in the section only if the respective functionality is allowed on the class level. For example, if the **Creation Mode** is set to **ContextDependent** for an object class while the **Editing Mode** and **Navigation Mode** are set to `None`, only the **Support Creation** attribute will be displayed in the class settings.

The attributes **Support Creation**, **Support Editing**, and **Support Navigation** in the **Support in AlfaBot** section of the relevant class settings must be set to `True` in order to allow the user to navigate to, create, or edit objects of the relevant object class via the AlfaBot. By default, the attributes are set to `False`.

If the AlfaBot is implemented to automatically generate reports for the ***Analyze*** intent of the AlfaBot, ad-hoc reports will not be generated if the user is searching for information about an object class for which the **Navigation Mode** attribute is set to `False` in the relevant class settings.

The object classes `ALFA_MM_CLASS_INFO` and `ALFA_PM_CLASS_SETTING_INFO` provide information about the current configuration of the object class model and the current configuration of the class settings defined for object classes. This includes information about the settings for the AlfaBot support. To find out which object classes support the AlfaBot, you can create a configured report with a query targeting these object classes.

Define AlfaBot Search Capabilities for Configured Reports

Configured reports are by default accessible via the AlfaBot functionality. Optionally, configured reports can be excluded from accessibility via the AlfaBot.

There are two ways to access reports via the AlfaBot:

- Users can search for a configured report by entering the name of the report into the AlfaBot. If the report caption entered by the user in the AlfaBot for a request to open a configured report does not match the caption of one of the available configured reports, the AlfaBot will split the caption entered by the user into keywords and will perform a keyword search on the **Caption**, **Description** and the **Business Problem Statement** attributes of the configured reports that are accessible via the AlfaBot. In addition to a keyword search, a synonym search is performed on the text provided in the **Business Problem Statement** attribute.
- Users can search for content in configured reports by selecting the `Analyze` intent from the list of intents displayed on opening the AlfaBot. The `Analyze` intent opens a search view in the Alfabet user interface. The user can enter a description of the information he/she is looking for. The `Analyze` intent includes synonyms and related words into the search in addition to the search terms

entered by the user. Further, the intent offers advanced analysis capabilities with regard to the configured reports. In addition to the **Caption**, **Description**, and **Business Problem Statement** attributes of a configured report, the search will also use the **Apply To** attribute and the information from the **Semantic Analysis** sub-node of the configured report to find relevant reports. Strings in the **Alias** attribute of object classes and object class properties are taken into account to identify object classes.

The `Analyze` intent can be used without a connection to an NLP provider.

The following configuration is required to make full use of the search capabilities of the AlfaBot in the context of configured reports:

- Configuring the Reports
- Enabling Semantic Search in the Attributes of Configured Reports
  - Defining a Search Index Directory
  - Defining a Search Group for the Object Class Report
  - Updating the Search Index in Regular Intervals
  - Configuring the AlfaBot to Use the Search Group
- Activating Search in Reports via the Analyze Intent
  - Defining the Configured Reports Included in the Search
  - Creating a Search Index for the Intent
  - Creating a Popularity Score for Reports
  - Defining a Search Result Threshold for the Semantic Search
  - Managing Automatically Generated Reports
    - Activating the Automatic Generation of reports
    - Configuring the Information to be Displayed in the Automatically Generated Reports
    - Deleting Unused Automatically Generated Reports from the Database
- Excluding Configured Reports from Results for Defined Users or User Profiles

Configuring the Reports

1) In Alfabet Expand, go to the **Reports** tab.
2) In the explorer, click the configured report for which you want to change the AlfaBot configuration.
3) In the attribute window, set the following AlfaBot specific attributes according to demand:
   - **Applicable for AlfaBot**: Specify whether the configured report is accessible via the AlfaBot. By default, the attribute will be set to `True` for new reports. Set the attribute to `False` if the configured report should not be opened outside of a specific context. For example, this may be relevant for reports that are specifically designed to be embedded in an object cockpit.

An additional method to further restrict access to configured reports marked as applicable for AlfaBot per user profile or user is available and described below in the section Excluding Configured Reports from Results for Defined Users or User Profiles. The restrictions configured with this method will be applied to both the navigation to configured reports via the ***Navigate to Report*** intent and the semantic search in reports via the ***Analyze*** intent.

- **Business Problem Statement**: Optionally enter a description that will help users to search for the configured report in the AlfaBot. The text is not displayed in the user interface, but it is of special relevance for the search mechanisms implemented for the AlfaBot.

  You must create a search index to enable the semantic search including synonym search on text entered as business problem statement. This configuration step is described below in the section Enabling Semantic Search in the Attributes of Configured Reports.

4) Click the **Save**button to save your changes.

Enabling Semantic Search in the Attributes of Configured Reports

A semantic or synonym search can be performed based on the user input in the **Business Problem Statement** attribute of the configured reports that are accessible via the AlfaBot. In addition to a keyword search, a synonym search queries the text. This search mechanism requires that a search index is available for the content of the **Business Problem Statement** attribute. It should be activated to make full use of the search capabilities available for the AlfaBot.

The semantic search does not require natural language processing via an NLP provider and can be implemented without a connection to DialogFlow. Nevertheless, it requires general activation of the AlfaBot and activation of the `Analyze` intent. If you want to implement it as a search functionality while not providing a connection to DialogFlow required for the other intents of the AlfaBot, you can de-activate all other intents as described in the section.

The following configuration is required

- Defining a Search Index Directory
- Defining a Search Group for the Object Class Report
- Updating the Search Index in Regular Intervals
- Configuring the AlfaBot to Use the Search Group

Defining a Search Index Directory

A search index directory must be defined for storage of the search indexes on the local file system. This is done by a system administrator in the server alias of the Alfabet Web Application and the Alfabet Server. The absolute path to the relevant directory must be defined in the **Search Index Directory** attribute on the **Overview** tab of the server alias editor. In most cases, the search index directory will already be specified because it is also used for the full text search and for the online help search.

For more information, see the section *Creating a Server Alias for the Alfabet Web Application* in the reference manual *System Administration*.

Defining a Search Group for the Object Class Report

An object centric search group must be defined in the XML object `SearchManager`:

1) In the **Presentation** tab in Alfabet Expand, expand the **XML Objects** folder.
2) Right-click the XML object ***SearchManager*** and select **Edit XML**.

   The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects

3) Enter the following XML element `SearchGroup` as child element of the XML object ***SearchManager***:

```
<SearchGroup Name="SemanticSearchInReportAttributes">
   <Query Name="SemanticSearchInReports" ClassName="ALFA_REPORT"
   ImageProps="CAPTION" ShowProps="CAPTION"
   ExportProps="NAME,CAPTION,DESCRIPTION,BUSINESS_COMMENT" />
</SearchGroup>
```

   you can optionally change the XML attributes `Name` of the configuration displayed above. The `Name` of the search group is the name of the search group folder in the search index directory. It is also to be used in command lines or editors of the mechanism available for search index update to identify the search group.

   For an explanation of the meaning of the XML attributes in a search group, see Configuring a Global Search Group in the chapter Configuring Custom Selectors and Search Functionalities.

4) In the toolbar, click the **Save** button to save the XML definition.

Updating the Search Index in Regular Intervals

The search index must be updated in regular intervals. This can be done using one of the following options:

- An administrative user can create and update the full-text search indexes for globally-defined search groups in the administrative **Full-Text Search** functionality on the Alfabet user interface. The functionality allows creation and update of indexes for all supported languages to be performed.

The view `ADMIN_FullTextSearch` must be added to the user profile of the administrative user to enable manual creation and update of the globally-defined full-text search. For information about how to manually create and update search indexes, see the section *Creating an Index for the Full-Text Search* in the reference manual *User and Solution Administration*.

- Via the **Job Schedule** functionality on the Alfabet user interface, administrative users can create job schedules for the automatic creation and update of search indexes for globally-defined full-text search groups in defined time intervals. A job schedule triggers the creation and update of the

search index of a single full-text search group. Please note that the job schedule functionality currently supports only definition of full-text search indexes for the language English (United States).

For information about how to implement and configure the **Job Schedule** functionality, see the section Activating the Job Schedule Functionality. For information about how to schedule full-text search index update via a job schedule, see the section *Creating a Job Schedule for the Generation of a Full-Text Search Index* in the reference manual *User and Solution Administration*.

- Via the executable `FullTextSearchUtil.exe`, system administrators can configure a batch process to regularly create and update search indexes for globally-defined full-text search groups. The command line tool can create and update indexes for all supported languages. It can either update an index for a defined search group or for all globally-defined full-text search groups.

For more information about the configuration of the batch process, see the section *Updating Indexes with the FullTextSearchUtil.exe* in the reference manual *System Administration*.

Configuring the AlfaBot to Use the Search Group

The AlfaBot must be configured to use the search group:

1) Go to the **Presentation** tab in Alfabet Expand.
2) Navigate to **XML Objects** > `IntegrationSolutions` > ***AlfaChatBotConfig***.
3) Right-click the XML object ***AlfaChatBotConfig*** and select **Edit XML**.
4) In the editor in the middle pane, enter the name of the new report in the XML attribute `AnalyzeIntentReport` of the child XML element `Settings` of the XML element `ChatBotInfo`:

```
<AlfaChatBotConfig>
    <Proxy url="" user="" psw="" domain=""></Proxy>
    <ChatBotInfo
        Type="Dialogflow"
        IsActive="true"
        ProjectId="project id"
        AccountId="account id"
        CertificateFile="path to P12 certificate file"
        CertificateSecret="password for P12 certificate">
        <Settings MaxSelectionOptions="5"
        AnalyzeIntentReport="CustomizedReportName"ReportSearchGroup="Seman
        ticSearchInReportAttributes" />
    </ChatBotInfo>
</AlfaChatBotConfig>
```

5) In the toolbar, click the **Save** button to save the XML definition.

Activating Search in Reports via the Analyze Intent

Users can search for content in configured reports by selecting the `Analyze` intent from the list of intents displayed on opening the AlfaBot. The `Analyze` intent opens a search view in the Alfabet user interface. The user can enter a description of the information he/she is looking for. The `Analyze` intent includes synonyms and related words into the search in addition to the search terms entered by the user. Further, the intent offers advanced analysis capabilities with regard to the configured reports. In addition to the **Caption**, **Description**, and **Business Problem Statement** attributes of a configured report, the search will also use the **Apply To** attribute and the information from the **Semantic Analysis** sub-node of the configured report to find relevant reports. Strings in the **Alias** attribute of object classes and object class properties are taken into account to identify object classes.

The following configurations and regular maintenance steps are required for implementation of the `Analyze` intent of the AlfaBot in addition to the general configuration of the AlfaBot functionality:

- Defining the Configured Reports Included in the Search
- Creating a Search Index for the Intent
- Creating a Popularity Score for Reports
- Defining a Search Result Threshold for the Semantic Search
- Managing Automatically Generated Reports
  - Activating the Automatic Generation of reports
  - Configuring the Information to be Displayed in the Automatically Generated Reports
  - Deleting Unused Automatically Generated Reports from the Database

Defining the Configured Reports Included in the Search

In the **Reports** explorer in Alfabet Expand, the private `AlfabetDefaultAnalysisIntentReport` report is available in a private `AlfaBotReports` report folder. The reports find configured reports relevant for the new ***Analyze*** intent via a query targeting the object class `ALFA_REPORT`. Only configured reports that are returned by this report are included in the search for the intent.

The configured report limits the search to configured reports that have the **Applicable for AlfaBot** attribute set to `True`. Furthermore, the **Apply to Class** attribute must either be not set, set to an artifact class in the IT landscape, or set to an object class involved in risk management, project management, or contract management. Configured reports are excluded if they provide editing capabilities for object classes or are configured in the scope of Alfabet functionalities (such as in the questionnaire or data capture template functionalities).

The standard ***AlfabetDefaultAnalysisIntentReport*** report can optionally be substituted by a configured report with a customized set of reports to be included. The following description of the configuration uses the existing private report as a template for creating an own report. The query finding the reports can also be defined from scratch, but working with the predefined query enhances configuration speed and ensures that relevant properties of the reports are taken into account for selection.

1) In the **Reports** explorer in Alfabet Expand, right-click the report folder that the report shall be created in and select **Create New Report**.

For information about structuring configured reports in report folders, see Managing and Structuring Your Configured Reports in Folders.

2) Click the new report and change the **Name** attribute of the configured report in the attribute window to a meaningful, unique name. the name should not contain special characters or whitespaces.
3) In the explorer, navigate to ***AlfaBotReports*** > ***AlfabetDefaultAnalysisIntentReport***.
4) Right-click the ***AlfabetDefaultAnalysisIntentReport*** report and select **Copy**.
5) Navigate back to your new report.
6) Right-click the new report and select **Paste**.
7) In the menu of Alfabet Expand click **Save** to save the new report definition.
8) Click the **Browse** button in the **Native SQL Query** attribute of the configured report.
9) In the editor, change the native SQL query according to demand and click **OK** to save your changes.
10) In the menu of Alfabet Expand click **Save** to save the new report definition.
11) Go to the **Presentation** tab in Alfabet Expand.
12) Navigate to **XML Objects** > ***IntegrationSolutions*** > ***AlfaChatBotConfig***.
13) Right-click the XML object ***AlfaChatBotConfig*** and select **Edit XML**.
14) In the editor in the middle pane, enter the name of the new report in the XML attribute `AnalyzeIntentReport`of the child XML element `Settings` of the XML element `ChatBotInfo`:

```
<AlfaChatBotConfig>
    <Proxy url="" user="" psw="" domain=""></Proxy>
    <ChatBotInfo
        Type="Dialogflow"
        IsActive="true"
        ProjectId="project id"
        AccountId="account id"
        CertificateFile="path to P12 certificate file"
        CertificateSecret="password for P12 certificate">
        <Settings MaxSelectionOptions="5"
        AnalyzeIntentReport="CustomizedReportName" />
    </ChatBotInfo>
</AlfaChatBotConfig>
```

15) In the toolbar, click the **Save** button to save the XML definition.

An additional method to further restrict access to configured reports marked as applicable for AlfaBot per user profile or user is available and described below in the section Excluding Configured Reports from Results for Defined Users or User Profiles. The restrictions configured with this method will be applied to both the navigation to configured reports via the ***Navigate to Report*** intent and the semantic search in reports via the ***Analyze*** intent.

Creating a Search Index for the Intent

The `Analyze` intent requires regular generation of a search index. This requires the following configuration:

- A search index directory must be defined for storage of the search indexes on the local file system. This is done by a system administrator in the server alias of the Alfabet Web Application and the Alfabet Server. The absolute path to the relevant directory must be defined in the **Search Index Directory** attribute on the **Overview** tab of the server alias editor. In most cases, the search index directory will already be specified because it is also used for the full text search and for the online help search.

For more information, see the section *Creating a Server Alias for the Alfabet Web Application* in the reference manual *System Administration*.

- The `Analyze` intent requires regular generation of a search index via the standard `SemanticSearch` ADIF import scheme available in the **Alfabet Standard Jobs** folder of the ADIF explorer in Alfabet Expand. It is recommended to run this ADIF job regularly (For example, daily) to include changes to configured reports, indicator types, and evaluation types. The ADIF job can be scheduled for automatic execution via the **Job Schedule** functionality.

The ADIF job must be executed without definition of a file to export to. The ADIF job will add a directory ***AnalyzeReports*** to the search index directory defined in the server alias configuration and add a complete search index file infrastructure to this directory.

For information about how to execute ADIF jobs, see *Configuring ADIF Schemes* in the reference manual *Alfabet Data Integration Framework*.

Please note:

- Execution of the ADIF job will fail if the configured report defined with the XML attribute `AnalyzeIntentReport` of the XML object **AlfaChatBotConfig** does not exist, does not return any results, or does not return results for the object class `ALFA_REPORT` only.
- The ADIF job for update of the search index will be run automatically on update of the meta-model via AMM file and after re-store of a database from ADBZ file to adapt the search results to changes in the meta-model configuration.

Creating a Popularity Score for Reports

The popularity score for configured reports included in the search results for the `Analyze` intent provides information to the user about whether the report is often or seldom visited by users. The popularity score will only be displayed if it is calculated in the background in regular intervals. The following is required to calculate the score:

- The **Presentation Usage Tracking** functionality needs to be activated in the server alias configuration of the Alfabet Web Application. Set the **Track Presentation Usage** attribute in the **Server Settings** > **Tracking** tab to **Local Database** to activate **Presentation Usage Tracking**. For information about the complete configuration options for **Presentation Usage Tracking**, see *Activating Presentation Usage Tracking* in the reference manual *System Administration*.

- An ADIF job based on the standard `UpdateReportsPopularity` ADIF import scheme available in the **Alfabet Standard Jobs** folder of the ADIF explorer in Alfabet Expand must be executed in regular intervals to generate and update the popularity score. The ADIF job can be scheduled for automatic execution via the **Job Schedule** functionality. It will enter the number of times the `POPULARITY` object class property of the `ALFA_REPORT` object class.

  The ADIF job for update of the popularity score will be run automatically on update of the meta-model via AMM file to adapt the search results to changes in the meta-model configuration.

Defining a Search Result Threshold for the Semantic Search

User input into the search field of the faceted semantic search is preprocessed prior to sending it to the search engine. Substrings consisting of multiple words are compared to entities like For example, object class names or indicator type name. If strings are matching an entity name, they are emphasized in the search string sent to the search engine. Emphasizing of search sub-strings is fine tuned if either no search results are returned with the current syntax or if too many results are found. The maximum number of results is 100 per default and can be changed in the XML object ***AlfaChatBotConfig***: that may be returned is configurable with the new XML attribute `AnalyzeIntentResultThreshold` which has been added to the XML element `ChatBotInfo` of the XML object . The default value for the XML attribute `AnalyzeIntentResultThreshold` is 100.

1) Go to the **Presentation** tab in Alfabet Expand.
2) Navigate to **XML Objects** > `IntegrationSolutions` > ***AlfaChatBotConfig***.
3) Right-click the XML object ***AlfaChatBotConfig*** and select **Edit XML**.
4) In the editor in the middle pane, add an XML attribute Anal`AnalyzeIntentResultThreshold` to the child XML element `Settings` of the XML element `ChatBotInfo` and set it to the maximum number of search results:

```
<AlfaChatBotConfig>
    <ChatBotInfo
        ...
        >
        <Settings MaxSelectionOptions="5"
            AnalyzeIntentResultThreshold="100"
            AnalyzeIntentReport="CustomizedReportName" />
    </ChatBotInfo>
</AlfaChatBotConfig>
```

5) In the toolbar, click the **Save** button to save the XML definition.

Managing Automatically Generated Reports

Automatically generated reports are simple tabular datasets that are generated on the fly if the user looks for objects of a defined object class or object class stereotype and the information about the object that

the user wants to know is either the setting of a defined indicator, a defined role for a user, or the value setting for an object class property based on an enumeration.

The automatically generated reports are simple tabular datasets listing all objects that match the search.

Reports will not be generated automatically if the user is searching for information about an object class for which the **Navigation Mode** attribute is set to `False` in the relevant class settings. The user profile permissions to access object class property, indicator, and role information is also considered for automatic generation of reports.

Automatically generated reports are stored persistently after creation. In the **Reports** explorer in Alfabet Expand, they are located in the private folder **Analysis Intent Ad-Hoc Reports** . The name and caption of the automatically generated reports is concatenated from the name or caption of the object class or object class stereotype the information in the automatically generated report is about and the hits from the user question that the information was generated from amended with a random number. For example, a report which is automatically generated for a user query to search components with a high usability, the name would be `ComponentIndicatorNameUsabilityIndicatorSemanticValueHigh_<number>` and the report caption would be `Component - Indicator Name - Usability - Indicator Semantic Value - High`.

Configured reports are stored in the object class `ALFA_REPORT`. The object class `ALFA_REPORT` has a boolean object class property `ADHOC` that is set to `True` for automatically generated configured reports only.

Automatically generated reports are assigned to the user profile the user posting the request that led to the generation of the report was logged in with. The report will be available to other users logged in with the same user profile. If a user logged in with the same user profile uses the faceted semantic search with a request matching the results in the already existing automatically generated report, the existing report will be re-used instead of generating a new report. In addition, the automatically generated report will automatically be visible in the **Configured Reports** functionality of all users with the same user profile. At the same time, it will be excluded from the functionality for all other user profiles and the **Selector Behavior** attribute of the configured report is set to 'Not Visible', which means that the automatically generated reports cannot be selected by a user via a reports selector.

The automatically managed reports will be available to report administrators in the **Report Administration** functionality and the availability for user profiles can be changed by the administrator.

The automatic generation of reports requires activation and maintenance:

- Activating the Automatic Generation of reports
- Configuring the Information to be Displayed in the Automatically Generated Reports
- Deleting Unused Automatically Generated Reports from the Database

Activating the Automatic Generation of reports

To activate automatic generation of reports:

1) Go to the **Presentation** tab in Alfabet Expand.
2) Navigate to **XML Objects** > `IntegrationSolutions` > ***AlfaChatBotConfig***.
3) Right-click the XML object ***AlfaChatBotConfig*** and select **Edit XML**.

4) In the editor in the middle pane, add the XML attribute `AnalyzeIntentType` to the XML element `ChatBotInfo` and set it to one of the following values:

   - `Search`: Automatic generation of reports is deactivated and only existing configured reports will be searched for results. This is the default.
   - `GenerateReports`: Automatic generation of reports is activated and existing configured reports are not searched for results.
   - `Both`: Search results will both include automatically generated reports and results from existing configured reports.

5) In the toolbar, click the **Save**  button to save the XML definition.

A clean-up mechanism for removing un-used automatically generated reports is available. This mechanism deletes all ad-hoc reports that have never been opened by any user. Whether an ad-hoc report has been opened will be read from the presentation usage table. Activation of presentation usage tracking is Therefore, a precondition for clean-up of un-used ad-hoc reports. It is Therefore, highly recommended to activate presentation usage tracking in the server alias of the Alfabet Web Application at the time generation of ad-hoc reports is activated. For information about activation of presentation usage tracking, see the section *Presentation Usage Tracking* in the reference manual *System Administration*.

Configuring the Information to be Displayed in the Automatically Generated Reports

By default, automatically generated reports are simple tabular datasets that provide a list of objects of an object class or object class stereotype that has been identified to match the search condition. Without any additional configuration, the automatically generated reports display all object class properties of the data types String, Text, Date, Integer, and Real which are defined in the **Properties in Preview** and **Image Properties** attributes of the relevant class settings for the object class or object class stereotype. In addition, all object class properties used to filter the ad-hoc report results are displayed. This includes information about the role type or about the indicator type and indicator semantic value, if applicable.

You can change the output of the automatically generated reports to show additional information with the following configurations:

- You can change the **Properties in Preview** section attribute of the class settings of the object class to provide other information. Please note however that this will also change the information displayed in object preview windows.

For information about changing the class setting attributes, see Creating a Custom Class Setting for a Protected Object Class or Object Class Stereotype in the chapter Configuring User Profiles for the User Community.

- Define database views which return the required information. The automatically generated reports are generated from both database tables and database views. If a report is automatically generated from a database view, it will display the dataset of the database view instead of the preview properties from the class settings of the object class.

Only information from database views with the **Applicable for AlfaBot** attribute set to `True` are considered for automatic generation of reports. The search will use the **Base Class** attribute of the database view and the information from the **Semantic Analysis** sub-node of the database view to find relevant database views for automatic generation of reports.

For information about defining database views, see Creating Database Views To Enhance Performance And Support Search Functionalities.

- Define report collections for object classes to provide additional information about the listed objects to the user. If report collections are defined for the object class or object class stereotype the automatically generated report is about, the user will see a number of tabs to open configured reports with additional information such as gantt charts or diagrams about all objects in the tabular dataset. Report collections defined for an object class or object class stereotype are displayed on both standard configured tabular reports and automatically generated reports finding objects of that object class.

For information about the configuration of report collections, see Integration of Configured Reports as Report Collection Into Tabular Configured Reports in the chapter Configuring Reports.

Deleting Unused Automatically Generated Reports from the Database

Automatically generated reports are stored persistently and are re-used, which means that they are added to the result dataset of similar matching user requests, if applicable. Multiple automatically generated reports might be generated per user request, and users might navigate only to one of the ad-hoc reports to find an answer to the current request or they might navigate to a customer configured report instead. Therefore, many ad-hoc reports will never be opened by any user. A mechanism is available to clean the database from un-used ad-hoc reports. This mechanism will delete all automatically generated reports that have not been used by any user and that are not generated on the current date according to the **Creation Date** object class property of the report.

The following configuration is required for the clean-up procedure:

- To clean the database from un-used automatically generated reports, presentation usage tracking needs to be activated when activating ad-hoc report generation. The presentation usage tracking will be read by the clean-up job to evaluate which ad-hoc reports have been opened by users. For information about activation of presentation usage tracking, see the section *Presentation Usage Tracking* in the reference manual *System Administration*.
- An ADIF job based on the standard `AdHocReportsCleanup` ADIF import scheme available in the **Alfabet Standard Jobs** folder of the ADIF explorer in Alfabet Expand must be executed in regular intervals to clean the database from automatically generated reports that have never been opened by any user. The ADIF job can be scheduled for automatic execution via the **Job Schedule** functionality. For information about how to execute ADIF jobs, see *Configuring ADIF Schemes* in the reference manual *Alfabet Data Integration Framework*

During update of the meta-model via an AMM file replacing the configuration in the target database, automatically generated reports are not deleted from the target database.

Excluding Configured Reports from Results for Defined Users or User Profiles

The availability of configured reports via the AlfaBot is controlled by the following mechanisms:

- ***Navigate to Report*** intent: Users have access to all configured reports with the **Applicable for AlfaBot** attribute set to `True`. For more information about the required configuration, see Configuring the Reports.

- ***Analyze*** intent: Users have access to all configured reports found by the private standard ***AlfabetDefaultAnalysisIntentReport*** report or a customer configured substitution of this report defined in the XML object ***AlfaChatBotConfig***. For more information about the required configuration, see Defining the Configured Reports Included in the Search.

In addition, access to configured reports via the AlfaBot can be restrict per user profile or per user. The restrictions configured with this method will be applied to both the navigation to configured reports via the ***Navigate to Report*** and the semantic search in reports via the ***Analyze*** intent.

1) In the **Reports** explorer in Alfabet Expand, right-click the report folder that the report shall be created in and select **Create New Report**.

   For information about structuring configured reports in report folders, see Managing and Structuring Your Configured Reports in Folders.

2) Click the new report and set the following attributes: the

   - **Name**: Change the default name to a meaningful, unique name. The name should not contain special characters or whitespaces.
   - **Type**: Select either `NativeSQL` to define a native SQL query to generate the result dataset, or `Query` to define an Alfabet query.
   - **Applicable for AlfaBot**: Select `True`.
   - **Caption**: Optionally, change the default caption to a meaningful caption.
   - **Description**: Optionally enter a description explaining the purpose of the configured report.
   - **Native SQL Query** / **Alfabet Query**: Click the **Browse** button to open the query editor and define a query as follows:
     - The query must find objects of the object class `ALFA_REPORT`. For an Alfabet query, the `FIND` class must be `ALFA_REPORT`. For a native SQL query, the first argument in the `SELECT` statement must return the `REFSTR` of objects of the object class `ALFA_REPORT`. All other arguments of the `SELECT` statement or show properties defined for Alfabet queries are not relevant for the functionality to restrict access to configured report.
     - Use the Alfabet query parameters `CURRENT_USER` and `CURRENT_PROFILE` to refer to the current working environment.

     The following example is a native SQL query using the parameter `CURRENT_PROFILE` to exclude the configured reports from access via the AlfaBot if they have been excluded for the user profile the user is currently logged in with via the **Reports Administration** functionality. The object class `ALFA_REPORT_USAGE` stores the setting done in the **Reports Administration** functionality. This object class includes all configured reports that have been excluded or included explicitly for a user profile or a user.

     The example query is based on a copy of the query defined in the standard, private configured ***AlfabetDefaultAnalysisIntentReport*** report to ensure that only reports available in general for the ***Analyze*** intent are included in the list of configured reports a user might access when working with a specific user profile:

```
SELECT rep.REFSTR, rep.NAME
FROM ALFA_REPORT rep
```



```
LEFT JOIN ALFA_REPORTUSAGE ru ON ru.REPORT = rep.REFSTR AND
ru.OWNER = @CURRENT_PROFILE

WHERE (rep.TEMPLATE IS NULL

OR (rep.TEMPLATE NOT IN
('ITMAP_TableReport','Relationships_TableReport','Relationshi
ps_EditableTableReport','EvaluationReport','MatrixMapReport_E
x','CustomPivotTable','GaugeReport','RefApiObjectReport','Wid
getReport','UserManagementReport','ContactManagementReport','
EditableClassViewReport','AugmentedAIReport','DiagramListRepo
rt','PercentageDistributionReport','DataCaptureTemplateStatus
Report','QuestionnaireEvaluationReport','DataQualityAIReport'
,'JobScheduleAdministrationReport')

AND rep.TEMPLATE NOT LIKE 'Capture%'))

AND (rep.APPLYTOCLASS IS NULL OR rep.APPLYTOCLASS IN (SELECT
A_NAME FROM ALFA_MM_CLASS_INFO WHERE PARENT IN
('Artifact','ArtifactAuthorized','ContractBase','ProjectClass
es','RiskManagement')))

AND rep.APPLICABLE_ALFABOT=1

AND (ru.A_TYPE = 'Excluded' or ru.A_TYPE IS NULL)
```

3) In the menu of Alfabet Expand click **Save** to save the new report definition.
4) Go to the **Presentation** tab in Alfabet Expand.
5) Navigate to **XML Objects** > `IntegrationSolutions` > ***AlfaChatBotConfig***.
6) Right-click the XML object ***AlfaChatBotConfig*** and select **Edit XML**.
7) In the editor in the middle pane, enter the name of the new report in the XML attribute `AnalyzeIntentUserPermissionReport`of the child XML element `Settings` of the XML element `ChatBotInfo`:

```
<AlfaChatBotConfig>
    <Proxy url="" user="" psw="" domain=""></Proxy>
    <ChatBotInfo
        Type="Dialogflow"
        ...>
        <Settings MaxSelectionOptions="5"
            AnalyzeIntentUserPermissionReport="PermissionReportName" />
    </ChatBotInfo>
</AlfaChatBotConfig>
```

8) In the toolbar, click the **Save**  button to save the XML definition.
9) Restart the web server hosting the Alfabet Web Application. The permissions will only be applied after a web server restart.

A web server restart is also required after changes have been made to the query of the already existing user permission report.

Activating Workflow Start via the AlfaBot for a Workflow Template

Workflows can only be started via the AlfaBot for workflow templates that are configured to accept workflow start via the AlfaBot:

1) In the Workflow tab in Alfabet Expand select the workflow template that shall be started via the AlfaBot.
2) In the attribute window define the following attributes:
   - **Allow AlfaBot to Create New Workflows for Existing Objects**: Set to `True` to allow workflows to be started for processing of existing objects.
   - **Allow AlfaBot to Create New Workflows for New Objects**: Set to `True` to allow workflows to be started for creation of new objects.

Activating the AlfaBot in the AlfaBot Configuration Functionality

The AlfaBot must be initialized in the **AlfaBot Configuration** (`CONF_AlfaBot`) functionality.

The functionality must be made available to an administrative user via the user profile configuration. For information about how to provide access to a functionality for a user, see Making Functionalities Accessible to a User Profile.

When the functionality opens without the AlfaBot being initialized, the explorer will be empty and a **Setup AlfaBot** button will be displayed. Click the button to initialize the AlfaBot. The explorer will be filled with the available intents and the button will no longer be displayed.

Configuring Training Phrases for the AlfaBot

Training phrases are configured in the **AlfaBot Configuration** (ChatBot_Config) functionality in the Alfabet user interface.

The functionality of the AlfaBot is based on intents. An intent is a defined functionality (such as the creation of a new object or the navigation to a configured report) that triggers a predefined response of the AlfaBot. Training phrases are defined for each intent. A training phrase is a phrase that might be entered by a user to trigger the intent. It can contain placeholders for all variables that may occur in the phrase. For example, if a user wants to create an object for an object class or object class stereotype, the placeholder can be entered in the training phrase at the position where the name of the object class or object class stereotype is expected. Placeholders have a predefined name starting with @. For example, a training phrase for the creation of an object could be:

```
I want to create a @classOrStereotype
```

Please note that the AlfaBot only understands English phrases.

If a user enters a phrase in the AlfaBot, the phrase will be sent to Dialogflow. Dialogflow analyzes the result and sends information about the identified intent and the placeholder values back to the AlfaBot. The AlfaBot will execute the intent if both the intent and the placeholders are meaningfully identified in a phrase. Otherwise, it will try to execute one of the intents that request additional information from the user.

The sentences the users type in the AlfaBot window as well as the interpretations returned by Dialogflow are stored in the Alfabet database and Alfabet will try to interpret user sentences based on the already stored and analyzed user sentences if a user uses the same wording for a similar action. If the meaning of the request can be evaluated based on stored prior requests, the sentence will not be sent to Dialogflow again. The costs for language analysis by Dialogflow can be reduced over time via this mechanism.

Internal processing of user input can be deactivated via the **Turn Config Mode On** button in the workspace of the root node of the explorer of the **AlfaBot Configuration** functionality.

For both internal processing and processing via Dialogflow, user input will be processed in different consecutive steps which are all executed with a fuzzy search mechanism to cope with misspellings:

- The user input is first compared to all available training phrases to identify the intent. If the training phrase matches an intent and placeholders are included in the training phrase, the words entered in the location of the placeholders will be identified.
- The captions of relevant configuration objects in the current configuration of the meta-model are provided to Dialogflow as a list of allowed values for the respective placeholders. For example, a value for the placeholder @classOrStereotype will only be resolved successfully if it is identical to the caption of an existing object class or object class stereotype in the current configuration. If user input cannot be mapped to one of the intents via the training phrases, it will be compared with the lists of allowed values for the entity definitions. If it matches an entity definition, the AlfaBot will process it as information about the content provided by the entity. For example, if the AlfaBot asks the user for which object class he/she would like to create an object, the user can enter the object class caption without any additional text. The entry will correspond to an allowed value for object class caption and will be processed as such. Fuzzy search within entity definitions is used to cope with typos in the requests.

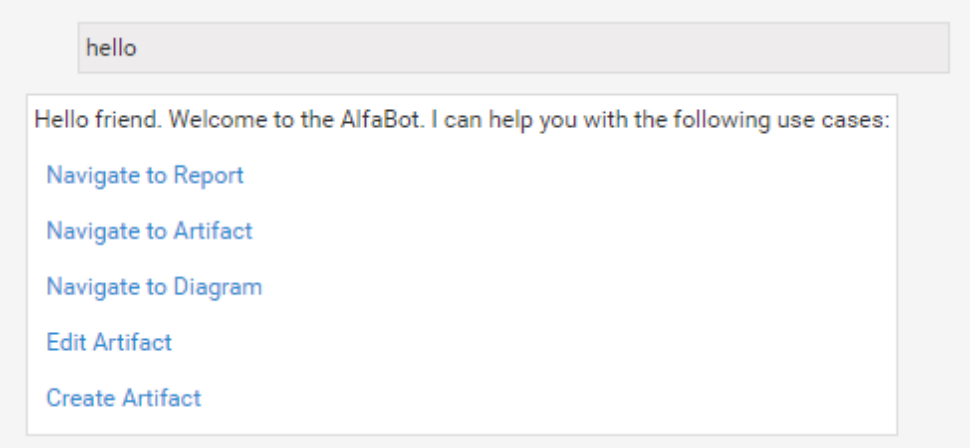
  The entities must be updated if the meta-model configuration changes. Details are provided below.

- If no matching results are found and the user input consists of more than one word, the user input will be split into separate words and a keyword search will be performed. For configured reports, the search mechanisms perform a synonym search on the text provided in the **Business Problem Statement** attribute in addition to a keyword search on caption and description of the configured report.

The font color and font style of the link lists in the AlfaBot are configurable. The styling of intents and options is carried out via the **Alfabot Options Styles** section of the GUI scheme. For more information, see the section Configuring GUI Scheme Definitions for the Alfabet Interface in the chapter Configuring the Visualization of the Alfabet Interface as well as a detailed documentation of all GUI scheme attributes in the chapter *Overview of GUI Scheme Attributes* in reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

All intents that can be processed by the AlfaBot are listed in the explorer of the **AlfaBot Configuration** functionality. The following intents are available:

- **Welcome**: Phrases that the user might enter as a welcome text with no additional intention, like "Hello" or "Good Morning". The AlfaBot will answer with any of the configured responses followed by a list of the activities that are supported by the AlfaBot. The user can click any of the activities and will then be asked immediately to provide the required information about the object, class or view names relevant for the intent of the selected use case without needing to type a phrase that identifies the intent.



The font color and font style of the link list can be changed via the GUI scheme configuration by means of the attributes in the attribute section **Application** > **AlfaBot Option Styles** > **Intent Style**. For more information about the configuration of GUI schemes, see Configuring GUI Scheme Definitions for the Alfabet Interface.

- **Create Artifact**: If the user wants to create a new object, he/she can use any of the configured training phrases. Object creation can be supported either context-free or context-dependent: The AlfaBot needs the caption of the object class or object class stereotype to process the intent. If the object class or object class stereotype is configured to allow creation via the AlfaBot, the AlfaBot will open the editor or wizard for creation of a new object. After the user has created the object, the object profile or object cockpit of the new object will open in the Alfabet user interface. If the object class or object class stereotype is configured to deny creation via the AlfaBot, the AlfaBot will inform the user.
  - Context-free object creation: Any object that can be created without knowing the defined context (such as an application) can be created context-free. The AlfaBot requires the caption of the object class or object class stereotype to process the intent. If the object class or object class stereotype is configured to allow creation via the AlfaBot, the AlfaBot will open the editor or wizard to create the new object. After the user has created the object, the object profile or object cockpit of the new object will open in the user interface.
  - Context-dependent object creation: Any object that requires a specific context (such as a business service which can only be created in the context of a providing application) is created context-dependent. The AlfaBot needs the caption of the object class or object class stereotype to process the intent. If the object class or object class stereotype is configured to allow creation via the AlfaBot, the AlfaBot will scan the configuration of the current user profile for the ability to create the object via the buttons in page views. If the object can be created in multiple views, the user will be asked for more details. The AlfaBot will also ask the user for the name of the object he/she wants to create a dependent object for. The AlfaBot will open the relevant view for the object that the user specifies as the context object.

If the object class or object class stereotype is configured to deny creation via the AlfaBot, the AlfaBot will inform the user.

- **Navigate to Artifact**: If the user wants to open a standard page view for an existing object or the object profile or object cockpit of an existing object, he/she can use any of the configured training phrases. The AlfaBot needs the caption of the object class or object class stereotype, the name of the object, and optionally the caption of a standard page view or object cockpit to process the intent. If the object class or object class stereotype is configured to allow navigation via the AlfaBot

and the user profile configuration the user is currently logged in with provides access to the view, the AlfaBot will open the defined view. If the user does not provide information about a standard page view or object cockpit to open, the object profile or first defined object cockpit will open. If the object class or object class stereotype is configured to deny navigation via the AlfaBot, the AlfaBot will inform the user.

- **Edit Artifact**: If the user wants to edit an existing object, he/she can use any of the configured training phrases. The AlfaBot needs the caption of the object class or object class stereotype and the name of the object to process the intent. If the object class or object class stereotype is configured to allow editing via the AlfaBot, the AlfaBot will open the editor or wizard for the object. If the user profile of the user is configured to open a wizard for editing the object, the user can add the caption of the wizard view he/she would like to edit to the request. The wizard will then open displaying the requested wizard step. After the user has edited the object, the object profile or object cockpit of the edited object will open in the Alfabet user interface. If the object class or object class stereotype is configured to deny editing via the AlfaBot, the AlfaBot will inform the user.
- **Navigate to Diagram**: If the user wants to open a configured report, he/she can use any of the configured training phrases. The AlfaBot identifies the diagram via a combination of diagram name, diagram type, and the name of the object that the diagram was created for. The AlfaBot will open the requested diagram.
- **Navigate to Report**: If the user wants to open a configured report, he/she can use any of the configured training phrases. The AlfaBot needs the caption of the report to process the intent. The AlfaBot will open the requested configured report.
- **Start Workflow**: If the user wants to start a workflow, he/she can use any of the configured training phrases. Workflows can be started via the AlfaBot for all workflow templates that have been enabled to start via the AlfaBot. In the AlfaBot, a user can ask to start a workflow for an existing object by entering the object class stereotype and object name. If multiple workflows can be started for the object, these will be listed in the AlfaBot and the user can start the relevant workflow. Alternatively, the user can ask to start a workflow by entering the workflow caption. Multiple captions defined to start a workflow with new objects or with existing objects will be taken into account. In this case, the AlfaBot will request information about the object that the workflow shall be started with. The caption of the workflow started via the AlfaBot is a concatenation of the caption configured for the workflow and the name of the object that the workflow has been started for.
- **Analyze**: If a user searches for specific information in reports that may not have relevant information in the title or description, he/she can use any of the configured training phrases. For example, a user searching for information about the usability of the enterprise's applications could find the report "Market Readiness of Applications" which includes an indicator for usability even though the term "usability" is neither in the title or description of the report.
- **Provide Keywords**: If the AlfaBot is not able to identify a configured report in the **Navigate to Report** request, for example, the user will be asked to provide keywords to identify the report. The user can answer this request with any of the configured training phrases for this request or provide the keywords without additional text. If multiple keywords are provided, they can be separated with either a comma, the word "`and`", or an ampersand (&).
- **Provide Class or Stereotype Name**: If the AlfaBot is not able to map the value of a placeholder that should return an object class caption or object class stereotype caption to any of the available object classes or object class stereotypes, the user will be asked to provide a correct caption. The user can answer this request with any of the configured training phrases for this request or provide the object class or object class stereotype caption without additional text.

- **Provide Object Name**: If the AlfaBot is not able to map the value of a placeholder that should return an object name to any of the available objects, the user will be asked to provide a correct object name. The user can answer this request with any of the configured training phrases for this request or provide the object name without additional text.
- **Fallback**: This intent will be executed if the phrase that the user enters in the AlfaBot cannot be mapped to any of the other intents. The AlfaBot will answer with any of the configured responses.
- **Provide Positive Confirmation**: If the AlfaBot requests a confirmation from the user that the interpretation of the user request is correct, the user can answer with any of the configured training phrases for this intent to confirm.
- **Provide Negative Confirmation**: If the AlfaBot requests a confirmation from the user that the interpretation of the user request is correct, the user can answer with any of the configured training phrases for this intent to confirm.
- **Cancel Conversation**: Phrases that the user might enter to indicate that he/she would like to end the conversation in the AlfaBot window. By default, no response is returned and the AlfaBot will close without a prior response. Optionally, responses can be configured. The AlfaBot will answer with any of the configured responses prior to ending the conversation.
- **Ask for Help**: Phrases that the user might enter if he/she does not know how to work with the AlfaBot. The AlfaBot will answer with any of the configured responses.

If you click an intent in the explorer, a workspace opens that lists all training phrases and, if applicable, responses that are available in a table. All training phrases are listed by default in ascending alphanumerical order. You can change the sort order by clicking the header of a column in the table. The table provides the following information:

- **Training Phrases** / **Responses**: The text of the training phrase or response is listed in the column. If an intent can have both training phrases and responses, the table will be an expandable dataset with a separate section for training phrases and responses.
- **Synchronized**: The column displays a selected checkbox for all training phrases and responses that have been sent to Dialogflow. If the selected checkbox is not displayed, the training phrase can currently not be used for the intent. Click the **Synchronize** button in the toolbar to send the phrases to Dialogflow.
- **Reserved**: If the column displays a selected checkbox, the training phrase or response is a standard phrase that cannot be edited or deleted.

The AlfaBot is preconfigured to understand standard training phrases. Some of these are reserved to ensure a basic functionality of the AlfaBot and cannot be changed or deleted. If the training phrases provided by default do not match the wording that is typically used by your users, you can add typical phrases that you expect to be entered in the AlfaBot for the selected intent.

It is recommended to use the **AlfaBot Interaction Analysis** functionality to analyze the outcome of user input in the AlfaBot and to decide about new training phrases based on actual user input. For more information, see the section *Training the AlfaBot to User Input* in the reference manual *User and Solution Administration*.

To create a new training phrase or a new response for the **Welcome**, **Fallback**, **Cancel Conversation**, and **Ask for Help** intents:

1) Click the **Intents** root node of the explorer.
2) Click the **Turn Configuration Mode On** button.

If the button is called **Turn Configuration Mode Off**, the configuration mode is already turned on. In configuration mode, training phrases can be edited and user input is exclusively processed via Dialogflow. Internal pre-processing based on existing intents is deactivated.

3) In the explorer, click the intent for which you want to add training phrases.

4) In the toolbar of the intent's workspace, click **New** > **Training Phrase** to enter a training phrase or **New** > **Response** to enter a response. An editor opens.

5) Enter text in the **Training Phrase** field for the training phrase or the **Response** field for the response. For a training phrase, enter a phrase that the user can enter to execute the intent. The training phrase must fulfill the following criteria:

- The training phrase must be written in English.
- The training phrase must not contain special characters and punctuation. A mechanism is implemented that strips the following characters from the training phrase or response on synchronization: comma (,) dot (.), semicolon (;), question mark (?), exclamation mark (!), slash (/), backslashes followed by quotation mark or single quotation mark (\', \"), curly brackets ({,}), parenthesis ((,)), square brackets ([,]), angle brackets (<,>), plus sign (+), equal sign (=), asterisk (*), underscore (_), dollar sign ($), hash (#), circumflex (^), percent sign (%).
- The guidelines for the intent provide information about the placeholders that can be used in the training phrase. A training phrase can only include each placeholder once. It can include only a subset or none of the placeholders listed in the guidelines. The AlfaBot will then ask the user to provide the missing information.

  Typically a placeholder is used for names of For example, objects, object classes, or configured reports. An exception of this rule is the `Analyze` intent where the placeholder `@report` is not standing for the name of a configured report but the complete information the user is looking for. Standard training phrases for this intent are For example, "What are `@report`" or "Where are `@report`". This ensures that most of the user input is analyzed via the search mechanism.

- The training phrase must not consist of only placeholders. The identification of a placeholder without text is managed by a separate mechanism that does not require training phrases and will not work properly if a training phrase consists of only placeholders.

  The identification of only placeholder values as input requires additional configuration described below.

For a response, enter a phrase that the AlfaBot can return to the user as a result of the intent. The phrase must not include placeholders and it must be written in English language.

6) Click **OK**. The training phrase or response is added to the table. The **Synchronized** column is empty.

7) In the toolbar, click **Synchronize**. The training phrase or response is sent to Dialogflow. The **Synchronized** column now displays a checkmark.

8) Click the **Intents** root node of the explorer.

9) Click the **Turn Config Mode Off** button.

Existing training phrases or responses that are not reserved can be edited via the **Edit** button or deleted via the **Delete** button. Both actions require that the changes to the training phrases for the intent are synchronized with Dialogflow via the **Synchronize** button afterwards.

Each time a customer-defined training phrase is changed or added in the AlfaBot configuration functionality or a standard training phrase is added or changed via an update of the meta-model (for example, on upgrade to a new Alfabet release), an example question will be added to the internal list of stored user input in the AlfaBot. This reduces the number of requests send to Dialogflow®. If a user enters a request corresponding to the new training phrase in the AlfaBot, the internal pre-processing can interpret the request based on the example phrase and does need to send this request to Dialogflow®. The example phrases are not visible in the **AlfaBot Interaction Analysis** functionality.

Deactivating Intents

Intents can be deactivated in the **AlfaBot Configuration** (`ChatBot_Config`) functionality in the Alfabet user interface.

If an intent is deactivated, it is not taken into account by the AlfaBot when trying to match the user input to an intent. In addition, it is removed from the list of intents displayed to the user on first opening of the AlfaBot in a user session. For example, for companies that have not implemented workflows, it would be confusing for the users to see the intent about start of a workflow in the intent list of the AlfaBot and the workflow intent should Therefore, be deactivated. Only intents listed on start of the AlfaBot can be deactivated.

1) Click the root node of the explorer. A view with a table listing relevant intents and the information about activation or deactivation is displayed. By default, all intents are activated and show a hook in the column **Active**.
2) In the table listing the intents, select one or multiple intents that you would like to deactivate.
3) In the toolbar above the table, select **Deactivate**.
4) End the user session and re-start the Alfabet Web Application to apply the changes.

To re-activate a deactivated intent, select the intent in the table and click **Activate** in the toolbar. Re-activation also requires a re-start of the Alfabet Web Application to apply the changes.

Updating Dialogflow Entities With Information About Customization of the Meta-Model

For some intents, the user may enter a placeholder value only. For example, if the AlfaBot asks the user to provide the name of an object class, it is highly probable that a user will not enter "The name is application", but "application" only.

Placeholders in training phrases correspond to entities in Dialogflow. Some of the entities for the intents that are preconfigured for Alfabet not only provide a name for the placeholder in the training phrase, but also provide a range of allowed content values. For example, for object classes, a list of object class captions is stored directly in Dialogflow. Captions of standard page views, wizard steps, and configured reports are also stored in the respective entities.

If a user enters text in the AlfaBot, Dialogflow first compares the text with the available training phrases. If no matches are found, the text is compared with the list of values defined for each entity. If the entry matches an entity value, the information is processed accordingly.

On update of the meta-model via AMM files, new entity values based on the changes applied to the meta-model are directly sent to Dialogflow if the connection to Dialogflow is configured and active while updating the meta-model. Customer configuration of the current database directly performed in Alfabet Expand must be updated manually. This includes stereotype definition, configured reports, and re-naming of standard page view captions and object class names via the vocabulary or via view customization.

Click the **Update AlfaBot Entities** button in the workspace of the root node of the explorer to update entity values for Alfabet -specific intents with strings resulting from configuration of the meta-model.

Running the AlfaBot in Offline Mode

The connection of the AlfaBot to Dialogflow can be deactivated temporarily or permanently if the high number of already processed intents enable the AlfaBot to handle user requests on basis of pre-processed intents only. The connection can be deactivated permanently or until the next web server restart only.

To deactivate the connection until the next web server restart:

- In the **AlfaBot Configuration** functionality on the Alfabet user interface, click the button **Use AlfaBot Offline**.

The Alfabet Web Application will check on each Web Server restart whether DialogFlow is accessible and will return to online mode if the check is positive.

To deactivate the connection to Dialogflow permanently:

- In Alfabet Expand, go to the **Presentation** tab.
- Expand the explorer nodes **XML Objects** > ***IntegrationSolutions***.
- Right click the node ***AlfaChatBotConfig*** and select **Edit XML**.
- In the editor, add the XML attribute `IsOffline` to the XML element `ChatBotInfo` and set it to `true`.

```
<AlfaChatBotConfig>
    <ChatBotInfo
        Type="Dialogflow"
        IsActive="true"
        IsOffline="true"
        [...]>
        <Settings [...]>
    </ChatBotInfo>
    </AlfaChatBotConfig>
```


Configuring the Extended Data Capture Functionality

The **Extended Data Capture Templates** functionality provides a sophisticated and comprehensive means to collect large sets of data for object classes as well as reference information for properties of the type `ReferenceArray` in the context of XLSX files. One or more data capture templates can be configured for a permissible object class in order to address multiple data capture approaches for different regional units, customer segments, products, etc. Data may be captured in any of the languages supported by your enterprise.

For more information about data collection via data capture templates, see the section *Capturing Data with Data Capture Templates* in the reference manual *User and Solution Administration*.

The following configuration is required to implement the **Extended Data Capture Templates** functionality:

- **Enable for Data Capture Templates**: Select `True` if the object class may be captured via data capture templates. Select `False` if the object class may not be captured via data capture templates. For each object class for which the **Enable for Data Capture Templates** attribute is set to `True`, you must enable the individual properties that shall be captured for the object class by setting the **Enable for Data Capture Templates** attribute for each object class property to `True`. For information about specifying the **Enable for Data Capture Templates** attribute for an object class property, see the section Editing a Protected Property.
- Specify the object class and object class properties for which data shall be captured:
  - Set the **Enable for Data Capture Templates** attribute to `True` for the object class for which a data capture template is to be configured. Data capture is limited to `Artifact` object classes only and the **Enable for Data Capture Templates** attribute may only be specified for those classes for which data capture is permissible. If data capture is not permissible for a class, the attribute will be greyed out.
  - Set the **Enable for Data Capture Templates** attribute to `True` for each public or protected object class property that shall be exported to the data capture template for the object class. If the protected property is inherited from the parent object class `Artifact` or `ArtifactAuthorized`, then the **Enable for Data Capture Templates** attribute will be greyed out. In this case, you must specify a local setting for the property that will apply to the property only in the context of this object class. To do so, expand the **Local Settings** section and set the **Enable for Data Capture Templates** attribute to `True` to override the inherited setting from the parent object class.

    Properties of the type `BinData`, `Picture`, `DateArray`, `POSIX`, `ByteArray`, `DateArray`, `DateTimeArray`, `BindataArray`, `IntegerArray`, `TimeArray`, `PropertySet`, `StringArray`, and `URLArray` are not suitable for data capture templates.

    For more information about configuring object classes and object class properties, see the chapter Configuring the Class Model in the reference manual *Configuring Alfabet with Alfabet Expand*.

- Specify the release statuses for class-based and reference-based date capture templates. Release status definitions must be configured in the XML object ***ReleaseStatusDefs*** for the classes `ALFA_DATACAPTURETEMPLATE:Class` and `ALFA_DATACAPTURETEMPLATE:Reference`. For more information about specifying release statuses, see the section Configuring Release Status Definitions for Object Classes in the reference manual *Configuring Alfabet with Alfabet Expand*.

- If necessary, create configured reports that shall be assigned to data capture templates. The configured report should be of the type `Query` or `NativeSQL` and should return the set of objects to be reviewed, updated, or deleted as well as the sort order of the returned objects. For each configured report, the **Category** attribute of the configured report must be set to ADIF or the relevant category specified for the use case `DataCaptureTemplates` in the XML object `UseCaseCategory`. The reports assigned the relevant category for data capture templates will be available in the Export Record Provider field in the **Data Capture Template** editor. For more information about specifying the XML object `UseCaseCategory`, see the section Assigning a Category for Specific Functional Use to a Configured Report For more information about how to create a configured report, see the chapter Configuring Reports in the reference manual *Configuring Alfabet with Alfabet Expand*. Configured reports may need to be created for the following:

  Please note that instead of specifying a configured report to limit the set of objects available in the XLSX file, it is also possible to specify that either all available objects or only objects based on a specified stereotype may be exported.

  - Configured reports may be configured to find the objects for the relevant object class targeted by the data collection activity that shall be exported to the XLSX file.
  - Configured reports may be configured to find the referenced objects for properties of the type `Reference`.
  - Configured reports may be configured to find the relevant objects for properties of type `ReferenceArray`. A configured report may be specified to find objects of the base class of the reference and another configured report may be configured to find the objects that the base class objects may reference.
  - Configured reports may be configured to find the relevant persons or organizations referenced by a role type.
  - Optional for either class-based or reference-based data capture templates: A configured report may be configured to find the objects to export as sample data to the XLSX file. Sample data may be exported to the XLSX file to provide examples of existing data for users to understand how to create or modify the data in the XLSX file. The sample data is exported to a **Sample Data** tab in the XLSX file where users can experiment with the data. The sample data will not be reimported to Alfabet and Therefore, the Alfabet database will not be impacted by changes made to the sample data.

- Ensure that the **Extended Data Capture** (`ADMIN_CaptureDataNextGen`) functionality is available to the user profile of the users that will be responsible for configuring data capture templates and importing and exporting the data. For more information about adding functionalities to a user profile, see Making Functionalities Accessible to a User Profile in the chapter Configuring User Profiles for the User Community.
- If necessary, configure the asynchronous import and export for large amounts of data. If asynchronous execution is configured, the user defining the data capture template may activate asynchronous import/export of data. If asynchronous import/export of data is not activated, the data will be imported and exported synchronously. To make asynchronous execution available for data capture.
  - The asynchronous import and export of data requires a running Alfabet Server to be connected to the same database as the Alfabet Web Application providing the **Extended Data Capture** functionality in the Alfabet user interface.

- A remote alias must be configured and the **Use Server to Execute ADIF Jobs** checkbox must be selected for the remote alias. For more information, see the section *Creating a Remote Alias* in the reference manual *User and Solution Administration*.
- The Alfabet Web Application must be configured to connect to the Alfabet Server via the remote alias and to use the Alfabet Server for the execution of ADIF jobs. The **Use Server to Execute ADIF Jobs** attribute in the **Application Server** tab of the server alias of the Alfabet Web Application must be selected.
- The RESTful services must be activated for the Alfabet Web Application as defined in the reference manual *Alfabet RESTful API*. The configuration of the RESTful services must grant at least the following access:
  - A user for execution of self-reflective events must be defined.
  - Both the access permissions for the RESTful services in the server alias of the Alfabet Web Application and for the user for execution of self-reflective events must grant the permissions **Has Batch Utilities API Access**.
- Specify the document folder in the **Internal Document Selector** where the XLSX files generated during asynchronous import/export shall be stored in. This is specified in the XML object ***IDocManagerConfiguration*** and is described in detail in the section Configuring Access Permissions to Folders in the Internal Document Selector for the Job Schedule. Please note that the configuration of the folder for data capture templates is included in the XML specification `<UseCase Name="JobSchedule">`. The caption specified in the XML attribute `Folder Name` will be displayed for the folder in the **Import/Export Asynchronously** tab of the **Data Capture Template - Class** editors. This caption may differ from the technical folder name in the XML attribute `Path`.
- In the **Import/Export Asynchronously** tab of the **Data Capture Template - Class** editor available in the **Extended Data Capture** functionality, ensure that the **Import/Export Asynchronously** checkbox is selected and select the relevant document folder in the **Internal Document Selector** where the export file shall be created. For more information about the required configuration of extended data capture templates, see the section *Capturing Data with Data Capture Templates* in the reference manual *Configuring Alfabet with Alfabet Expand*.

Activating the Job Schedule Functionality

With the **Job Schedule** functionality, users can schedule execution of the following functionalities in regular intervals.

- ADIF import and ADIF export.
- Removal of old information about ADIF session execution from the Alfabet database.
- Update and creation of full text search indexes.
- Re-computation of automatically calculated indicators.
- Assignment of coloring defined by color rules.
- Starting workflows configured to be started automatically, for closing workflow steps configured to be closed automatically, for automatic deletion of finished workflows, and for rescanning and updating roles and responsibilities for workflows and workflow steps.

In the **Job Schedule** functionality, users can create objects of the object class Job Schedule that provide information about the functionality to execute and the time table for execution. Directly after creation of a job schedule, a wakeup event is added to a job scheduler event queue with the information about the job schedule and the next execution time.

The Alfabet Server checks the event queue in regular intervals for job schedules that need to be executed. At execution time, the following is done:

- A RESTful service call to the Alfabet RESTful services schedules an ADIF job for immediate execution in the queue for ADIF execution. All job schedule functionalities are executed via ADIF.

  For the execution of ADIF import and ADIF export, the defined ADIF job is scheduled for execution. For all other functionalities, standard private ADIF jobs are available to execute the functionalities. The ADIF jobs are visible in the ADIF explorer in Alfabet Expand in the folder ***StandardJobs***. They cannot be edited.

- A new wakeup event for the next execution in the job schedule's time table is added to the job scheduler queue.

The following preconditions must be met to enable the **Job Schedule** functionality:

- The job scheduler requires a running Alfabet Server to be connected to the same database as the Alfabet Web Application providing the **Job Schedule** functionality on the user interface.
- The Alfabet Web Application must be configured to to use the Alfabet Server for execution of ADIF jobs. The **Use Server to Execute ADIF Jobs** attribute in the **Application Server** tab of the server alias of the Alfabet Web Application must be selected.
- The RESTful services of the Alfabet Web Application must be activated and configured as described in the chapter *Activating the Alfabet RESTful API on Server Side* of the reference manual *Alfabet RESTful API*.
- A user must be configured for execution of RESTful service calls via self-reflective events. For more information, see Configuring a User to Execute Self-Reflective Events below.
- The **Job Schedule** (`JobSchedule`) functionality must be added to the user profile of the user that will be responsible for scheduling the jobs. For more information about adding functionalities to a

user profile, see Making Functionalities Accessible to a User Profile in the chapter Configuring User Profiles for the User Community.

- Optionally, the interval that the Alfabet Server checks the queues for scheduled wakeup events and ADIF jobs can be adapted to your demands. The default intervals are optimized for the functionality and usually do not need to be changed. For more information, see Changing the Interval for Checking the Queues of Scheduled Events and ADIF Jobs below.
- For the execution of ADIF export and import, a number of additional configurations are required that are described below in the section Configuration Required for Scheduling ADIF Jobs.
- For the rescan of indicators, a number of additional configurations are required that are described below in the section Configurations for Scheduling Rescan of Indicators.
- For each functionality to execute, the required configuration for activation and execution of the functionality in Alfabet has to be available. For information about the configuration of the basic functionalities provided by the job schedules, see one of the following:
  - For the configuration of ADIF export schemes, see *Configuring Data Export with ADIF* in the reference manual *Alfabet Data Integration Framework*.
  - For the configuration of ADIF import schemes, see *Configuring Data Import with ADIF* in the reference manual *Alfabet Data Integration Framework*.
  - For the configuration of the full-text search functionality, see Configuring the Full-Text Search Capability.
  - For the configuration of color rules, see *Configuring Color Rules for Map Views and Diagram Views* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.
  - For the definition of automatically calculated indicators for evaluation types, see *Configuring Evaluation Types* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.
  - For the configuration of workflows, see Configuring Workflows

The following information is available:

- Configuring a User to Execute Self-Reflective Events
- Changing the Interval for Checking the Queues of Scheduled Events and ADIF Jobs
- Defining Maintenance Windows
- Configuration Required for Scheduling ADIF Jobs
  - Creating Categories for the Job Schedule Use Case
  - Configuring the ADIF Scheme to be Executable via the Job Schedule Functionality
  - Configuring Access Permissions to Folders in the Internal Document Selector for the Job Schedule
- Configurations for Scheduling Rescan of Indicators
  - Creating Report Categories for the Job Schedule Use Case
  - Defining the Query in a Configured Report

Configuring a User to Execute Self-Reflective Events

The user selected for execution of events of the type `SelfReflective` will be used for authentication of RESTful service calls to the RESTful API of the Alfabet Web Application for both wakeup events involved in job scheduling and for all other event based activities that involve execution via the RESTful API.

For more information about triggering actions via event templates, see the chapter Configuring Events.

It is recommended to create a user that is exclusively used for executing events via the RESTful services. The user can be excluded from access to any objects except for executing job schedules, asynchronously starting ADIF jobs via the user interface, and start of workflows, RESTful service calls and/or execution of ADIF jobs via events based on event templates, but it is also possible to use an existing user.

The user must be a named user with at least one user profile assigned. The user profile is not used for evaluation of access permissions. A ReadOnly user profile is sufficient to execute the RESTful services in the context of events.

Only one user can be selected for execution of events of the type `SelfReflective`. If you assign this functionality to a user while another user has already been selected to execute self-reflective events, the setting is removed from that user when it is set for the user you are currently assigning it to. Therefore, if a user has already been defined in the context of the activation of any other functionality, you should add the required access permissions described below to this user instead of creating another user to make sure that the permissions for the functionality the user was created for is maintained.

To create a user for execution of self-reflective events in the **User Administration** functionality on the Alfabet user interface:

The same functionality is also available via the Alfabet Administrator. For information about accessing the user management functionalities of the Alfabet Administrator see *Functionalities Available via the Expanded Explorer of the Connected Alias* in the reference manual *System Administration*.

1) In the toolbar of the **User Administration** view, select **New** > **Create New User**. An editor opens.
2) In the editor, define the following:

   Basic Data tab:

   - **Name**: Enter a meaningful name for the user. The user is a technical user. You can either assign the name of an existing person to it or give it a name that indicates that this is a virtual person defined to execute a functionality.
   - **User Name**: Enter a user name. The user name is used by the RESTful services to identify the user.
   - **Type**: Select `NamedUser`.

   API Permissions tab:

   - **Has API V2 Access**: Select the checkbox.
   - **API Access Options**: Make sure that the following permission are selected:
     - **Has ADIFAPIInvocation Access** for the execution of ADIF schemes via the **ADIF Import Job Schedule** and the **ADIF Export Job Schedule**.

- **Has Batch Utilities API Access** for the execution of all other job schedules.

- **Generate API Password**: Click the button. The API Password field is filled. For events of the type `Query`, copy the **API User Name** and the **API Password** . These have to be entered into the specification in the XML object `AlfabetIntegrationConfig` described below in the section Configuring the Connection Parameters in the XML Object AlfabetIntegrationConfig.

3) Click **OK** to close the editor. The new user is added to the table in the **User Administration** view.
4) Select the user in the table and select **Action** > **Set as Executes Self-Reflective Events User** in the toolbar.
5) Select the user in the table and click the **Navigate**  button.
6) In the object profile of the user, click **Assigned User Profiles**.
7) In the toolbar of the **Assigned User Profiles** view, click **New** > **Assign User Profiles**.
8) In the selector, select a user profile and click **OK** to assign it to the user. For a user that is exclusively used for events, it is recommended to use a ReadOnly user profile.

The configuration described above is the configuration required for the **Job Schedule** functionality only. In addition, you can set any other properties of the user. For security reasons you might also consider to assign a login password to the user although the user password is not required for the **Job Schedule** functionality. For information about the available configuration options for users, see *Defining and Managing Users* in the reference manual *User and Solution Administration*.

Changing the Interval for Checking the Queues of Scheduled Events and ADIF Jobs

This configuration is optional. The default interval of the Alfabet Server for checking the wakeup events created for execution of job schedules for ADIF jobs due for execution is 500 milliseconds and the default interval of the Alfabet Server for checking the ADIF jobs queue and execute the next ADIF job is 3000 milliseconds. Optionally, you can adapt these values to your demands.

The configuration has to be performed in the tool Alfabet Administrator:

1) In the explorer of the Alfabet Administrator, click the **Alfabet Aliases** node.
2) In the table, select the server alias configuration that is used for starting the Alfabet Server.
3) In the toolbar, click the Edit  button. The server alias editor opens.
4) Go to the **Server Settings** > **General** tab and edit the following attributes:
   - **ADIF Job Server Sleep Time (milliseconds)**: Enter the time between checking of the queue of ADIF jobs to be executed.
   - **Event Server Sleep Time (milliseconds)**: Enter the time between checking the event queue for scheduled wakeup events to be executed.
5) Click **OK** to save your changes.

Defining Maintenance Windows

This configuration is optional. Maintenance work on the Alfabet components, like For example, update of the meta-model, conflicts with the execution of scheduled jobs. To ensure that no scheduled jobs are executed during maintenance, maintenance windows can be defined in Alfabet. Job schedules can then be configured to read the current maintenance window specification and schedule any job that is due during a maintenance window to start 1 minute after the end time of the maintenance window instead.

Maintenance windows can either be configured to be recurring once a week or once a month or they can be scheduled for a specified date. All specifications refer to the time of the server host.

Maintenance windows are configured in the XML object ***MaintenanceWindows*** in Alfabet Expand:

1) In the Presentations tab, expand the explorer nodes **XML Objects** > **Administration**.
2) Double-click the ***MaintenanceWindows*** node. The XML editor opens in the middle pane with an empty root XML element:

   ```
   <MaintenanceWindows></MaintenanceWindows>
   ```

3) For each maintenance window that you want to define, add a child XML element `MaintenanceWindow` to the XML element `MaintenanceWindows`.
4) The definition of the XML attributes for the XML element `MaintenanceWindow` depends on the scheduling options:

   To define a recurring maintenance window, add the following XML attributes:

   - `DayOfWeek`: If the maintenance window occurs once a week, define the day of the week that the maintenance window shall occur. Allowed values are the English names of the days of the week starting with a capital letter.
   - `DayOfMonth`: If the maintenance window occurs once a month, define the day of the month that the maintenance window shall occur as integer.
   - `StartTime`: Define the start time of the maintenance window in the twenty-four hour time in the format hh:mm.
   - `EndTime`: Define the end time of the maintenance window in the twenty-four hour time in the format hh:mm.

   Example for maintenance windows recurring once a week and once a month:

   ```
   <MaintenanceWindows>
      <MaintenanceWindow DayOfWeek="Friday" StartTime="20:30"
      EndTime="23:00"/>
      <MaintenanceWindow DayOfMonth="12" StartTime="08:00"
      EndTime="12:00"/>
   </MaintenanceWindows>
   ```

   To define a maintenance window for a defined date, add the following XML attributes:

   - `Format`: Define the date format that you will use for definition of the date for the maintenance window. the XML attribute is optional. If it is not defined, the date format defined in the culture settings that are configured to be the primary culture are used.Allowed formats are:

- MM/dd/yyyy
- dd/MM/yyyy
- M/d/yyyy
- d/M/yyyy

- `Date`: Define the date the maintenance window will occur in the format defined with the XML attribute `Format`.
- `StartTime`: Define the start time of the maintenance window in the twenty-four hour time in the format hh:mm.
- `EndTime`: Define the start time of the maintenance window in the twenty-four hour time in the format hh:mm.

Example for a one time maintenance window:

```
<MaintenanceWindows>
   <MaintenanceWindow Date="12/13/2019" Format="MM/dd/yyyy"
   StartTime="08:00" EndTime="12:00"/>
</MaintenanceWindows>
```

5) In the toolbar, click the **Save**  button.

Maintenance work is usually scheduled by system administrators with no access to Alfabet Expand. The XML object ***MaintenanceWindows*** can Therefore, also be defined in the Alfabet Administrator or in an XML file located on the local file system and imported into the XML object in the Alfabet database via a command line tool.

To provide access to the XML object ***MaintenanceWindows*** via the Alfabet Administrator, set the attribute **Visible In Administrator** of the XML object to `True`.

For more information about the methods for defining maintenance windows that are available for system administrators, see *Defining Maintenance Windows for Scheduled Jobs* in the reference manual *System Administration*.

Configuration Required for Scheduling ADIF Jobs

The following configuration is required for the execution of ADIF import or ADIF export via the **Job Schedule** functionality. These configuration steps are not required for other job schedules.

The following information is available:

- Creating Categories for the Job Schedule Use Case
- Configuring the ADIF Scheme to be Executable via the Job Schedule Functionality
- Configuring Access Permissions to Folders in the Internal Document Selector for the Job Schedule

Creating Categories for the Job Schedule Use Case

ADIF schemes can only be scheduled via the **Job Schedule** functionality if they are assigned to a category defined for the functionality in the XML object ***UseCaseCategories***.

To set the categories for the **Job Schedule** functionality in the XML object ***UseCaseCategories***:

1) In the **Presentation** tab of Alfabet Expand, expand the explorer node **XML Objects**.
2) Right-click on the XML object ***UseCaseCategories*** in the explorer and select **Edit XML** from the context menu. The XML object opens in the middle pane.
3) Enter the following code as child element of the XML element ***UseCaseCategories***:

```
<UseCaseInfo UseCase="JobScheduler">
   <ScopeInfo Scope="ADIF" Categories="CommaSeparatedListOfCategories" />
</UseCaseInfo>
```

   The XML attribute `Categories` must be set to either one category name or a comma separated list of category names. Please note that a category name is a technical name and should not contain special characters or whitespaces. All category names that are defined can be used in the **Category** attribute of an ADIF scheme to activate scheduling for the ADIF scheme.

   Private ADIF schemes are delivered with Alfabet to trigger For example, automated translation or import or export in the scope of an integration interface. If you would like to schedule execution of these private ADIF schemes, the list of categories must include the respective category that is preset for the ADIF scheme. The following category names are set for private ADIF schemes:

   - `Translation` for all ADIF jobs for batch processing associated with automated translation.
   - `Technopedia` for the ADIF import scheme `ALFABET_TECHNOPEDIA_UPDATE`.
   - `CentraSite` for all ADIF jobs that are associated with CentraSite.
   - `APIGateway` for the ADIF import scheme `Alfabet_APIGateway_Synchronization`.
   - `Apigee` for the ADIF import scheme `Alfabet_Apigee_Synchronization`.
   - `APIPortal` for the ADIF import scheme `Alfabet_APIPortal_Synchronization`.

4) In the toolbar, click the **Save**  button.

Configuring the ADIF Scheme to be Executable via the Job Schedule Functionality

The ADIF Job must meet the following preconditions to be scheduled via the **Job Schedule** functionality:

- It must be assigned to one of the categories defined for the use case `JobScheduler` and the scope `ADIF` in the XML object ***UseCaseCategories***.

  For information about defining category names in the XML object ***UseCaseCategories***, see Creating Categories for the Job Schedule Use Case.

- The ADIF scheme must not include parameters defined in compatibility mode or with the data type `StringArray` or `ReferenceArray`.

To activate job scheduling for an ADIF import scheme:

1) In the **ADIF** tab of Alfabet Expand, click the ADIF scheme.
2) In the attribute window, select one of the categories defined for the use case `JobScheduler` and the scope `ADIF` in the XML object ***UseCaseCategories***.

Configuring Access Permissions to Folders in the Internal Document Selector for the Job Schedule

ADIF import scheduled via the job scheduler can be performed from a file located in the **Internal Document Selector** of the Alfabet database. This method is optional. Import can also be performed from a file on the local file system. ADIF export to file via the job scheduler can exclusively target the **Internal Document Selector**. Export of data to the local file system is not available for scheduled jobs.

The job scheduler can only access folders of the **Internal Document Selector** with explicit access permissions for the job scheduler.

To define the IDOC folders that are accessible via scheduled ADIF jobs:

1) In the **Presentation** tab of Alfabet Expand, expand the explorer node **XML Objects** > **Administration**.
2) Right-click on the XML object ***IDocManagerConfiguration*** in the explorer and select **Edit XML** from the context menu. The XML object opens in the middle pane.
3) Enter the following code:

```
<IDocManager>
   <UseCase Name="JobSchedule" >
      <Folder Name="FolderShowName" Path="IDOC:\Path" />
   </UseCase>
</IDocManager>
```

   The XML element `UseCase` can have multiple child elements `Folder`, each defining one folder that shall be accessible via the job scheduler. Define the folder with the following XML attributes of the XML element `Folder`:

   - `Name`: Define a caption that shall be displayed for the folder in the editor for scheduling ADIF jobs. The name does not have to be identical to the name of the folder in the **Internal Document Selector**.
   - `Path`: Define the path to the folder in the **Internal Document Selector** . The path must start with `IDOC:\` and end with the name of the selected folder. Backslashes must be used between folder names. The job scheduler can read and write in the defined folder and all subordinate folders of the defined folder.

4) In the toolbar, click the **Save**  button.

Configurations for Scheduling Rescan of Indicators

The following configuration is used for the rescan of indicators via the **Job Schedule** functionality. These configuration steps are not required for other job schedules.

Rescanning of indicators can optionally be limited to a subset of objects of a defined object class found by a query. The query must be provided via a configured report. This section describes the required configurations for using a configured report for the rescan of indicators via the job schedule functionality.

The following information is available:

- Creating Report Categories for the Job Schedule Use Case
- Defining the Query in a Configured Report

Creating Report Categories for the Job Schedule Use Case

Configured reports can only be used to restrict rescan of indicators via the **Job Schedule** functionality if they are assigned to a category defined for the functionality in the XML object ***UseCaseCategories***.

To set the categories in the **Job Schedule** functionality in the XML object ***UseCaseCategories***:

1) In the **Presentation** tab of Alfabet Expand, expand the explorer node **XML Objects**.
2) Right-click on the XML object ***UseCaseCategories*** in the explorer and select **Edit XML** from the context menu. The XML object opens in the middle pane.
3) Enter the following code as child element of the XML element ***UseCaseCategories***:

```
<UseCaseInfo UseCase="JobScheduler">
    <ScopeInfo Scope="Report" Categories="CommaSeparatedListOfCategories"
    />
</UseCaseInfo>
```

   The XML attribute `Categories` must be set to either one category name or a comma separated list of category names. Please note that a category name is a technical name and should not contain special characters or whitespaces. All category names that are defined can be used in the **Category** attribute of a configured report to activate it for the use in job schedule definitions for rescan of indicators.

4) In the toolbar, click the **Save**  button.

Defining the Query in a Configured Report

If only a subset of objects of a class shall be scanned for re-calculation of indicators, the information about which objects shall be included must be defined in a configured report.

The following configuration is required for the configured report:

For general information about configured reports, see Configuring Reports. For general information about defining queries for Alfabet configurations, see Defining Queries.

- The **Type** attribute of the configured report must be `Query` or `NativeSQL`.
- The **Category** attribute of the configured report must be set to one of the categories defined for the use case `JobScheduler` and the scope `Report` in the XML object ***UseCaseCategories***.
- The rescan mechanism reads the information about which objects are found from the query. For Alfabet queries, these are the objects defined as `FIND` class. For native SQL queries, the first argument in the `SELECT` statement must return the `REFSTR` of the objects to be included into the rescan process. All other data returned via the `SELECT` statement of native SQL queries or the **Show Properties** of Alfabet Queries will be ignored.
- The configured report must have the **State** set to `Active` to be used for a job schedule.

Configuring the Web Polling Mechanism

The XML object `SolutionOptions` allows you to specify the Web polling mechanism that allows the user interface to poll the server for the completion status of background processes for which completion shall be announced via an **Event Feedback** message.

Each user who wants to implement the Web polling mechanism must activate Web polling in their user interface by selecting the **Enable User Interface to Poll Web Server** checkbox in the **User Settings** editor. For more information about defining user settings, see the section *Defining Your User Settings in Alfabet* in the reference manual *Getting Started with Alfabet*.

To edit the XML object `SolutionOptions`:

1) Go to the **Presentation** tab and expand the **XML Objects** folder.
2) Right-click the XML object `SolutionOptions` and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) Specify the following XML attributes:
   - `WebUIPollingInterval`: Specify an integer for the interval of the polling in milliseconds. The default value is 30000 milliseconds.
   - `AsynchronousPublication`: Set to `True` to enable Web polling.
4) In the toolbar, click the **Save**  button.

Configuring the Propagation of Organizational Changes

The XML object ***OrganizationalChanges*** allows you to define settings for the propagation of changes to the organizational structure defined in the Alfabet solution. Users may view and potentially process organizational changes in the **Organizational Changes** page view available on the root node in the **Organizations** explorer as well as for an organization.

In the Alfabet solution, users can document changes to the enterprise's organizations and their dependent objects when a change is made to the organizational structure. For example, if an organization is merged with another organization, the objects dependent on that organization such as subordinate organizations or applications that provide support to the merged organization may need to be assigned to another organization.

Possible changes to the organizational structure include, for example:

- merging an organization with another organization
- absorbing an organization by another organization
- creating a new spin-off organization
- changing an organization's parent organization
- deleting an organization

Whenever a change occurs in the organizational structure, an automatic assignment will be generated for the authorized user(s) of the objects dependent on the organizational change. This allows the authorized user(s) to assess whether the organizational change impacts the object and what action should be taken.

If a changed organization is defined in a business support map, an automatic assignment will be generated for the authorized user(s) of the associated master plan map, IT strategy map, or solution map, unless specified otherwise. Additionally, an automatic assignment will be generated for the authorized user(s) of objects belonging to the following object classes, unless otherwise specified:

- `Application`
- `BusinessProcess`
- `Component`
- `Demand`
- `OrgaUnit`
- `BusinessSupport`
- `Project`
- `StrategicBusinessSupport`
- `TacticalBusinessSupport`

The batch processing tool `AlfaBatchExecutor.exe` must be executed in order for assignments to be generated for the authorized user(s) of the objects dependent to the organizational change. For more information about executing the batch job, see the section *Batch Processing for Monitors and Change Management with AlfaBatchExecutor.exe* in the reference manual *System Administration*.

Changes to the organizational structure can be carried out in two ways in Alfabet. Changes can either be made manually through the Alfabet interface, or automatically via an input file from an external application. Typically, changes to the organizational structure are committed by means of an external system such as a human resources management solution. In this case, you may want to configure the functionality so that

changes to the organizational structure may only be incorporated in Alfabet via an update from the external system.

The XML object ***OrganizationalChanges*** allows you to configure such aspects in the change propagation process including:

- for whom assignments should be generated if there is no authorized user defined for a dependent object
- the default target date for when such assignments should be closed
- whether automatic emails should be sent to the authorized users of dependent objects
- whether organizational changes may be manually defined by users
- whether organizational changes may be automatically defined via an external application
- exclusion rules that define which dependent objects are not considered in an organizational change (for example, all applications with the object state Retired or all demands with the release status Completed, etc.)

The following configuration requirements must be completed by the system administrator in the tool Alfabet Administrator:

- All Alfabet functionalities for which the email capability is to be implemented require the setup of a connection to an SMTP server for outgoing email. For more information, see the section *Activating the Dispatch of Email Notifications in Alfabet* in the reference manual *System Administration*.
- The user profile used to open the Alfabet views targeted by hyperlinks in the email is, by default, the user profile that the sender was logged in with when the view was sent/triggered. However, your system administrator may configure that Alfabet views in email notifications are to be opened using the recipient's user profile. For more information, see the section *Configuring the Setup To Open the Alfabet Interface via Links in Email Notifications* in the reference manual *System Administration*.

The text displayed in automatically-generated email notifications is also configured in Alfabet Expand. For an overview of all text templates available for organizational changes, see *Text Templates for Organizational Changes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*. Please note that you should not set the **Is HTML** attribute to `True` in the context of the text templates available for organization changes. For more information, see the section Configuring Text Templates for Email Notifications in the reference manual *Configuring Alfabet with Alfabet Expand*.

To access the XML object ***OrganizationalChanges***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder, and expand the **ObjectChanges** folder.
2) Right-click the XML object ***OrganizationalChanges*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) Define the XML attributes, as needed. The table below describes the XML elements and XML attributes that can be edited for the XML object ***OrganizationalChanges***:

XML Element (bold) / XML Attribute	Explanation
ObjectChangesDef	
`DefaultResponsible`	Enter the name of the default user that should receive assignments associated with the propagation of changes if the impacted object has no authorized user defined or if the relevant organization does not have a parent organization. The name should be written as follows: [USERNAME]. The text template `TextTemp_ObjectChangeDefaultResponsible` will be sent to the user specified in this attribute if no authorized user is defined for the object impacted by the change. For more information, see *Text Templates for Organizational Changes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.
`DefaultTargetSpanInDays`	Enter an integer to establish the target date when an assignment generated due to an organization change should be completed. The target date is computed based on the date when the assignment is generated plus the value defined in the XML attribute `DefaultTargetSpanInDays`.  If thd XML attribute `DefaultTargetSpanInDays` is defined as 30, every assignment that is generated in the context of an organizational change will have a target date that is 30 days after the date that the assignment was generated. An assignment generated on Jan. 1, 2008 will have a target date of Jan. 30, 2008.
`Profile`	Enter the name of the user profile required to access the Alfabet interface via the hyperlink displayed in the email notification that is generated for the assignment triggered by an organizational change.
`SendEmails`	Enter "`true`" if notification via email should be sent in addition to the assignment to the authorized user(s) of the dependent object informing them about the organizational change. Enter "`false`" if no email notification should be sent.
`AllowObjectChangeOnUI`	Enter "`true`" if organizational changes may be manually defined by users in the **Organizational Changes** page view. Enter "`false`" if users may not manually define organizational changes. The respective functionalities will be disabled in the **Organizational Changes** page view and thus hidden in the user interface.

XML Element (bold) / XML Attribute	Explanation
`ProcessExternal-Changes`	Enter "`true`" if organizational changes are defined via the input of data via an external application. Enter "`false`" if organizational changes should not be triggered via the input of data from an external application. Depending on the value you define, only the relevant functionalities will be displayed in the **Organizational Changes** page view.
ExclusionRule	
`ClassNames`	Enter the values of the **Name** attribute of the object classes for which you want to define an exclusion rule. Multiple classes should be separated by a comma. Enter the symbol * (asterisk) if all classes relevant for the organizational changes capability should be included. Object class names that are misspelled are ignored.
ExclusionRuleDetail	Define criteria that determines which dependent objects should be ignored in the organizational change functionality.
`PropertyName`	Enter the technical name of the public and/or custom object class property (for example, Status) that determines which dependent objects should be ignored in the organizational change functionality. The object class property is relevant for all object classes listed in the XML attribute `ClassNames`.
`ExcludedValues`	Enter one or more values (for example, Discarded, Closed) relevant to the object class property that determine its exclusion.

4) In the toolbar, click the **Save**  button to save the XML definition.

Configuring the Propagation of Business Process Changes

The XML object ***BusinessProcessChanges*** allows you to define settings for the propagation of changes to the business process hierarchy defined in the Alfabet solution. In Alfabet, users may view and potentially process business process changes in the **Process Changes** page view available on the root node in the **Business Processes** explorer as well as for a business process.

In the Alfabet solution, users can document changes to the enterprise's business processes and their dependent objects when a change is made to the business process model. For example, if a business process is merged with another business process, the objects dependent on that business process such as subordinate business processes or applications that provide support to the merged business process may need to be assigned to another business process.

Possible changes to the business process hierarchy include, for example:

- merging a business process with another business process
- absorbing a business process by another business process
- creating a new spin-off business process
- changing a business process's parent business process
- deleting a business process

Whenever a change occurs in the business process hierarchy, an automatic assignment will be generated for the authorized user(s) of the objects dependent to the business process change so that he/she can consider whether the business process change impacts the object and what action should be taken.

If a changed business process is defined in a business support map, an automatic assignment will be generated for the authorized user(s) of the associated master plan map, IT strategy map, or solution map, unless specified otherwise. Additionally, an automatic assignment will be generated for the authorized user(s) of objects belonging to the following object classes, unless otherwise specified:

- `Application`
- `BusinessProcess`
- `Component`
- `Demand`
- `OrgaUnit`
- `BusinessSupport`
- `Project`
- `StrategicBusinessSupport`
- `TacticalBusiness` Support

The batch processing tool `AlfaBatchExecutor.exe` must be executed in order for assignments to be generated for the authorized user(s) of the objects dependent to the process change. For more information about executing the batch job, see the section *Batch Processing for Monitors and Change Management with AlfaBatchExecutor.exe* in the reference manual *System Administration*.

Changes to the business process hierarchy can be carried out in two ways in Alfabet. Changes can either be made manually through the Alfabet interface, or automatically via an input file from an external application. Typically, changes to the business process hierarchy are committed by means of an external system such as a human resources management solution. In this case, you may want to configure the functionality so that changes to the business process hierarchy may only be incorporated in Alfabet via an update from the external system.

The XML object ***BusinessProcessChanges*** allows you to configure such aspects in the change propagation process including:

- for whom assignments should be generated if there is no authorized user defined for a dependent object
- the default target date for when such assignments should be closed
- whether automatic emails should be sent to the authorized users of dependent objects
- whether business process changes may be manually defined by users
- whether business process changes may be automatically defined via an external application

- exclusion rules that define which dependent objects are not considered in a business process change (for example, all applications with the object state Retired or all demands with the release status Completed, etc.)

The following configuration requirements must be completed by the system administrator in the tool Alfabet Administrator:

- All Alfabet functionalities for which the email capability is to be implemented require the setup of a connection to an SMTP server for outgoing email. For more information, see the section *Activating the Dispatch of Email Notifications in Alfabet* in the reference manual *System Administration*.
- The user profile used to open the Alfabet views targeted by hyperlinks in the email is, by default, the user profile that the sender was logged in with when the view was sent/triggered. However, your system administrator may configure that Alfabet views in email notifications are to be opened using the recipient's user profile. For more information, see the section *Configuring the Setup To Open the Alfabet Interface via Links in Email Notifications* in the reference manual *System Administration*.

The text displayed in automatically-generated email notifications is also configured in Alfabet Expand. Please note that you should not set the **Is HTML** attribute to `True` in the context of the text templates available for process changes. For an overview of all text templates available for business process changes, see *Text Templates for Organizational Changes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*. For more information, see the section Configuring Text Templates for Email Notifications in the reference manual *Configuring Alfabet with Alfabet Expand*.

To access the XML object ***BusinessProcessChanges***:

1) Go to the **Presentation** tab and expand the **XML Objects** folder and expand the **ObjectChanges** folder.
2) Right-click the XML object ***BusinessProcessChanges*** and select **Edit XML**. The XML editor is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) The table below describes the XML elements and XML attributes that can be edited for the XML object ***BusinessProcessChanges***:

XML Element (bold) / XML Attribute	Explanation
ObjectChangesDef	
`DefaultResponsible`	Enter the name of the default user that should receive assignments associated with the propagation of changes if the impacted object has no authorized user defined or if the relevant business process does not have a parent business process. The name should be written as follows: [USERNAME].

XML Element (bold) / XML Attribute	Explanation
	The text template `TextTemp_ObjectChangeDefaultResponsible` will be sent to the user specified in this attribute if no authorized user is defined for the object impacted by the change. For more information, see *Text Templates for Organizational Changes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.
`DefaultTargetSpanInDays`	Enter an integer to establish the target date when an assignment generated due to a business process change should be completed. The target date is computed based on the date when the assignment is generated plus the value defined in the XML attribute `DefaultTargetSpanInDays`.  If the XML attribute `DefaultTargetSpanInDays` is defined as 30, every assignment that is generated in the context of a business process change will have a target date that is 30 days after the date that the assignment was generated. An assignment generated on Jan. 1, 2008 will have a target date of Jan. 30, 2008.
`Profile`	Enter the name of the user profile required to access the Alfabet interface via the hyperlink displayed in the email notification that is generated for the assignment triggered by a process change.
`SendEmails`	Enter "`true`" if notification via email should be sent in addition to the assignment to the authorized user(s) of the dependent object informing them about the business process change. Enter "`false`" if no email notification should be sent.
`AllowObjectChangeOnUI`	Enter "`true`" if business process changes may be manually defined by users in the **Process Changes** page view. Enter "`false`" if users may not manually define business process changes. The respective functionalities will be disabled in the **Process Changes** page view and thus hidden in the user interface.
`ProcessExternalChanges`	Enter "`true`" if business process changes are defined via the input of data via an external application. Enter "`false`" if business process changes should not be triggered via the input of data from an external application. Depending on the value you define, only the relevant functionalities will be displayed in the **Process Changes** page view.
ExclusionRule	
`ClassNames`	Enter the values of the **Name** attribute of the object classes (for example, `Project`, `ICTObject`) for which you want to define an exclusion rule. Multiple classes should be separated by a comma. Enter the symbol * (asterisk)

XML Element (bold) / XML Attribute	Explanation
	if all classes relevant to business process changes are included. Class names that are misspelled are ignored.
ExclusionRuleDetail	Define criteria that determines which dependent objects should be ignored in the business process change functionality.
`PropertyName`	Enter the technical name of the public and/or custom object class property (for example, Status) that determines which dependent objects should be ignored in the business process change functionality. The object class property is relevant for all object classes listed in the XML attribute `ClassNames`.
`ExcludedValues`	Enter one or more values (for example, Discarded, Closed) relevant to the object class property that determine its exclusion.

4) Click the **Save** button to save the XML definition.

Chapter 15: Configuring the Visualization of the Alfabet Interface

The visualization of the Alfabet interface can be specified in a variety of ways to align with your corporate guidelines. This includes, for example, the font family and colors used in the interface, the inclusion of the company logo in the header, the icons used to visualize indicators in object cockpits, configured reports, custom editors, the images used to visualize enterprise milestones as well as object classes in the interface.

The following information is available:

Configuring GUI Scheme Definitions for the Alfabet Interface

You can align the Alfabet interface according to your corporate design guidelines by specifying GUI scheme definition. Multiple GUI schemes can be created thus ensuring the styling requirements for different entities in a federated enterprise as well providing user profiles that support barrier-free accessibility. The GUI schemes can be assigned to the relevant user profile via the **GUI Scheme** attribute available on the user profile. For more information about assigning the GUI scheme to a user profile, see the section Making Functionalities Accessible to a User Profile.

The ***AlfaGuiScheme*** definition allows you to specify the font type, colors used for various aspects of the interface, line widths for the focus as well as the logos to display in the header of the Alfabet interface. The ***AlfaGuiScheme*** definition is applied to all aspects of the Alfabet interface including any configured guide pages. For more information about the configuration of guide pages, see the reference manual *Designing Guide Pages for Alfabet*.

Please note the following regarding the configuration of GUI schemes:

- The GUI scheme labelled `Default Scheme` will be automatically implemented if no further custom GUI schemes are defined or no GUI scheme is assigned to a user profile. This GUI scheme labelled `Default Scheme` cannot be deleted.
- The protected GUI scheme `SoftwareAG_103Styles` is available for reference and specifies all standard settings in the Alfabet user interface. Customers wishing to align their GUI scheme with the new standard Software AG styling can simply copy this scheme to their respective (default) GUI scheme.

- You can return to the default settings predefined for a GUI scheme at any time by right-clicking the ***AlfaGuiScheme*** object and selecting **Reset to Default Settings**.
- Several attributes are especially important in order to specify the ways that the focus is visualized in the user interface or the font color is displayed. For more information about configuring user profiles for users using screen readers and keyboard shortcuts, see the section Configuring Barrier-Free User Profiles.
- For an explanation and example of each attribute that can be defined for a GUI scheme, see the section *Overview of GUI Scheme Attributes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

To create an ***AlfaGuiScheme***:

1) Click the **Presentation** tab and expand the **GUI Schemes** folder by clicking the + symbol.
2) Right-click the **GUI Schemes** folder and select **Add New GUI Scheme** . The ***AlfaGuiScheme*** is added below the **GUI Schemes** folder.
3) To ease the definition of the GUI scheme attributes, ensure that the attributes are structured according to their purpose by clicking the **Categorized** button in the upper left corner of the attribute window.
4) In the **Common** section of the attribute window, specify a name for the GUI scheme in the **Name** field.
5) In the **Application** section of the attribute window, specify general aspects of the interface. Please note the following:
   - The **Application Font** attribute specifies the font to be used. The `Roboto` font is not a standard font available via the web browser, and Therefore, must be uploaded from the Alfabet Web Application at runtime. The upload of runtime fonts is not permitted per default in the configuration of the Alfabet Web Application. Therefore, if the `Roboto` font should be used for the user interface, the use of runtime fonts must be explicitly enabled via the server alias. If the `Roboto` font is not explicitly enabled via the server alias, the fonts defined for the **Application Font Fallback 1** and **Application Font Fallback 2** attributes of the GUI scheme will be used in the Alfabet user interface. For more information about enabling the `Roboto` font via the server alias, see the section *Activating Roboto as the Standard Font for the Alfabet Interface* in the reference manual *System Administration*.
   - When you define the font type that should be used in the interface, the font size/weight will be automatically adjusted based on where the font is used in the interface. For example, the font size/weight will differ for menu items, explorer nodes, toolbar buttons, column captions, data in datasets, etc.
   - Please note that if a font name with spaces (for example, `Arial Narrow`) is configured in the ***AlfaGuiScheme*** object, the font type `Times` will be displayed per default if the Alfabet interface is rendered using the Mozilla® Firefox® browser. This is a limitation of the Firefox browser.
   - You must specify the rendering style of the interface controls in the custom editor. Two options are possible and the method you choose will determine how you go about designing the custom editor. You can choose the traditional rendering style, which reflects the explicit layout of all interface controls in the custom editor as designed by the solution designer, or the stack rendering style which automatically positions the visible interface controls in a one- or two-column linear list. For an overview of configuring the rendering styles of standard and

custom editors, see the section Specifying the Rendering Definition of the Custom Editor. In the **Application** section of the attribute window, expand the **Editor Rendering Options** attribute and define the following:

- **Rendering Style**: Select either **Traditional** to display standard and custom editors as the have been explicitly designed or select **Stack** to display standard and custom editors as a one- or two-columnar linear list. If you select **Stack**, define the attributes listed below.

  Please note that if you change the Rendering Style attribute, you must delete the value in the **Editor Rendering Options** field and click the **Save** button to update the attributes in the **Editor Rendering Options** section of the attribute window.

- **Stack Layout Type**: If **Stack** has been selected for the **Rendering Style** attribute: Select **One Column** if the captions and interface controls shall be displayed as a single left-aligned column in the editor. Select **Two Columns** if the captions and interface controls shall be displayed as two left-aligned columns in the editor. Please note that the sequence of the interface controls is determined by the **Tab Index** attribute defined for each interface control.
- **Preserve Layout in Group Boxes**: Select `True` if the position of interface controls in a Group Box interface control should follow a traditional rendering style and not be changed to have a stack rendering style. Select `False` if the interface controls in a Group Box interface control should have a stack rendering style.
- **Show Hint In-Line**: Select `True` if the help texts defined via the **Hint** attribute of the interface controls shall be displayed in the editor below the editor field. The hint icon specified in the **Interface Control Hint Icon** attribute will not be displayed if the **Show Hint In-Line** attribute is set to `True`. Select `False` if the help texts should not be displayed below the editor field.
- **Interface Control Hint Icon**: Displays the icon displayed to show the help texts defined for the **Hint** attribute of interface controls. It is recommended that this attribute is not changed. The hint icon will only be displayed if the **Show Hint In-Line** attribute is set to `False`.

- For an explanation of each attribute in the **Application** section of the attribute window, see the section *Attributes Displayed in the Grid Section: Application* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

6) In the **Application Header** section of the attribute window, specify the visualization of the main header in the Alfabet interface including the masthead, toolbar buttons, etc. For an explanation of each attribute in the **Application Header** section of the attribute window, see the section *Attributes Displayed in the Grid Section: Application Header* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

7) In the **Automated Assistant Positions** section of the attribute window, specify the size and location of configured automated assistants for various configuration objects. For an explanation of each attribute in the **Automated Assistant Positions** section of the attribute window, see the section *Attributes Displayed in the Grid Section: AlfaBot and Automated Assistant Positions* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*. For more information about configuring automated assistants, see the section Providing Custom Online Help to the User Community.

8) In the **Dataset Rows Highlighting** section of the attribute window, specify the visualization of datasets (page views with tables) in the Alfabet interface. For an explanation of each attribute in the **Dataset Rows Highlighting** section of the attribute window, see the section *Attributes Displayed in the Grid Section: Data Rows Highlighting* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

9) In the **Explorer Tree** section of the attribute window, specify the visualization of explorers. For an explanation of each attribute in the **Explorer Tree** section of the attribute window, see the section *Attributes Displayed in the Grid Section: Explorer Tree* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

10) In the **Floating Presentation Toolbar** section of the attribute window, specify the **Buttons Full Opacity** attribute. Select `True` to turn off the opacity for the floating menus. The default value is `False`.

For example, when set to `True`:

When set to `False`:

11) In the **Light-Weight Text Pop-Ups** section of the attribute window, specify the attributes to visualize the pop-up windows available for editor field help texts in editors. For an explanation of each attribute in the **Light-Weight Text Pop-Ups** section to of the attribute window, see the section *Attributes Displayed in the Grid Section: Light-Weight Text Pop-Ups* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

12) In the **Login Form** section of the attribute window, specify the attributes to visualize the login window. For an explanation of each attribute in the **Login Form** section of the attribute window, see the section *Attributes Displayed in the Grid Section: Login Form* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

13) In the **Logos** section of the attribute window, define the following, as needed:

- **Custom Logo**: Click the **Browse** button and select the relevant file type and file that you want to upload and click **OK**. It is recommended that only icons in PNG format are imported to Alfabet. The logo you plan to display in Alfabet may not be larger 36 x 400 pixels.
- **Custom Logo Location**: Specify the location of the header where the custom logo should be displayed.
- **Custom Logo Tooltip**: Enter a short sentence for the tooltip when the user points to the custom logo.
- **Custom Logo URL**: Enter a URL that shall open in the browser when the user clicks the custom logo.
- **Product Logo Image**: Select one of the predefined visualizations of the Alfabet product logo.

- **Product Logo Tooltip**: Enter a short sentence for the tooltip when the user points to the Alfabet product logo.
- **Product Logo URL**: Enter a URL that shall open in the browser when the user clicks the Alfabet product logo.

14) In the **Slide-In Toolbar** section of the attribute window, specify the visualization of the slide-in toolbar available on the right edge of the user interface which provides various forms of user assistance such as AlfaBot user assistance, automated assistants, and quality widgets. For an explanation of each attribute in the **Main Slide-In Bar** section to of the attribute window, see the section *Attributes Displayed in the Grid Section: Slide-In Toolbar* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

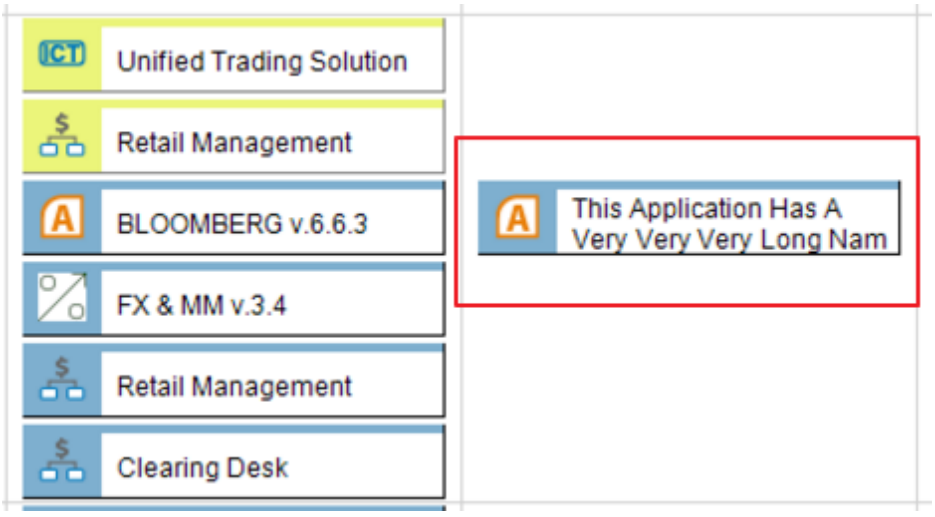
15) In the **Message Boxes** section of the attribute window, specify the attributes to visualize the error, info, question, and warning messages that may open in Alfabet. For an explanation of each attribute in the **Message Boxes** section of the attribute window, see the section *Attributes Displayed in the Grid Section: Message Boxes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

16) In the **Pop-Up Window** section of the attribute window, specify the colors used in the header of pop-up windows used to display wizards, for example. For an explanation of each attribute in the **Application Header** section of the attribute window, see the section *Attributes Displayed in the Grid Section: Popup Window* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

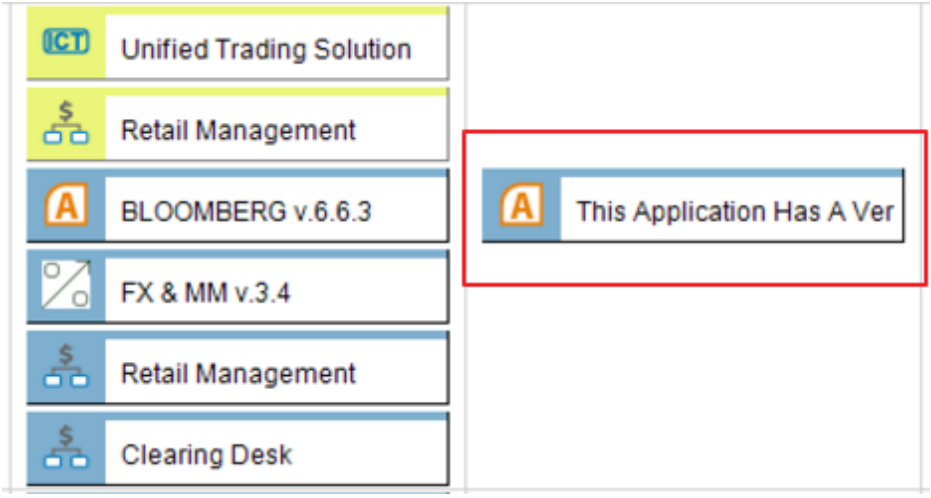
17) In the **Presentation Gantt** section of the attribute window, define the **Maximal Width of Gantt Dataset** attribute. Specify the maximum size in pixel of Gantt reports. This is may be particularly relevant for Gantt reports embedded in object cockpits or console reports. The value is applied if one of the rows in the dataset requires more space than what is defined. In this case, a scroll bar will be added to the dataset (potentially in addition to the scroll bar for the timeline section of the Gantt chart). It is recommended that the value is either -1 or no smaller than 300. The default value for Max Width Gantt Dataset is -1.

18) In the **Presentation Matrix** section of the attribute window, define the **Allow Multiline Matrix Item Caption** attribute. Select `True` if matrix captions should be distributed with a linebreak over two lines or select `False` if matrix captions should be truncated if longer than one line.

For example, when set to `True`:



When set to `False`:



19) In the **Secondary Window Skins** section of the attribute window, specify the visualization of the various forms of user assistance available in the slide-in toolbar displayed on the right edge of the user interface. These attributes allow you to specify the button icon, notch color, and colors of the pop-up windows used for AlfaBot user assistance, automated assistants, and quality widgets. For an explanation of the attributes available to specify the visualization of AlfaBot user assistance, automated assistants, and quality widgets in the **Secondary Window Skins** section of the attribute window, see the section *Attributes Displayed in the Grid Section: Secondary Window Skins* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

20) In the **View Tabs** section of the attribute window, specify the visualization of tabs in editors, wizards, and guide views with tabbed pages. For an explanation of each attribute in the **View Tabs** section of the attribute window, see the section *Attributes Displayed in the Grid Section: View Tabs* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

21) In the **Login Form** section of the attribute window, wxpand the **Usage Terms Acknowledgement Options** section to configure a login acknowledgement screen. If configured, users will be mandated to click the acknowledgement button before the actual content of the Alfabet application is shown. The acknowledgement timestamp will be logged in the database table `ALFA_USERLOGIN_DETAILS`. Define the following attributes:

- **Button Text**: Specify a text to be displayed for the acknowledgement button.
- **URL**: Specify a URL that opens automatically when the user has been authenticated successfully. Enter a valid URL starting with the prefix `https://www.` or `http://www.` The URL link may contain up to 511 characters.

22) In the toolbar, click the **Save** button to save the XML definition.

23) To assign the GUI scheme to the relevant user profile, go to the **Admin** tab, click the relevant user profile and select the GUI scheme in the **GUI Scheme** attribute. For more information about assigning the GUI scheme to a user profile, see the section Making Functionalities Accessible to a User Profile.

24) In the toolbar, click the **Save** button to save the user profile definition.

Specifying a Company Logo in the Alfabet Interface

You may specify a custom logo to display in the header of the Alfabet interface. You can also determine where the logo should be displayed in the header, a tooltip for the logo and a URL that shall open in the browser when the user clicks the custom logo. It is recommended that only icons in PNG format are imported to Alfabet. The logo you plan to display in Alfabet may not be larger 36 x 400 pixels.

Furthermore, you can select which predefined visualization of the Alfabet product logo should be displayed, where it should be displayed, and a URL that shall open in the browser when the user clicks the Alfabet product logo.

1) Click the **Presentation** tab in the tab bar and expand the **GUI Schemes** folder by clicking the + symbol.

2) Right-click the **GUI Schemes** folder and select either the `Default Scheme` or a custom GUI scheme created by your enterprise.

3) To ease the definition of the GUI scheme attributes, ensure that the attributes are structured according to their purpose by clicking the **Categorized** button in the upper left corner of the attribute window.

4) In the **Logos** section of the attribute window, define the following, as needed:

   - **Custom Logo**: Click the **Browse** button and select the relevant file type and file that you want to upload and click **OK**. It is recommended that only icons in PNG format are imported to Alfabet. The logo you plan to display in Alfabet may not be larger 36 x 400 pixels.

     The following special characters <,>,*,%,&,:,\,? may not be used in the names of icons available in Alfabet.

   - **Custom Logo Location**: Specify the location of the header where the custom logo should be displayed.
   - **Custom Logo Tooltip**: Enter a short sentence for the tooltip when the user points to the custom logo.
   - **Custom Logo URL**: Enter a URL that shall open in the browser when the user clicks the custom logo.
   - **Product Logo Location**: Select one of the predefined visualizations of the Alfabet product logo.
   - **Product Logo Tooltip**: Enter a short sentence for the tooltip when the user points to the Alfabet product logo.
   - **Product Logo URL**: Enter a URL that shall open in the browser when the user clicks the Alfabet product logo.

5) In the toolbar, click the **Save** button to save the XML definition.

6) To assign the GUI scheme to the relevant user profile, go to the **Admin** tab, click the relevant user profile and select the GUI scheme in the **GUI Scheme** attribute.

7) In the toolbar, click the **Save** button to save the user profile definition.

Adding and Maintaining Icons for the Alfabet Interface

A standard set of icons is provided with Alfabet Expand for use in the Alfabet interface. Software AG provides a number of icons stored in the **Icon Gallery** folder in Alfabet Expand that can be implemented in the solution configuration. These icons are bundled in icon groups. For example, the icon group Color Balls consists of the icons Green, Orange, Red that could be implemented to represent traffic light signals for indicators.

Customized icons may be displayed in a number of contexts in Alfabet. All icons that are to be implemented in Alfabet must first be added to the **Icon Gallery** folder in Alfabet Expand. Custom icons of varying sizes of varying sizes can be imported to Alfabet Expand for use in the Alfabet interface.

Please note the following:

- It is recommended that only icons in PNG format are imported to Alfabet with the exception of images used in guide view. In this case, any file format supported by the browser used to render Alfabet may be uploaded to the **Original Images** library:
- .NET technology requires that custom icons have a transparent background. The color of the bottom left pixel will be rendered transparent in the graphic. To ensure that the graphic can be properly rendered, you must ensure that the bottom left pixel of the graphic displays a color that is NOT used in the custom icon.
- The following special characters <,>,*,%,&,:,\,? may not be used in the names of icons available in Alfabet.
- A preconfigured set of icons are used to represent object classes in Alfabet. We recommend that you do not implement an icon that is already being used to represent an object class or relationship in the Alfabet solution. For example, it is recommended that you do not add the icon to an icon gallery because it represents an application in the Alfabet interface.
- The standard buttons **Delete**, **Detach**, **Edit**, **Export**, **Navigate**, and **Show** displayed in the toolbars in Alfabet views cannot be replaced with custom icons.
- The icon gallery `ENT_Milestones` is a protected icon gallery You may only add icons to or remove icons from this icon gallery. Attributes for the icon gallery cannot be modified.

The table below provides an overview of the various contexts in which custom icons may be used and where they are configured.

Custom Icon Can Be Implemented For:	Where Configured;	For More Information:
Company logo	**Custom Logo** attribute of the GUI scheme	Specifying a Company Logo in the Alfabet Interface
Evaluation types and indicator types	**Evaluations and Portfolios** functionality in Alfabet	*Configuring Evaluations, Prioritization Schemes, and Portfolios* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*
Specified objects for preconfigured set of object classes. The custom ions will be displayed in diagrams, matrices, object profiles/object cockpits, previews, and standard and custom explorers.	**Icon** field of the editor for the relevant object class For example, `Application`, `Component`, `Domain`, `LocalComponent`, `ICTObject`, `Device`, `Location`, `OrgaUnit`, `SystemBuildingBlock`, and `StandardPlatform`.	See the relevant context-sensitive editor help.
Custom editors and object cockpits	Icon interface control in custom editors and object cockpits	Adding Icons to the Custom Editor and Adding Icons to the Object Cockpit
Configured reports	`PictureAssignment` instruction in queries	Using Instructions to Format the Results of an Alfabet or Native SQL Query
Class icon for an object class/object class stereotype. The custom ions will be displayed in diagrams, matrices, object profiles/object cockpits, previews, and standard and custom explorers.	**Icon** attribute of the class setting	Specifying the Icons Implemented for Object Classes and Object Class Stereotypes
Custom diagram item templates for diagrams	**Icon** attribute of the custom diagram item template	Configuring Custom Diagram Item Templates
Object classes displayed in views (such as matrices) and	XML object ***GeneralViewMap***	Specifying Icons for Object Classes in Presentation Objects and the

Custom Icon Can Be Implemented For:	Where Configured;	For More Information:
in the Alfabet Diagram Designer		Alfabet Diagram Designer
Guide View Picture interface controls in guide views	Guide Pages Designer ; Please note that the **Original Images** library is available only for images used in guide views. Any file format supported by the browser used to render Alfabet may be uploaded to the **Original Images** library: BMP files will be automatically converted to PNG since most browsers do not support the BMP format.	*Adding Guide View Pictures to the Guide View* in the reference manual *Designing Guide Pages for Alfabet*.
Content voting capability	XML object ***ObjectAssocationsConfig***	Configuring the Content Voting Capability via Object Associations

The following information is available:

- Uploading Custom Icons to the Icon Gallery
- Creating a New Icon Group and Adding Icons to the Icon Group
- Deleting an Icon or Icon Group

Uploading Custom Icons to the Icon Gallery

You can upload icons that are relevant for your company to Alfabet Expand, making them available for use in the solution interface. Custom icons can be structured in an icon group, if needed. Once an icon has been uploaded, it may be implemented in the solution interface to represent qualitative or performance information (such as indicators and evaluations) as well as associated with functionalities. It is recommended that only icons in PNG format are imported to Alfabet.

Icons of varying sizes can be uploaded to the **Icon Gallery** in Alfabet Expand. In addition to the library which contains the standard Alfabet icons that are 22x22 pixel in size, a **Large Icon** library allows you to add icons that are 30x30 pixel in size, and a **Free Icon** library allows you to add icons of an arbitrary size. The **Free Icon** library in particular allows icons of various sizes to be added to object cockpits, for example. The **Original Images** library is available only for images used in guide views. Any file format supported by the browser used to render Alfabet may be uploaded to the **Original Images** library:

The following special characters <,>,*,%,&,:,\,? may not be used in the names of icons available in Alfabet.

To upload custom icons to Alfabet Expand:

1) Go to the **Presentations** tab and right-click the **Icon Gallery**  folder and select one of the following libraries, depending on the icons you want to upload to Alfabet Expand:

- **Edit Icons (22x22)** to add PNG files with images that are 22x22 pixel in size
- **Edit Large Icons (30x30)** to add PNG files with images that are 30x30 pixel in size
- **Edit Free Icons (any size)** to add PNG files with images that are of an arbitrary size
- **Edit Original Images (any size)** to add any file format with an arbitrary size. Please note that the **Original Images** library is available only for images used in guide views. Any file format supported by the browser used to render Alfabet may be uploaded to the **Original Images** library: BMP files will be automatically converted to PNG since most browsers do not support the BMP format.

2) You will see the **Icons** editor in the center pane displaying any previously upload custom icons. In the toolbar, select **Icon Manager** > **Add New Icons**.
3) In the window that opens, select the file type and icon that you want to add and click **OK**. The icon is automatically displayed in the **Icon Manager** in the center pane of Alfabet Expand.
4) In the **Name** attribute, edit the name as needed. The following special characters <,>,*,%,&,:,\,? may not be used in the names of icons available in Alfabet.
5) Click the **Save** button in the toolbar to save your changes.

Creating a New Icon Group and Adding Icons to the Icon Group

An icon group allows custom and standard icons to be bundled and made available in an icon gallery.

You can create a new icon group and assign icons that you have added to Alfabet Expand as well as icons that exist in the predefined icon gallery provided by Software AG. To create an icon group in order to structure icons:

1) Go to the **Presentations** tab, and right-click the **Icon Gallery** node and select **New Icon Group**. A new icon group is displayed below the folder in the explorer.
2) Click the icon group to open the attribute window.
3) In the attribute window, enter a technical name for the icon group in the **Name** field. This is the icon group name that is visible in Alfabet Expand. In the **Caption** field, enter a caption for the icon group. This is the icon group name that users will see in Alfabet.
4) To add icons to the icon group, right-click the new icon group and select **Create Icon Definition**.
5) Expand the icon group to display the new icon definition and click the new icon definition.
6) Define the following in the attribute window:
   - **Name**: Enter a name for the icon
   - **Caption**: Specify a caption for the icon. This may be useful in the content voting capability, for example. For more information, see the section Configuring the Content Voting Capability via Object Associations.
   - **Icon**: Select the icon that you want assign to the icon definition.
7) Click the **Save** button in the toolbar to save your changes.

Deleting an Icon or Icon Group

You can delete an icon or icon group. To do so, right-click the icon or icon group and select Delete. If you delete an icon, the icon will be irrevocably deleted from the Alfabet database. If you delete an icon group, the icon group including all icons will be irrevocably deleted from the Alfabet database.

Specifying the Icons Implemented for Object Classes and Object Class Stereotypes

Any image that is to be configured as an icon for an object class or object class stereotype must first be uploaded to the icon gallery as described in the section Adding and Maintaining Icons for the Alfabet Interface. If custom icons have been defined for an object in the Alfabet interface, the custom icon defined for the object will have precedence over the icon configured for the object class/object class stereotype.

If you have configured a custom class setting for an object class or object class stereotype, you can replace the standard icon used for the object class in explorers, diagrams, matrices, etc. by defining the **Icon** attribute for that custom class setting. Please note that the images must first be uploaded to the icon gallery as described in the section Adding and Maintaining Icons for the Alfabet Interface. For general information about the configuration of class settings for a user profile, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes.

To change the icon for a custom class setting:

1) Go to the **Presentation** tab and expand the **Class Settings** node.
2) In the **Class Settings** folder, navigate to the object class/object class stereotype folder that you want to define class settings for and expand the folder.
3) Right-click the public class setting template  that you want to specify for the relevant user profile.
4) In the **Icon** field, select the image that should replace the standard object class icon in the Alfabet interface.
5) In the toolbar, click the **Save**  button to save the class setting definition

Specifying Icons for Object Classes in Presentation Objects and the Alfabet Diagram Designer

It may be that a custom icon is specified for an object class via the class setting. In this case, the custom icon will be displayed in the object profile and explorer tree. However, if the class icon should be displayed in presentation objects such as matrix views (For example, the *Business Support Map Page View* or *Platform Architecture Page View*), you must explicitly define the replacement of the default standard class icon with the custom icon in the XML object ***GeneralViewMap***. The icon you specify in the XML object ***GeneralViewMap*** will replace the standard icon used for the object class in matrices and the **Toolbox Items** pane in the Alfabet Diagram Designer.

Please note that the images must first be uploaded to the icon gallery as described in the section Adding and Maintaining Icons for the Alfabet Interface.

An icon cannot be configured for an object class stereotype in the XML object ***GeneralViewMap***.

For more information about configuration possibilities for the XML object ***GeneralViewMap***, see the section Configuring the Alfabet User Interface to Map Standard Configuration Objects to Custom Configuration Objects in the chapter Executing Administrative Tasks in Alfabet Expand.

To change the default class icon for presentation objects:

1) Click the **Presentation** tab in the tab bar and expand the **XML Objects** folder by clicking the + symbol.
2) Right-click the XML object ***GeneralViewMap*** and select **Edit XML**. The **XML Editor** is displayed in the center pane.
3) Create an XML element `<MapEntry Type="Picture"/>` for each standard icon that you want to replace with a custom icon.
4) For each XML element `<MapEntry Type="Picture"/>`, enter the name of the standard Alfabet icon to be replaced in the `Source` XML attribute and enter the name of the custom icon to replace the standard Alfabet icon in the `Target` XML attribute. In order to enter the correct name of the standard Alfabet icon that you want to replace, see the class settings of the respective object class.

```
<AlfaViewMap Name="Map_WithExcludedElements">
    <MapEntry Type="Picture" Source="ICTObject"
    Target="Custom_MyICTObject"/>
    <MapEntry Type="Picture" Source="ICTObject_sbs"
    Target="Custom_MyStratBS"/>
    <MapEntry Type="Picture" Source="Application"
    Target="Custom_MyApplication"/>
</AlfaViewMap>
```

5) In the toolbar, click the **Save** button to save the XML definition.

Chapter 16: Configuring Events

Event management provides a means to automatically execute one of the following automatically as a consequence of a defined user action on the Alfabet user interface:

- Execution of an ADIF import or export job. The ADIF import or export can be executed either on the same or on a different Alfabet database.
- Start of workflows. A workflow can be started either on the same or on a different Alfabet database.
- Generation of a publication.
- Generation of questionnaire indicators for a defined object for an already running questionnaire.
- Score computation for questionnaire indicators.
- Sending of RESTful service calls to any RESTful service. The RESTful service call can target any third-party component or the Alfabet RESTful API.

The following user activities can be specified to trigger events:

- When a wizard step is entered, exited, or cancelled.
- When a workflow step is entered, refused, expired, or exited.
- When a rating or a contact request is submitted via the Feedback Bot.
- When an event template configured to send a RESTful service call is finished.

For the execution of an ADIF import or export job or the start of a workflow, execution of the event is performed via a RESTful service call to the RESTful API of the Alfabet Web Application. The service call triggers the ADIF scheme execution or starts the workflow as defined in the event. For events that send a RESTful service call to a third-party RESTful API, the defined service call is send on triggering the event.

There are two methods for executing the event:

- **Asynchronous execution**

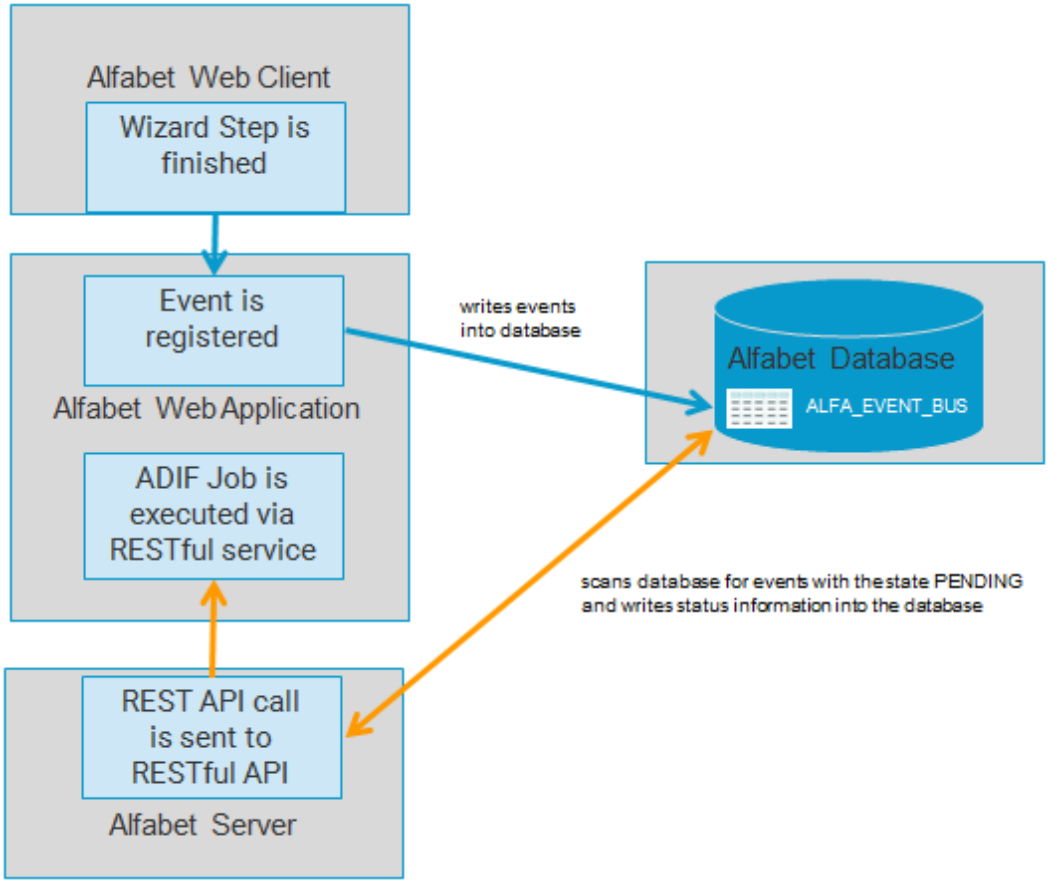
Asynchronous execution is available for all kinds of event.

The RESTful service call for the event is run in the background by an Alfabet Server connecting to the same Alfabet database. If a user triggers an event, the event will be registered. This means that the event is written to the database table of the object class `ALFA_EVENT_BUS` with the property `STATE` set to `PENDING`.

The Alfabet Server scans the database table of the object class `ALFA_EVENT_BUS` in regular intervals for events with the property `STATE` set to `PENDING`. Registered events are executed asynchronously in the order of registration. If multiple events are triggered at the same time, the events might be executed with a delay.

If queries are defined for the event with parameter settings (such as the referencing of a BASE object), the parameters will be substituted with the current values when the event is registered. The query is then executed with these settings when the event is executed.

The following figure shows an overview of the involved Alfabet components and event execution steps for the example of an asynchronous ADIF scheme execution based on an event triggered by a user leaving a wizard step:



- **Synchronous Execution**

Synchronous execution is available for events with the exception of events triggering execution of an ADIF export or an ADIF import.

Synchronously executed events are triggering immediate execution of the REST API call. The event is nevertheless written into the `ALFA_EVENT_BUS`, but with the `STATE` set to `FINISHED` or, if the execution of the event was not successful, to `ERROR`. This ensures that there is no gap between the triggering of the event and For example, the start of the workflow. The Alfabet Server is not required to execute synchronous events.

The following information is available:

- Configuring an Event Triggering Start of a Workflow, ADIF Execution, Publication, or Questionnaire
  - Related Functionalities
    - Configuring an ADIF Job or a Workflow Template To Be Executed via the Event
    - Enabling Execution of RESTful Service Calls
    - Configuring a User to Execute REST API Calls for Events
    - Defining the Event Template
      - Defining an Event Template to Start a Workflow

Configuring an Event Triggering Start of a Workflow, ADIF Execution, Publication, or Questionnaire Related Functionalities

A number of configuration objects must be configured for the event. The central configuration object is an event template that contains references to all components required for event execution. All other involved components are also used in other contexts.

Configuration depends on the connection type of event. There are two connection types of event:

- `SelfReflective`: The event is executed on the same Alfabet database it is triggered from. The REST API call is then directed to the same Alfabet Web Application and executed with a user configured to be used for self-reflective event executions.

- `Query`: The event is targeting either the same or another Alfabet database via an **Alfabet Database Connection** that specifies the connection to the RESTful service interface of a running Alfabet Web Application.

A complete event includes the following configurations depending on the connection type of event:

- If ADIF shall be activated or a workflow shall be started, the ADIF scheme or workflow template must be configured to be executable via the event.
- The RESTful services of the Alfabet Web Application must be activated.
- A call to the RESTful API of the Alfabet Web Application must be specified. The specification depends on the connection type of event:
  - For events of the connection type `Query`, the configuration is done in two steps:
    - First, the parameters for the service call are defined in the XML object `AlfabetIntegrationConfig`.
    - In a second step, an object of the object class **Alfabet Database Connection** must be defined that references the connection. The Alfabet database connection is an object in the Alfabet database, which means that it is subject to access permission concepts and that it can be found via a query.
  - For events of the connection type `SelfReflective`, a user must be configured for execution of the event.
- An event template must be defined that couples the information about which action shall be executed with the information about the target of the REST API call triggering the event.
- A workflow step action, a wizard step action or an ADIF scheme must be defined for the wizard step or workflow step which shall trigger the event.

The following figure shows the way an event is triggering the execution of an ADIF job when a user leaves a wizard step and the configurations that are required to trigger the event.

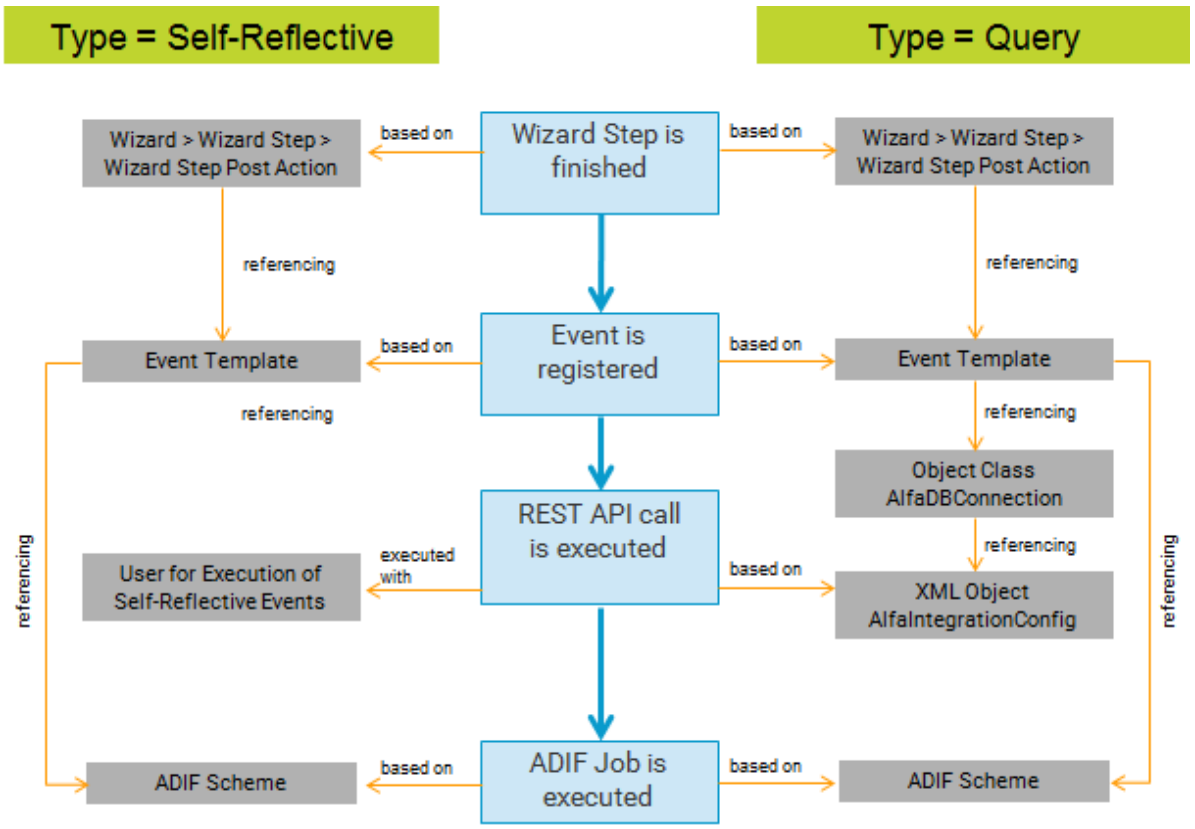


FIGURE: Execution of an ADIF Job triggered via an event and the necessary configuration

The RESTful service call is executed with the access permissions of the user defined to execute self-reflective events or the user specified in the XML Object ***AlfaIntegrationConfig***. Access permissions for the RESTful service call are Therefore, independent from the user triggering the event. Nevertheless, the user triggering the event as well as the user profile that the user is logged in with when triggering the event is stored as the executing user and user profile for the execution of the functionality triggered by the event. This mechanism enables asynchronous response for events and also ensures that workflows started for an existing object can be executed without access permission issues. This means that the user operating the Alfabet user interface and thereby triggering an event that in turn creates a workflow will be captured as the owner of the workflow.

The following information is available:

- Configuring an ADIF Job or a Workflow Template To Be Executed via the Event
- Enabling Execution of RESTful Service Calls
- Configuring a User to Execute REST API Calls for Events
- Defining the Event Template
  - Defining an Event Template to Start a Workflow
  - Defining an Event Template for the Execution of an ADIF Import Scheme
  - Defining an Event Template for the Execution of an ADIF Export Scheme
  - Defining an Event Template for Adding Questionnaire Indicators for an Object to a Questionnaire

Configuring an ADIF Job or a Workflow Template To Be Executed via the Event

An event can trigger the execution of any valid ADIF import scheme, ADIF export scheme or workflow template.

Please note the following preconditions for the configuration of ADIF schemes if they shall be executed via RESTful service calls:

- ADIF schemes that involve files located on the local network, either by exporting to files or importing from files cannot be executed via events. An exception for this rule is the execution of ADIF import events on completion of events triggering RESTful service calls. For more information about this exception, see the section Configuring an Event Template Sending a REST API Call to Trigger Another Event.

In addition to the configuration required for the ADIF scheme or workflow execution, the ADIF scheme or workflow template must be activated for execution via the REST API with the following setting:

1) In Alfabet Expand, click on the ADIF scheme or workflow template in the explorer.
2) Set the attribute **Applicable for REST API** to `True`.
3) In the toolbar, click the **Save** button.

Enabling Execution of RESTful Service Calls

The Alfabet database for which the ADIF scheme is executed or the workflow is started must be configured to allow access to the Alfabet RESTful services. This Alfabet database might be the same database as the one configured to trigger the event, but the event can also be executed on another Alfabet database. The required configuration is described below in a short overview. For a detailed description of the configuration required for the RESTful API, see the reference manual *Alfabet RESTful API*.

- A license for the Alfabet Data Integration Framework (ADIF) must be active.
- Configure the `web.config` file of the Alfabet Web Application to enable Alfabet RESTful services.
- Configure the Alfabet Web Application on the Web Server to enable access to the Alfabet RESTful services.
- Configure the server alias of the Alfabet Web Application to enable access to the Alfabet RESTful services.

Configuring a User to Execute REST API Calls for Events

It is recommended to create a new user that is exclusively used for triggering events via the RESTful services. The user can be excluded from access to any objects except for triggering workflows and/or ADIF jobs via RESTful service calls, but it is also possible to use an existing user. The user must be a named user with at least one user profile assigned. The user profile is not used for evaluation of access permissions. A ReadOnly user profile is sufficient to execute the RESTful services in the context of events.

The same user can be used for events of the type `Query` and `SelfReflective`. Please note however, that the user must be configured in the database that is the target of the REST API call. If the event is triggering

execution of an ADIF scheme or a workflow template in another Alfabet database, the user must be defined in the target database the REST API call is sent to by the event.

Only one user can be selected for execution of events of the type `SelfReflective`. If you assign this functionality to a user while another user has already been selected to execute self-reflective events, the setting is removed from that user when it is set for the user you are currently assigning it to.

The selected user will also be used for execution of events in the scope of the **Job Schedule** functionality.

To create a user for execution of events in the **User Administration** functionality on the Alfabet user interface:

The same functionality is also available via the Alfabet Administrator. For information about accessing the user management functionalities of the Alfabet Administrator see *Functionalities Available via the Expanded Explorer of the Connected Alias* in the reference manual *System Administration*.

1) In the toolbar of the **User Administration** view, select **New** > **Create New User**. An editor opens.
2) In the editor, define the following:

   Basic Data tab:

   - **Name**: Enter a meaningful name for the user. The user is a technical user. You can either assign the name of an existing person to it or give it a name that indicates that this is a virtual person defined to execute a functionality.
   - **User Name**: Enter a user name. The user name is used by the RESTful services to identify the user.
   - **Type**: Select `NamedUser`.

   API Permissions tab:

   - **Has API V2 Access**: Select the checkbox.
   - **API Access Options**: Make sure that the permissions **Has ADIFAPIInvocation Access** for the execution of ADIF schemes and/or **Has WorkflowADIFInvocation Access** for the execution of workflows are selected.
   - **Generate API Password**: Click the button. The API Password field is filled. For events of the type `Query`, copy the **API User Name** and the **API Password** . These have to be entered into the specification in the XML object `AlfabetIntegrationConfig` described below in the section Configuring the Connection Parameters in the XML Object AlfabetIntegrationConfig.
3) Click **OK** to close the editor. The new user is added to the table in the **User Administration** view.
4) If the user shall be used for execution of events of the type `SelfReflective`, select the user in the table and select **Action** > **Set as Executes Self-Reflective Events User** in the toolbar.
5) Select the user in the table and click the **Navigate** button.
6) In the object profile of the user, click **Assigned User Profiles**.
7) In the toolbar of the **Assigned User Profiles** view, click **New** > **Assign User Profiles**.

8) In the selector, select a user profile and click **OK** to assign it to the user. For a user that is exclusively used for events, it is recommended to use a ReadOnly user profile.

The configuration described above is the configuration required for the event management functionality only. In addition, you can set any other properties of the user. For security reasons you might also consider assigning a login password to the user although the user password is not required for the event management functionality. For information about the available configuration options for users, see *Defining and Managing Users* in the reference manual *User and Solution Administration*.

Configuring the Connection for Execution of an Event of the Connection Type Query

The following configuration is only applicable for events of the connection type `Query`.

The connection to the RESTful API of the Alfabet Web Application is done in two configuration steps:

- Configuring the Connection Parameters in the XML Object AlfabetIntegrationConfig
- Configuring the Alfabet Database Connection for Execution of the Event

Configuring the Connection Parameters in the XML Object AlfabetIntegrationConfig

The Alfabet Web Application acts as RESTful client that connects to the RESTful API of either itself or another Alfabet Web Application. The connection parameters must be configured in the XML object ***AlfabetIntegrationConfig*** in Alfabet Expand:

1) Go the **Presentation** tab of Alfabet Expand and expand the explorer node **XML Objects** > **Integration Solutions**.
2) Right click the XML object `AlfabetIntegrationConfig` and select **Edit XML** from the context menu. The XML object opens in the middle pane.
3) In the attribute window, edit the XML as described below.
4) Click the **Save**  button to save your changes.

The XML object must contain the following XML structure:

```
<AlfabetIntegrationConfig>
   <Connection
      name = "My Test Connection"
      service="http://localhost/ALFABET"
      active="true"
      user="DAME"
      psw="H7GLVUGZWQETKJFSX7HY6OK2CUB4WRGK"
      profile="MASTER"/>
</AlfabetIntegrationConfig>
```

The following XML elements and their attributes are part of the specification:

Element (Bold) / Attribute	Configuration Requirements
AlfabetIntegrationConfig	Root node of the configuration.
Connection	The configuration parameters for sending the RESTful service call and request the data from the source database. This XML element can be added multiple times to define multiple source databases.
`name`	Enter a unique name for the connection configuration. The name is used to identify the connection configuration For example, in editors on the Alfabet user interface. This attribute is mandatory.
`service`	Enter the URL of the Alfabet Web Application for access to the source database. This attribute is mandatory.
`active`	This attribute can be defined to deactivate (`false`) or activate (`true`) the connection. If the attribute is set to `false`, the connection will not be displayed in the drop-down list for the connection specification in the editor **Alfabet Database Connection** on the Alfabet user interface. The default value is `true`.
`user`	Enter the user name of the user configured to have access to the RESTful services of the Alfabet Web Application connected to the target database the ADIF scheme is executed on or the workflow is started for. For information about the required configuration of the user, see Configuring a User to Execute REST API Calls for Events. This attribute is mandatory.
`psw`	Enter the REST API password of the user defined in the `user` attribute. For more information about the configuration of the user, see the reference manual *Alfabet RESTful API*. This attribute is mandatory.
`profile`	Enter the user profile configured in the source database to grant the correct access permissions to the relevant object classes.

Configuring the Alfabet Database Connection for Execution of the Event

Configuration is done in the **Integration Solutions Configuration** functionality on the Alfabet user interface.

1) Go to the **Integration Solutions Configuration** functionality and click the **Alfabet Database Connection** node in the **Integration Solutions Configuration** explorer.

2) In the view, click **New** > **Create Alfabet Database Connection**.

> If you have already defined a similar connection and want to take over the settings of that connection for your new connection, you can alternatively click **New** > **Create Alfabet Connection as Copy** and select the existing connection the new connection should base on from the selector that opens. The editor for the new connection will then open with all settings identical to the copied connection and the name set to "Copy of <base connection name>".

3) In the **Alfabet Database Connection** editor, define the following fields as needed:

Basic Data tab:

- **ID**: Alfabet assigns a unique identification number to each Alfabet database connection. This number cannot be edited.
- **Name**: Enter a unique name for the Alfabet database connection.
- **Release Status**: Select the Alfabet database connection's current release status.

  The set of release statuses available for an object class are configured by your solution designer in the configuration tool Alfabet Expand. For more information, see the section Configuring Release Status Definitions for Object Classes in the reference manual *Configuring Alfabet with Alfabet Expand*. For general information about release statuses, see the section *Understanding Release Statuses* in the reference manual *Getting Started with Alfabet*.

- **Description**: Enter a meaningful description that will clarify the purpose of the Alfabet database connection.

Authorized Access tab:

- **Authorized User**: Click the **Search** icon to assign an authorized user to the selected Alfabet database connection. The authorized user will have Read/Write access permissions for the object and is responsible for the maintenance of the object.
- **Authorized User Groups**: Select one or more checkboxes to assign Read/Write access permissions to all users in the selected user group(s).

Connection tab:

- **Alfabet Connection**: Select the connection to the relevant Alfabet database that is configured in the XML element ***Connection*** in the XML object ***AlfabetIntegrationConfig*** in Alfabet Expand.
- **Allowed Classes**: This field is for implementation of the **Import Data Search** functionality only. No settings are required for event management.

4) Click **Test Alfabet Database Connection**. If your settings are correct, a message "The connection is valid" is displayed. Otherwise, an error message is displayed.

Defining the Event Template

An event template links to both the Alfabet database connection for execution of the event and to the ADIF scheme or workflow template to be executed. For each type of event a different type of event template must be defined:

- Defining an Event Template to Start a Workflow
- Defining an Event Template for the Execution of an ADIF Import Scheme
- Defining an Event Template for the Execution of an ADIF Export Scheme
- Defining an Event Template for Adding Questionnaire Indicators for an Object to a Questionnaire

Defining an Event Template to Start a Workflow

Event templates are defined in the **Reusable Elements** tab of Alfabet Expand:

1) Go to the **Reusable Elements** tab of Alfabet Expand.
2) Right click the node **Events** and select **Create New Workflow Event Template** from the context menu. The new workflow event template is added on the root level of the event management tree.
3) Click on the new event template and edit the following attributes:
   - **Name**: Enter a unique name for the event template. The name is used to identify the event templates in configurations.
   - **Description**: Optionally enter a short text describing the functionality of the event. This information is only visible in the event template attributes in the **Reusable Elements** tab to inform, For example, other solution designers about the functionality implemented with the event template.
   - **Workflow Template Name**: Select the workflow template from the drop-down list, to start a workflow based on that template via the event. The drop-down list displays all workflow templates that have the attribute **Applicable for REST API** set to `True` and the attribute **State** set to `Active`.
   - **Connection Type**: Select `SelfReflective`, if the event shall be executed for the same Alfabet database it is triggered from, or `Query`, if the event shall be executed on an Alfabet database that is defined via an **Alfabet Database Connection**.
   - **Connection via Query** / **Connection via Query as Text**: This attribute is only applicable and visible if the **Connection Type** is `Query`. Define a query that returns the `REFSTR` of the Alfabet database connection to be used for execution of the event. The query must return a single `REFSTR`. It can either be defined as native SQL query or Alfabet query:
     - Use the **Connection via Query** attribute to define an Alfabet query. Show properties are not required for the query.

```
ALFABET_QUERY_500
FIND Alfabet_DBConnection
WHERE Alfabet_DBConnection.Name = 'EventConnection'
```


- Use the **Connection via Query as Text** attribute to define a native SQL query with the `REFSTR` of the Alfabet database connection as the only `SELECT` argument.

```
SELECT REFSTR
FROM ALFABET_DBCONNECTION
WHEREALFABET_DBCONNECTION.NAME = 'EventConnection'
```

- **Execute Immediately**: Setting this attribute to `True` will cause the Alfabet Web Application to execute the event immediately by triggering the REST API call starting the workflow. The event is nevertheless written to the `ALFA_EVENT_BUS` table, but with the `STATE` set to `FINISHED` or, if execution was not successful, to `ERROR`. Setting this attribute to `False` will lead to standard scheduling of the event via the `ALFA_EVENT_BUS` table and execution via the Alfabet Server.
- **Base Objects via Query** / **Base Objects via Query as Text**: Optionally, a query can be defined to find the start objects for the workflow. This query is usually defined directly in the workflow template. If the workflow template does not contain a base object definition or if the event requires different search options for the base object (For example, to take parameters from the wizard or workflow it is triggered from into account), the query defined in the workflow template can be substituted with a query defined in the event. Use the **Base Objects via Query** attribute to define an Alfabet query. Use the **Base Objects via Query as Text** attribute to define a native SQL query. For information about how to define the start base class for an automatically started workflow, see Specifying the Objects Targeted by the Workflows
- **Source Objects via Query** / **Source Objects via Query as Text**: Optionally, a query can be defined to create new objects via the workflow on basis of a query defining which objects to create. This query is usually defined directly in the workflow template. If the event requires different search options for the base object (For example, to take parameters from the wizard or workflow it is triggered from into account), the query defined in the workflow template can be substituted with a query defined in the event. Use the **Source Objects via Query** attribute to define an Alfabet query. Use the **Source Objects via Query as Text** attribute to define a native SQL query. For information about how to define the start base class for an automatically started workflow, see Specifying the Objects Targeted by the Workflows.
- **Group**: If you would like to structure your event templates in folders, enter the name of the event template folder that the event template shall be located in. If event template folders have already been defined for other event templates, they will be displayed in a drop-down list and can be selected. If you want to define a new event template folder, enter a new name into the field. The folder is then created in the explorer. The event template is moved to the selected event template folder.

4) Click the **Save** button to save your changes.
5) Right-click the event template and select **Set Event State to Active** from the context menu.

Defining an Event Template for the Execution of an ADIF Import Scheme

Event templates are defined in the **Reusable Elements** tab of Alfabet Expand:

1) Go to the **Reusable Elements** tab of Alfabet Expand.

2) Right click the node **Events** and select **Create New ADIF Import Event Template** from the context menu. The new ADIF import event template is added on the root level of the event management tree.

3) Click on the new event template and edit the following attributes:

- **Name**: Enter a unique name for the event template. The name is used to identify the event templates in configurations.
- **Description**: Optionally enter a short text describing the functionality of the event. This information is only visible in the event template attributes in the **Reusable Elements** tab to inform, For example, other solution designers about the functionality implemented with the event template.
- **ADIF Scheme**: Select the ADIF import scheme that shall be executed from the drop-down list. The drop-down list displays all ADIF import schemes that have the attribute **Applicable for REST API** set to `True` and the attribute **State** set to `Active`.
- **Connection Type**: Select `SelfReflective`, if the event shall be executed for the same Alfabet database it is triggered from, or `Query`, if the event shall be executed on an Alfabet database that is defined via an **Alfabet Database Connection**.
- **Connection via Query** / **Connection via Query as Text**: This attribute is only applicable and visible if the **Connection Type** is `Query`. Define a query that returns the `REFSTR` of the Alfabet database connection to be used for execution of the event. The query must return a single `REFSTR`. It can either be defined as native SQL query or Alfabet query:
  - Use the **Connection via Query** attribute to define an Alfabet query. Show properties are not required for the query.

```
ALFABET_QUERY_500
FIND Alfabet_DBConnection
WHERE Alfabet_DBConnection.Name = 'EventConnection'
```

  - Use the **Connection via Query as Text** attribute to define a native SQL query with the `REFSTR` of the Alfabet database connection as only `SELECT` argument.

```
SELECT REFSTR
FROM ALFABET_DBCONNECTION
WHERE ALFABET_DBCONNECTION.NAME = 'EventConnection'
```

- **Variable Names**: If the ADIF scheme contain parameter definitions, the names of the parameters must be written as comma separated list into this attribute. The specification must be case sensitive. The values for the parameters must then be provided with the attribute **Values for Variables via Query** / **Values for Variables via Query as Text**.
- **Values for Variables via Query** / **Values for Variables via Query as Text**: The query must return the values for the variables defined in the attribute **Variable Names** in the same order. The query can be defined either via native SQL or via Alfabet query. Use the **Value for Variables via Query** attribute to define an Alfabet query. Use the **Values for Variables via Query as Text** attribute to define a native SQL query. Alfabet query language parameters can be used in the query. The base object returned via the Alfabet query language parameter `BASE` depends on the way an event has been triggered. For events triggered via a workflow or wizard, `BASE` returns the REFSTR of the object the user was working on in the wizard step or

workflow step triggering the event. For events triggered via an REST API call event, the `BASE` object is the same as the one for the event triggering the event.

The query must return a dataset with the column names identical to the parameter names defined with the attribute **Variable Names**.

For example, if **Variable Names** is defined as `@AppName,@AppVersion`, the query might be defined as:

```
SELECT REFSTR, NAME AS '@AppName', VERSION AS
'@AppVersion'
FROM APPLICATION
WHERE REFSTR = @BASE
```

- **Verbose Logging for ADIF Job**: Set the attribute to `True` if the ADIF job should be started with the command line parameter `-logverbose`, resulting in additional information about the running process to be logged. The default for this attribute is `False`. Verbose logging is in most cases not required and can lead to a decrease in performance.
- **Import File** If the event is triggered on completion of an event that has executed a RESTful service call, the return value from the REST API call can be handed over to the ADIF import event and from the ADIF import event to the ADIF import scheme for execution of the ADIF job. To enable this feature, a file name must be entered in the **Import File** attribute. The event template triggering this event will create the file and write the return value into it prior to triggering this event. Please note that the **Import File** attribute of the ADIF import scheme's JSON import set processing the return value must be set to the same file name as the **Import File** attribute of the event template. For more information about configuring the execution of an event on completion of an event sending a RESTful service call, see Configuring an Event Template Sending a REST API Call to Trigger Another Event.
- **Group**: If you would like to structure your event templates in folders, enter the name of the event template folder that the event template shall be located in. If event template folders have already been defined for other event templates, they will be displayed in a drop-down list and can be selected. If you want to define a new event template folder, enter a new name into the field. The folder is then created in the explorer. The event template is moved to the selected event template folder.

4) Click the **Save** button to save your changes.
5) Right-click the event template and select **Set Event State to Active** from the context menu.

Defining an Event Template for the Execution of an ADIF Export Scheme

Event templates are defined in the **Reusable Elements** tab of Alfabet Expand:

1) Go to the **Reusable Elements** tab of Alfabet Expand.
2) Right click the node **Events** and select **Create New ADIF Export Event Template** from the context menu. The new ADIF export event template is added on the root level of the event management tree.
3) Click on the new event template and edit the following attributes:

- **Name**: Enter a unique name for the event template. The name is used to identify the event templates in configurations.
- **Description**: Optionally enter a short text describing the functionality of the event. This information is only visible in the event template attributes in the **Reusable Elements** tab to inform, For example, other solution designers about the functionality implemented with the event template.
- **ADIF Scheme**: Select the ADIF export scheme that shall be executed from the drop-down list. The drop-down list displays all ADIF export schemes that have the attribute **Applicable for REST API** set to `True` and the attribute **State** set to `Active`.
- **Connection Type**: Select `SelfReflective`, if the event shall be executed for the same Alfabet database it is triggered from, or `Query`, if the event shall be executed on an Alfabet database that is defined via an **Alfabet Database Connection**.
- **Connection via Query** / **Connection via Query as Text**: This attribute is only applicable and visible if the **Connection Type** is `Query`.Define a query that returns the `REFSTR` of the Alfabet database connection to be used for execution of the event. The query must return a single `REFSTR`. It can either be defined as native SQL query or Alfabet query:
  - Use the **Connection via Query** attribute to define an Alfabet query. Show properties are not required for the query.

    ```
    ALFABET_QUERY_500
    FIND Alfabet_DBConnection
    WHERE Alfabet_DBConnection.Name = 'EventConnection'
    ```

  - Use the **Connection via Query as Text** attribute to define a native SQL query with the `REFSTR` of the Alfabet database connection as only `SELECT` argument.

    ```
    SELECT REFSTR
    FROM ALFABET_DBCONNECTION
    WHEREALFABET_DBCONNECTION.NAME = 'EventConnection'
    ```

- **Variable Names**: If the ADIF scheme contain parameter definitions, the parameters must be written as comma separated list into this attribute. The definition must be case sensitive.The values for the parameters must then be provided with the attribute **Values for Variables via Query** / **Values for Variables via Query as Text**.
- **Values for Variables via Query** / **Values for Variables via Query as Text**: The query must return the values for the variables defined in the attribute **Variable Names** in the same order. Use the **Value for Variables via Query** attribute to define an Alfabet query. Use the **Values for Variables via Query as Text** attribute to define a native SQL query. Please note that the first column of the native SQL query is ignored. Alfabet query language parameters can be used in the query. For events triggered via a workflow or wizard, `BASE` returns the REFSTR of the object the user was working on in the wizard step or workflow step triggering the event. For events triggered via a REST API call event, the `BASE` object is the same as the one for the event triggering the event.

The query must return a dataset with the column names identical to the parameter names defined with the attribute **Variable Names**.

For example, if **Variable Names** is defined as `@AppName,@AppVersion`, the query might be defined as:

```
SELECT REFSTR, NAME AS '@AppName', VERSION AS
'@AppVersion'
FROM APPLICATION
WHERE REFSTR = @BASE
```

- **Verbose Logging for ADIF Job**: Set the attribute to `True` if the ADIF job should be started with the command line parameter `-logverbose`, resulting in additional information about the running process to be logged. The default for this attribute is `False`. Verbose logging is in most cases not required and can lead to a decrease in performance.
- **Group**: If you would like to structure your event templates in folders, enter the name of the event template folder the event template shall be located in. If event template folders have already been defined for other event templates, they will be displayed in a drop-down list and can be selected. If you want to define a new event template folder, enter a new name into the field. The folder is then created in the explorer. The event template is moved to the selected event template folder.

4) Click the **Save** button to save your changes.
5) Right-click the event template and select **Set Event State to Active** from the context menu.

Defining an Event Template for Adding Questionnaire Indicators for an Object to a Questionnaire

Starting a questionnaire will create the questionnaire indicators for all objects found by the query in the assigned questionnaire policy.

The number of relevant objects may change during the time period for answering the questionnaire. New objects may be created or the data for existing objects may be changed to match the search conditions in the questionnaire policy.

To add additional questionnaire indicators for a new object to a started questionnaire, an event can be added to a wizard or workflow.

For example, an active questionnaire is currently answered by the authorized users of all applications with a criticality indicator set to 'High' or 'Very High'. The evaluation page view for the application is available as a wizard step in a wizard. If a user changes the criticality indicator and moves on to the next page of the wizard, the event will be triggered and questionnaire indicators will be created for the application.

Event templates are defined in the **Reusable Elements** tab of Alfabet Expand:

1) Go to the **Reusable Elements** tab of Alfabet Expand.
2) Right click the node **Events** and select **Create New Questionnaire Add Indicators Event** from the context menu. The new event template is added on the root level of the event management tree.
3) Click on the new event template and edit the following attributes:

- **Name**: Enter a unique name for the event template. The name is used to identify the event templates in configurations.
- **Description**: Optionally enter a short text describing the functionality of the event. This information is only visible in the event template attributes in the **Reusable Elements** tab to inform, For example, other solution designers about the functionality implemented with the event template.
- **Execute Immediately**: Setting this attribute to `True` will cause the Alfabet Web Application to execute the event immediately by triggering the REST API call for the event. The event is nevertheless written to the `ALFA_EVENT_BUS` table, but with the `STATE` set to `FINISHED` or, if execution was not successful, to `ERROR`. Setting this attribute to `False` will lead to standard scheduling of the event via the `ALFA_EVENT_BUS` table and execution via the Alfabet Server.
- **Connection Type**: Select `SelfReflective`, if the event shall be executed for the same Alfabet database it is triggered from, or `Query`, if the event shall be executed on an Alfabet database that is defined via an **Alfabet Database Connection**.
- **Connection via Query** / **Connection via Query as Text**: This attribute is only applicable and visible if the **Connection Type** is `Query`. Define a query that returns the `REFSTR` of the Alfabet database connection to be used for execution of the event. The query must return a single `REFSTR`. It can either be defined as native SQL query or Alfabet query:
  - Use the **Connection via Query** attribute to define an Alfabet query. Show properties are not required for the query.

    ```
    ALFABET_QUERY_500
    FIND Alfabet_DBConnection
    WHERE Alfabet_DBConnection.Name = 'EventConnection'
    ```

  - Use the **Connection via Query as Text** attribute to define a native SQL query with the `REFSTR` of the Alfabet database connection as only `SELECT` argument.

    ```
    SELECT REFSTR
    FROM ALFABET_DBCONNECTION
    WHERE ALFABET_DBCONNECTION.NAME = 'EventConnection'
    ```

- **Group**: If you would like to structure your event templates in folders, enter the name of the event template folder the event template shall be located in. If event template folders have already been defined for other event templates, they will be displayed in a drop-down list and can be selected. If you want to define a new event template folder, enter a new name into the field. The folder is then created in the explorer. The event template is moved to the selected event template folder.

4) Expand the event's node in the explorer and expand the **Parameters** node. Three parameter nodes are displayed.
5) Click each of the parameter nodes and define the following in the attribute section:
   - ***@Questionnaire***: This parameter is mandatory. Define a query finding the relevant questionnaire. The questionnaire must have the release status defined as `ApprovedStatus` for the object class `Questionary` in the XML object ***ReleaseStatusDef*** and the questionnaire target date must be in the future.

For more information about the definition of release status values, see Configuring Release Status Definitions for Object Classes.

You can either define an Alfabet query in the **Value for Parameter via Query** attribute or a native SQL query in the **Value for Parameter via Query as Text** attribute. The query must return the `REFSTR` of the relevant questionnaire in a dataset with one column and one row.

For example, a native SQL query to find the questionnaire may read:

```
SELECT NULL, QUESTIONARY.REFSTR
FROM QUESTIONARY
WHERE QUESTIONARY.NAME = 'Security Assessment for Cloud
Servers'
```

- ***@Object***: This parameter is mandatory. Define a query finding the relevant object only. Typically, this will be the base object of the wizard or workflow you are currently working with. You can either define an Alfabet query in the **Value for Parameter via Query** attribute or a native SQL query in the **Value for Parameter via Query as Text** attribute. The `FIND` class of the Alfabet query must be the object class of the relevant object. The native SQL query must return the `REFSTR` of the object class of the relevant object as first `SELECT` argument. The query must contain conditions that limit the result to a single object.

For example, an Alfabet query to find the object may read:

```
ALFABET_QUERY_500
FIND Device
WHERE Device.REFSTR CONTAINS:BASE
AUTODSINFO
QUERY_XML
<QueryDef>
   <ShowProperty Type="Property" ClassName="Device"
   Name="REFSTR" />
</QueryDef>
```

- ***@UserName***: This parameter is optional. If specified, the string defined in the parameter will be written into the `CREATION_USER` object class property of the questionnaire indicators generated by the event. Please note that according to the setting of the **Update History User Name** in the server alias of the Alfabet Web Application, the `CREATION_USER` object class property stores either the user name or the technical name of users. The information returned for the parameter must match the setting in the server alias.

For information about the setting in the server alias, see *Configuring User Information Displayed in History Tracking* in the reference manual *System Administration*.

You can either define an Alfabet query in the **Value for Parameter via Query** attribute or a native SQL query in the **Value for Parameter via Query as Text** attribute. The query must return the user name or the technical name of an Alfabet user in a tabular dataset with a single row and a single column. Instead of the

defining the user via a query, a fixed user name or technical name can be defined in the **Default Value** attribute.

For example, a native SQL query to set the `CREATION_USER` to the user that is triggering the event may read:

```
SELECT PERSON.REFSTR, PERSON.USER_NAME
FROM PERSON
WHERE PERSON.REFSTR = @CURRENT_USER
```

6) Click the **Save**  button to save your changes.

7) Right-click the event template and select **Set Event State to Active** from the context menu.

Configuring an Event for the Execution of a RESTful Service Call

A number of configuration objects must be configured for the event. The central configuration object is an event template that contains references to all components required for event execution. All other involved components are also used in other contexts.

A complete event triggering execution of a RESTful service call includes the following configurations:

- The parameters for the service call must be defined in the XML object `GenericRestConfig`.
- An object of the object class **REST API Connection** must be defined that references the connection. The REST API connection is an object in the Alfabet database, which means that it is subject to access permission concepts and that it can be found via a query.
- If the REST API call includes definition of a body, a configured report of the type `NativeSQL` must be created that returns the data to be included into the call in a format that can be converted to JSON by the Alfabet components.
- A REST API call event template must be defined that includes information about all required configuration objects.
- A workflow step action, a wizard step action or an ADIF scheme must be defined for the wizard step or workflow step which shall trigger the event.

The following figure shows how an event triggers the execution of a REST API call when a user leaves a wizard step and the configurations that are required to trigger the event:

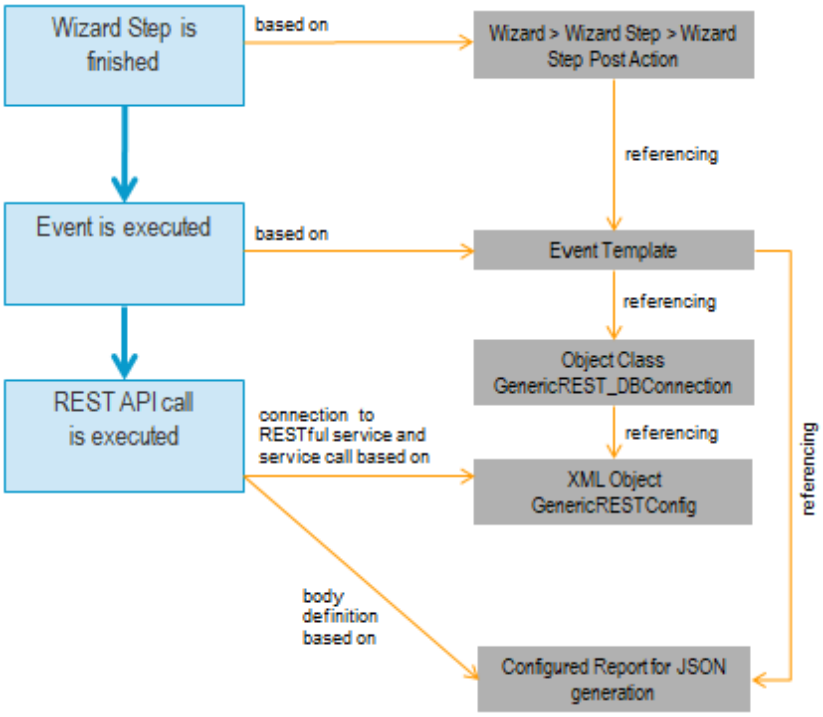


FIGURE: Execution of REST API call triggered via an event and the necessary configuration

The following information is available:

- Configuring the Connection Parameters in the XML Object GenericRestConfig
  - Defining a Dynamic Method Path for the RESTful Service Call
- Configuring the REST API Connection for Execution of the Event
- Defining the JSON Payload for the Body of the Service Call
  - Specifying Properties returning Text, String, Boolean, Date/Time Integer or Float
  - Specifying Properties returning an Array of Property Values
  - Specifying Properties Returning an Object
  - Specifying Properties Returning a List of Objects
- Creating an Event Template for Sending of a RESTful service call

Configuring the Connection Parameters in the XML Object GenericRestConfig

The Alfabet Web Application acts as RESTful client that connects to the RESTful API of any defined third-party component offering a RESTful service. The connection parameters must be configured in the XML object ***GenericRestConfig*** in Alfabet Expand:

1) of Alfabet Expand and expand the explorer node **XML Objects** > **Integration Solutions**.
2) Right click the XML object `GenericRestConfig` and select **Edit XML** from the context menu. The XML object opens in the middle pane.

3) Edit the XML as described below.

4) Click the **Save**button to save your changes.

The XML object must contain the following XML structure:

```
<GenericRestConfig>
    <Connection
        name = "ARIS Object Update"
        active="true"
        service="$ARIS_Service"
        authorization="tenant=default&amp;name=${ARIS_UserName}&amp;password=${ARIS_Password}&amp;key=${ARIS_Key}"
        auth_path = "/umc/api/tokens"
        auth_type = "token"
        auth_method_type = "POST"
        api="/abs/api"
        method_type = "PUT"
        method_content_type = "application/json"
        method_path = "/abs/api/objects/${dbname}/${objectGUID}"
        auth_release_type = "DELETE"
        auth_release_path = "/umc/api/tokens/${token}" />
</GenericRestConfig>
```

The following XML elements and their attributes are part of the specification:

Element (Bold) / Attribute	Configuration Requirements
GenericRestConfig	Root node of the configuration.
Connection	The configuration parameters for sending the RESTful service call. This XML element can be added multiple times to define multiple targets for RESTful service calls.
`name`	Enter a unique name for the connection configuration. The name is used to identify the connection configuration For example, in editors on the Alfabet user interface. This attribute is mandatory.
`active`	This XML attribute can be defined to deactivate (`false`) or activate (`true`) the connection. If the attribute is set to `false`, the connection will not be displayed in the drop-down list for the connection specification in the editor **Alfabet Database Connection** on the Alfabet user interface. The default value is `true`.

Element (Bold) / Attribute	Configuration Requirements
service	Enter the URL of the external application offering the RESTful service. This XML attribute is mandatory. The service XML attribute can be defined via a server variable to read the value of the XML attribute at runtime from the server alias configuration of the Alfabet Web Application for each service call. Only the complete value can be substituted with a server variable by specifying an $ sign followed by the server variable name. For information about how to define and use server variables, see Using Server Variables in Configurations.
auth_type	Enter the type of authentication at the external RESTful service. Supported values are: • basic for basic authentication with user name and password. The XML attributes authorization, auth_username and auth_password must also be defined for this authentication type. • token for request of a user token from the external RESTful service in a separate service call that is then valid for authentication during the actual service call. The XML attributes authorization, auth_path and auth_method_type must also be defined for this authentication type.
authorization	If auth_type is basic, enter Basic. Please note that Basic must start with a capital letter. If auth_type is token, enter the body definition for the authorization call. Please note that the specification must be compliant with definitions in XML. For example, ampersands must be written as & instead of &. The definition of the authorization XML attribute can include parts that are not directly defined in the authorization XML attribute but in the server alias of the Alfabet Web Application as server variable, to substitute the variable definition at runtime with the value from the server alias configuration of the Alfabet Web Application. Server variables can be included into the string with a $ sign followed by the server variable name in curly brackets. For example,: ${ServiceUserName} to include the server variable ServiceUserName. For information about how to define and use server variables, see Using Server Variables in Configurations.
auth_username	If auth_type is basic: Enter the user name for authentication at the external RESTful service.
auth_password	Enter the user password for authentication at the external RESTful service.

Element (Bold) / Attribute	Configuration Requirements
auth_path	If auth_type is token: Enter the path to the authentication end point of the external RESTful service relative to the URL defined with the attribute service.
auth_method_type	If auth_type is token: Enter the HTTP method used for the authentication call.
method_type	Enter the HTTP method used for the RESTful service call. The currently supported methods are GET, HEAD, POST, PUT, DELETE, and PATCH.
method_content_type	Enter the content type for the HTTP request. This attribute is optional. If not defined, the Content-Type header of the RESTful service call will be set to application/json.
method_path	Enter the path to the end point of the external RESTful service the service call should be made to, relative to the URL defined with the attribute service. The method_path specification can include dynamic parts that will be substituted at runtime from the values returned from a configured report. For more information, see Defining a Dynamic Method Path for the RESTful Service Call below.
auth_release_path	If auth_type is token, and the external RESTful service provides an endpoint to release tokens after use, the call to that endpoint to release the current token can be defined as part of the event execution. The path to the endpoint releasing the token including the required parameters for the service call must be defined in the auth_release_path XML attribute to activate the token release. The path definition should include the variable ${token}, that will be substituted at runtime with the token generated for the current RESTful service call. The attribute auth_release_type must also be defined to send the RESTful service call for releasing the token.
auth_release_type	If auth_type is token, and the external RESTful service provides an endpoint to release tokens after use, the call to that endpoint to release the current token can be defined as part of the event execution. Enter the HTTP method for the call to release the token into the auth_release_type XML attribute to activate the token release. The auth_release_path XML attribute must also be defined to send the RESTful service call for releasing the token.

Defining a Dynamic Method Path for the RESTful Service Call

The call to the endpoint of the external service defined with the `method_path` XML attribute in the `GenericRestConfig` XML object may contain parts that depend on the current call, like For example, adding the name of the object in the Alfabet database the user was working with in the wizard step that triggered the event as parameter handed over in the service call. Such parts can be filled dynamically via a configured report of the type `NativeSQL`.

For information about how to create a configured report of the type `NativeSQL`, see Creating a Tabular Configured Report of the Type NativeSQL.

The configured report definition must meet the following criteria:

- The **Category** attribute of the configured report must be set to the category specified for the use case `Events` in the XML object ***UseCaseCategories***.

  For information about the category definition in the XML object ***UseCaseCategories***, see Assigning a Category for Specific Functional Use to a Configured Report.

- The **Type** attribute of the configured report must be `NativeSQL`.
- The SQL query defined for the configured report must return a report with one row and a column for each variable to be used in the `method_path` XML attribute of the ***Connection*** XML element in the ***GenericRestConfig*** XML object.

  Please note that only visible columns in the result dataset of the query are taken into account. The first `SELECT` argument in the native SQL query is not visible in configured reports. In this context, it does not provide any functionality and can be set to NULL.

- Alfabet query language parameters can be used in the query. Please note that the parameter `BASE` returns the `REFSTR` of the object the user is currently working on in the wizard step or workflow step triggering the event. For events triggered via another REST API call event, the `BASE` object is the same as the one for the event triggering the event.

The column names from the result dataset of the configured report can then be used as variables in the definition of the method_path XML attribute. The required syntax is:

```
${ColumnName}
```

The variable will then be substituted with the value returned in the respective column at runtime.

Configuring the REST API Connection for Execution of the Event

Configuration is done in the **Integration Solutions Configuration** functionality on the Alfabet user interface.

1) Go to the **Integration Solutions Configuration** functionality and click the **REST APIConnection** node in the **Integration Solutions Configuration** explorer.
2) In the view, click **New** > **Create REST API Connection**.

If you have already defined a similar connection and want to take over the settings of that connection for your new connection, you can alternatively click **New** > **Create REST Connection as Copy** and select the existing connection the new connection should base on from the selector that opens. The editor for the new connection will then open with all settings identical to the copied connection and the name set to "Copy of <base connection name>".

3) In the **REST API Connection** editor, define the following fields as needed:

Basic Data tab:

- **ID**: Alfabet assigns a unique identification number to each REST API connection. This number cannot be edited.
- **Name**: Enter a unique name for the REST API connection.
- **Release Status**: Select the REST API connection's current release status.

  The set of release statuses available for an object class are configured by your solution designer in the configuration tool Alfabet Expand. For more information, see the section Configuring Release Status Definitions for Object Classes in the reference manual *Configuring Alfabet with Alfabet Expand*. For general information about release statuses, see the section *Understanding Release Statuses* in the reference manual *Getting Started with Alfabet*.

- **Description**: Enter a meaningful description that will clarify the purpose of the REST API connection.

Authorized Access tab:

- **Authorized User**: Click the **Search** icon to assign an authorized user to the selected Alfabet database connection. The authorized user will have Read/Write access permissions for the object and is responsible for the maintenance of the object.
- **Authorized User Groups**: Select one or more checkboxes to assign Read/Write access permissions to all users in the selected user group(s).

Connection tab:

- **REST Connection**: Select the connection to the relevant Alfabet database that is configured in the XML element ***Connection*** in the XML object ***GenericRestConfig*** in Alfabet Expand.

4) Click **Test REST API Connection**. If your settings are correct, a message "The connection is valid" is displayed. Otherwise, an error message is displayed.

Defining the JSON Payload for the Body of the Service Call

If the service call includes a body definition, the JSON object to be send as payload of the call has to be defined via a configured report. The configured report must be of the type `NativeSQL`.

For information about how to create a configured report of the type `NativeSQL`, see Creating a Tabular Configured Report of the Type NativeSQL.

The configured report definition must meet the following criteria:

- The **Category** attribute of the configured report must be set to the category specified for the use case `Events` in the ***UseCaseCategories*** XML object.

  For information about the category definition in the XML object ***UseCaseCategories***, see Assigning a Category for Specific Functional Use to a Configured Report.

- The **Type** attribute of the configured report must be `NativeSQL`.
- The SQL query defined for the configured report must return a report with a simple table that returns data of a given format in columns with given column names as described below.
- Alfabet query language parameters can be used in the query. For events triggered via a workflow or wizard, `BASE` returns the REFSTR of the object the user was working on in the wizard step or workflow step triggering the event. For events triggered via another REST API call event, the `BASE` object is the same as the one for the event triggering the event.

The `SELECT` statement of the configured report must return columns with a given name scheme returning specific parts of the JSON. Each row in the configured report corresponds to a row in the report.

In general, the following definition is required:

Column Name	Data to be returned in the column
`REFSTR`	The first `SELECT` argument in the native SQL query is not visible in configured reports. In this context, it does not provide any functionality and can be set to `NULL`.
`ObjID`	Each object in the JSON must have an identifier. The identifier is not visible in the JSON, but required to build the structure. The identifier can be any string and must be unique for the object within the report. Special characters are not allowed in identifier names. This column is mandatory. An identifier is required in each row of the report to assign the data defined in the row to the correct JSON object.
`ParentObjID`	If the data in a row belongs to a JSON object that is child of another JSON object, this column must return the identifier of the parent object. This column is not required for rows returning properties of the root object(s).
`ParentPropName`	If the data in a row belongs to a JSON object that is child of another JSON object, this column must return the name of the parent object's property containing the subordinate object. This column is not required for rows returning properties of the root object(s).
`PropName`	The name of the property that is defined with this row. This value is shown in the JSON output.

Column Name	Data to be returned in the column
	This column is mandatory.
PropIsList	The column defines whether the property values is a list of JSON objects. Allowed values are: • 1: the property value is a list of JSON objects. • 0: the property value in not a list of JSON objects. This column is mandatory.
ValueInteger	If the property value is an integer, this column must return the property value. Otherwise, the column must return NULL.
ValueFloat	If the property value is a float value, this column must return the property value. Otherwise, the column must return NULL. If non of the properties in the JSON output has a float value, the column can be left out in the report definition.
ValueText	If the property value is a text, this column must return the property value. Otherwise, the column must return NULL.
ValueBoolean	If the property value is a boolean, this column must return the string true or false. Otherwise, the column must return NULL. If non of the properties in the JSON output has a boolean value, the column can be left out in the report definition.
ValueDate	If the property value is a date in a valid date or date time type, this column must return the property value. Otherwise, the column must return NULL. In case a string type instead of a date type value is used, the data and time format must be ISO 8601 compliant, e.g. 2018-09-17 08:57:22.260. If non of the properties in the JSON output has a date value, the column can be left out in the report definition.

It is recommended to create the event template for the event prior to configuring the report and link the report to the event template in an early stage of configuring the report. The context menu of the event template provides a functionality for testing the JSON output of a configured report. It is not required to set the report state to **Active** for testing the JSON output via the event template. The functionality can be used to test intermediate stages of the JSON definition via the configured report. For more information, see Creating an Event Template for Sending of a RESTful service call.

The following sections provide details about the definition of different structures in the JSON:

- Specifying Properties returning Text, String, Boolean, Date/Time Integer or Float

- Specifying Properties returning an Array of Property Values
- Specifying Properties Returning an Object
- Specifying Properties Returning a List of Objects

Specifying Properties returning Text, String, Boolean, Date/Time Integer or Float

A JSON object consists of a number of properties that are a name/value pair. The JSON body of a RESTful service request can have either one single root JSON object or a list of JSON objects.

To define the properties of the type text, string, boolean, date/Time and number in a root JSON object, you must specify an internal object ID for the root object, like For example, the string `root`.

If the JSON contains a list of objects on the root level, each root level object must have a different internal object ID.

In addition, the attribute **Is List** of the event template must be set to `True` for processing of an object list on the root level.

For each property of a root JSON object, the configured report the JSON serialization is based on must contain a row that returns the following:

- The internal JSON object ID of the object in the column `ObjID`.
- The name of the property in the column `PropName`.
- The property value in either the column `ValueInteger`, `ValueFloat`, `ValueText`, `ValueBoolean`, or `ValueDate`, dependent on the data type of the value.
- The `PropIsList` column must be set to `0`.
- All remaining columns must be set to `NULL`.

For example, for creating a simple JSON object with the structure:

```
{
    "TextProperty": "This is a root property",
    "DateProperty": 2019-02-11 10:59:30.333,
    "IntegerProperty": 16,
    "BooleanProperty": true
}
```

The native SQL query of the configured report must result in the following dataset:

	ObjID	ParentObjID	ParentPropName	PropName	PropIsList	ValueInteger	ValueText	ValueDate	ValueBoolean
1	root			TextProperty	0		This is a root property		
2	root			DateProperty	0			2018-02-13 0...	
3	root			IntegerProperty	0	16			
4	root			BooleanProperty	0				1

The example is based on the following native SQL query:

```
SELECT NULL As REFSTR, 'root' As ObjID, NULL As ParentObjID, NULL As
ParentPropName, 'TextProperty' As PropName, 0 As PropIsList, NULL As
'ValueInteger', 'This is a root property' As 'ValueText', NULL As
'ValueDate', NULL As 'ValueBoolean'

UNION ALL

SELECT NULL As REFSTR, 'root' As ObjID, NULL As ParentObjID, NULL As
ParentPropName, 'DateProperty' As PropName, 0 As PropIsList, NULL As
'ValueInteger', NULL As 'ValueText', '2019-02-11 10:59:30.333' As
'ValueDate', NULL As 'ValueBoolean'

UNION ALL

SELECT NULL As REFSTR, 'root' As ObjID, NULL As ParentObjID, NULL As
ParentPropName, 'IntegerProperty' As PropName, 0 As PropIsList, 16
As 'ValueInteger', NULL As 'ValueText', NULL As 'ValueDate', NULL As
'ValueBoolean'

UNION ALL

SELECT NULL As REFSTR, 'root' As ObjID, NULL As ParentObjID, NULL As
ParentPropName, 'BooleanProperty' As PropName, 0 As PropIsList, NULL
As 'ValueInteger', NULL As 'ValueText', NULL As 'ValueDate', 1 As
'ValueBoolean'
```

If the property is not defined for the root XML object but a subordinate object included via a property returning a list of objects, the specification described above must be amended with the following:

- The JSON object ID of the parent JSON object in the column `ParentObjID`.
- The name of the property in the parent JSON object that returns the list of subordinate objects in the column `ParentPropName`.

The complete requirements for defining properties returning one JSON object is described in the section Specifying Properties Returning an Object, and for properties returning a list of JSON objects in the section Specifying Properties Returning a List of Objects.

Specifying Properties returning an Array of Property Values

For a property in a JSON object that shall contain an array of values, the configured report must return one row for each value in the array. Each row must be identical except for the defined value. The following must be returned:

- The JSON object ID of the JSON object containing the property with the array values in the column `ObjID`.
- The name of the property containing the array values in the column `PropName`.
- One of the values of the array in either the column `ValueInteger`, `ValueFloat`, `ValueText`, `ValueBoolean`, or `ValueDate`, dependent on the data type of the value. All rows defining values for the array should have the value defined in the same column.
- The `PropIsList` column must be set to `1`.

- The columns `ParentObjID` and `ParentPropName` must be set to `NULL` for a root node property or to the object ID and property name of the parent object and parent property if the array property is defined for a subordinate object.

  The complete requirements for defining properties returning one JSON object is described in the section Specifying Properties Returning an Object, and for properties returning a list of JSON objects in the section Specifying Properties Returning a List of Objects.

For example, for creating a property with an array in the root JSON object with the structure:

```
{
    "ArrayProperty": ["Value 1","Value2"]
}
```

The native SQL query of the configured report must result in the following dataset:

	ObjID	ParentObjID	ParentPropName	PropName	PropIsList	ValueInteger	ValueText	ValueDate	Valu
1	root			ArrayProperty	1		Value1		
2	root			ArrayProperty	1		Value 2		

Specifying Properties Returning an Object

For a property in a JSON object that shall contain a single JSON object, the configured report must return one row for the definition of the property as returning an object and an additional row for each property of the subordinate object.

For the definition of the property as a property containing an object, the following must be defined:

- The JSON object ID of the JSON object containing the property in the column `ObjID`.
- The name of the property in the column `PropName`.
- The columns `ParentObjID` and `ParentPropName` must be set to `NULL` for a root node property or to the object ID and property name of the parent object and parent property if the property is defined for a subordinate object.
- `NULL` in all of the columns `ValueInteger`, `ValueFloat`, `ValueText`, `ValueBoolean`, or `ValueDate`.
- The `PropIsList` column must be set to `0`.

For the definition of the properties of the subordinate object, the following must be defined:

- The JSON object ID of the subordinate JSON object containing the property defined with this row in the column `ObjID`.
- The name of the property defined with this row in the column `PropName`.
- The name of the property containing the subordinate object in the column `ParentPropName`.

- The object ID of the JSON object that the property containing the subordinate object belongs to in the column `ParentObjID`.
- The property value of the property defined with this row in either the column `ValueInteger`, `ValueFloat`, `ValueText`, `ValueBoolean`, or `ValueDate`, dependent on the data type of the value. All other of the property value columns must be set to `NULL`.
- The `PropIsList` column must be set to `0` if the property defined with this row specifies a simple value or to `1`, if the property defines an array.

For example, for creating a property with a subordinate object in the root JSON object with the structure:

```
{
    "ObjectProperty": {
        "SubTextProperty":"This is a subordinate object",
        "SubIntegerProperty": 1
        }
}
```

The native SQL query of the configured report must result in the following dataset:

	ObjID	ParentObjID	ParentPropName	PropName	PropIsList	ValueInteger	ValueText	ValueDate	ValueBoolean
1	root			ObjectProperty	0				
2	sin...	root	ObjectProperty	SubTextProperty	0		This is a s...		
3	sin...	root	ObjectProperty	SubIntegerPro...	0	1			

Specifying Properties Returning a List of Objects

For a property in a JSON object that shall contain a list of JSON objects, the configured report must return one row for the definition of the property as returning an object list and an additional row for each property of the subordinate objects.

For the definition of the property as a property containing a list of objects, the following must be defined:

- The JSON object ID of the JSON object containing the property in the column `ObjID`.
- The name of the property in the column `PropName`.
- The columns `ParentObjID` and `ParentPropName` must be set to `NULL` for a root node property or to the object ID and property name of the parent object and parent property if the property is defined for a subordinate object.
- `NULL` in all of the columns `ValueInteger`, `ValueFloat`, `ValueText`, `ValueBoolean`, or `ValueDate`.
- The `PropIsList` column must be set to `1`.

For the definition of the properties of the subordinate objects, the following must be defined:

- The JSON object ID of the subordinate JSON object containing the property defined with this row in the column `ObjID`. Please note that each object in the list of objects must have a different object ID.

- The name of the property defined with this row in the column `PropName`.
- The name of the property containing the list of subordinate objects in the column `ParentPropName`.
- The object ID of the JSON object that the property containing the list of subordinate objects belongs to in the column `ParentObjID`.
- The property value of the property defined with this row in the column `ValueInteger`, `ValueFloat`, `ValueText`, `ValueBoolean`, or `ValueDate` that corresponds to the data type of the value. All other of the property value columns must be set to `NULL`.
- The `PropIsList` column must be set to `0` if the property defined with this row specifies a simple value or to `1`, if the property defines an array.

For example, for creating a property with a list of two subordinate objects in the root JSON object with the structure:

```
{
   "ObjectProperty": [{
      "SubTextProperty":"This is the first object in a list",
      "SubIntegerProperty": 1
      },
      {"SubTextProperty":"This is the second object in a list",
      "SubIntegerProperty": 2
      }]
}
```

The native SQL query of the configured report must result in the following dataset:

	ObjID	ParentObjID	ParentPropName	PropName	PropIsList	ValueInteger	ValueText
1	root			ObjectListProp...	1		
2	firstinlist	root	ObjectListProperty	SubTextProperty	0		This is the first object in a list
3	firstinlist	root	ObjectListProperty	SubIntegerPro...	0	1	
4	secondinlist	root	ObjectListProperty	SubTextProperty	0		This is the second object in a l
5	secondinlist	root	ObjectListProperty	SubIntegerPro...	0	2	

Creating an Event Template for Sending of a RESTful service call

To define an event template for an event sending a RESTful service call:

1) Go to the **Reusable Elements** tab of Alfabet Expand.
2) The new RESTful service call event template is added on the root level of the event management tree.
3) Click the new event template and edit the following attributes in the attribute window:
   - **Name**: Enter a unique name for the event template. The name is used to identify the event templates in configurations.

- **Description**: Optionally enter a short text describing the functionality of the event. This information is only visible in the event template attributes in the **Reusable Elements** tab to inform, For example, other solution designers about the functionality implemented with the event template.
- **Connection via Query** / **Connection via Query as Text**: Define a query that returns the `REFSTR` of the REST API connection to be used for execution of the event. The query must return a single `REFSTR`. It can either be defined as native SQL query or Alfabet query:
  - Use the **Connection via Query** attribute to define an Alfabet query. Show properties are not required for the query.

    ```
    ALFABET_QUERY_500
    FIND GenericREST_DBConnection
    WHERE GenericREST_DBConnection.Name = 'EventConnection'
    ```

  - Use the **Connection via Query as Text** attribute to define a native SQL query with the `REFSTR` of the Alfabet database connection as only `SELECT` argument.

    ```
    SELECT REFSTR
    FROM GENERICREST_DBCONNECTION
    WHERE GENERICREST_DBCONNECTION.NAME = 'EventConnection'
    ```

    If the query returns no result, the event is not executed, but no error will be thrown. This behaviour can be used to define a query that couples the return value to a condition for executing the RESTful service call. For example, data about Alfabet objects to be exported to a third part application shall only be executed if the required external object ID is set for all objects included in the export.
- **Connection Report**: If the connection definition in the ***GenericRestConfig*** XML element includes dynamic parts in the specification of the `method_path` XML attribute, select the configured report returning the current values for the dynamic elements from the drop-down list.
- **Execute Immediately**: Setting this attribute to `True` will cause the Alfabet Web Application to execute the event immediately by triggering the REST API call starting the workflow. The event is nevertheless written to the `ALFA_EVENT_BUS` table, but with the `STATE` set to `FINISHED` or, if execution was not successful, to `ERROR`. Setting this attribute to `False` will lead to standard scheduling of the event via the `ALFA_EVENT_BUS` table and execution via the Alfabet Server.
- **Completion Event**: Select an existing event of any event type that shall be executed on completion of the REST API call event. The following is relevant for the execution of an event triggered on completion of an event sending a RESTful service call:
  - The base object of the current event is handed over as base object to the event triggered on completion. Queries defined for the completion event can reference this base object via the Alfabet query language parameter `BASE`.
  - If the completion event is triggering the execution of an ADIF import job, the return value from the REST API call executed via this event can be handed over to the completion event. The event template triggering the ADIF job must then include the definition of an import file with the **Import File** attribute of the event template for the

completion event. The file will be created and the return value written to it prior to triggering the completion event. Please note that the **Import File** attribute of the ADIF import scheme's JSON import set processing the return value must be set to the same file name as the **Import File** attribute of the event template.

- **Group**: If you would like to structure your event templates in folders, enter the name of the event template folder the event template shall be located in. If event template folders have already been defined for other event templates, they will be displayed in a drop-down list and can be selected. If you want to define a new event template folder, enter a new name into the field. The folder is then created in the explorer. The event template is moved to the selected event template folder.
- **Report**: If the service call should include a body, select the configured report defining the JSON object, which will be send as body, from the drop-down list.
- **Is List**: Set this attribute to `True` if the configure report specified with the attribute Report will result in a JSON having a list of JSON objects on the root level instead of a single root JSON object.

4) Click the **Save**button to save your changes.
5) A new window opens that displays the JSON request as generated from the data returned by the configured report defined with the **Report** attribute of the event template. Check the JSON request for correctness of structure and content. If the result is not as expected, the configured report has to be corrected.
6) Right-click the event template and select **Set Event State to Active** from the context menu.

Configuring Triggering of an Event

One of the following configurations can be used to configure execution of events on user interactions:

- Configuring a Wizard or Workflow to Trigger the Event
- Configuring an Event Template Sending a REST API Call to Trigger Another Event
- Configuring the Feedback Bot to Trigger an Event

Configuring a Wizard or Workflow to Trigger the Event

Events can be triggered via an on enter, on cancel, or on exit step action of a wizard step or via an on enter, on refuse, on expired, or on exit step action of a workflow step. The implementation is the same for all wizard step actions and workflow step actions:

1) In Alfabet Expand, navigate to the wizard step or workflow step that you want the event to be triggered from.
2) Create a new wizard step action or workflow step action, as needed:
   - To create a new wizard step action for a wizard step, right-click the relevant **Action On Entered Step**, **Action On Exited Step**, or **Action On Cancelled Step** node and select the option to create the new wizard step action. For more information about configuring wizard step actions, see the section Configuring Wizard Step Actions for a Wizard Step.

- To create a new workflow step action for a workflow step, right-click the relevant **Action On Entered Step**, **Action On Refused Step**, **Action On Expired Step**, or **Action On Exited Step** node of the workflow step and select the option to create the new workflow step action. For more information about configuring workflow step action to trigger an event, see the section Configuring Events to Be Triggered for a Workflow Step.

3) Click the new wizard step action/workflow step action in the explorer and edit the following attributes in the attribute window:

- **Technical Name**: Enter a name for the wizard step action/workflow step action. This name is not displayed in the Alfabet interface.
- **Action Type**: Select `TriggerEvent`.
- **Technical Comment**: Enter any relevant information about the wizard step action/workflow step action.
- **Event Template**: Select the relevant event template from the drop-down list.

4) Optionally, you can define a condition to execute the event depending on the availability of specific data returned by a query. For example, you could specify that the event may only be executed if a property of the base object of the wizard is set to a specific value. Edit the following attributes in the attribute window to define the condition:

- **Check Query** / **Check Query as Text**: Define a query that will return a result only if a condition is met. The query can be defined either as an Alfabet query in the **Check Query** attribute or as a native SQL query in the **Check Query as Text** attribute.
- **Result Type**: Select `Positive` if the event shall only be executed if the query returns a result. Select `Negative` if the event shall only be executed if the query does not return a result.

For example, an event to update an application's indicators when a wizard step is exited shall be triggered if the application's release status is set to the value "Approved". The following query will only return a result if the application that is the base application of the wizard step has the correct release status:

```
SELECT REFSTR
FROM APPLICATION
WHERE APPLICATION.STATUS = 'Approved'
AND APPLICATION.REFSTR = @WIZARDBASE
```

The **Result Type** is set to `Positive` sothat that the event is only executed if the query returns a result.

5) Click the **Save**  button to save your definitions.

Configuring an Event Template Sending a REST API Call to Trigger Another Event

Events can be triggered on completion of an event for sending of a RESTful service call.

On triggering of the event for sending a RESTful service call, the RESTful service call is executed, and if the event is configured to trigger another event on completion of the service call, the completion event is

triggered as soon as the return value from the external RESTful service has been received. The base object of the current event is handed over as base object to the event triggered on completion. Queries defined for the completion event can reference this base object via the Alfabet query language parameter `BASE`.

If the completion event is triggering the execution of an ADIF import job, the return value from the REST API call can be handed over to the completion event in JSON format. The event template triggering the ADIF job must then include the definition of an import file with the **Import File** attribute of the event template for the completion event. The file will be created and the return value written to it prior to triggering the completion event. Please note that the **Import File** attribute of the ADIF import scheme's JSON import set processing the return value must be set to the same file name as the **Import File** attribute of the event template.

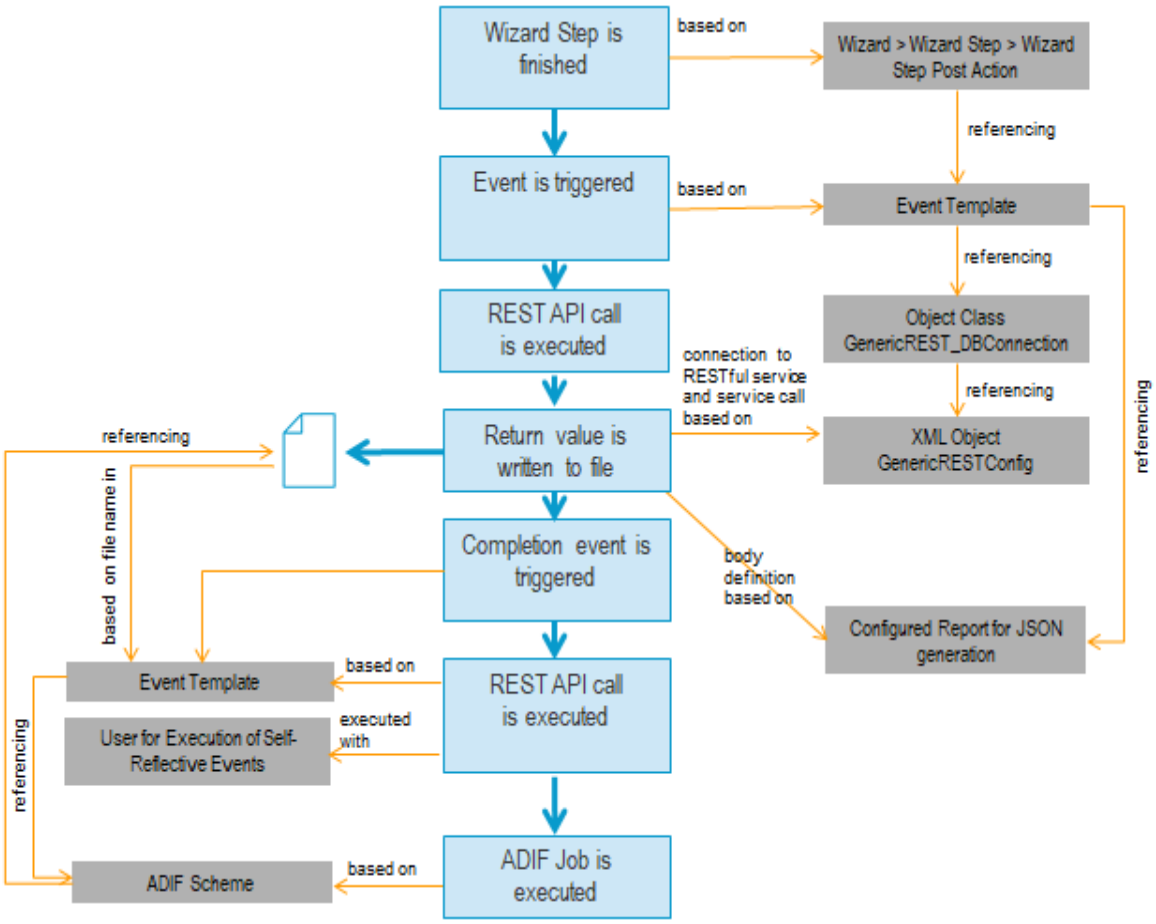


FIGURE: Triggering of an ADIF import job that processes the return value from an event template

To configure an event for execution of a RESTful service call to trigger another event on completion:

1) In Alfabet Expand, go to the **Reusable Elements** tab.
2) In the explorer, click the event template for execution of a RESTful service call that should trigger the event.
3) In the attribute window select the event template that shall be triggered on completion of this event in the drop-down list of the attribute **Completion Event**.
4) Click the **Save** button to save your definitions.

The following additional configuration is required to process the return value in an ADIF import triggered by the completion event:

- In the event template triggering the ADIF import, specify a file name with the extension JSON in the **Import File** attribute.
- In the ADIF scheme triggered by the completion event, create a JSON import set and set the **Import File** attribute of the JSON import set to the file name specified in the **Import File** attribute of the completion event template.

For information about creating and configuring an ADIF import scheme, see the reference manual *Alfabet Data Integration Framework*.

Configuring the Feedback Bot to Trigger an Event

Events can be triggered when a user sends a contact request or rating via the Feedback Bot, For example, to create a ticket for the feedback in a ticketing system via a RESTful service call.

1) Go to the **Presentation** tab and expand the **XML Objects** folder. The XML object ***AlfaFeedbackBotConfig*** opens.
2) Right-click ***AlfaFeedbackBotConfig*** and select **Edit XML**. The XML editor is displayed in the center pane.
3) To trigger an event when the user submits a rating, add an XML attribute `EventTemplate` to the XML element `SpecificFeedbackPage` and set it to the name of the event template:

   ```
   <SpecificFeedbackPage EventTemplate="NameofEventTemplate" [...]/>
   ```

4) To trigger an event when the user submits a contact form, add an XML attribute `EventTemplate` to the XML element `ContactUsPage` and set it to the name of the event template:

   ```
   <ContactUsPage EventTemplate="NameofEventTemplate" [...]/>
   ```

5) In the toolbar, click the **Save**  button to save the XML definition.

Checking the Execution of Events

Registered events are stored in the Alfabet database as objects of the object class `ALFA_EVENT_BUS` with the state `PENDING`. The state and other attributes of the event in the table are changed and amended during execution, informing about the success of execution. If errors occur, the error messages are also written to the database table of the object class `ALFA_EVENT_BUS`.

Two business functions are available that provide an overview over the execution of event as stored in the `ALFA_EVENT_BUS` table:

- The business function **Event Administration** (`ADMIN_Events`) provides information about all events that are triggered by any user.
- The business function **My Events** (`USER_Events`) provides information about all events that are triggered by the current user. The user triggering the event is the user that was using the wizard or workflow that triggered the event via one of its conditions.

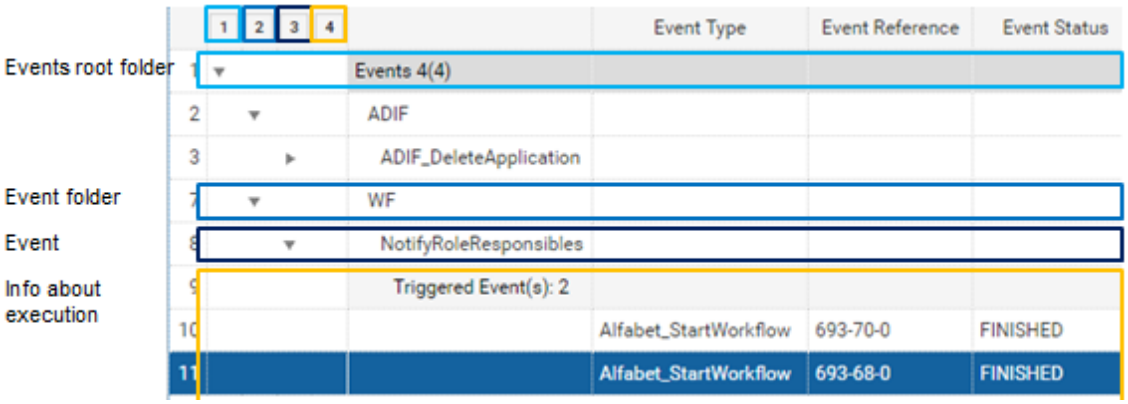
The views can be made available to the user via the user profile, guide page or guide view configuration.

For information about adding business functions directly to user profiles, see Making Functionalities Accessible to a User Profile. For information about adding business functions to user profiles via guide pages or guide views, see the reference manual *Designing Guide Pages for Alfabet*.

Both views show the same tabular view of the information about event execution.

The table is an expandable dataset that shows information about event execution in the structure that is defined for the event templates in Alfabet Expand:

- The first level displays the root node and informs about the number of executed events.
- The second level displays event folders and events that are located directly under the root node.
- The third level displays the events in the event folder, or, for events located directly under the root node, the list of triggered events.
- The fourth level displays the triggering for events located in event folders.



If an event has been triggered one or multiple times, each triggering of the event is listed as a separate row under the heading **Triggered Event(s)** with the following information about the event execution:

- **Event Type**: Displays whether the event triggers the start of a workflow (`Alfabet_StartWorkflow`), a RESTful service call (`RestCall`), execution of an ADIF import (`Alfabet_StartAdifImport`), or execution of an ADIF export (`Alfabet_StartAdifExport`). Monitoring events are reported as the event type `Task`.
- **Event Reference**: The `REFSTR` of the triggered event. This is a unique identifier for the scheduled or executed event.
- **Event Status**: The status can be one of the following:
  - **Pending**: The event has been triggered and is queued for execution.
  - **Executing**: The event is currently executed.
  - **Finished**: The event has been successfully executed.
  - **Canceled**: The event has been canceled.
  - **Error**: The event has been executed and an error occurred during execution of the event.

- **Execution Error**: The event has been executed and an error occurred during execution of the functionality triggered by the event.
- **Execution Forcefully Terminated**: The execution of the event has been terminated in the context of a server shutdown.

If monitoring events are scheduled for cyclic execution, regular executions will be managed via a single event. After the monitoring is triggered, the event will be immediately re-scheduled for execution ten minutes later. Therefore, the event status value of the event will remain **Pending** after monitoring has been triggered.

- **Event Callback Status**: This column is only filled for events triggering an ADIF import or export that are configured to wait for a callback from the triggered ADIF process. The event will have the status **Pending** as long as the ADIF process is executed. The status will then change to **Finished** or **Execution Error** depending on the success of the ADIF process execution.
- **Occurrence Time**: The date and time when the event was triggered.
- **Start Time**: This column is only filled for triggered cyclic monitoring events. Cyclic execution is based on a single event. After monitoring is executed, the event will not be terminated, but re-scheduled immediately for the next execution of monitoring. Therefore, the event status of the event will remain **Pending**. The next scheduled start time of monitoring will be displayed in the **Start Time** column.
- **Comment**: This field is currently not used.
- **Trigger Type**: Currently, the only trigger type is `ALFABET`. This means that the event is triggered from an Alfabet component.
- **Sender**: Currently, the only sender is `Alfabet`. This means that the event is triggered from an Alfabet component.

A filter is available to reduce the content of the view to a subset of the triggered events.

If the execution of an event failed, and the **Event Status** is set to **ERROR**, a detailed error message, including error messages that were thrown during execution of the RESTful service call for the event, can be downloaded for the failed event. Select the event in the table and click the **Show Event Message** button in the toolbar. The event message is written into a file that is then provided for download via the download mechanism of your browser.

Changing or Deleting an Event Template

Whether an event template can be executed, edited or deleted depends on the setting of the **Event State** of the event template:

- Event templates can only be edited if the **Event State** is set to **Plan**.
- Event templates can only be added to workflow or wizard step actions and executed by a user if the **Event State** is set to **Active**.
- Event templates can only be deleted if the **Event State** is set to **Plan** or **Retired**. Please note that event templates that are implemented to be executed via wizard or workflow step actions are ignored, if the **Event State** is set to **Retired**. That means that the event is not executed any longer without causing an error.

To edit an active event template:

1) In the explorer, right-click the event template and select **Show Usage**. A new window opens that lists all wizard and workflow steps triggering the execution of an event on basis of the selected event template.
2) On basis of the information, check whether re-configuration must be performed prior to altering the **Event State** of the event template. If a user executes a wizard or workflow that triggers the event and the **Event State** of the event template is not **Active**, an error message is displayed informing the user about the inactive **Event State** of the event template.
3) Right-click the event template and select **Set Event State to 'Plan'**.
4) Edit the event template.
5) Right-click the event template and select **Set Event State to 'Active'**.
6) If you changed the configuration of wizard or workflow step actions to prevent error messages caused by currently not active event template, revert the preliminary changes.

To delete an event template:

1) In the explorer, right-click the event template and select **Show Usage**. A new window opens that lists all wizard and workflow steps triggering the execution of an event on basis of the selected event template.
2) On basis of the information, remove the configuration using the event template from wizard and workflow configurations.
3) Right-click the event template and select **Set Event State to 'Retired'**.
4) Right-click the event template and select **Delete**.

Structuring Events in Event Folders

You can structure and organize event templates in group folders:

- To create an event folder for an existing event template, enter a name for the event folder in the **Group** attribute and press ENTER. The event folder appears ascendant to the selected event template.
- To create a new event template for an existing event folder, right-click the event folder and select **Create New Workflow Event Template**, **Create New ADIF Import Event Template**, or **Create New ADIF Export Event Template**.
- To add an event template to an existing event folder, click the **Group** attribute and select the relevant event template group. The event template is moved to the selected event folder.
- Any changes you make to the **Group** attribute of the event template must be saved by clicking the **Save**  button.

The context menu of the event folders includes options to alter the event state. If you select one of the options to alter the event state on an event folder, the event state of all events subordinate to the folder is changed.

Saving and Restoring the Configuration of Event Templates

It is recommended to configure events in a development environment and apply it to the production environment after complete testing only.

To take over the configuration done in a development environment to the Alfabet database of a test or production environment, the general mechanisms for merging or replacing solution configurations in one Alfabet database with the complete or partial solution configuration of another Alfabet database described in the section Applying Configuration Changes to Other Databases can be used.

Alternatively, the configuration can be saved to an XML file and merged from the XML file into the configuration of the target database:

1) Connect to the source database with Alfabet Expand.
2) In the **Reusable Elements** tab, do one of the following:
   - To store the complete event template configuration in an XML file, right-click the folder **Event Templates** and select **Save As**.
   - To store a single event template in an XML file, right-click the event template and select **Save As**.
3) In the explorer window that opens, select a location for the XML file that is accessible via Alfabet Expand in the target environment and click **Save** to save the file.
4) Connect to the target database using Alfabet Expand.
5) In the **Reusable Elements** tab, right-click the folder **Event Templates** and select one of the following:
   - **Replace from File** to overwrite the current configuration with the configuration in the XML file. The whole event template configuration is deleted and then the configuration stored in the XML file is applied. That means that all event templates in the XML file will overwrite corresponding event templates in the target database. Event templates in the target database that have no corresponding event template in the XML file will be deleted. Event templates that are only available in the XML file will be added.
   - **Merge from File** to merge the current configuration with the configuration in the XML file. All event templates in the XML file will overwrite corresponding event templates in the target database. Event templates that have no corresponding event template in the XML file will remain unchanged. Event templates that are only available in the XML file will be added.
6) In the explorer window that opens, select the XML file with your new event template configuration.
7) Confirm the warning message. The event template configuration in the target database is changed.

Chapter 17: Configuring Reports

Configured reports can be generated and configured in the **Reports** tab of Alfabet Expand for the following purposes:

- **Providing customer-specific information to the user community**

Standard Alfabet reports about objects in the Alfabet database are included in the standard Alfabet product. However, you can also configure and provide customized reports to the user community via the Alfabet interface.

- **Defining matrices or multi-editors for batch data capture**

The standard editors and views for the Alfabet interface allow the user to define data per object. In some special situations that require to change the data for a high number of objects, this can be a time consuming process. To ease data capture for multiple objects, a configured report can be defined that allows multiple business supports or relationships between objects to be defined for a customizable set of objects by clicking cells in a matrix. Another type of configured report provides editors for multi-editing of object class properties of selected objects either by defining individual values or by batch setting of a property to a defined value for all selected objects.

- **Defining compliance policies for the Compliance Management functionality**

For more information, see Configuring Queries for Compliance Policies in the section Configuring the Compliance Management Capability.

- **Defining datasets to export data via the Alfabet RESTful API**

For security reasons, the Alfabet RESTful service reads data from a query in a configured report that is executed at runtime instead of directly accessing the Alfabet database. For more information, see the reference manual *Alfabet RESTful API*.

- **Defining data export conditions for batch data export to HTML, XML or PDF files with the tool QueryExecutor.exe**

For more information, see the section *Exporting Data to XML, HTML, or Microsoft® Excel® Format with the QueryExecutor.exe* in the reference manual *System Administration*.

This chapter provides an overview of the types of configured reports that can be specified, explains how configured reports are implemented in the Alfabet interface, and describes in detail the necessary configuration steps.

The following information is available regarding reports:

Types and Features of Configured Reports

Alfabet offers a wealth of standard reports in the standard Alfabet solution. In addition to the standard reports that are available, customers can create configured reports required by the enterprise and display them in the Alfabet interface.

The following overview provides information about the different mechanisms to display the information in configured reports and about how configured reports can be integrated in the Alfabet interface.

Software AG provides mechanisms to create five types of configured reports:

Report Type	Description
`Query`	Configured report based on an Alfabet query that displays the query results as a tabular report. You can either allow the user to simply execute the configured report based on the configured Alfabet query, or you can allow the user to limit the Alfabet query results to a subset of the data by setting filters before the configured report is executed. For more information, see Query-Based Tabular Reports.
`NativeSQL`	Configured report based on a native SQL query that displays the query results as a tabular report. You can either allow the user to simply execute the configured report based on the configured native SQL query, or you can allow the user to limit the search results to a subset of the data by setting filters before the configured report is executed. For more information, see Query-Based Tabular Reports.
`Custom`	This type of configured report can result in any of the following report types: • Configured reports that display results as a graphic report, For example, as portfolio, treemap or chart report. The configured reports are based on a template and are defined in the **Report Assistant**. For more information, see Query Based Graphic Representations of Configured Reports. • A template-based matrix for multi-editing of objects. Three templates are available for the creation of a matrix for multi-editing of business supports, object relationships, and indicators. For more information, see Configured Reports Providing Ability for Mass Data Changes. • Console reports that display an object cockpit with a filter area. An object cockpit is a collection of subordinate configured reports displayed in a tabular grid. The filter settings of the console report are applied to all subordinate configured reports assigned to the object cockpit. For more information, see Configuring Console Reports. • Cascading report that displays at least one master report and one subordinate report. When the user selects an object in the master report, the content of the subordinate report changes and displays data related to the subordinate object. For more information, see Combining Multiple, Cascading Reports in One View. • Configured reports that display a geographic map with relevant data displayed by color of region or color and size of markers and connectors between markers.

Report Type	Description
ObjectView	Configured report displaying an object view or object cockpit configured for objects in a selected object class. For more information, see Configured Reports opening Existing Object Views or Object Cockpits.
Extern	Configured report opening a URL specified by the customer. This type of configured report allows for the display of reports showing data in the Alfabet database or external data generated by third-party reporting tools. For more information, see Integration of External Reports.

The following information is available about the features that are provided by the different kind of reports:

- Features Shared by All Types of Configured Reports
  - Providing Custom Online Help Files for a Configured Report
  - Publishing Current Report Results
  - Navigating to Objects Displayed in a Configured Report
  - Navigating to an URL From a Configured Report
  - Assigning Configured Reports to an Object Class
  - Specifying Base Classes Required for the Configured Report
  - Configuring Filters to Define Search Criteria
  - Invoking Start of a Workflow
  - Providing Additional Information in a Quality Widget
  - Hiding Toolbar Buttons
  - Restricting Access to the Alfabet database Via Configured Reports
- Query-Based Tabular Reports
  - Expandable Result Table
  - Definition of Frozen Columns
  - Adding Data to the Clipboard
  - User Specific Table Layout
    - Allowing the User to Hide Columns and Change the Column Order
    - Allowing the User to Alter the Structure and Design of the Report
  - Display of Results in Standard Alfabet Report Formats
  - Display of Results in Customer Configured Views
  - Configuring User Workspaces to Edit Objects in the Configured Report

Features Shared by All Types of Configured Reports

The following features are available for all of the types of configured reports that can be configured for your enterprise:

Providing Custom Online Help Files for a Configured Report

You can create context-sensitive online Help files that provide information about the configured report to the user community and add them to the standard Alfabet online Help. The help file is stored externally and the URL of the help file is defined in the configured report. When the user views the configured report in the Alfabet interface, he/she can click the **Help** button to open a dialog providing hyperlinks to the Help available for the current view. The user will not be able to distinguish between links to external help files and links to the standard Alfabet online Help. When the user clicks a link leading to an external custom help file, the URL specified in the configured report will open in a new window.

For more information about the structure and use of the Alfabet online Help, see the section *Using the Help in Alfabet* in the reference manual *Getting Started with Alfabet*.

Publishing Current Report Results

Alfabet allows for the export of views via an **Export** button in the toolbar. Tables can be exported to HTML or Microsoft® Excel® files and graphics can be exported to Microsoft® PowerPoint® files, Adobe® PDF files and HTML files with embedded graphics in a variety of graphic formats. This functionality is also available by default for all configured reports except for configured reports of the type `Extern`. Template-based matrix reports that support the simultaneously editing of multiple objects can only be exported to Microsoft® Excel® files.

The filter area of a configured report can be defined as included or excluded from export when configuring the report.

Please note the following:

- Gauge reports and geo map reports embedded in an object cockpit cannot be published to DOC or PDF format.
- Reports that return object class properties of the Property Type Text can only be exported to Microsoft® Excel® files if the content of the text fields does not contain more than 32768 characters, which is the length restriction for Microsoft® Excel® data sheet cells.
- For the following configured reports, the export functionality is only available if the third-party component Essential Objects is activated in the web.config file of the Alfabet Web Application.
  - Gallery reports
  - Branching diagram reports
  - Gauge reports
  - Circular roadmap reports
  - Map Charts

  For more information about the activation of the component, see *Prerequisits for Using Essential Objects®* in the reference manual *Technical Requirements*.

For more information about the export of views in Alfabet, see the section *Exporting Data* in the reference manual *Getting Started with Alfabet*.

For graphic reports, a printing functionality is additionally available via the floating toolbar. Clicking the **Print** button in the floating toolbar opens a view that allows to fine-tune the output of the graphic. Confirming the layout by clicking **OK** in the new window will open a new browser tab containing the graphic without the surrounding interface elements, like header and toolbar. The image can then be printed via the print functionality of the browser. This feature is not available for branching diagram reports.

For more information about the printing of graphic views in Alfabet, see the section *Printing a View in Alfabet* in the reference manual *Getting Started with Alfabet*.

Navigating to Objects Displayed in a Configured Report

The user can navigate to the object view of an object displayed in any type of configured report. Please note the following:

- In configured reports of the type `Query`, `NativeSQL`, or `Custom`, the user can access the object view of the object either by double-clicking the object in the results or by selecting the object in the results and clicking the **Navigate** button in the toolbar. Navigation to a base class without further configuration of the report is only possible for a preconfigured subset of object classes.

  - For configured reports based on an Alfabet query, navigation to the preconfigured subset of base classes is available by default. For configured reports based on a native SQL query, a special configuration is required. For more information see Definition of the SELECT Clause in the section Defining Native SQL Queries.
  - For an overview of the navigable object classes for that navigation is activated by default, see *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.
  - For information about how to activate navigation for base object classes that do not offer navigation by default, see Activating Navigation From Reports to Object Classes Not Automatically Offering Navigation.
  - For configured reports of the type `Custom` that display a chart report, like For example, a bar or line chart, navigation is not available by default. The required configuration is described in the section Configuring Navigation from a Chart Report.
  - For configured reports of the type `Custom` that display a custom pivot table for analysis of cube data, navigation is not available by default. The required configuration is described in the section Activating Navigation From Reports to Object Classes Not Automatically Offering Navigation.

- For configured reports of the type `Extern` with a URL that access reports generated with external reporting tools, Software AG provides a link syntax that can be used to open Alfabet views via the external report. For information about the implementation of links to the Alfabet interface in external reports, see *Links to Alfabet Views from External Applications* in the reference manual *System Administration*.
- Configured reports of the type `ObjectView` already display the object view or object cockpit of an object as the report result.
- Alternatively, most configured reports can be configured to provide navigation directly to a page view of the selected object instead of the object's profile. For example, the configuration can specify that if the user double-clicks on an application group in a configured report, the application's portfolio view for the application group will open instead of the application group's object view. For

tabular configured reports, configuration is done by adding an Alfabet instruction. The configuration is described in the section Providing a Link to Alfabet Views, Editors or Wizards from Cells in a Report of the chapter Defining Queries. For most types of graphic representations of report results, the configuration can be done via the report assistant. The required setting in the report assistant is described in the section Defining Navigation from the Report to Alfabet Views.

- Reports can be configured to provide navigation to a related object class that is included in the result dataset instead of navigation to the base object class. This feature requires configuration of the query the report is based on by adding an Alfabet instruction. The configuration is described in the section Changing the Navigation Behavior of the Base Object Class in a Configured Report of the chapter Defining Queries.

Navigating to an URL From a Configured Report

If a cell in a dataset returns a valid URL without any other content, the URL is rendered as a link. If the user clicks the link, the URL opens in a new browser tab.

Assigning Configured Reports to an Object Class

Configured reports can be assigned to an object class by means of the **Apply to Class** attribute or the **Base Object Query** attribute, which are defined in the attribute window for the configured report. A configured report can only be assigned to one object class.

When a configured report of the type `Query`, `NativeSQL`, or `Custom` is assigned to an object class, the configured report automatically refers to a single object of the assigned object class with the Alfabet parameter `BASE` in the configured report's Alfabet query or native SQL query. The parameter `BASE` can be used in the query of the configured report to refer to a selected object of the assigned object class.

When the **Apply to Class** attribute is set, a selector field is automatically displayed on top of the configured report that allows an object in the assigned object class to be selected. If the configured report is displayed in the **Reports** page view of an object view or is assigned to an object view, the object specified with the Alfabet parameter `BASE` is by default the object the user is currently working with. The user can then change to another object by means of the selector.

For configured reports of the type `Query`, `NativeSQL`, `Extern` and `Custom`, the **Apply to Class** attribute also determines the display of the configured report in the **Configured Reports** page view. Configured reports are only available in the **Configured Reports** page view of the object view of an object class that has been defined in the **Apply to Class** attribute. If the **Apply to Class** attribute is not specified, the configured report will not be available in the **Configured Reports** page view for any object class.

The **Apply To Class** attribute is mandatory for configured reports of the type `ObjectView` and for configured reports of the type `Custom` displaying an HTML report about indicators. It is optional for all other configured reports

If the **Apply to Class** attribute is defined, the configured report will only be executed if a base object is defined. If you want a configured report to be executed both for a single selected object or for all objects of a class, you must define a configured report with a filter field that allows the report output to be limited to selected objects.

When the base class of a configured report is set with the **Base Object Query** attribute, the base object is determined by a query that returns a single object. The query can use Alfabet parameters referring to the

current working environment. The base object is determined by execution of the query when the user opens the configured report. Changing to another base object via an automatic filter is not possible when the base class is determined via a base object query.

When the configured report is applied to an object view, the Alfabet parameter `BASE`, referring to the object the user is currently working with, can be used in the **Base Object Query** attribute to determine the base object of the configured report. For example, when the configured report is assigned to an object view of an application, the **Base Object Query** attribute can find the ICT object the current application is assigned to as base object for the configured report.

The Alfabet parameter `BASE` can also be used in configured reports if the configured report is not applied to an object class by means of the attribute **Apply to Class** or **Base Object Query** but added to an object view. In this case, the Alfabet parameter `BASE` automatically specifies the object the user is currently working with. The selection of a different `BASE` object via a filter is not possible. A configured report using the `BASE` parameter in the query definition without being applied to a single class can be added to any class that can meaningfully be used as `BASE` definition. For example, a configured report displays all projects that have the current object as part of the target architecture definition. The project's target architecture can include objects of several classes. The same configured report can be added to the object view of all classes that may be part of a project's target architecture.

Specifying Base Classes Required for the Configured Report

When configuring view schemes for a user profile, object classes can be excluded from a view scheme. All Alfabet views for which the excluded class is the base class of the Alfabet view will be excluded.

The **Base Classes** attribute available for the configured report allows one or more object classes to be specified as the base classes for the configured report. This ensures that the configured report will not be visible if the user is logged in with a user profile for which the object class is excluded.

For more information about the exclusion of classes, see Refining Visibility Issues in the View Scheme in the chapter Configuring User Profiles for the User Community.

Configuring Filters to Define Search Criteria

A configured report can be specified to include filter fields that allow the user to specify search criteria to find objects for the report results. You can define filters for configured reports of the type `Query`, `NativeSQL` and `Custom`.

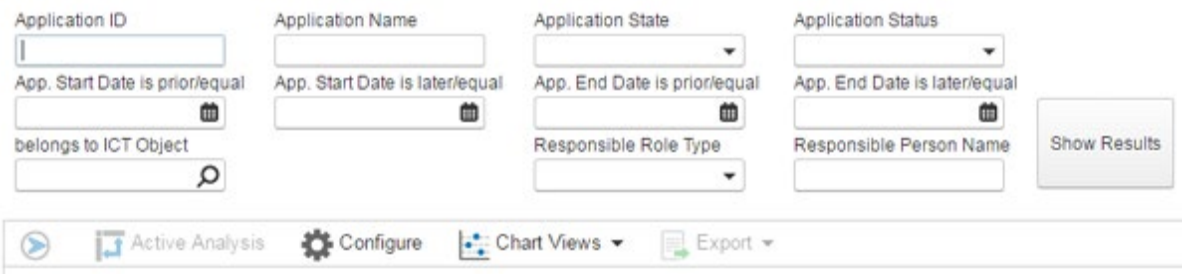


FIGURE: Example of filters that allow the user to define search criteria

Optionally, the filter area can be configured to provide the following functionality:

- Individual filter fields can be configured as mandatory fields. Mandatory filter fields are marked in red and with a red star in the user interface. If the user submits the configured report without setting the mandatory fields, a warning will be displayed.

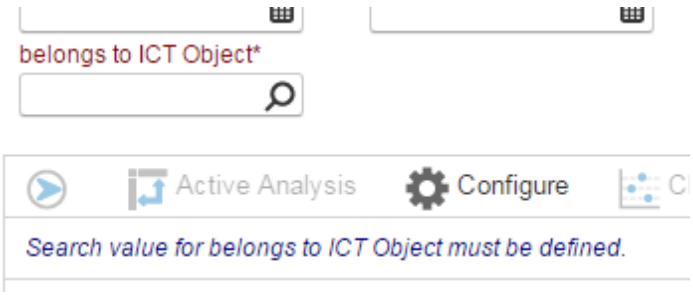


FIGURE: Warning message displayed if mandatory filter field is not defined

- The filter area can have a **Clear Search Patterns** button that allows the current filter settings to be cleared. If applicable, individual filter fields can be configured to be excluded from the clear action.
- The filter area can have a **Hide Filter Panel** button that allows the filter area to be collapsed. This is useful for configured reports with a significant number of filter fields.

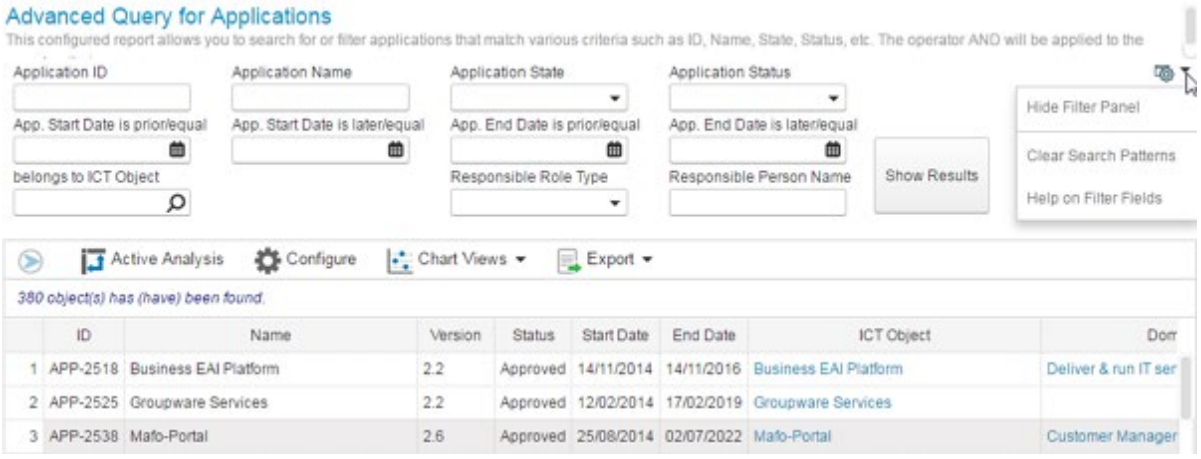


FIGURE: Configured report with expanded filter area

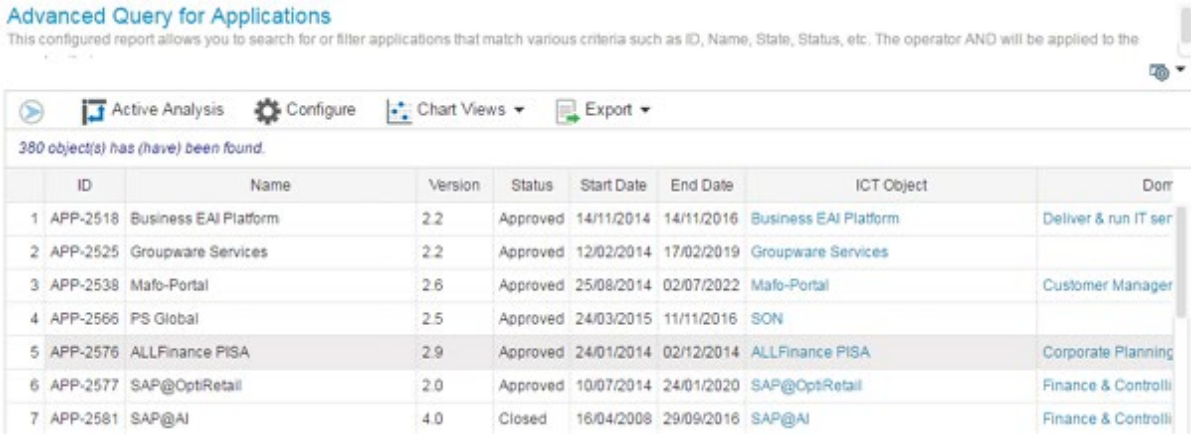


FIGURE: Configured report with collapsed filter area

- The selection that can be done in a drop-down list or a combo-box may depend on the setting that the user performed in another filter field. For example, a filter field allows an application to be selected and the drop-down list of the filter field available to select a local component will be filled with the local components of the currently selected application.
- For edit fields automatic settings of wild cards before and after the string that the user has entered into the field can be configured.
- By default, configured reports display unfiltered results when the user first opens the configured report. Filters can than be set by the user and the dataset will be reduced according to the filter conditions when the user clicks the **Submit** button. The configured report can be configured to show results after the user has clicked the **Submit** button for configured reports that would result in very large datasets when opening without filter settings.
- Filter settings can be reused for other configured reports. This requires a special configuration of the filter field to store the setting in a property of the class `UserGlobalData`.

The use of filters require a additional configuration which is described in the section Defining Filters for Configured Reports and Selectors in the chapter Defining Queries.

Invoking Start of a Workflow

A workflow can be started via a button in a configured report for either the base object or a selected object in a configured report. The menu button to start the workflow cannot be created or configured by the customer but must be ordered from Software AG.

Providing Additional Information in a Quality Widget

Additional information about data in a configured report or object cockpit can be provided via a quality widget displayed in a pop-up window. The quality widget is either a configured widget report or a configured business chart or gantt chart report that can be assigned to multiple configured reports or object cockpits. When the user opens or loads the configured report or object cockpit that the quality widget is assigned to, the quality widget will be displayed in a separate pop-up window that opens for a short time. The quality widget pop-up window opens by default in the upper right corner of the Alfabet user interface below the main menu. This position is configurable.

Depending on the configuration of the primary configured report, either the complete quality widget will be displayed or only the caption will be displayed in a title bar and the user can click the title to open the quality widget.

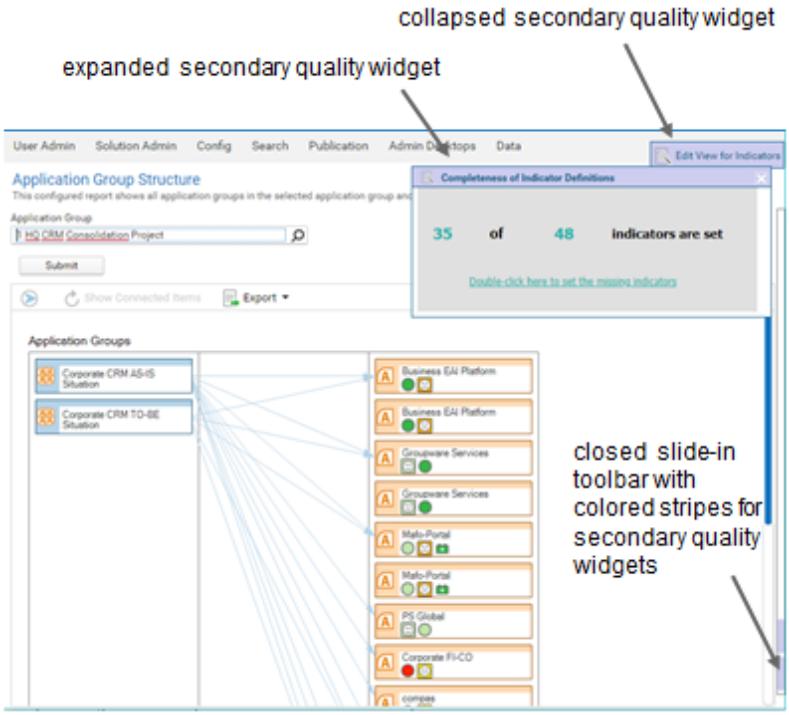


FIGURE: Configured report with a collapsed and an expanded quality widget

If the user does not click the quality widget during the short time that it is visible, the quality widget view will automatically drop into the slide-in toolbar on the right edge of the Alfabet user interface. The slide-in toolbar displays a colored notch for each of the available quality widgets (as well as for other forms of user assistance). The user can access the quality widget anytime by pointing to the notch in the slide-in toolbar and then clicking the quality widget icon to open it. The quality widget will remain open until the user clicks outside of the pop-up window or clicks the close button.

If navigation is configured from the quality widget view to another configured report or graphic view, the navigation target will open in a new browser tab.

A typical use case for quality widgets is a widget report that opens to provide information to the user about the quality of the data in the current view, which can be either a configured report or object cockpit. For example, if objects displayed in a configured report are colored according to an indicator value, the user could be provided with information about the completeness of the indicators specified for the objects in the configured report. A configured link available in the quality widget could then open a view that allows the user to provide the missing data and thus correct the issue. When the user clicks the link available in the quality widget report, the view will open in a new browser tab. The user can then define the relevant indicators and reload the original view with the updated data.

For information about how to define a secondary widget report for a configured report, see Integration of Quality Widgets in Configured Reports, Object Cockpits and Wizards.

Hiding Toolbar Buttons

Each configured report has a toolbar with all relevant buttons for the functionality assigned to the configured report. For example, report results may be analyzed in a Pivot table or the configured report can be exported to files.

You can configure the toolbar to hide buttons in the configured report. Configuration is done per view scheme so that a button may be visible to one user while another user will not have access to the button.

The configuration of the toolbar is described in the context of the view scheme configuration in the section Hiding Functionalities in a Page View or Configured Report in the chapter Configuring User Profiles for the User Community.

Restricting Access to the Alfabet database Via Configured Reports

A number of security mechanism has been implemented to restrict access to configured reports or to Alfabet objects via configured reports:

- In general, the Alfabet Web Application connects to the Alfabet database via a single database user that has ReadWrite access permissions for all tables in the Alfabet database independent from the individual access rights of Alfabet users for which requests are processed. For all or individual configured reports, a database user with restricted access permissions can be used for data requests. Two different methods are available:
  - The Alfabet Web Application can be configured to send requests for all configured reports via an Alfabet internal ReadOnly database user.
  - If the Alfabet database is hosted on a Microsoft® SQL server, customer defined database user with explicit access rights for indiviual database tables can be used for excecution of configured reports. Which database user is used for execution can be specified for all or for individual configured reports.

For information about the database access right restrictions that can be defined for configured reports, see Restriction of Database Access Permissions on Execution of Configured Reports.

Query-Based Tabular Reports

Configured reports of the type `Query` are based on the specification of an Alfabet query and configured reports of the type `nativeSQL` are based on a native SQL query. For both configured reports, the query defines which data should be displayed for a subset of objects in the configured report. The results are displayed in a table similar to the tables in standard Alfabet views.

The user can alter the width of the columns in the configured report at runtime. The column width is stored in the user context settings.

In configured reports displaying large datasets, the first 100 of those results will be displayed directly in the page. You can navigate to a next page showing the next 100 results by means of the floating toolbar in the lower right corner of the configured report and navigate to the next page, previous page, first page, or last page of results. The paging is also available in configured reports with large grouped datasets.

Tabular reports offer a wealth of functionality and display options that are described below.

If a feature is not available by default in the configured report, a link to the documentation describing the required configuration will be available. The basic configuration of configured reports of the type `Query` is described in the section Creating a Tabular Configured Report of the

Type Query. The basic configuration of configured reports of the type `native SQL` is described in the section Creating a Tabular Configured Report of the Type NativeSQL.

- Expandable Result Table
- Definition of Frozen Columns
- Adding Data to the Clipboard
- User Specific Table Layout
  - Allowing the User to Hide Columns and Change the Column Order
  - Allowing the User to Alter the Structure and Design of the Report
- Display of Results in Standard Alfabet Report Formats
- Display of Results in Customer Configured Views
- Configuring User Workspaces to Edit Objects in the Configured Report
- Analyzing a Configured Report in a Pivot Table Layout
- Asynchronous Execution For Long Running Reports

Expandable Result Table

Instead of displaying search results in a flat list, results may be hierarchically structured. The user can click an arrow in the second column on the left of the report table to expand the data table to display the next level of objects. In the example displayed below, domains are displayed on the first level, relevant business functions on the second level, and the applications providing the business functions on the third level.

	1 2 3	Domain / Function Name	Application	Status	Start Date	Er
1	▼	A.3.2 Product Business & Development Planning				
2	▼	Coordinate Business Case Approval				
3			Investment Banking Portal 1.0	Approved	18/07/2015	1(
4			SAP Enterprise Portal 4.7.c	Approved	29/01/2013	0(
5	▶	Define Business Case				
7	▶	A.3.1 Product Capability Definition				
17	▶	A.3.3 Product Design				
22	▶	A.3.5 Product Retirement				

FIGURE: Example for expandable report table

The first column in the report displays the running number of the row in the report. If a level is collapsed, rows are hidden, but nevertheless counted. In the example above, the riw number 5 is followed by the row number 7, indicating that there is one third level entry hidden business function 'Define Business Case' displayed at the second level in row 5.

The example also demonstrates that there are two ways to handle expandable levels:

- Two levels can be displayed in one column. Within the column, objects of the subordinate level are indented. In the example, domains and business functions are displayed in the same column with the business functions in an indented second level below the domain names.
- Two levels can be displayed in two different columns. In the example, applications are displayed in a separate column subordinate to the business functions that they are associated with.

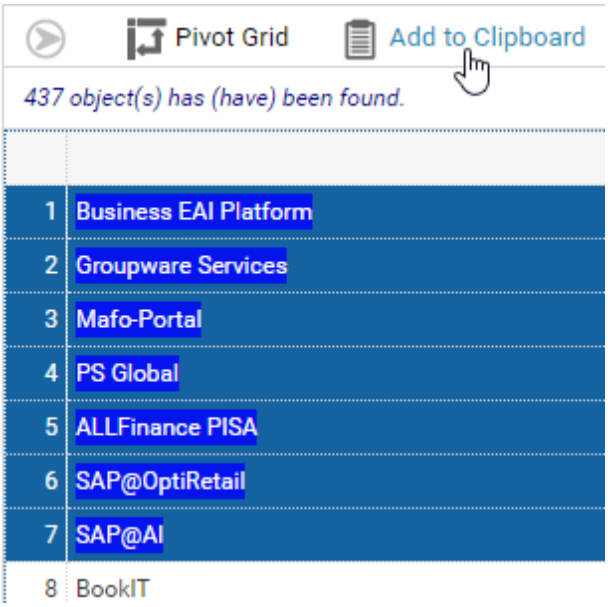
Expandable report tables require additional configuration which is described in the section Grouping Query Results in Expandable Reports in the chapter Defining Queries.

Definition of Frozen Columns

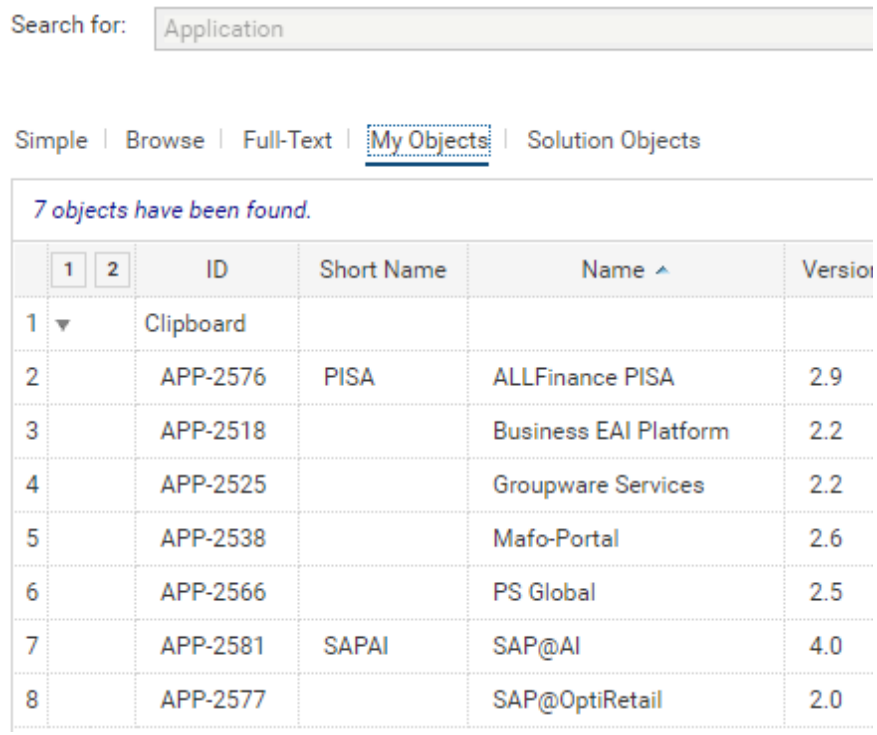
The first column or columns in the tabular dataset can be defined as frozen. Frozen columns remain visible during horizontal scrolling. The definition of frozen columns is done by a solution designer in the query definition. For more information, see Freezing Columns in Tabular Datasets.

Adding Data to the Clipboard

In configured tabular reports that are defined with Alfabet 10.6.1 or higher, an **Add to Clipboard** button is available in the toolbar of the report. The clipboard capability allows the user to select data in the table and click the **Add to Clipboard** button to copy the data.



The data will then be available in relevant standard selectors as well as custom selectors that are configured to display the **My Objects** tab. The saved objects are displayed in the **Clipboard** section in the **My Objects** tab, where they can be selected when defining references.



The **My Objects** view is available in standard selectors. It will only be available in custom selectors if the **Add My Objects Tab** attribute in the **Class Entry** of the custom selector is set to `True`. For more information about the configuration of custom selectors, see Configuring a Custom Selector for Search Functionalities.

Please note the following about the clipboard functionality:

- Data added to the clipboard will remain in the clipboard during the running sub-session. If the user opens another browser tab to open Alfabet or if the user logs out, the data will be removed from the clipboard. Changing the user profile will not empty the clipboard.
- Each time a user adds objects to the clipboard within a running session, the data will be added to the existing data in the clipboard.
- The **My Objects** tab will only display the sub-set of objects in the clipboard that may be selected via the selector. For example, if components and applications have been added to the clipboard, only applications will be displayed in an application selector, only components will be displayed in a components selector, and none of the objects will be displayed in a device selector.

User Specific Table Layout

Configured tabular reports can be designed to provide end user design capabilities. The user can either adapt the number and position of columns to his/her demands or re-define the complete design including coloring and picture assignment.

The following options are available:

- Allowing the User to Hide Columns and Change the Column Order
- Allowing the User to Alter the Structure and Design of the Report

Allowing the User to Hide Columns and Change the Column Order

User configuration can be enabled for individual configured reports. If user configuration is enabled, the table configured by a solution designer in Alfabet Expand as output of the configured report can be changed by the user opening the configured report on the Alfabet user interface:

A button **Configure**  is displayed in the toolbar of tabular configured reports. If the user clicks the button, an editor opens that displays all columns in the configured report. The user can hide columns in the configured report and change the order of columns in the table. The settings will then not only be applied at the current view, but stored in the user context settings and applied to the configured report each time this user opens the configured report, even if she/he is logged in with a different user profile. If another user opens the same configured report, she/he will see the default layout or a layout configured by her/him.

Configurations can only be applied to static tables. If the configured report displays a dynamic dataset where the number and kind of columns depends on the current search results, a layout defined by the user via the **Configure**  button will be applied to the current view and saved in the user context settings, but only applied to the configured report again if the configured report is displayed with the same number and kind of columns.

For information about the configuration of a configured tabular report to allow user configuration, see Allowing the User to Change the Number and Order of Columns in the Table for native SQL based tabular reports or Allowing the User to Change the Number and Order of Columns in the Table for Alfabet query language based tabular reports.

Allowing the User to Alter the Structure and Design of the Report

Tabular configured reports can be designed to provide a wealth of options for individual re-structuring and re-design by the user opening the configured report. This includes:

- Hiding data from the report or adding data to the report. If the report is based on Alfabet query language, the definition of columns defined by the solution designer is only a default layout that can be amended with any other object class property information about the involved object classes by the user. If the report is based on native SQL, the information accessible via the tabular report is limited to the data defined in the `SELECT` statement of the native SQL query. Columns can only be excluded, but additional data about the involved object classes cannot be added to the table.

- Data can be completely restructured. Columns can be moved, hidden and added and the layout can even be changed to a matrix layout with both column and row headers that can also be ordered in a hierarchy.
- Any data can be hidden or filters can be applied independent from the filter area on top of the configured report that is provided by the solution designer. All filters in the configured report are AND combined.
- Cells in the report can be highlighted with colors and pictures can be added according to end user defined rules.

For example, the following SQL based report shows application costs per year and cost type. the default layout displayed when the user opens the report for the first time is a simple table:

Number	Application	Cost Type Name	Year	Cost Value
53	Administrative Gen...	Expense	2017	3.00
54	Administrative Gen...	Expense	2018	31.00
55	Administrative Gen...	Expense	2019	112.00
56	Administrative Gen...	Maintenance & Su...	2019	168.00
57	Administrative Gen...	Other	2017	21.00
58	Administrative Gen...	Other	2018	24.00
59	Administrative Gen...	Other	2019	40.00
60	Administrative Gen...	Subscription	2017	85.00
61	Administrative Gen...	Subscription	2018	85.00
62	Administrative Gen...	Subscription	2019	700.00
63	AF Enterprise ConT...	Consulting	2015	69.00
64	AF Enterprise ConT...	Consulting	2016	323.00
65	AF Enterprise ConT...	Consulting	2017	378.00
66	AF Enterprise ConT...	Consulting	2018	106.00
67	AF Enterprise ConT...	Expense	2015	244.00
68	AF Enterprise ConT...	Expense	2016	1,129.00

The same report will look completely different if the user decides to structure it as a matrix providing an overview for costs per year for each cost type, where the costs for all applications matching the year and cost type definitions are summed up in the respective matrix cell. Icons are used to highlight higher or lower cost values:

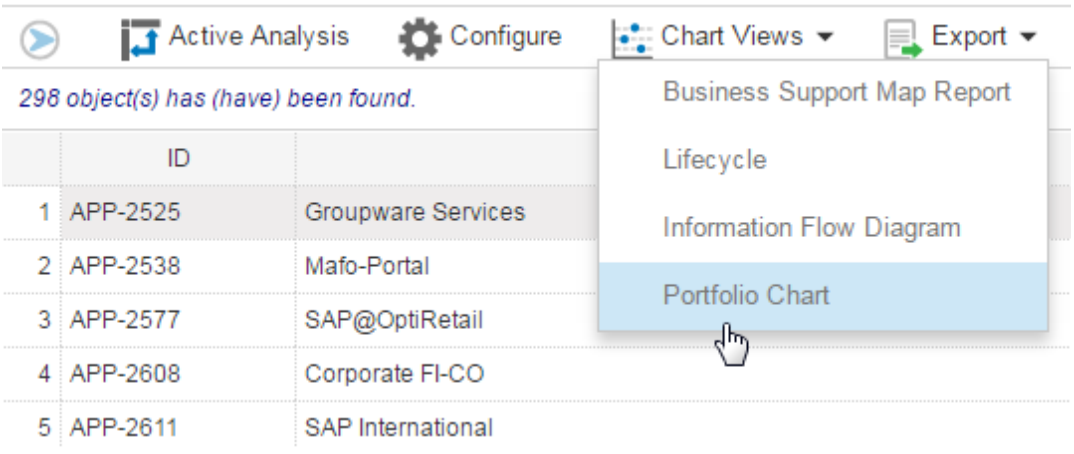
Year	Consulting Cost Value	Expense Cost Value	Other Cost Value	Subscription Cost Value
2008	667.00	2,307.00	316.00	705.00
2009	1,038.00	3,728.00	498.00	915.00
2010	1,393.00	5,013.00	758.00	1,670.00
2011	1,902.00	6,656.00	1,273.00	3,265.00
2012	3,051.00	10,802.00	2,330.00	6,170.00
2013	5,160.00	18,395.00	3,249.00	7,615.00
2014	9,612.00	34,051.00	5,640.00	13,120.00
2015	14,787.00	51,854.00	8,496.00	19,165.00
2016	23,254.00	81,617.00	12,029.00	24,935.00
2017	29,350.00	102,432.00	13,922.00	26,520.00
2018	33,181.00	115,694.00	15,729.00	29,325.00
2019	33,804.00	106,580.00	15,589.00	30,897.00

The end user design option are available if the template `DataTableReport` is assigned to the configured report. For information about defining a tabular configured report with enhanced user design capabilities, see Configuring a Data Table Report with Restructuring Options for the End User for native SQL based tabular reports or Configuring a Data Table Report with Restructuring Options for the End User for Alfabet query language based tabular reports.

A context sensitive help file is available in the standard Alfabet online help that explains the configuration capabilities of data table reports to end users. The context sensitive help is automatically available as soon as the template is selected for the configured report.

Display of Results in Standard Alfabet Report Formats

If the configured report presents the results in a simple table and one of the standard Alfabet reports listed above is available for the base class of the Alfabet query, the toolbar will automatically include a **Chart Views** button that provides access to relevant standard Alfabet report(s).



The following standard Alfabet reports may be available in a configured report for the objects found by the configured report's query:

- Lifecycle Chart
- Portfolio Chart
- Business Support Map Report
- Information Flow Diagram
- Project Tracking Overview (for configured reports with the base class Project)

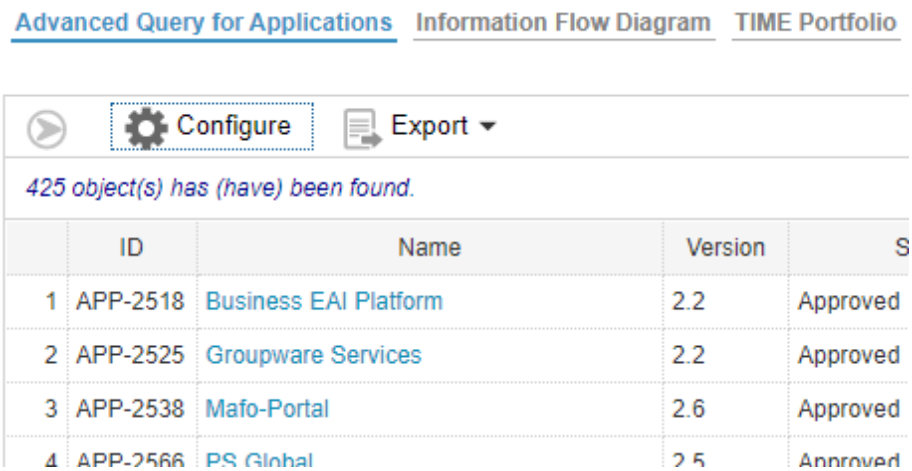
The drop-down list that opens when the **Chart Views** button is clicked will automatically display only the standard Alfabet reports that are available for the base class of the Alfabet query.

The **Save Layout** button of information flow diagrams opened via the **Chart Views** menu is hidden, because the layout cannot be saved for diagrams opened from configured reports. If filter settings for the diagram are stored in the user context settings, the diagram opens with filter settings as stored in the user context settings.

- This feature is only available for configured tabular reports of the type `Query` that do not include Alfabet query language instructions.
- For more information about which Alfabet report is available for which object class, see *Available Standard Reports for Base Object Classes in Configured Reports* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.
- When filters are defined for the configured report, the filter settings are not applied to the chart views. For example, when the filter settings of a configured report limit the display of applications to applications starting with the letter "a", chart views opened from the configured report may nevertheless include information about applications not starting with an "a".
- For an information flow diagram opened via the **Chart Views** button the layout cannot be saved.
- The **Chart Views** button can be excluded from a configured report by setting the attribute **Chart Views Mode** of the configured report to `None` or substituted with access to configured reports via tabs on top of the toolbar of the configured report.

Display of Results in Customer Configured Views

A report collection can be configured for object classes and object class stereotypes via the class settings. This report collection consists of specifically configured reports that provide additional information about the objects found in tabular configured reports returning objects of the object class or object class stereotype. Users can access reports from the report collection via tabs on top of the toolbar of the tabular dataset to view additional aspects like gantt charts or portfolios related to the set of objects currently displayed in the table.



If the user changes the tabular dataset via filters, the information in the reports from the report collection will change accordingly. With report collections different aspects that might be interesting for users interested in a sub-set of objects of a defined object class can be presented directly to the user opening the tabular report. As such, the tabular report is acting like a temporary group of objects that shall be analysed together.

For information about configuring report collections, see Integration of Configured Reports as Report Collection Into Tabular Configured Reports.

Configuring User Workspaces to Edit Objects in the Configured Report

Standard Alfabet views typically have toolbar buttons that allow objects to be edited. The available buttons could open a wizard or editor for the object or delete the object, or provide more complex functionalities such as the creation of a new object based on a copy of an existing object.

Per default, configured reports are for viewing purposes only. However, it is possible to provide edit functionalities in a configured report, if necessary:

- For the basic objects of the IT architecture, buttons can be added to a tabular reports via a preconfigured template. The templates add the buttons that are available in the standards Alfabet data capture functionality to the configured report. The template is available for the following classes:
  - Application
  - Component
  - Contract
  - Demand
  - Device
  - ICT object
  - Peripheral
  - Project

- Service product
- Vendor product
- Vendor

For information about the configuration of a report based on a data capture template, see Defining a Data Capture Environment for a Configured Report of the Type Query or Defining a Data Capture Environment for a Configured Report of the type NativeSQL.

- The functionalities for user management available in the **User Administration** view (`ADMIN_UsersOverview`) can be added to a tabular report via a preconfigured template. The template adds the buttons available for user administration to the configured report, with the exception of button sub-menu interactions that are triggering functionalities for all users in the Alfabet database. The underlying query of the configured report must find objects of the object class `Person`. The template can be used to provide limited access to the user management functionalities to administrators that are For example, only allowed to edit user data for a specific group of users or that are only responsible for a limited subset of administration processes.
- Edit functionalities can be implemented in other configured reports if they are available as the standard for the object class. Configured reports that allow objects to be edited are called "user workspaces". If a user workspace is required by your enterprise, a specific configuration is necessary that must be ordered from Software AG.

Buttons can be configured to be active for a subset of the objects in the configured report. For example, a configured report provide the means to edit only the applications based on a specified object class stereotype while applications based on other object class stereotypes are displayed in the configured report. Limiting button actions to a subset of the objects displayed in a configured report requires a special configuration of the query that the configured report is based on. The configuration is described in the section Limiting Toolbar Button Functionality To a Subset of Objects in a Configured Report in the chapter Defining Queries.

When planning to include a user workspace or a report based on a template for data capture or user management, you must consider access permissions and object availability as well as the location of the configured report in the Alfabet interface.

Analyzing a Configured Report in a Pivot Table Layout

A pivot table is an interactive table that you can use to view, organize, and analyze configured report data dynamically from within your Web browser. Additionally, you can define a layout for the data and then export the data to a Microsoft® Excel® XML Spreadsheet 2003, a CSV file or a PDF file.

Alfabet includes the DevExpress® component, providing capabilities for visualizing and analyzing data in chart and pivot table layouts.

The functionality is available for the following configured reports:

- Tabular reports with the exception of expandable report tables. Depending on the report configuration, the Pivot grid analysis either opens directly when the user accesses the report, or the tabular dataset of the configured report is displayed, and the user must click the **Pivot Grid**  button in the toolbar of the configured report to access the Pivot grid analysis view. Please note that it is not possible to open the Pivot grid analysis in a new tab.

The caption displayed in the column headers of the configured report must each be unique. For more information about the difference between column name and column caption and about altering the column caption, see Defining Column Names and Captions.

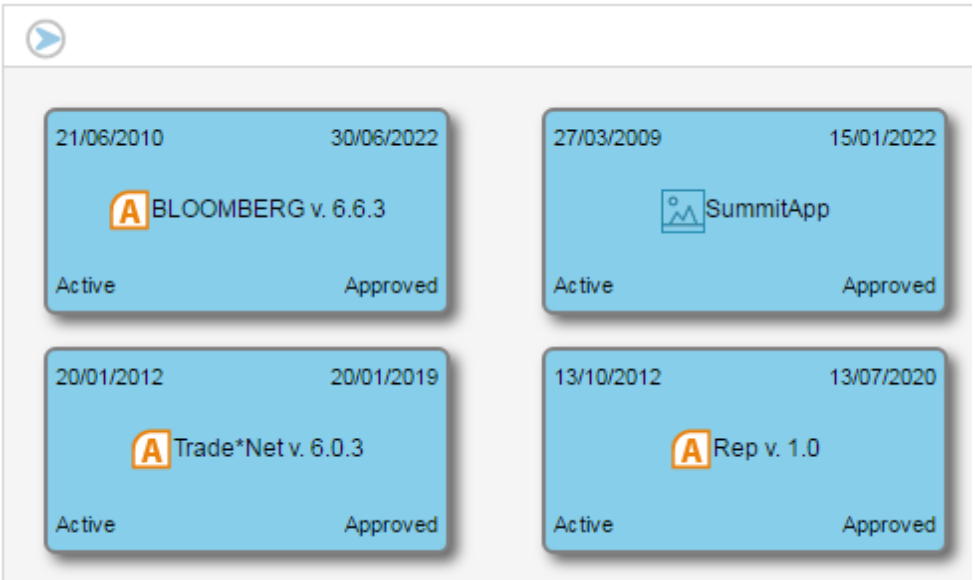
- Configured reports that retrieve data from an external data cube. For cube-based reports, the Pivot grid analysis view opens directly when you open the configured report. For more information about the configuration of cube based reports, see Defining Reports to Analyse Data Cubes.

Asynchronous Execution For Long Running Reports

Query Based Graphic Representations of Configured Reports

Configured reports of the type `Custom` allow search results from an Alfabet query or native SQL query to be displayed in preconfigured graphic formats that are described below. For the Alfabet user, the look and handling of a configured report is identical to the handling of a standard Alfabet report:

- The floating toolbar available for standard Alfabet reports with functionalities for zoom in and zoom out, printing, and for display of a legend is also available for configured reports.
- The design of boxes displayed in configured treemap, diagram matrix, lane and layered diagram reports is identical to the design of standard Alfabet reports, with text on white background displayed within a colored field.
- Links are displayed in the identical coloring and design as all links on the Alfabet user interface.
- If object icons are visible in the boxes of the report, and an object specific icon is defined for the object, the object specific icon is displayed:



The following types of graphic reports can be created and configured in Alfabet Expand:

- Branching Diagram Report

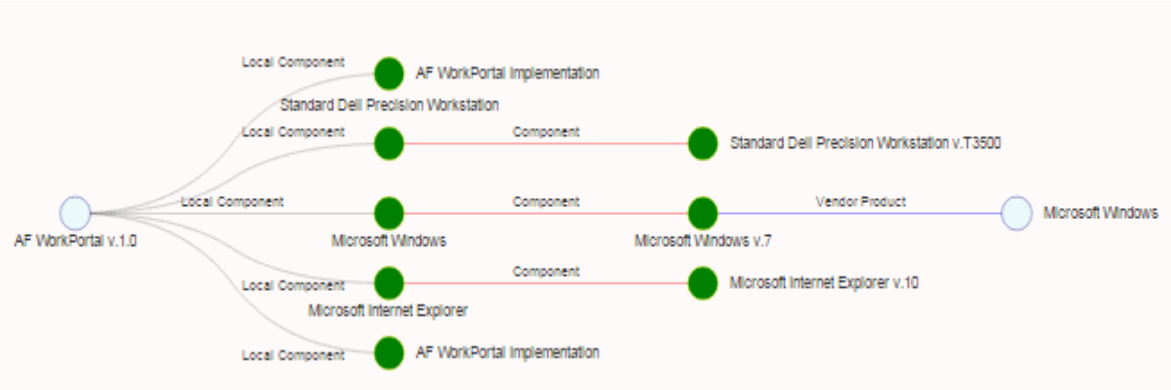
- Business Chart Reports
  - Bar Charts
  - Waterfall Charts
  - Line Charts And Spline Charts
  - Area Charts and Spline Area Charts
  - Pie Charts and Doughnut Charts
  - Multi-Level Pie Charts
  - Radar Charts
  - Combination Charts
- Circular Roadmap Reports
- Gallery Reports
- Node Arc Reports
- Portfolio Diagnostics Report
- Treemap Reports
- Rectangular Treemap configured reports
- Layered Diagram Reports
- Tree Reports
- Cluster Map Grid Reports
- Matrix Reports
- Gantt Chart Reports
-
  - Portfolio Reports
- Lane Report
- HTML Table Report for Display of Indicators
- Widget Reports
- Words Cloud Report

Branching Diagram Report

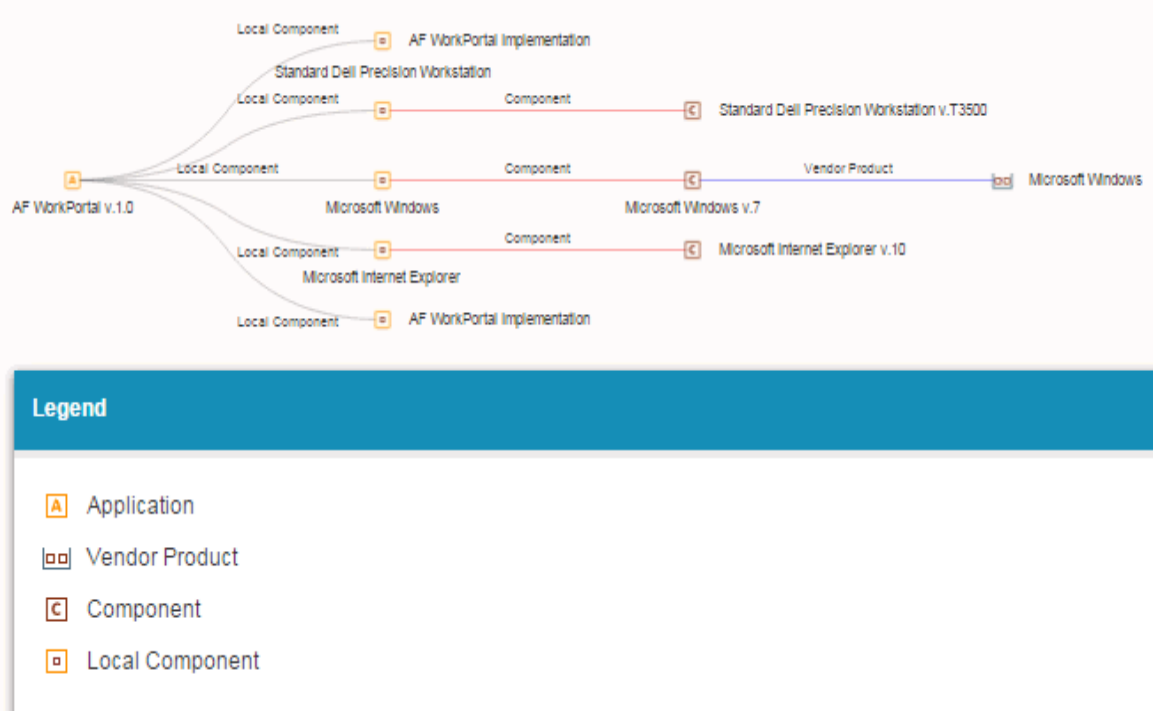
A branching diagram report is a graphic representation of a grouped tabular dataset that displays each level of objects as nodes in a column with the relation between objects represented by lines between the nodes.

The objects can be displayed in the following ways:

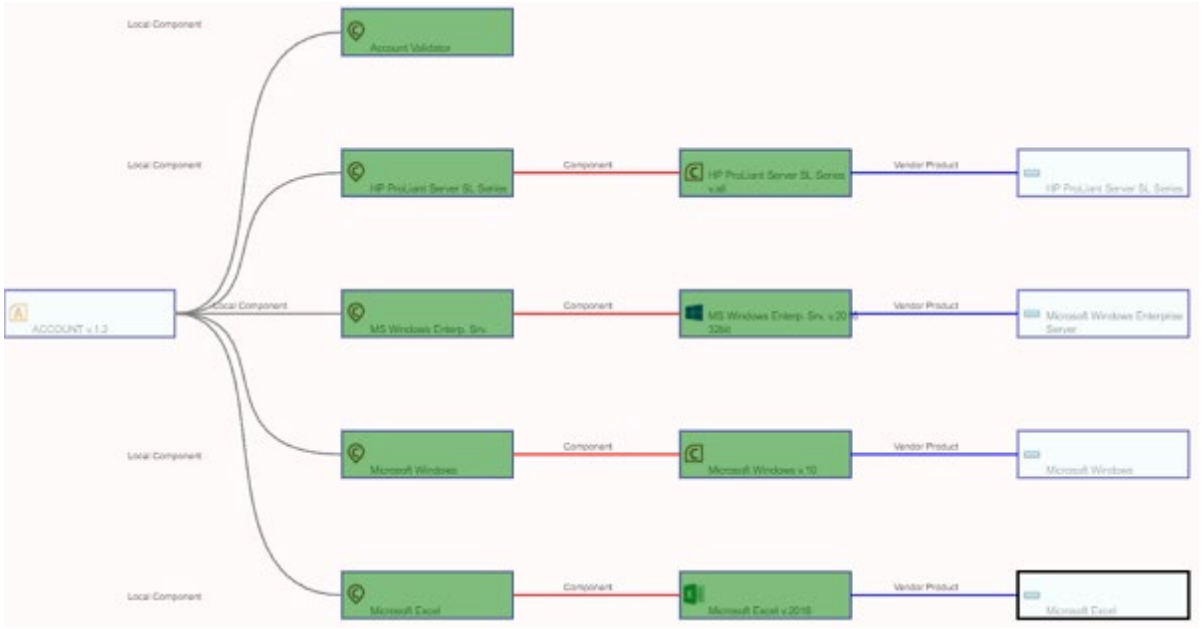
- Objects can be represented by colored circles with the a label:



- Objects can be represented by the icon configured for the object class in the class settings of the respective object class.



- Object can be represented by boxes with the label inside the box.



Branching diagram reports can also be displayed with a vertical orientation of the relation between objects. For branching diagrams with a vertical orientation the text can optionally also be displayed vertically.

FIGURE: Example of a branching diagram displaying object relations in a vertical direction

The branching diagram report can include the following options to display information for the user:

- The coloring of the circular nodes and the coloring of the lines between the nodes can be customer configured per node and line.
- A legend can be added explaining each node type.
- Text can be defined for the lines between the objects and for the nodes.

By default, the user can navigate to an object's profile by clicking a node and clicking the **Navigate** button or by double-clicking on the object node. The branching diagram report can be configured to open an alternative view if the user clicks a node in the report, For example, another configured report with the current object as base object.

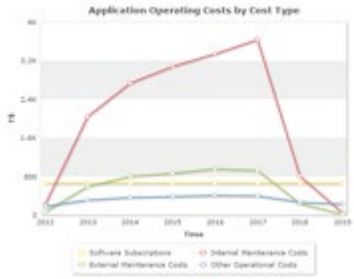
For more information about defining branching diagrams, see Defining Branching Diagram Reports.

Business Chart Reports

Business chart reports display numerical values (For example, indicator values or the number of results in a report) in a graphic. The following types of business charts are available:

Bar Charts

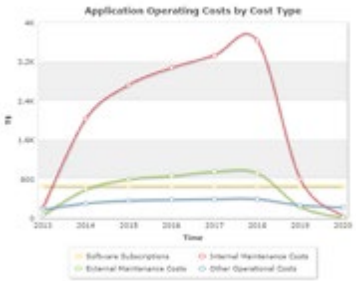
Line Charts



Pie Charts

Waterfall Charts

Spline Charts



Doughnut Charts

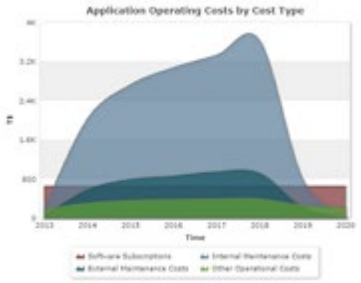
Radar Charts

Area Charts

Multi-Level Pie Charts

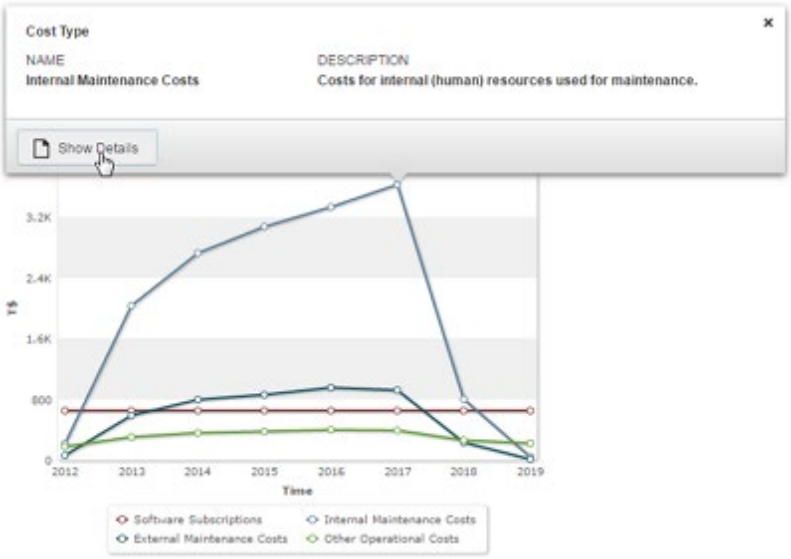
Combination Chart

Spline Area Charts



Configured business chart reports offer the following common additional features:

- The color used for objects in the chart can be changed by the user if the **Colors** button of the report is enabled or if a color selector is defined in a custom editor for the respective object class. Alternatively, the report can be configured to assign a fixed color to each graphic element in the result via a definition in the dataset the report is based on. The coloring of multi-level pie charts is an exception. It is not possible to define the coloring in the report.
- A tooltip is displayed when the user moves the mouse over a part of the graphic. The tooltip displays the combination of values that apply to that part of the chart, that means for line, spline, bar, area, spline area, waterfall and radar charts, the data series the value belongs to, the x-value and the y value are displayed comma separated.
- For all configured business chart reports except pie charts and doughnut charts, the color of the label text for values can be customized in the definition of the configured report.
- For bar, line, area, spline, spline area and combination charts, the color of the background area, and the color and thickness of the x-axis and the y-axis can be customized in the definition of the configured report. In addition, the color and thickness of the borders for area, spline area, and bar charts can be customized.
- A maximum and minimum value can be defined for the y-axis of configured business chart reports except for pie charts, doughnut charts and and multi level pie charts. By default, the y-axis starts with zero and the maximum value is evaluated from the data displayed in the chart. The definition of a fixed maximum and minimum value for the charts results in a fixed axis definition that is usefull for direct comparison of multiple chart reports displayed next to each other, For example, in an object cockpit. If an y value in a chart report is outside of the configured range for the y-axis, the configured maximum or minimum value is ignored to ensure that all data is displayed.
- Navigation from the configured business chart report is optionally available to open either the object profile of an object represented For example, by a bar in a bar chart or a sector in a pie chart, or to open any object view, standard Alfabet view or configured report defined for the respective result in the underlying query of the configured report. If navigation is activated, the user can click on a result displayed in the business chart to open a preview window. If the result represents an object, the preview window provides additional information about the object and displays a **Show Details** button that opens the object profile:



Otherwise the information is limited to the information about the current series:

Please note that for activation of navigation, you must click the part of the report representing a single result. For an area chart or spline area chart, this is any point representing a value on the border of the area. Clicking into the area itself will not activate any navigation.

For a detailed description about how to configure a business chart report, see Defining Chart Reports.

Bar Charts

Bar Charts can display results in bars that are either vertically or horizontally orientated.

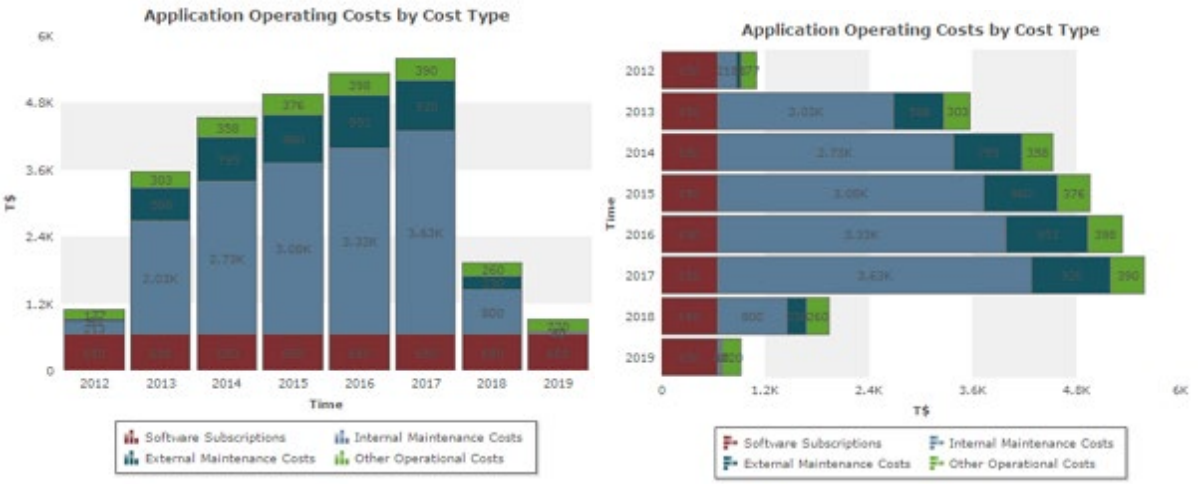


FIGURE: The same bar chart in vertical and in horizontal direction

In addition to the x and y-axis, bar charts can have a third dimension. Multiple series of results can be defined. The bars with the result for each serie can be displayed next to each other for each x value or on top of each other:

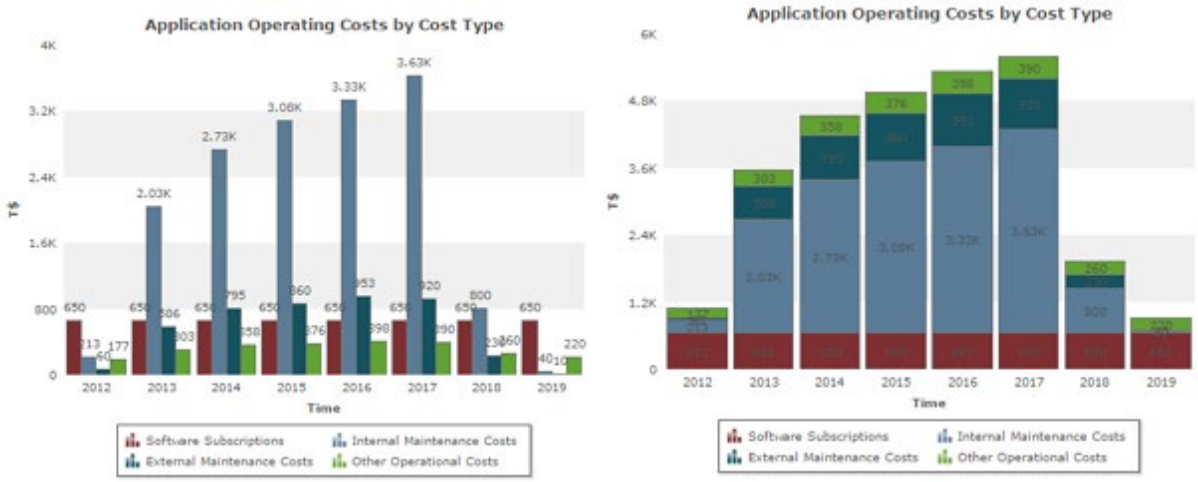


FIGURE: A same bar chart with independent and with stacked results

A legend beneath the report displays all series of bars that are visible in the bar chart. Clicking on a series in the legend removes the series from the chart. Clicking on the legend item will re-integrate the series into the report.

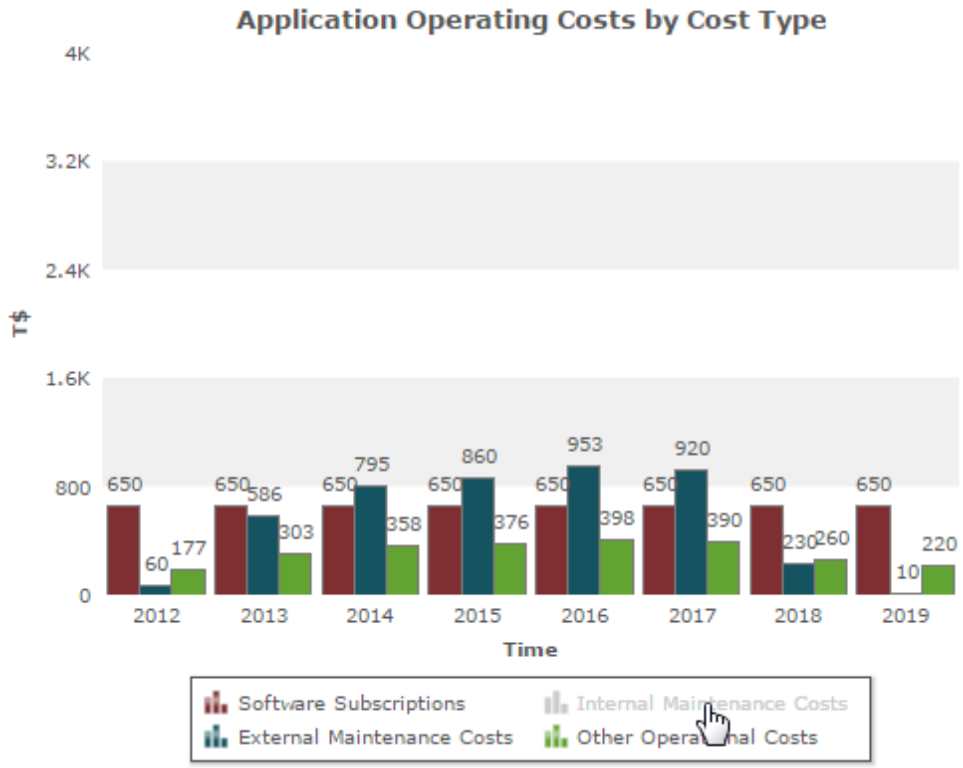


FIGURE: The bar chart in the example above with reduced dataset

Waterfall Charts

Waterfall charts display results for one or multiple series of data in a cumulative bar chart. While the first bar in the report starts at the baseline, the next bars are then each starting at the endpoint of the previous bar, cumulating the results of both bars. The transition of one bar to the other is optionally marked by a dotted line. The following figures show the same report both as a bar chart and as a waterfall chart to depict the differences between the two types of report. The example report shows both costs and income resulting from a project in a time line. In the bar chart, the costs and incomes are displayed, but the values are not related with each other. As a result, costs are always displayed in the negative range and incomes in the positive range:

In the waterfall chart, the graphic shows that in the first years the costs are exceeding the income, resulting in the columns of the report to be displayed in the negative y range, while at the end of the project, the sum of all costs and incomes results in a positive value:

The coloring of the bars in a waterfall chart can either be defined by color definition in the underlying query of the configured waterfall chart report to define coloring per bar or by defining a fixed color for positive and for negative values.

SUM columns can be added to the report at any position. The SUM colum can either display the SUM of all values between this SUM column and the last SUM column left to it, or a cumulative SUM column can be added that sums up all values of all columns displayed left to it no matter how many SUM columns are

included. If SUM columns are included into the report, they are not taken into account as column values by the cumulative SUM calculation.

In the follwing example, the costs and incomes of a project are summed up for each year (grey) and at the end of the report, a cumulative sum of all project costs and incomes is displayed in dark green:

Line Charts And Spline Charts

Line charts and spline charts show one or multiple series of values as lines connecting the y values in an x and y-axis diagram. While line charts connect the values with straight lines, spline charts draw a curve through the values:

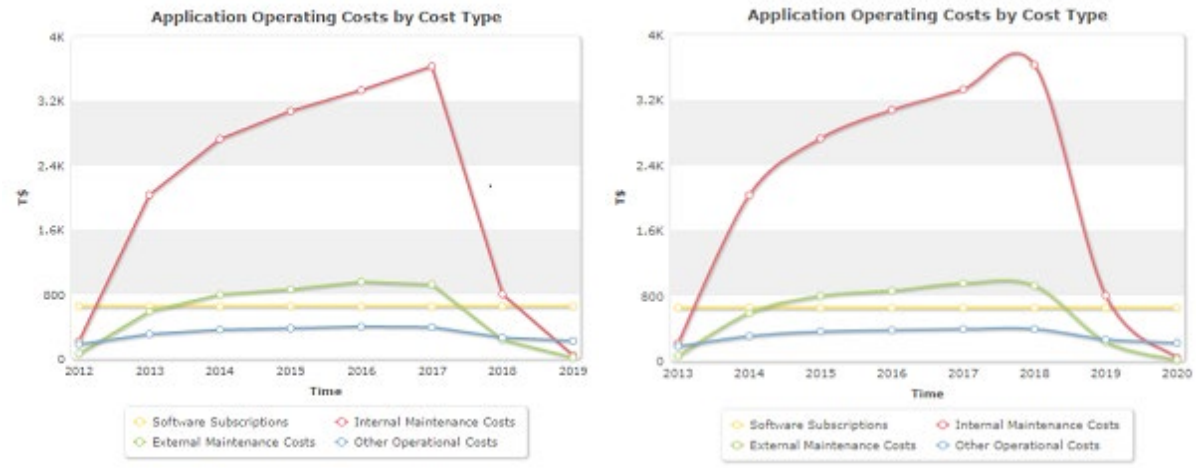
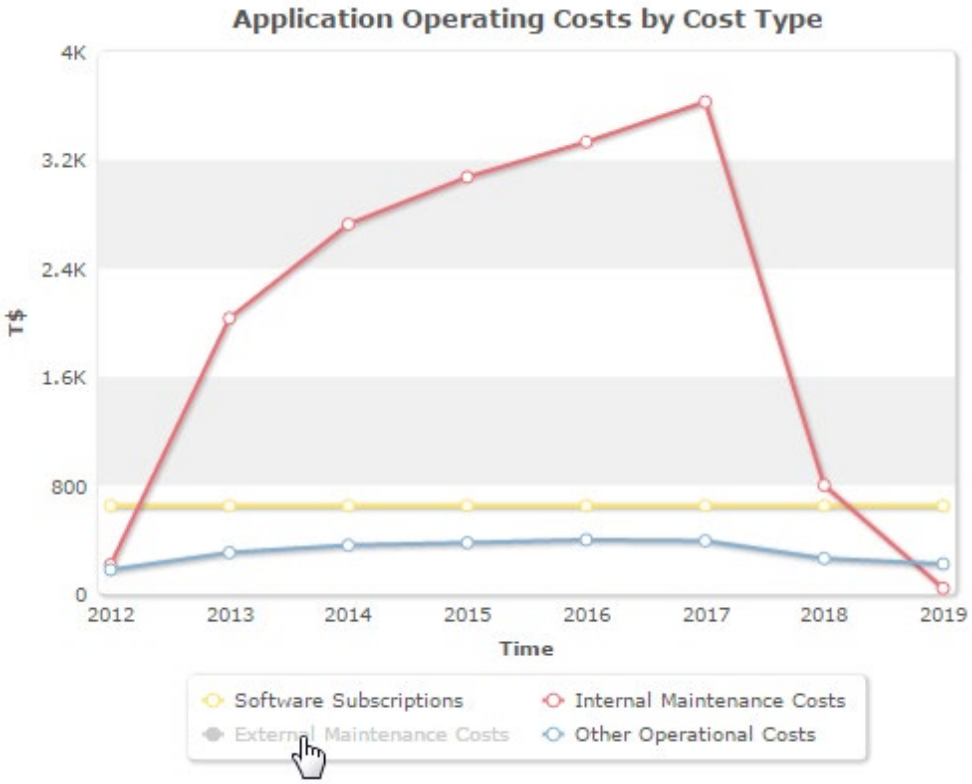


FIGURE: A line chart (left) and a spline chart (right) displaying the same data

A legend beneath the report displays all lines that are visible in the line or spline chart. Clicking on a series in the legend removes the series from the chart:



The values of the lines are independent from each other. Removing one series does not alter the results for the other series. Clicking on the legend item will re-integrate the series into the report.

Area Charts and Spline Area Charts

Area charts show one or multiple series of values as areas. The boundaries of an area connect the y values of a data series in an x and y-axis diagram. While in area charts the boundary of an area connects the values with straight lines, spline area charts draw a curve through the values as boundary of the area:

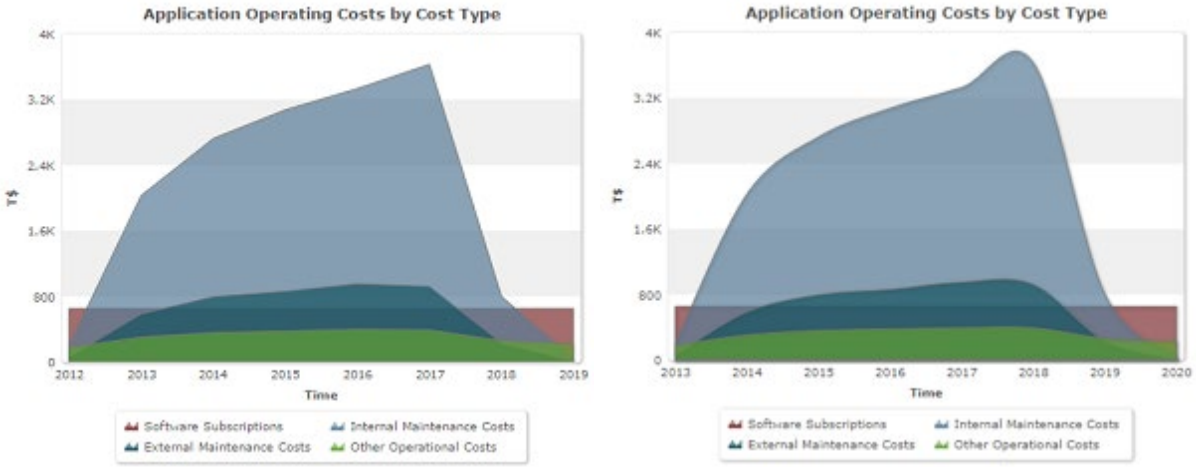


FIGURE: An area chart (left) and a spline area chart (right) both displaying the same data

In area charts, results of multiple series can either be displayed stacked on each other or independent from each other.

If the areas are displayed independent from each other, the areas overlap each other. Therefore, colors are displayed with opacity:

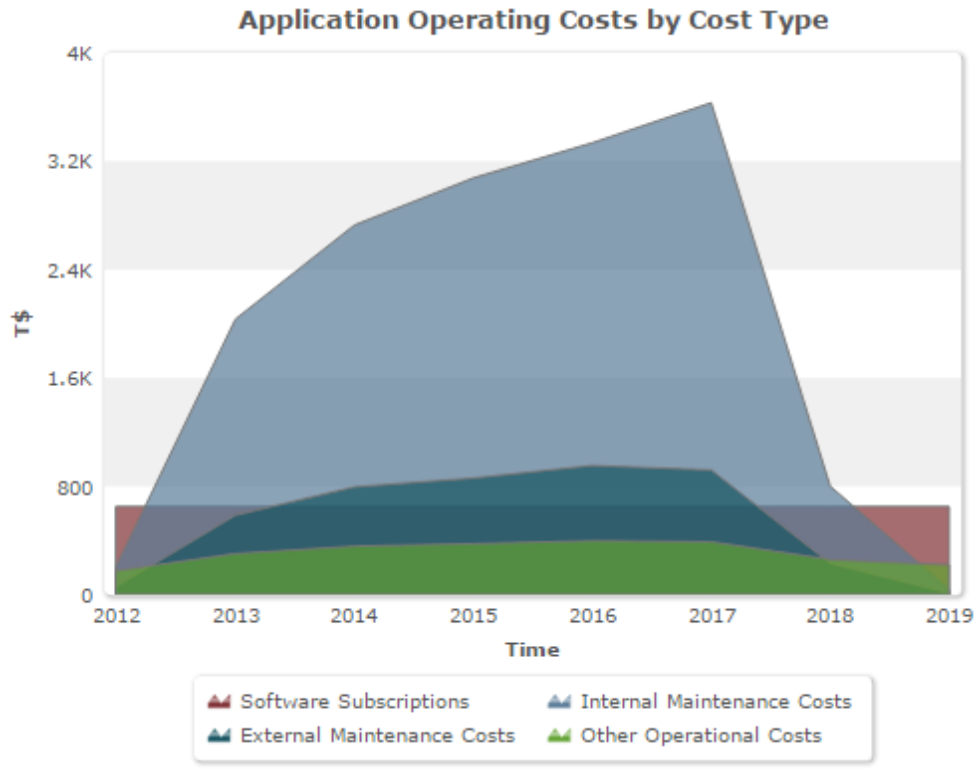


FIGURE: Example of an area chart with independent area

Stacked display means for two series, the first having a value of 3 and the other one a value of 2 for an x value, that the first series ends at the y value 3 and the second one at the y value 5 (3+2). Stacked area charts are displayed with solid colors:

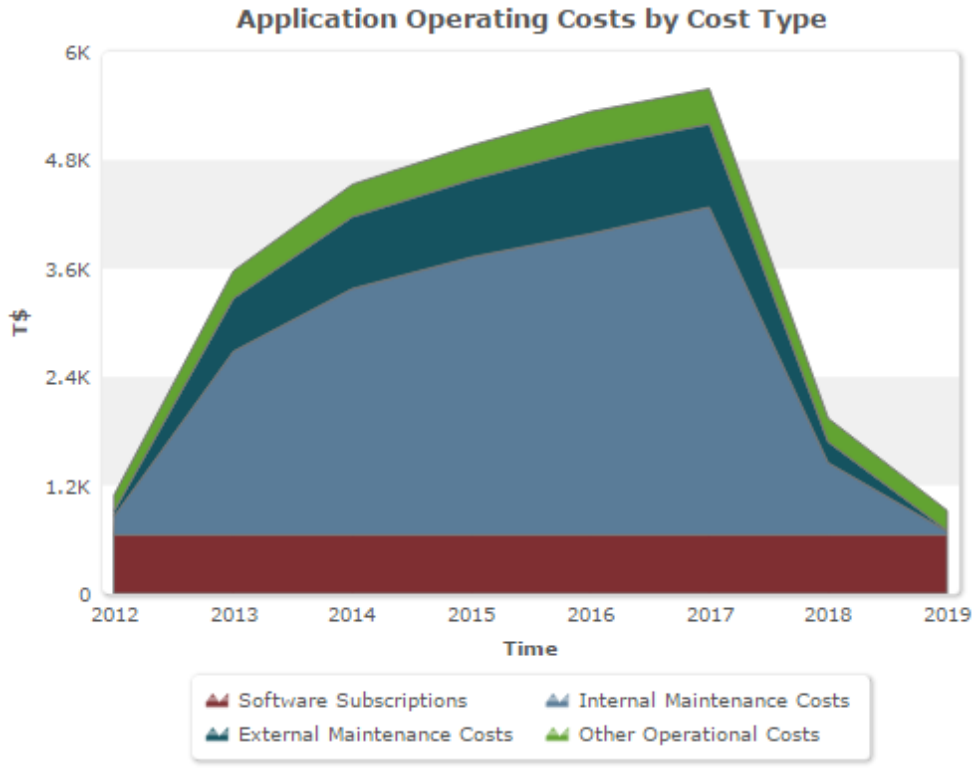


FIGURE: Example of an area chart with stacked areas

Spline area chart only render results independend from each other. The stacked display is not available.

A legend beneath the report displays all areas that are visible in the area chart or spline area chart. Clicking on a series in the legend removes the series from the chart. If an area chart displays stacked areas, removing one series will alter the sums displayed for the y values:

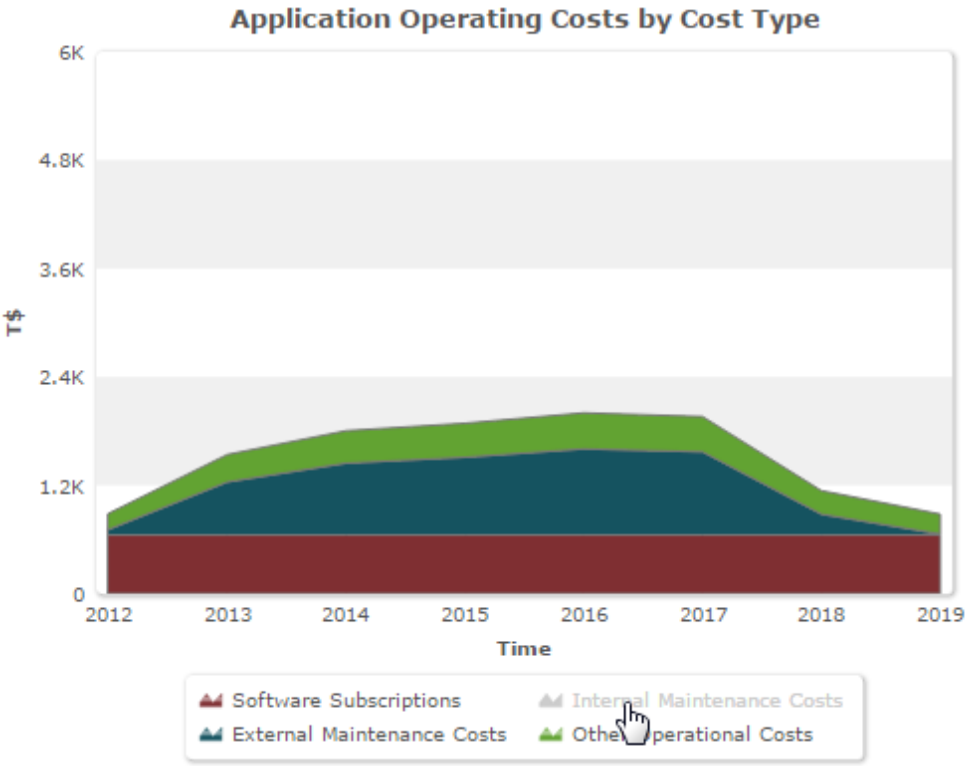


FIGURE: Stacked area chart with one area hidden from the report

Clicking on the legend item again will re-integrate the series into the report.

Pie Charts and Doughnut Charts

Pie charts and doughnut charts display an x and an y dimension. Each value of the x-axis is displayed as a slice of a pie or doughnut. The sizing of the pie or doughnut is evaluated in relation of the y-axis values for each slice.

The sum of all y values, that means the total sum represented by all slices in the pie can be displayed as subtitle of the graphic as part of a configurable text. In doughnut charts, the subtitle is displayed in the middle of the doughnut:

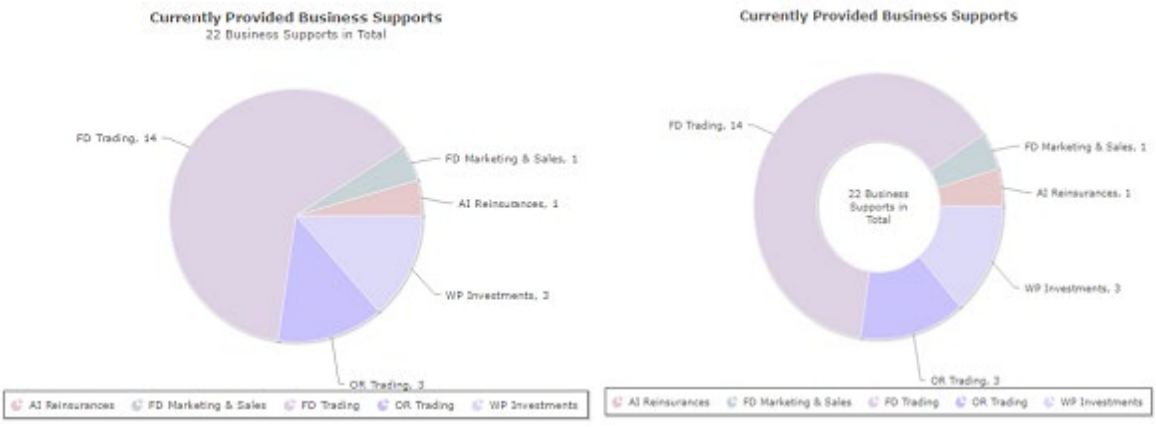


FIGURE: Example of a pie chart and a doughnut chart displaying the same data including a total sum as subtitle

Beneath the pie or doughnut, a legend informs the user, which object or value is represented by the slice. If the user clicks a value in the legend, the respective slice is moved out of the pie or doughnut. Clicking on a slice will also move the slice. The slice is moved in again by repeating the action.

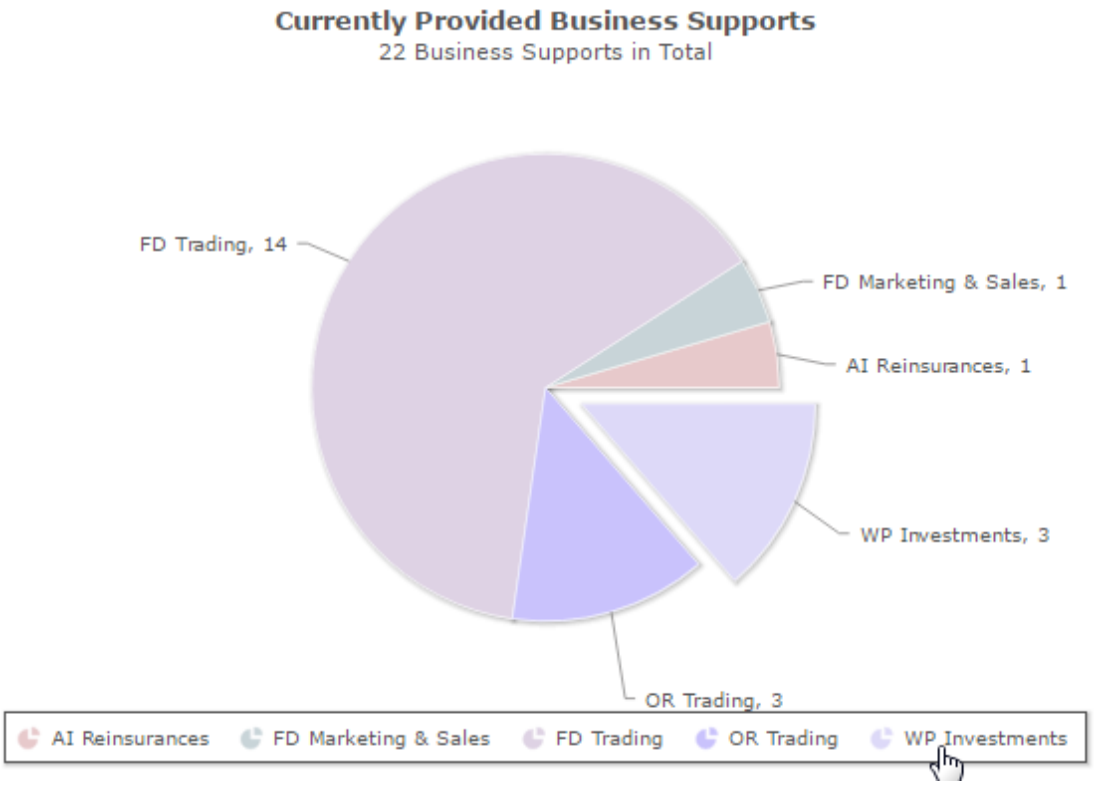


FIGURE: Example of a pie chart with one slice highlighted

Multi-Level Pie Charts

Multi-level pie charts display a pie split into multiple rings. Each ring displays an x and an y dimension whereby the x dimensions of the rings are related with each other. For example, the middle area of a multi-level pie chart can display applications that are part of a project architecture and the next ring displays the business support provided by each application.

If no data is available for a level, the level or part of the level are not displayed:

FIGURE: A pie chart where second level data is not available for some objects of the first level

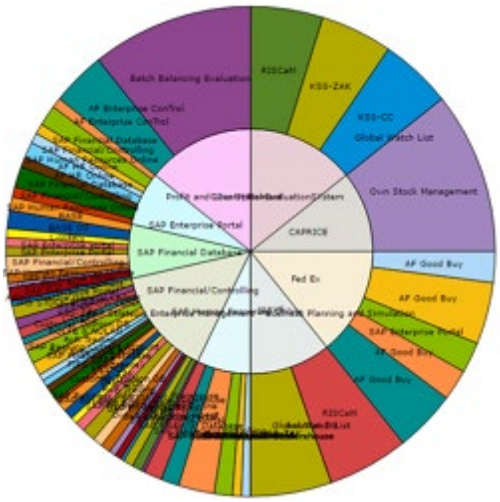
The y dimension is optional for multi-level pie charts. If no y dimension is defined, the objects are equally distributed.

FIGURE: The same multi-level pie chart report with and without y dimension

The distribution of the objects in the center of the multi-level pie chart depends on the defined y dimension only. Therefore, the sizing of the slices of the pie is directly giving information about the relation of the y values within the level. In the second level of the multi-level pie charts, objects are also sized according to the y dimension value, but the space available for the objects depend on the sizing of the element of the level below that the object belongs to and on the number of objects that are assigned to the same parent

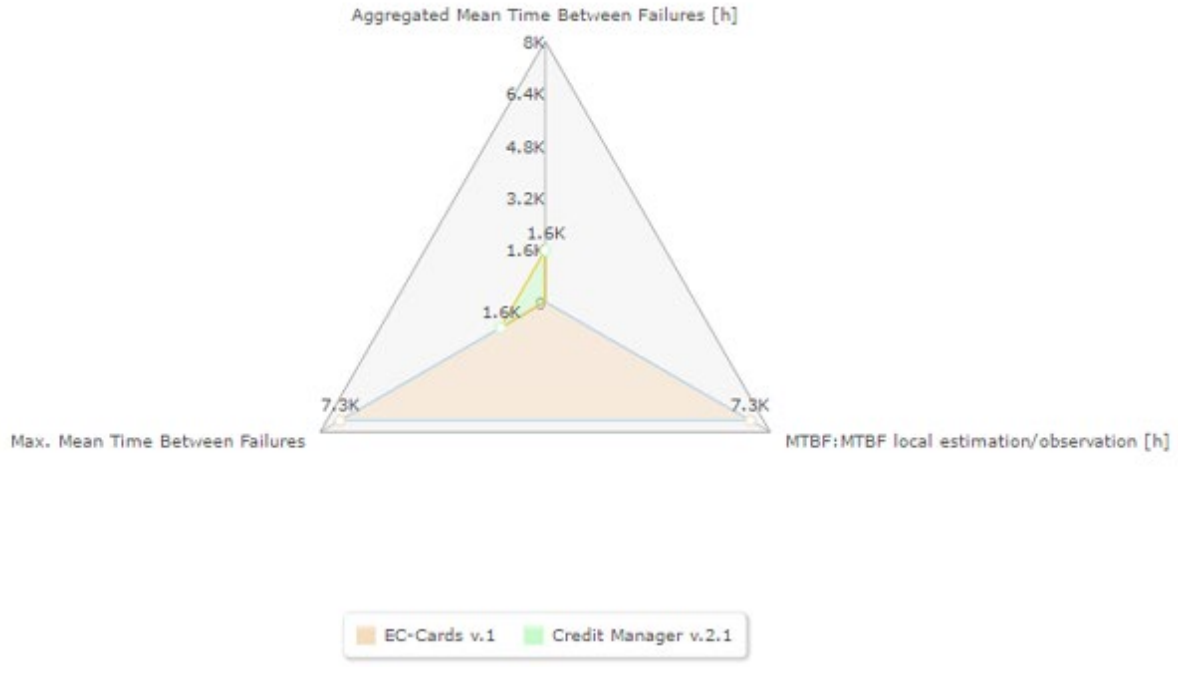
element in the inner level. Therefore, two objects with the same y value can be sized differently in the outer level.

In the following example, the y dimension has four values: 1, 2,3,4. While the sizing of the inner level is done according to the y dimension, the outer level displays objects on the left of the chart in very small slices, because multiple objects are assigned to a first multi-level object, while in the upper right part, the slices are much bigger because only up to three objects are assigned to the first level object. More than 4 different sizes are used for objects of the second level.

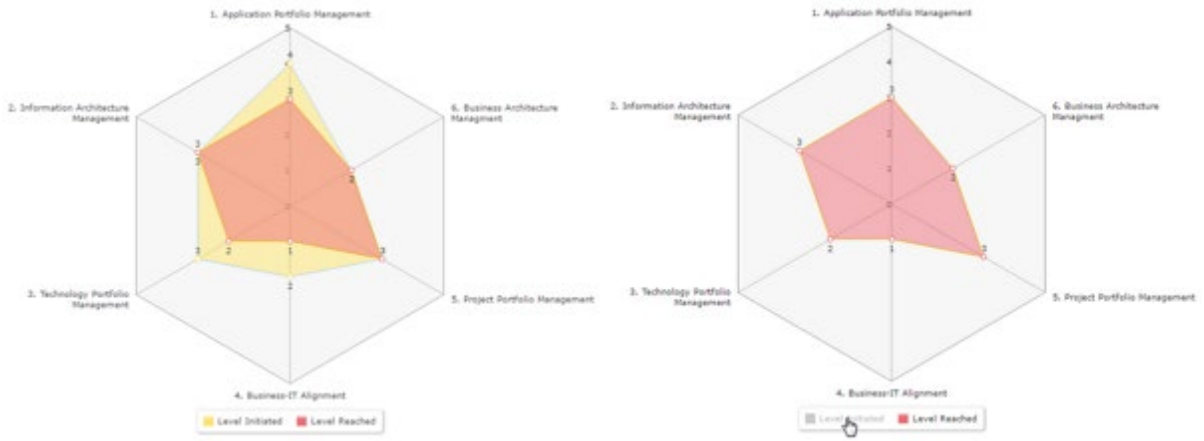


Radar Charts

Radar charts display multiple series of y values for an x value each on a different axis that start at the same point. The y values for the same x value are connected to each other and span a net each in a different color. For example, multiple indicators for applications can be displayed, with each indicator on a different axis and the indicator values for each application beeing connected:



A legend beneath the report displays all value nets that are visible in the radar chart. Clicking on an entry in the legend removes the value net from the chart:

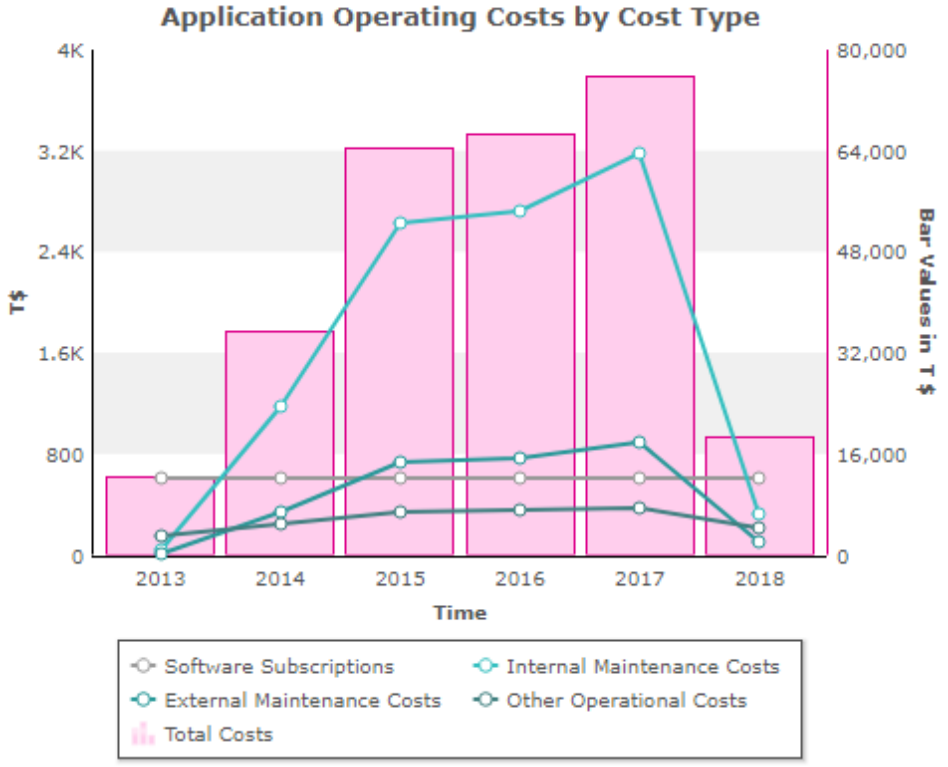


Combination Charts

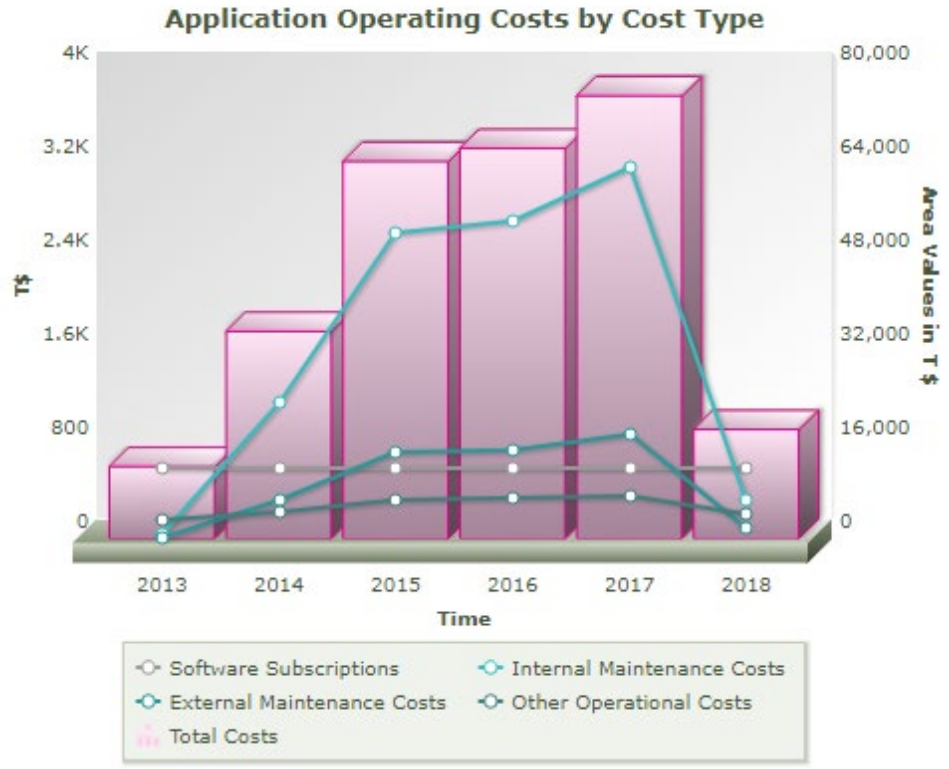
Line, area, and bar charts can be combined in a combination chart. The combination chart offers the following features for display of different kind of charts in the same graphic:

- Business charts of different chart types can be combined, For example, some series can be displayed as a bar and some as a line.
- Two different y axes can be defined, For example, to combine charts that require a different scale because the series are having different ranges of values or units of measure.
- For a better graphic display of the combined chart types, 3D effects can be added to the configured report. Only line and bar charts can be displayed in 3D. If the configured report contains areas, these are converted to bars.

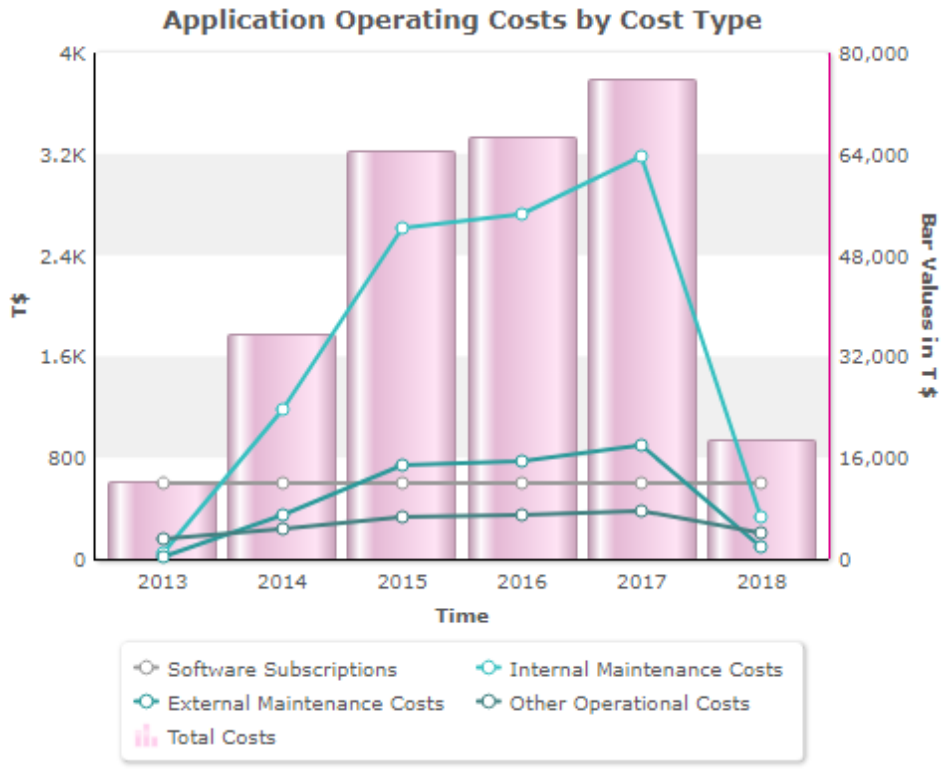
The following configured report shows the different types of costs for an application over the years as lines while the sum of the costs is displayed as bars with a separate axis with a higher range than the one for the single cost types:



The combination chart can alternatively be displayed in 3D-Design. Please note that coloring of the second y-axis is not visible in 3D design:



A 3D effect can alternatively be applied within the bars only:



Circular Roadmap Reports

Circular roadmap reports display objects as items in a circle that is divided both in horizontal and vertical orientation.

From the center of the circle until the border of the circle a time scale or a numerical scale is established, that is optically visible by dividing the circle into differently couloured shells. In addition, the circle is divided into sectors that typically represent objects related to the object elements in the circle. For example, if the objects displayed as items are applications, the sectors may represent domains that the applications are assigned to.

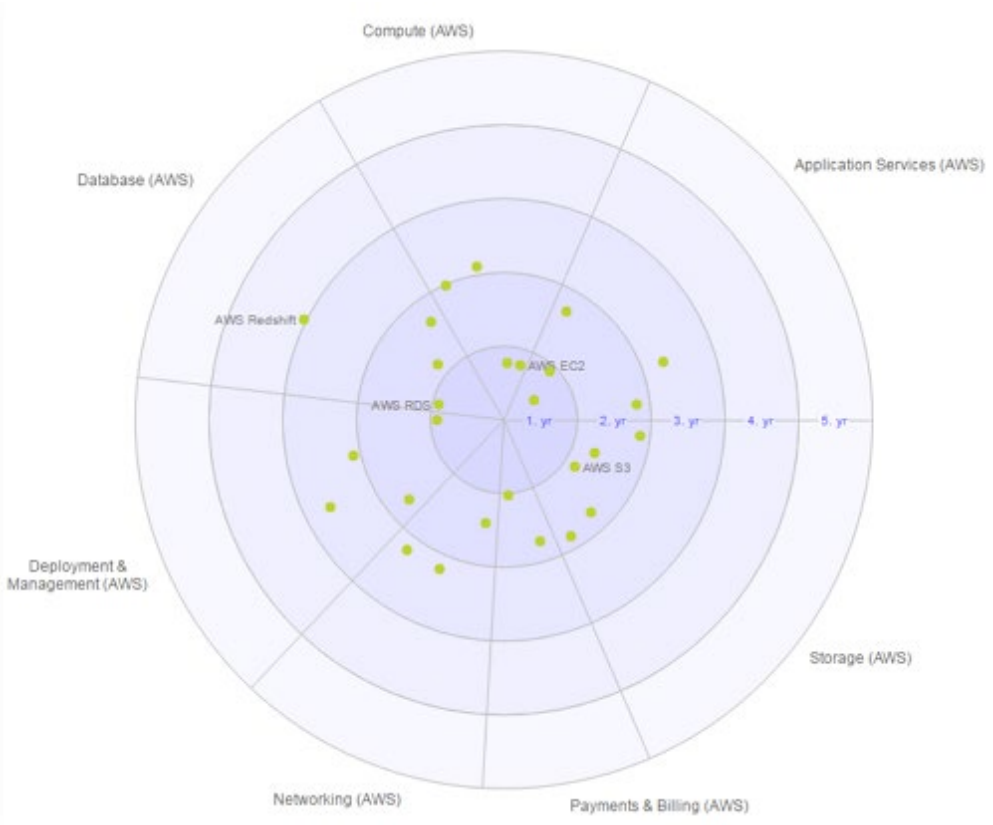


FIGURE: Circular roadmap displaying components placed into the circle according to their lifecycle end date and domain assignment

The object items are defined with a relation to both the shells and the sectors and are placed within the circle according to the values that they have on the shell scale and for the sector placement. To avoid placement of two object items on top of each other if they have the same value for shell and sector placement, a relative position from the middle axis of the sector can be defined for each object.

The items in the report can be represented by either graphic shapes, that may vary in size, color and shape according to defined criteria, or icons can be used to display the items.

In addition, relations between objects, like For example, versions of the same application or component succeeding each other, or information flows between components, can be displayed as arrows within the graphic.

FIGURE: Circular roadmap report displaying components as icons and arrows connecting successor versions

The circular roadmap report can provide navigation from both items and connecting arrows.

Gallery Reports

A gallery report displays data about objects in the Alfabet database as a gallery of identically sized graphic boxes:

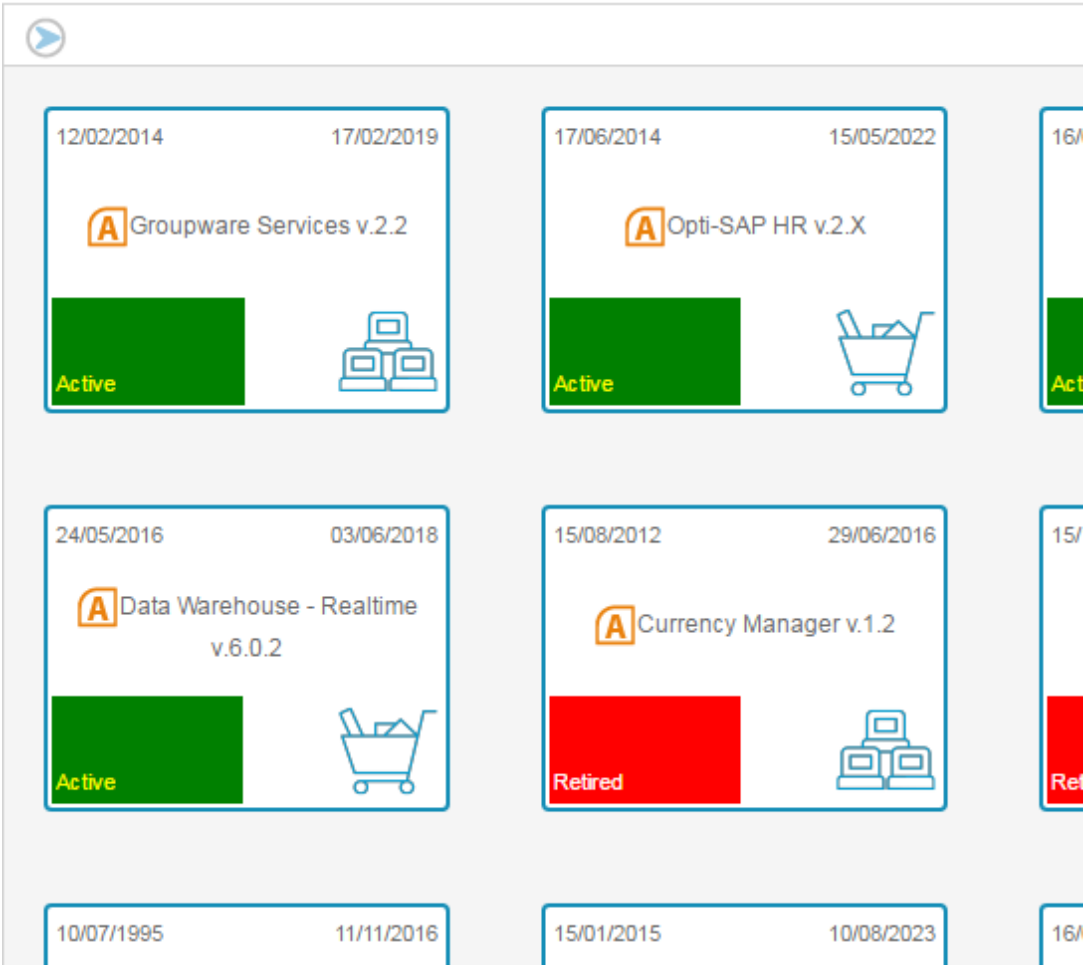


FIGURE: Part of a gallery report displaying the applications the current user is resonsible for

The boxes display configurable text in the center of the box. Typically the name of the object represented by the box is displayed in the center. Optionally, the icon of the object class the object belongs to can be displayed on the left of the text. The color of the text, the box and the border of the box can either be configured to be identical for all boxes or to depend on values from the return dataset of the underlying query of the configured gallery report.

In addition, four different attribute values related to the object can be displayed in the corners of each box. The information can either be displayed as plain text or as text on a colored background or represented by an icon. In the example above the application type information displayed in the lower right corner of each box is represented by an icon. A legend is available for the report that explains the meaning of the colors and icons used in the report.

The user can navigate ot the object profile of the object represented by a box in the configured gallery report by double-clicking on the box. Optionally, the configured report can be configured to open any other view of the Alfabet user interface instead of the object profile via double-click on an object box.

Node Arc Reports

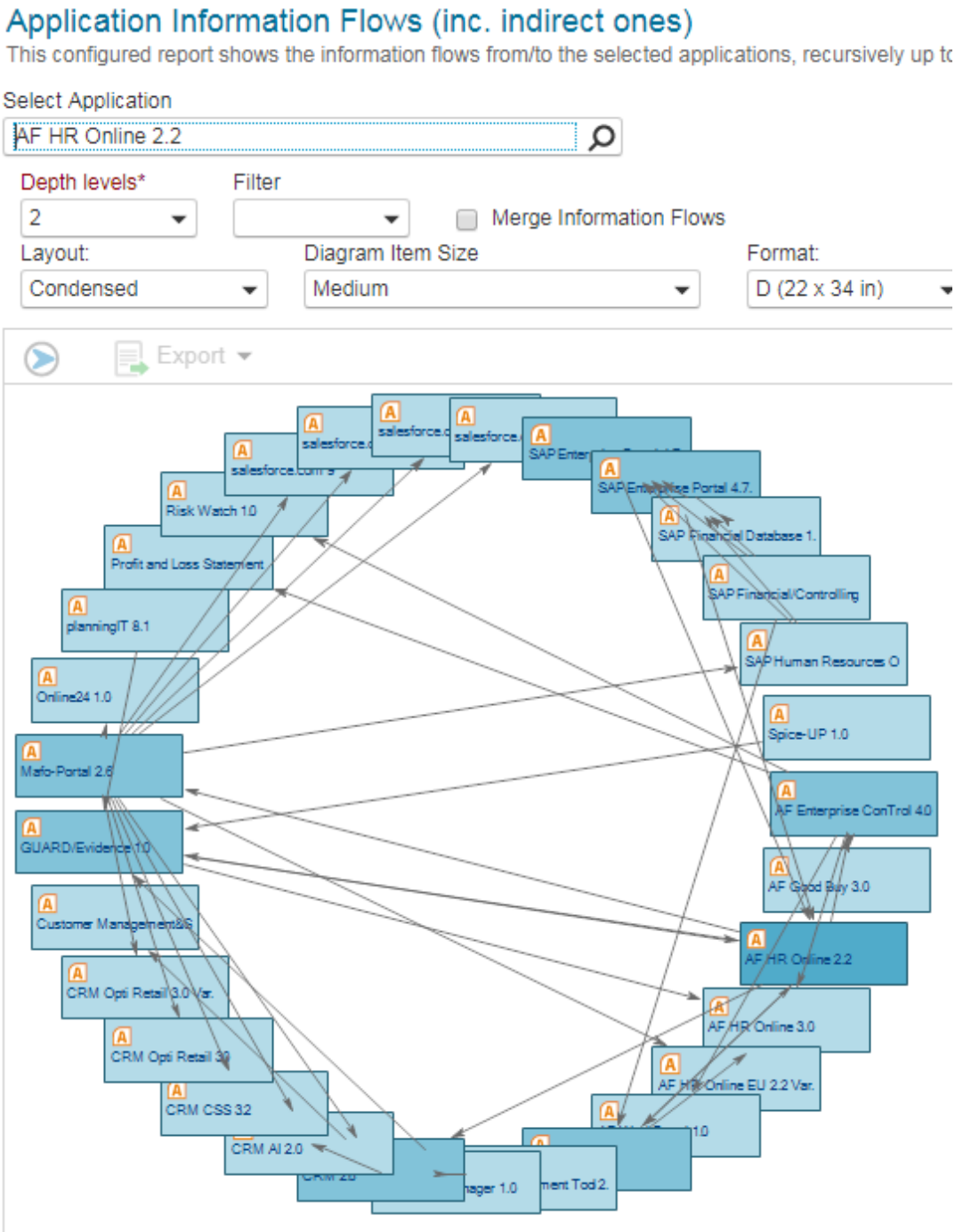
Configured node arc reports display connections between objects in the diagram layouts that are also available for standard Alfabet diagrams that were defined with the Alfabet Diagram Designer. Objects are displayed in object boxes that are based in design on the diagram item templates that can be customer

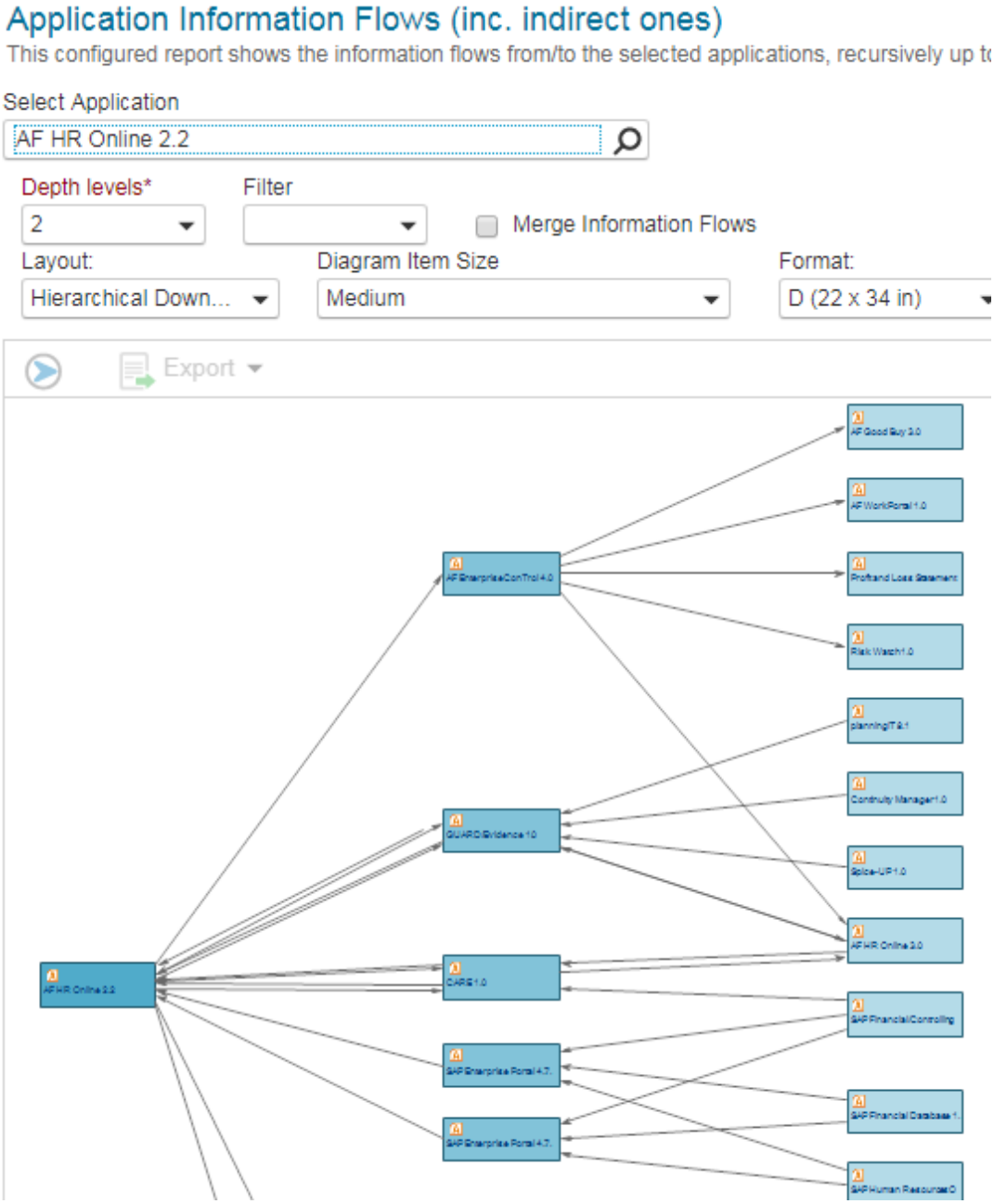
defined for use in diagrams. There are two different types of node arc reports which are different in the final layout options for the user:

- Node arc reports based on the template `NodeArcReport`:

The way boxes and connections between boxes are displayed is dynamically rendered at runtime according to the diagram layout selected by the customer. The filters that are available for all diagrams in standard Alfabet views are also added to the node arc reports designed by the customer. The standard filters can be set to hidden when the user should not be able to alter the design of the node arc report.

The following example displays the same node arc report with two different layouts applied:

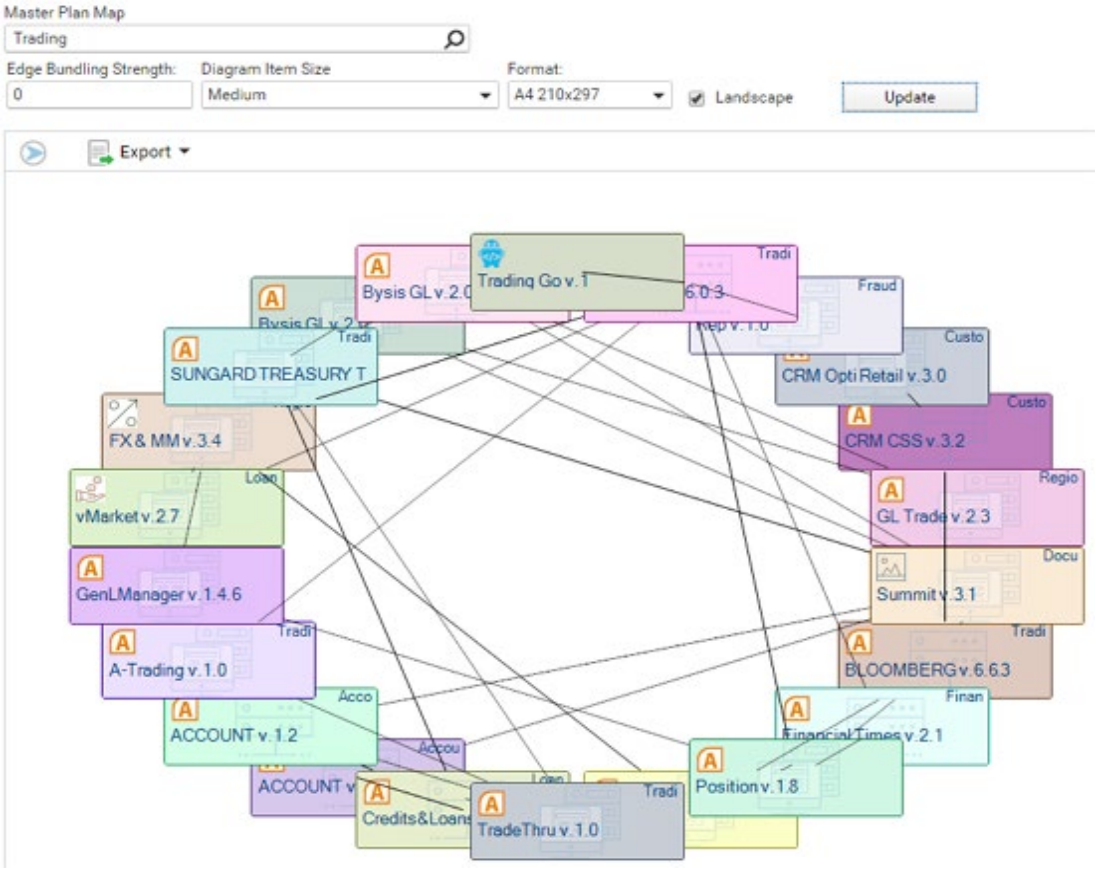


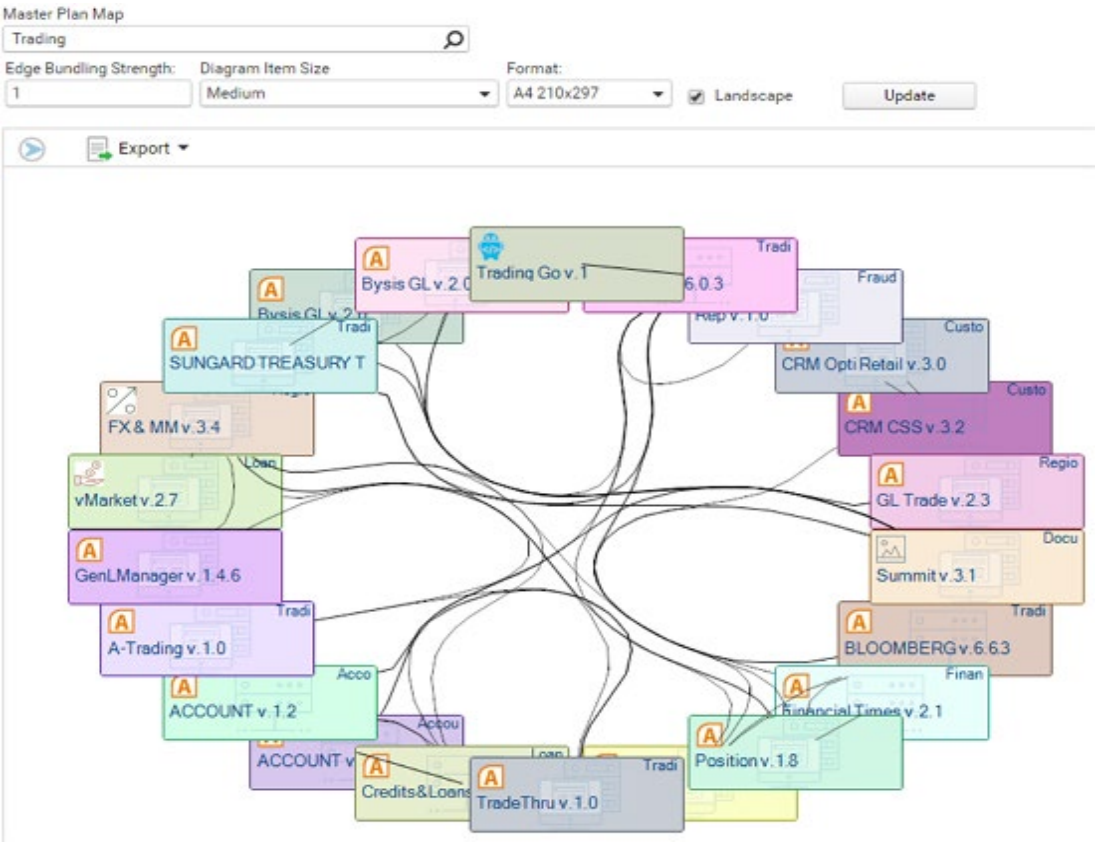


- Node arc reports based on the template `NodeArcReportWithBundling`:

Boxes and connections between boxes are displayed in the `Spring Layout` with connections beeing bundled to enhance the general overview for diagrams with a high number of connections. The user can decide at runtime whether and how much edge bundling is performed on the connections. The filters that are available for all diagrams in standard Alfabet views are also added to the node arc reports designed by the customer except for the **Layout** filter that is substituted with the **Edge Bundling Strength** filter. Edge bundling can be defined with a rational number between 0 and 1. Edge bundling of zero is identical with the normal condensed diagram layout with straight lines. Edge bundling strength will increase with a maximum at 1. The standard filters can be set to hidden when the user should not be able to alter the design of the node arc report.

The following examples show the same node arc report with edge bundling set to zero and to one:





The line style for connections and the design of the boxes representing objects in the configured report is highly configurable to provide additional information by using different coloring, line styling or labels in boxes.

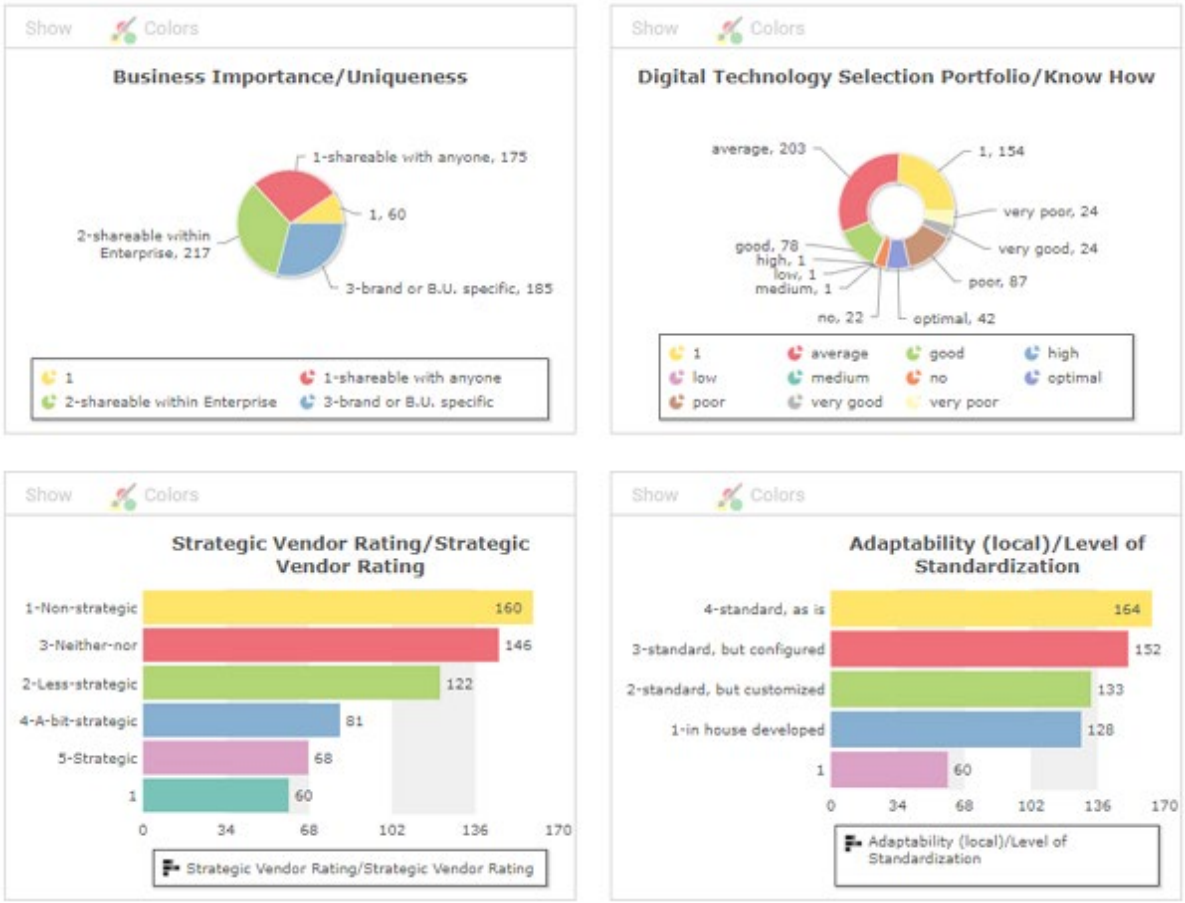
By default, navigation from the boxes to the object view of the object is enabled. Optionally, the node arc report can be configured to provide navigation from the object boxes to a standard page view or configured report instead of the standard navigation behaviour. If the connections between the boxes represent objects, navigation can be enabled for the connections as well. The user must then click either end of the connection and click the **Navigate** button. Clicking in the middle of the connection or the connection label does not enable the navigation option.

For information about configuring node arc reports, see Defining Node Arc Reports or Node Arc Reports With Edge Bundling.

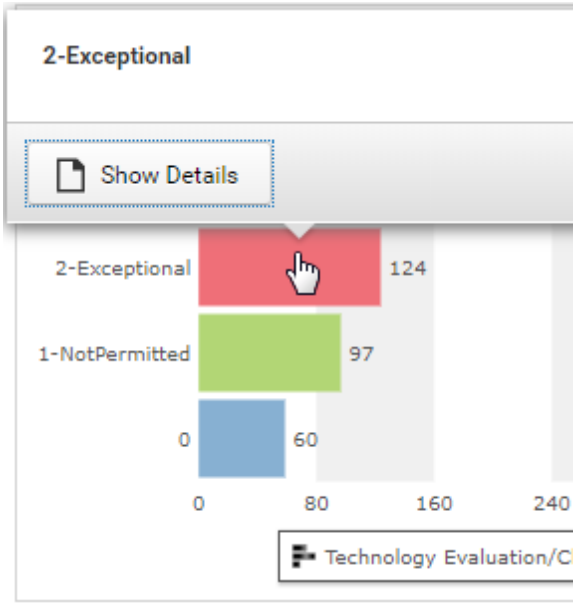
Portfolio Diagnostics Report

Portfolio Diagnostics reports use variance and factor analysis of a wide scope of measures to understand pain points in the IT architecture. The portfolio diagnostics report uses Software AG's augmented AI technology to provides users without prior knowledge of the details of the IT landscape with an entry point to explore and understand which aspects of the IT architecture may require change. Portfolio diagnostic reports are configured reports that analyze all or a subset of relevant indicators and numeric property settings for a defined object class. The report scans indicators and measures for inconsistencies to point the user to objects with a high deviation from the company's standard. Users can drill-down through report results for a deeper understanding where problems exist in the IT architecture and identify where action is needed.

The first level of the report uses variance analysis and displays the results for the indicators and measures that show the highest score of inconsistency in value settings. For each of these indicators and measures, a chart provides information about the number of objects with the occurring value for the indicator or measure:



The user can click-and-hold an area in one of the charts and click **Show Details** in the preview window to navigate to the next level of analysis:



The reports on all subsequent levels are factor analyses that take only the objects represented in the selected bar into account. The analysis searches for indicators and measures with values that closely correlate to the values of the indicator or measure in the previous selection. The indicators and measures with the highest correlation are displayed as charts with the number of objects per value. The user can then

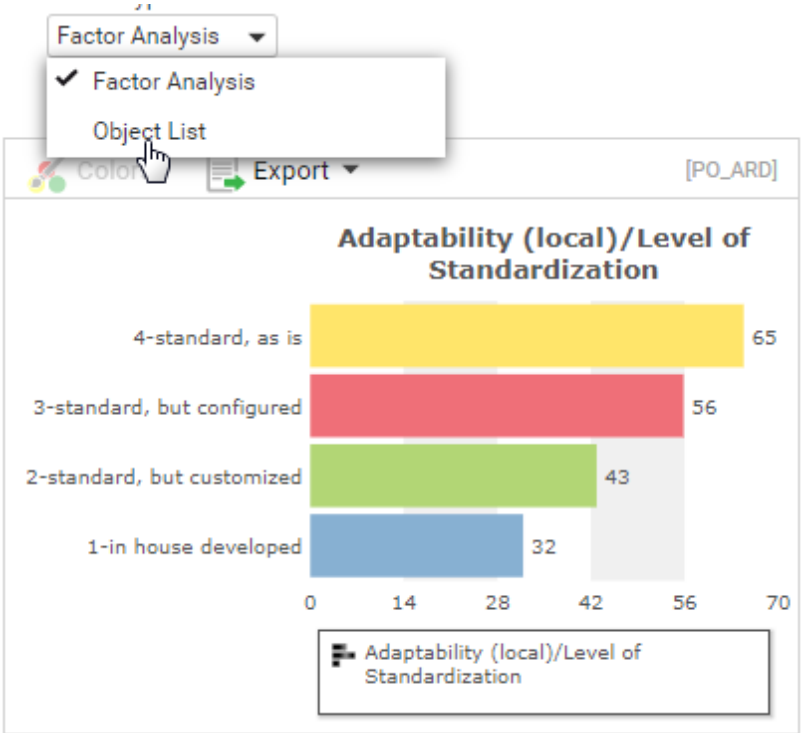
continue to drill down to the next level of factor analysis, whereby each level will display a smaller number of objects with the same combination of potentially problematic indicator settings. The selections made on each level are displayed as a text above the report.

The kind of chart that is displayed depends on the number of results to display. The following special rules apply:

- If results are displayed in a horizontal bar chart, the results are sorted top to bottom by Y-value, which means that For example, for an indicator, the order of the bars does not follow the order of the allowed indicator values from the lowest to the highest score, but the number of objects that are set to the indicator value.
- If results are displayed in a pie chart or doughnut chart, the results are sorted alphanumerically by X-value.
- If there are more than 20 values, X-Values are combined in buckets. The number of buckets is configurable for the configured report. The type of chart displayed depends on the number of buckets defined for the report. For example, for a report displaying 30 different X-values, representing values from 1 to 30, and a number of buckets of 10, the result dataset will display 10 results, one combining all Y-values for the X-values from 1 to 3, one combining all Y-values for the X-values from 4 to 6, and so on.

The user can exclude charts that are not regarded as relevant for analysis. To exclude a chart, the user must select the chart caption in the **Critical Measures** field above the, click the **Exclude** button, and then click the **Submit** button. If the user wants to re-include an excluded chart, he/she can select the excluded chart in the **Ad-Hoc Excluded Measures** field, click **Reinclude** and then click **Submit**.

At any point, the **Object List** option can be selected in the **Action Type** field above a report prior to opening the next level.



The next level will then be a dataset showing the relevant objects and their indicators and measures that were selected in the course of the analysis. The user can drill-down to the object profile/object cockpit of a selected object in order to understand the details about the object.

If a group infrastructure exists for the objects shown in the list, like For example, application groups for applications or component groups for components, the user can select one or multiple objects in the list and either assign them to an existing group or create a new group and add the selection to the new group in a single step. Both actions are available via the button **Action**. The **Action** button is currently available for the following object classes:

- Application
- Business Process
- Component
- Contract
- Cost Center
- Demand
- Device
- Domain
- ICT Object
- Policy
- Market Product
- Migration
- Organization
- Peripheral
- Project
- Service Product
- Technology
- Threat
- Value Stream
- Vendor
- Vendor Product

For information about how to create a portfolio diagnostics report, see Defining Portfolio Diagnostics Reports.

Treemap Reports

A treemap report shows Alfabet objects from the perspective of a specified object class. The configured report is structured in columns that represent an object class that is associated with the specified object class. For example, a configured report can show application groups in the column headers and the applications assigned to the respective application group as objects in the columns.

Each object in the configured report is displayed as a box with a configurable size, color and indicator icon in the object box. The color, size, and indicator symbol represent dimensions of object evaluation configured by the report designer.

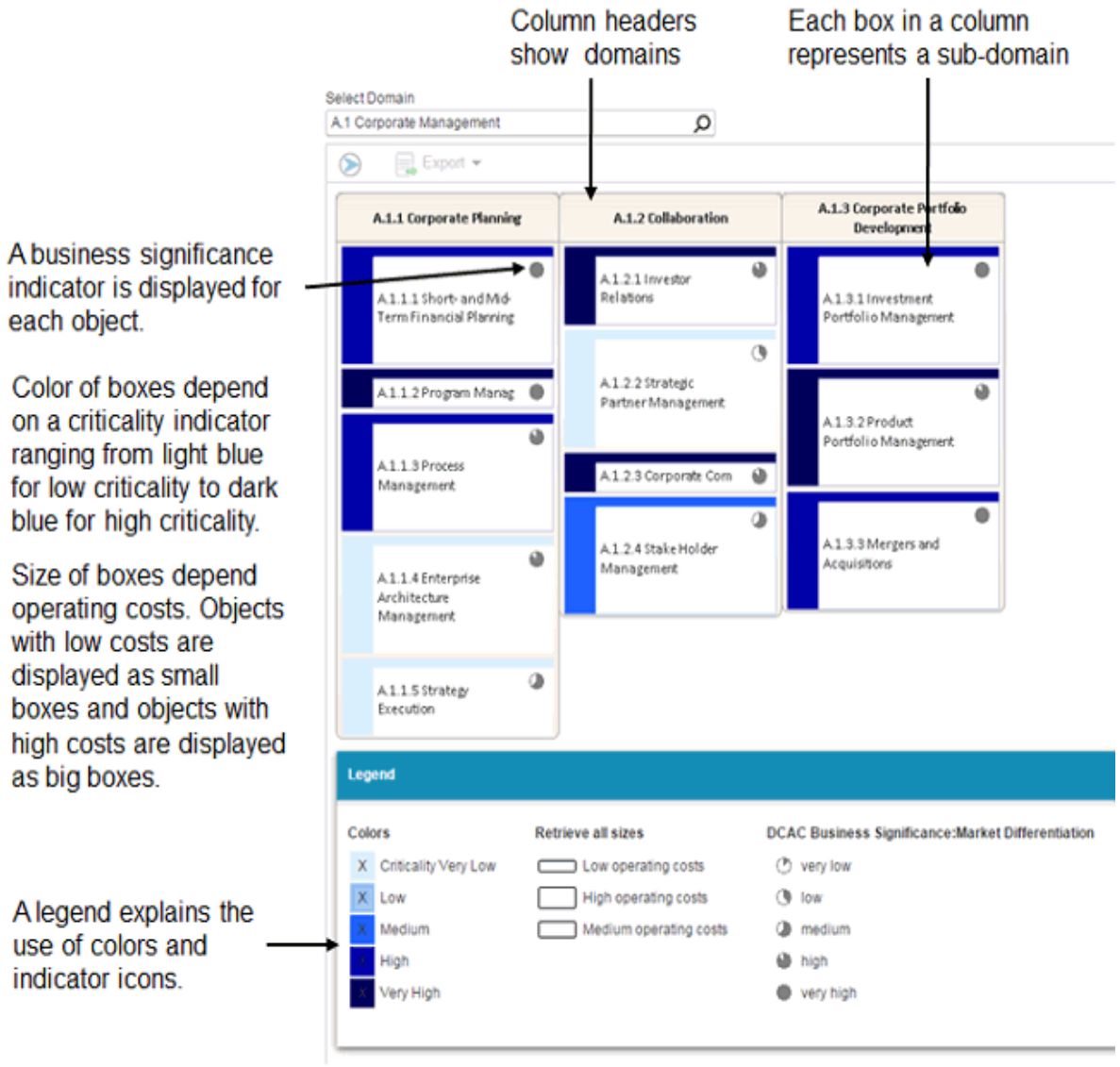


FIGURE: Example for treemap report displaying sub-domains sorted by parent domain.

In the example above, a domain with high criticality is displayed in dark blue whereas a domain with low criticality is displayed in light blue. The size of the domain represents its operating costs. The lower the operating costs, the smaller the box representing the object in the configured report. The indicator displayed on the box highlights a third aspect that is critical to the evaluation of the domain. In this case, a risk assessment for the domain is visualized.

The size, color and indicator are assigned automatically to the displayed objects based on configured rules. Users can easily find and select an object that requires an action or change and thus navigate to its object profile by double-clicking it in the configured report.

For a detailed description about how to configure a treemap configured report, see the section Defining a Treemap Report.

Rectangular Treemap configured reports

A rectangular treemap displays a hierarchical relation between objects as nested boxes. In Alfabet, rectangular treemap configured reports can display two levels of a hierarchy only. The objects of the subordinate level are displayed as boxes within the box of the superordinate object they belong to. The size of the object boxes depends on a measurable aspect of the object, like an indicator value or the number of relations to another object class.

For each object a name is displayed as text in the object box and in addition, a tooltip is displayed when the user moves the mouse over the text in the object box. The text and the tooltips are configurable.

An indicator can be displayed in each object box. In addition, the coloring of boxes can be defined to depend on a property of the object that the box represents. A legend is available for the indicator and the color coding in the configured report.

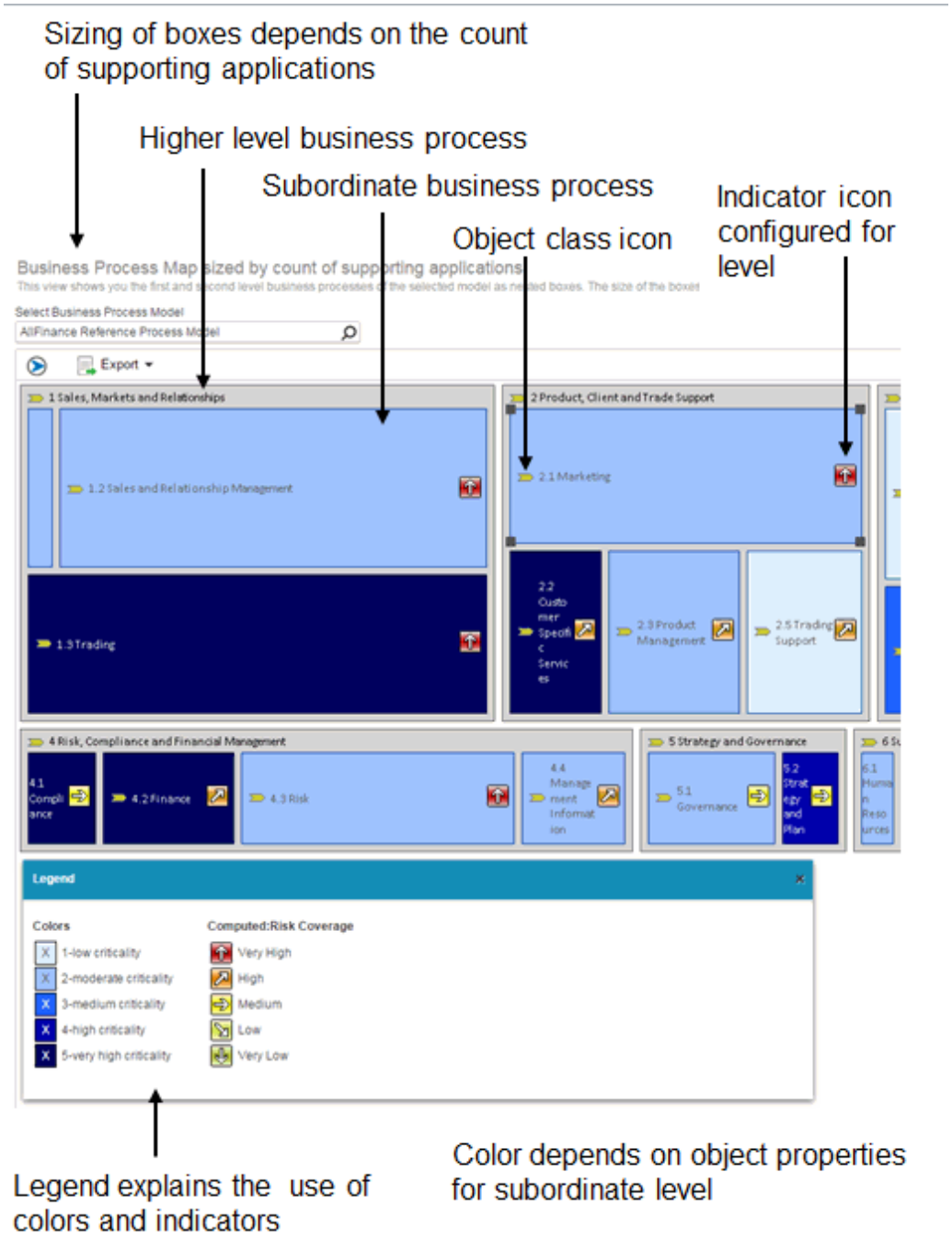


FIGURE: Example of a rectangular treemap report showing business processes and their child business processes

For a detailed description about how to configure a rectangular treemap report, see the section Defining a Rectangular Treemap Report.

Layered Diagram Reports

A layered diagram report shows objects from various object classes that are related to each other. Each object class is displayed in a separate layer and is related to the objects displayed in the layer directly above and below the current layer.

Each object in the report is displayed as a box showing a configured color and indicator icon. The color and indicator icon represent dimensions of object evaluation defined by the designer of the report. Various color coding and indicators can be defined per object class in the report.

For example, the colors of an object in an object class may depend on the number of open assignments while the colors displayed for objects of another object class may represent the object's criticality for the business.

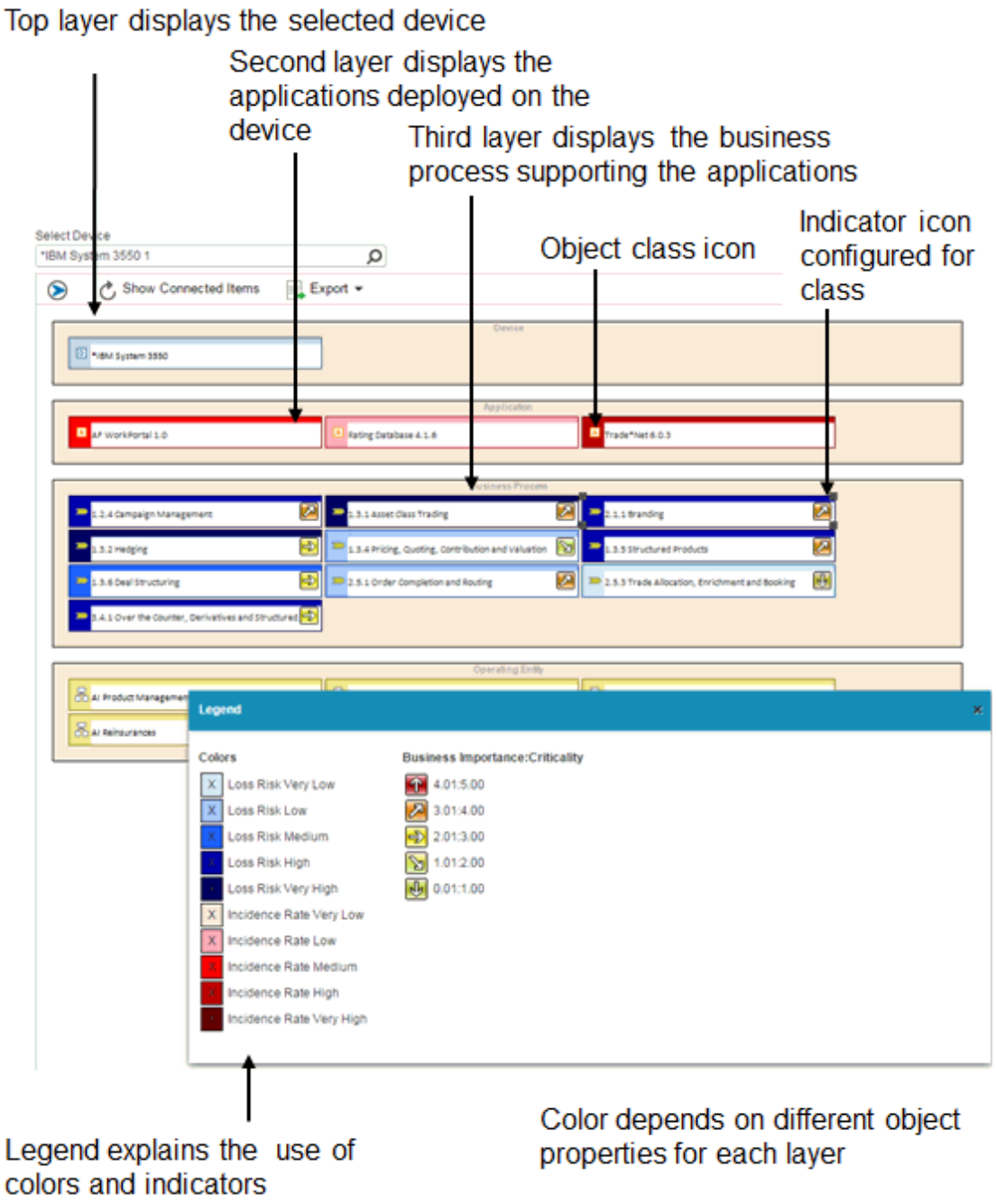


FIGURE: Example for a layered diagram report displaying devices in a selected device group and related objects

When the user selects an object in the report and clicks the **Show Connected Items** button in the toolbar, objects related to the selected objects are displayed in the colors determined by the color definition. Unrelated objects are displayed in grey.

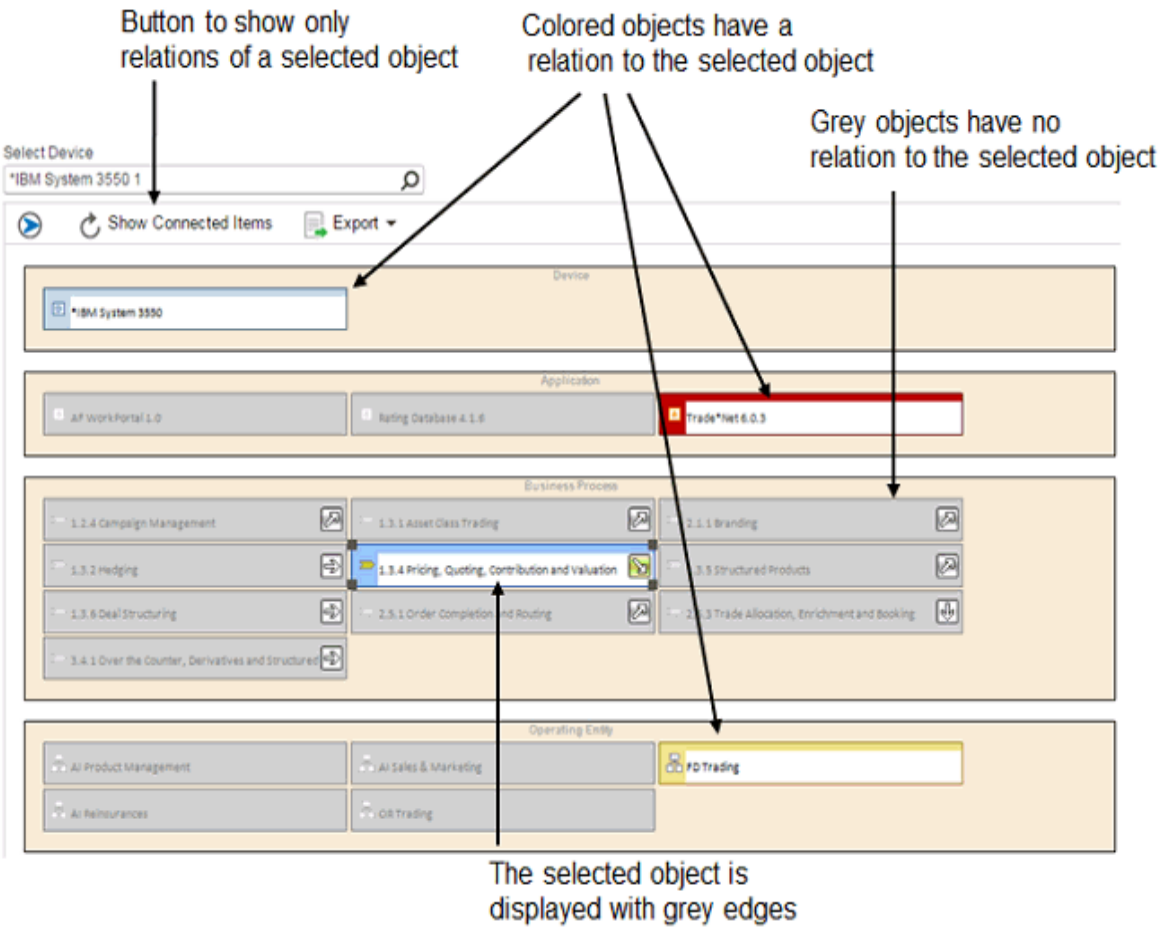


FIGURE: Example of a layered diagram report displaying devices and related objects and the relation of the objects in the layer with the selected business process

To display a report as a layered diagram report and configure the rules for the display of objects, an XML object must be configured in Alfabet Expand. The XML object is available via the attribute window of the report. For a detailed description about how to configure a layered diagram report, see Defining a Layered Diagram Report.

Tree Reports

A tree report shows a bi-directional tree displaying the relations between a selected object and related objects. In the middle of the tree, one base object or a set of base objects is displayed. Related objects found by queries are then displayed in a tree structure either in one or in two directions. The tree may be configured to span horizontally or vertically.

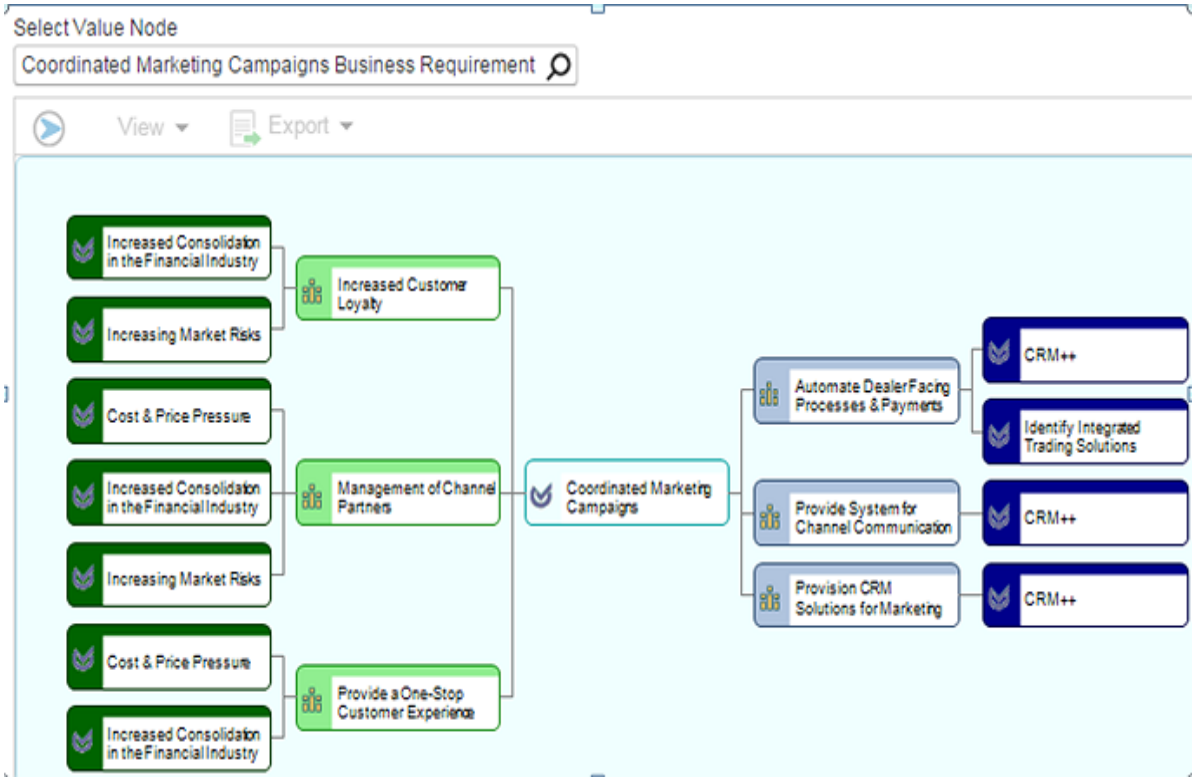


FIGURE: Example of a grid report with two trees in horizontal direction spanning from one selected object

Each object in the report is displayed as a box showing a configured color and indicator icon. The color and indicator icon represent dimensions of object evaluation defined by the report designer. Various colors and indicators can be defined per object class in the report.

For example, the colors of an object in an object class may depend on the number of open assignments while the colors displayed for objects of another object class may represent the object's criticality for the business.

A grid report can display multiple trees reflecting relations:

- The report can display more than one object as the starting point for the tree. The trees are then displayed next to each other. In this way, the queries defined to find related objects and the color rules and indicator rules used for the report are applied to all trees. All elements are displayed in the same size on the same background color.
- Multiple tree reports can be displayed in one report view. Cells in the report view may be filled with another definition of a tree report. The content of a cell is completely independent of the content of other cells in the report. Other objects can be displayed as the tree roots and the trees can reflect other relations. Colors and indicators are also specified separately for each cell. This makes it possible to specify, For example, multiple aspects of the same object or to compare different relations with one another.

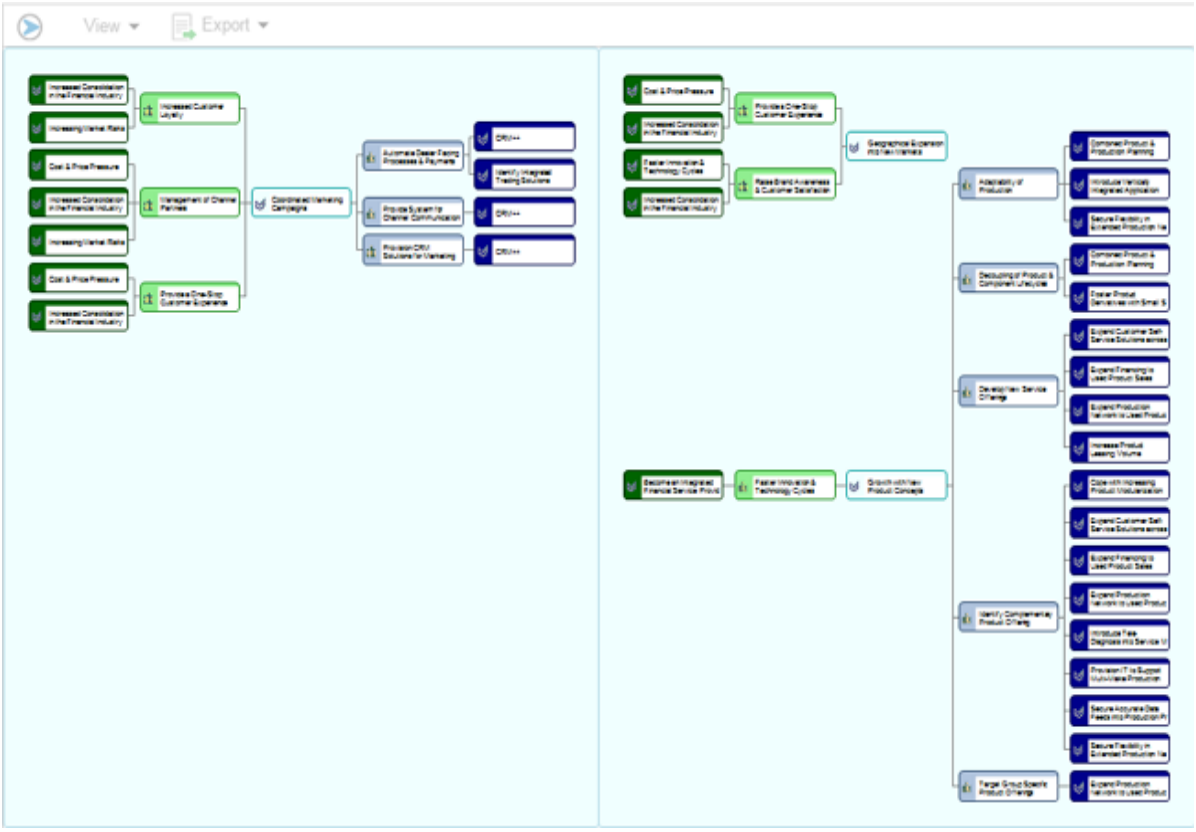


FIGURE: Example of a grid report with two cells. Cell one displays a one relation tree, while the second cell displays relations of three different base objects.

A legend is available in the report if color rules or indicator rules are specified. Color rules and indicator rules will be evaluated as described above for layered diagram reports.

For a detailed description about how to configure a grid report, see Defining a Grid Report.

Cluster Map Grid Reports

A cluster map report shows related objects in multiple levels of nested boxes. The outermost boxes could be arranged vertically or horizontally. The boxes represent objects that are found by means of queries that define the object relationships.

FIGURE: Example for a cluster map report (section thereof) showing sub-organizations of a base organization, the business processes executed by the organization and the applications providing business support for the business processes.

Each object in the report is displayed as a box highlighted by a configured color and an indicator icon (optional). The color and indicator icon represent dimensions of object evaluation defined by the report designer. Colors and indicators can be defined for each object class displayed in the report.

For example, the colors of an object in an object class may depend on the number of open assignments while the colors displayed for objects of another object class may represent the object's criticality for the business.

A legend is available in the report if color rules or indicator rules are specified. Color rules and indicator rules will be evaluated as described above for layered diagram reports.

Multiple cluster maps can be displayed in one report view. Cells in the report view may be filled with another definition of a cluster map.

For a detailed description about how to configure a grid report, see the section Defining a Grid Report.

Matrix Reports

Matrix reports shows object dependencies in a matrix. Columns and rows of the report represent objects in the Alfabet database. The cells display objects that are related to both the object in the row header and the object of the column header of the respective cell.

Each object in the report is displayed as a box displaying a configured color, border color and optionally an indicator icon. The colors and indicator icon represent dimensions of object evaluation defined by the report designer. Various color coding and indicators can be defined per object class in the report.

Two different designs are available for matrix reports: the affinity matrix and diagram matrix. The matrix types differ in style and functionality as well as in configurability:

Affinity Matrix

- Column and row headers are fixed during scrolling. The column header is excluded from vertical scrolling and the row header is excluded from horizontal scrolling.
- All cells in the report are displayed with sharp edges and the color is plain fill.
- Indicators cannot be displayed for objects in the matrix.

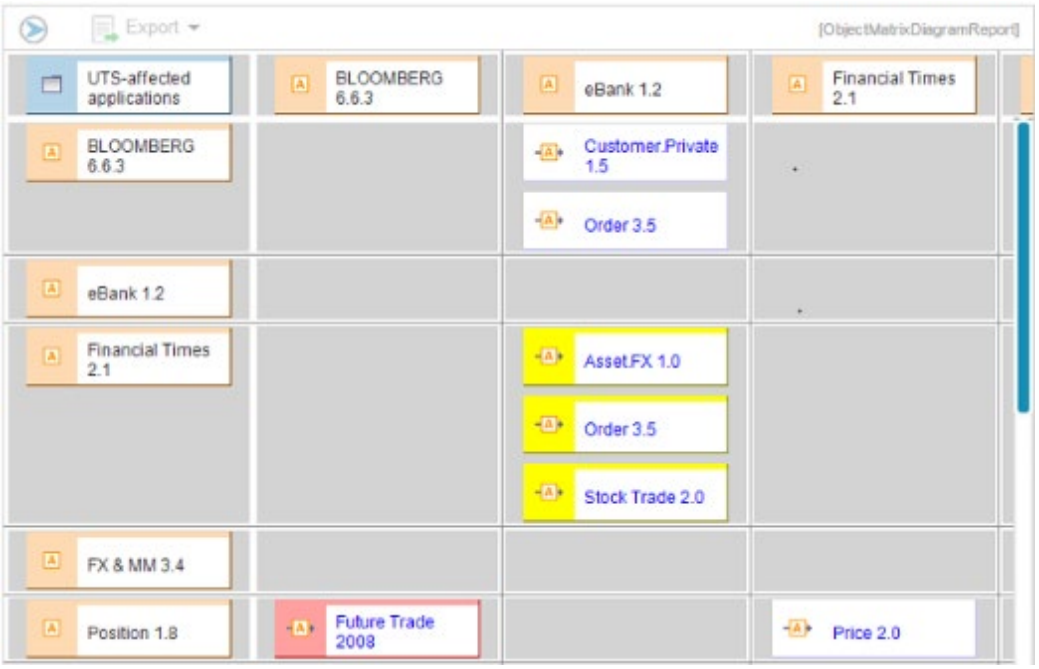


FIGURE: Upper-left corner of an affinity matrix before scrolling

FIGURE: Upper-left corner of the affinity matrix above after vertical scrolling

Diagram Matrix

- Column and row headers are included in scrolling and move out of sight when scrolling.
- All cells in the report are displayed either with rounded or sharp edges and with a color gradient.
- Indicator icons can be displayed in the object boxes in the cells of the matrix representing the indicator value for the object.

FIGURE: Matrix report of the type Diagram with indicators displayed for some objects on the right of the object boxes

For a detailed description about how to configure a matrix report, see the section Defining a Matrix Report.

Gantt Chart Reports

Gantt charts can be defined to display information about object lifecycle phases and start and end dates.

Gantt charts display the lifecycle and/or the start and end dates of objects selected via queries. The chart can display objects that depend on each other in a hierarchy. There are two different types of Gantt charts:

Hierarchical Gantt Charts

In the chart, there is a row for each object, with the hierarchy being displayed as a grouped dataset. The objects in the next level hierarchy in the report can be collapsed or expanded by the user.

The timeframe can be displayed per year, per quarter, per month, or per week. Changing the display of the timeframe is done via the zoom in and zoom out options of the floating toolbar.

The current date is displayed as a blue vertical line in the report.

Start and end times are either displayed as a line on top of the lifecycle, as a rectangle spanning the whole period between the start and end dates or as a bar indicating the start of the lifecycle. In the example below, the start and end dates of the project (displayed in the first row of the report) is displayed as a blue bar while the start and end dates of the other objects are displayed as a line on top of the lifecycle. Projects do not have a lifecycle defined but are included to enhance the visibility of the information.

Optionally, enterprise milestones and customer configured milestones can also be displayed in the report.

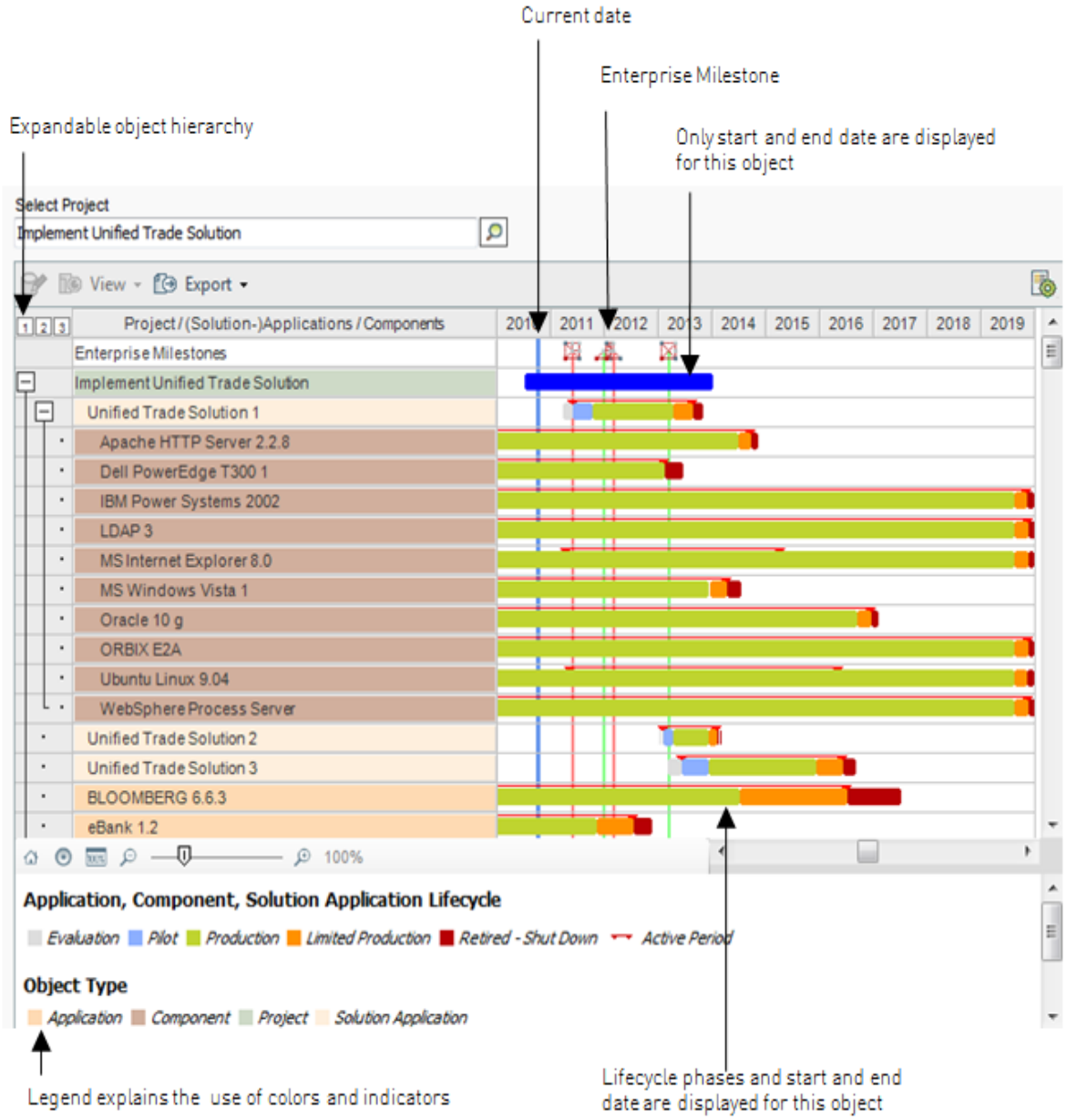


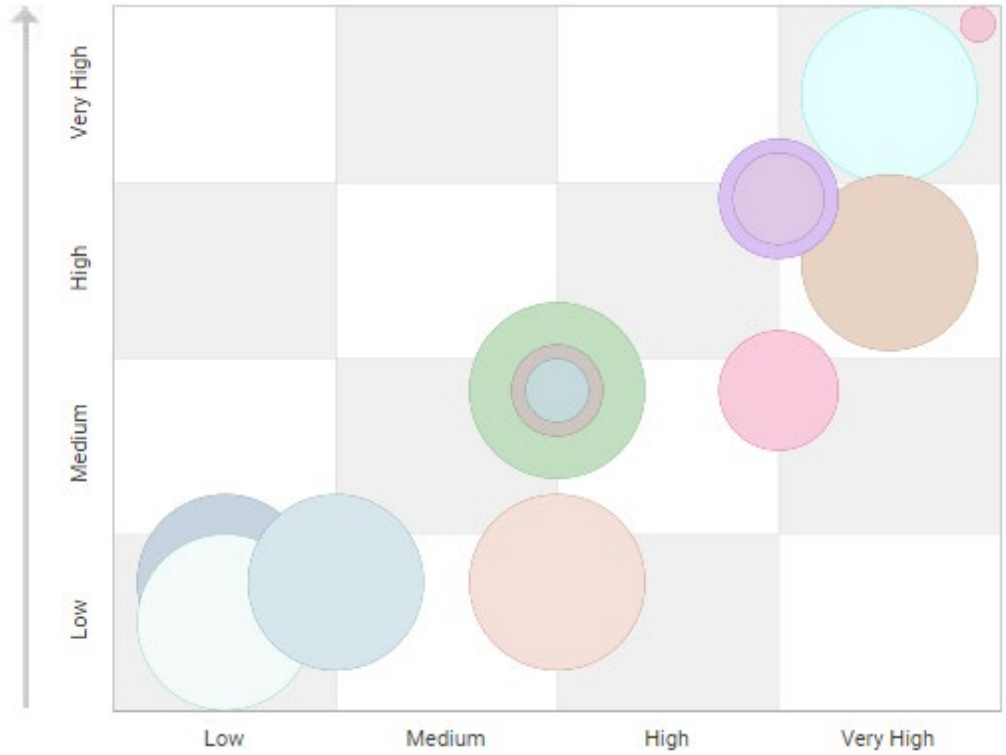
FIGURE: Gantt chart report displaying the lifecycle of applications and components relevant for a selected project

Clustered Gantt Chart

For a detailed description about how to configure a Gantt chart report, see Defining Gantt Chart Reports.

Portfolio Reports

Portfolio reports can be configured to show information that is not available in the standard portfolio reports in Alfabet. The objects displayed in the portfolio as well as the indicators that the evaluation is based on can be configured by the customer.



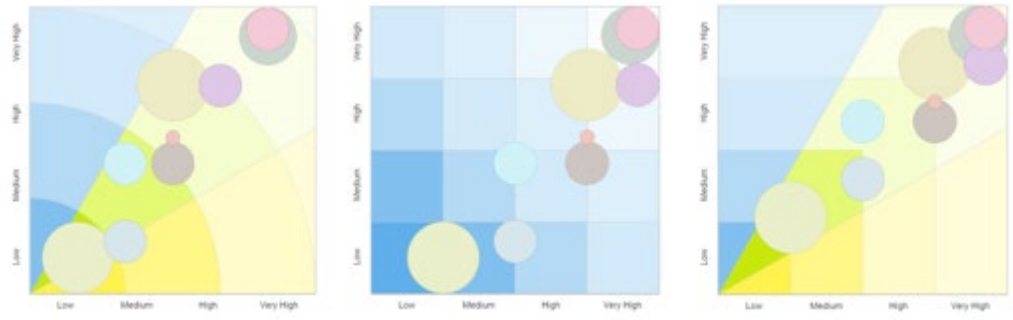
Configured portfolio reports differ from standard portfolio reports in the following:

- The underlying calculation mechanisms are different. Standard Alfabet portfolios are based on prioritization schemes that calculate values by weighting indicator values. The prioritization schemes have evaluation types assigned, which have at least one indicator type assigned. All indicators for an object associated with the indicator types assigned to an evaluation type are weighted according to the configuration of evaluation type weighting in the prioritization scheme. For more information about standard portfolio definition, see the chapter *Configuring Evaluations, Prioritization Schemes, and Portfolios* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.

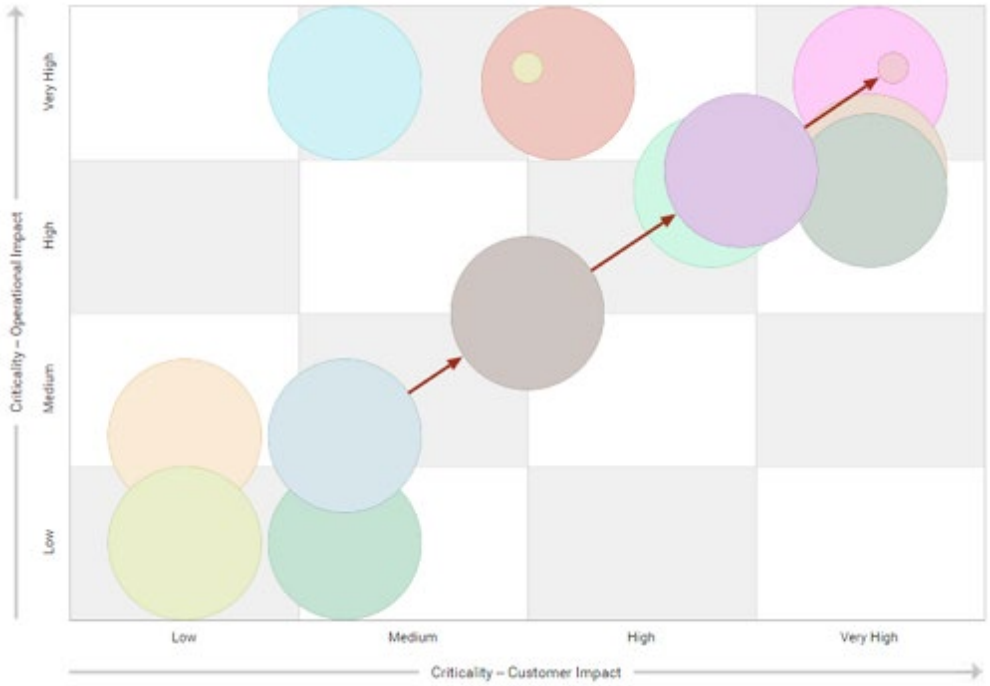
Configured portfolios show evaluations based on single indicator types. The configured report finds objects and their indicator values. The location of the object bubbles in the portfolio depends on the values defined for the indicator types for the X- and Y-axis. Optionally, the size of the bubbles in the configured report can vary depending on a third indicator defined as a third dimension (Z-axis) of evaluation.

- The way objects are represented in the portfolio can be changed from bubbles to different geometric shapes or icons. The coloring of geometric shapes is also configurable directly in the configured portfolio report.

- The background of the portfolio area can be colored with three different styles. A legend can be defined for the coloring.



- Connections can be drawn between objects in the portfolio report. A legend can be defined for the coloring.

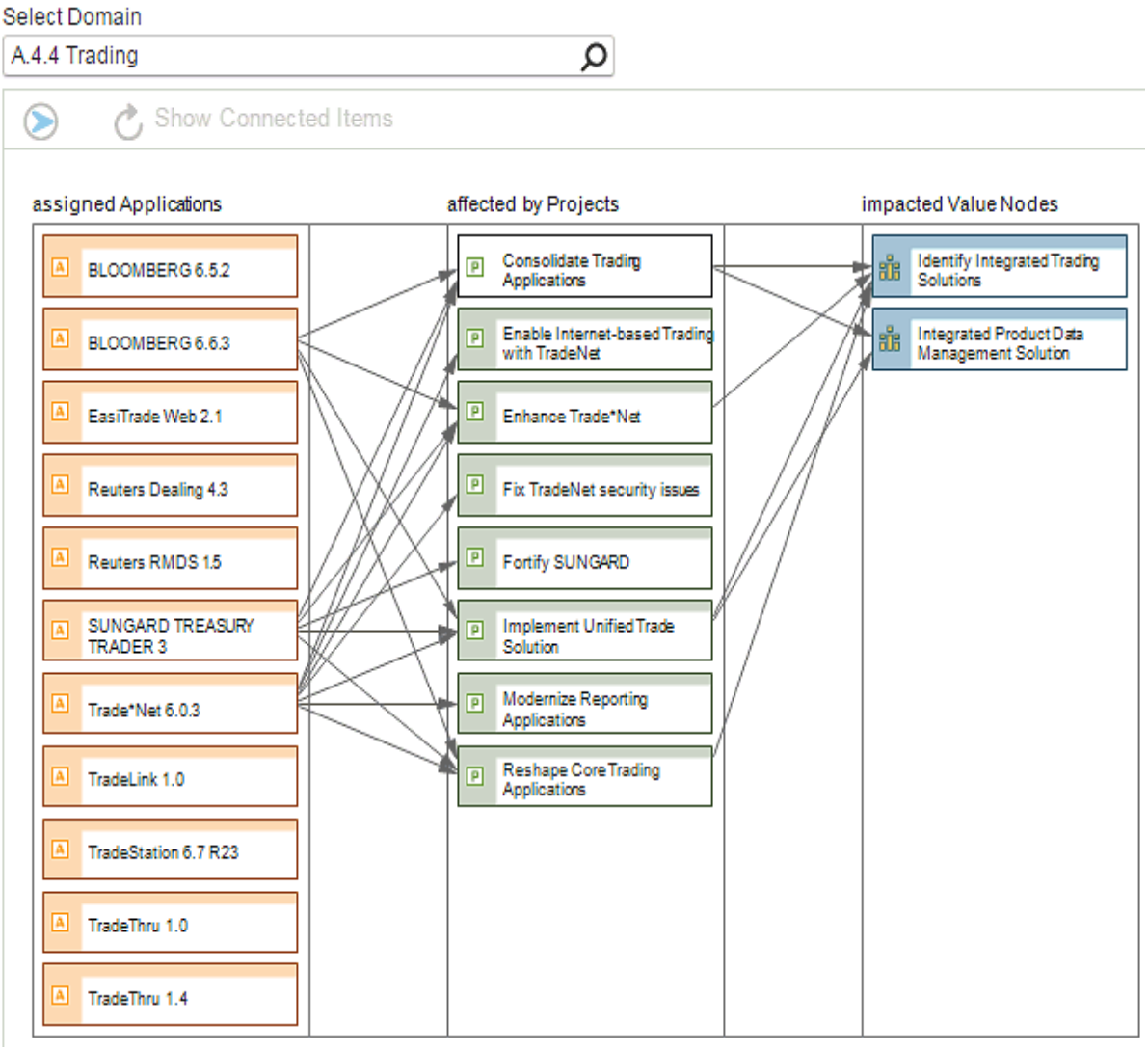


- The link target for navigation from objects in the portfolio is configurable and can open any Alfabet view. In standard portfolio views, navigation opens the object profile or object cockpit of the selected object.

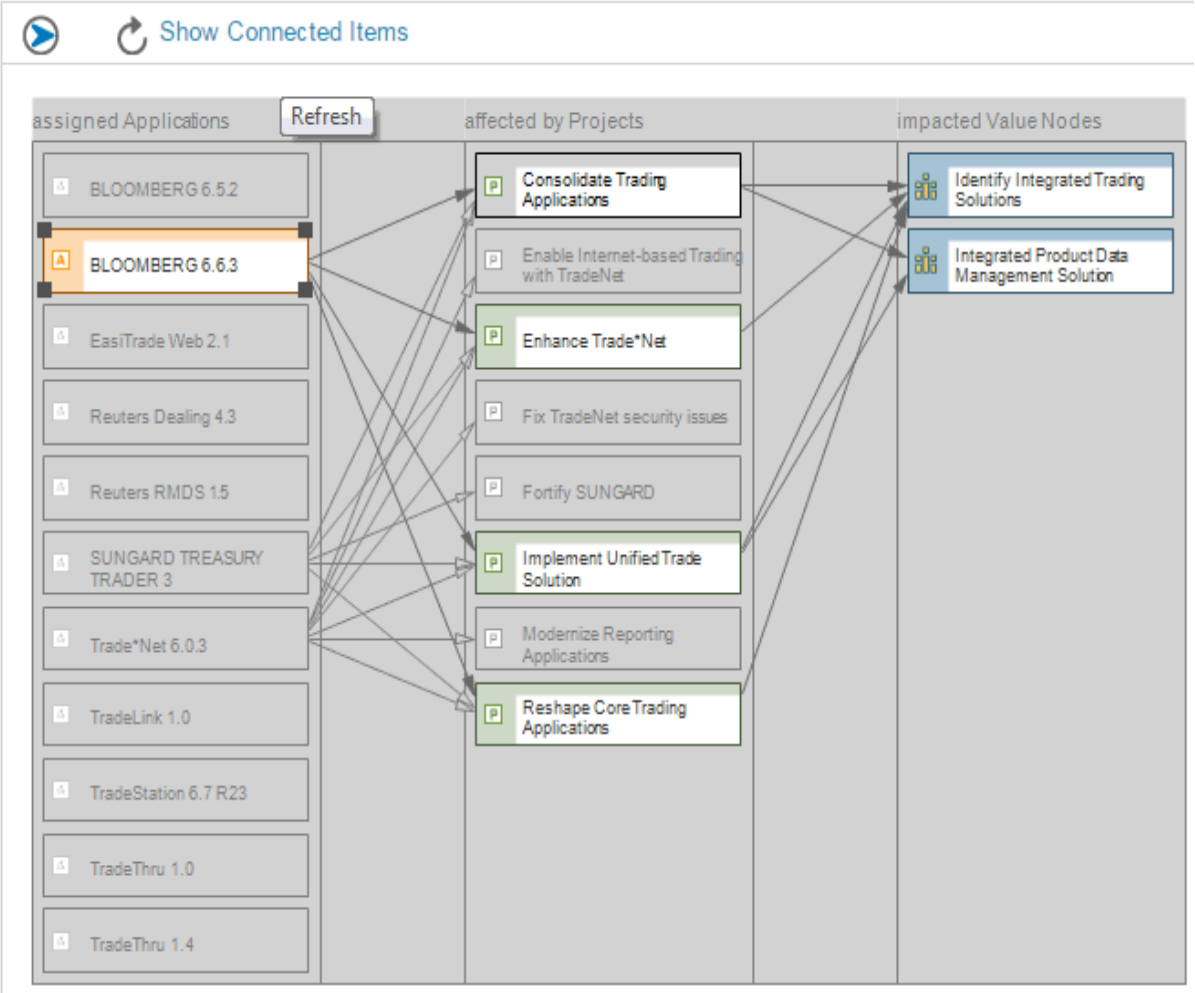
For a detailed description about how to configure a Portfolio report, see Defining Portfolio Reports.

Lane Report

A lane report displays different sets of objects in rows that are connected via links representing relations between the objects.



For a better overview, the objects directly connected to a selected object in the report can be highlighted by clicking **Show Connected Items** in the toolbar. All objects and connections not relevant for the selected objects are then displayed in light grey while the connections and objects related to the connected objects are displayed in the configured colors for the report:



For a detailed description about how to configure lane reports, see Defining Lane Reports.

HTML Table Report for Display of Indicators

Indicators for an object can be displayed in a table in HTML format in order to enhance the graphic visualization of indicator values.

For each indicator, a caption is displayed in the report that can be different from the name of the indicator. The value of the indicator for the selected object is displayed on the right. The indicator value is also displayed in a graphic format. Please note the following:

- If the indicator type that the indicator is based on has been configured to display icons rather than numerical values, the icon can be used to display the indicator.
- A scale can be included that displays the gradient of a configured maximum value that is reached by the indicator. The number of fields representing the gradient as well as the coloring and size of the gauge fields are configurable. A field will be colored if a numeric value lying in the range is reached. In the example below, a maximum of 200 was configured for the scale of Disaster Financial Amount of Loss. The indicator value of 70 is within the numerical range of 40 and 80 (2/5). Therefore, the first two fields are colored.

- The report can display up to 5 indicators. If you want more indicators to be displayed directly in an object view, then you must configure indicators to be displayed in any of the other types of configured reports, assign the report to the object view, and ensure that the report is expanded in the profile.

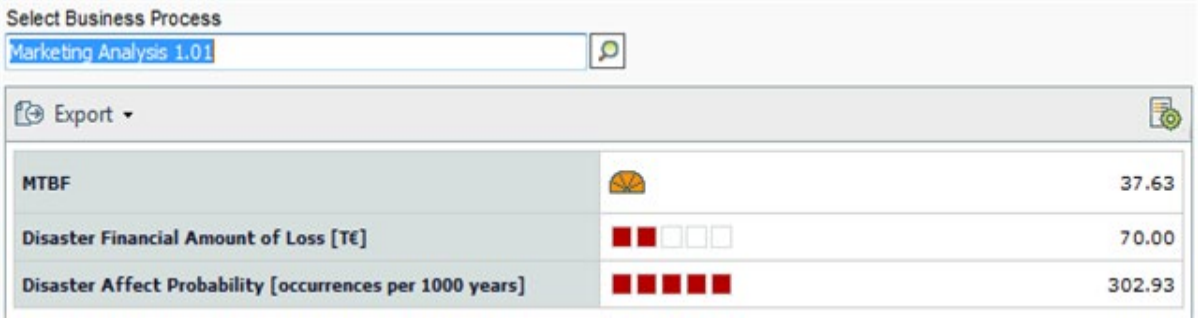


FIGURE: Example of HTML table report displaying indicator values for a selected business process.

HTML reports are configured reports of the type `Custom` and must be applied to an object class to specify the base object. An XML object must be configured in Alfabet Expand in order to define the graphic style and indicators to be displayed. The XML object is available via the attribute window of the report.

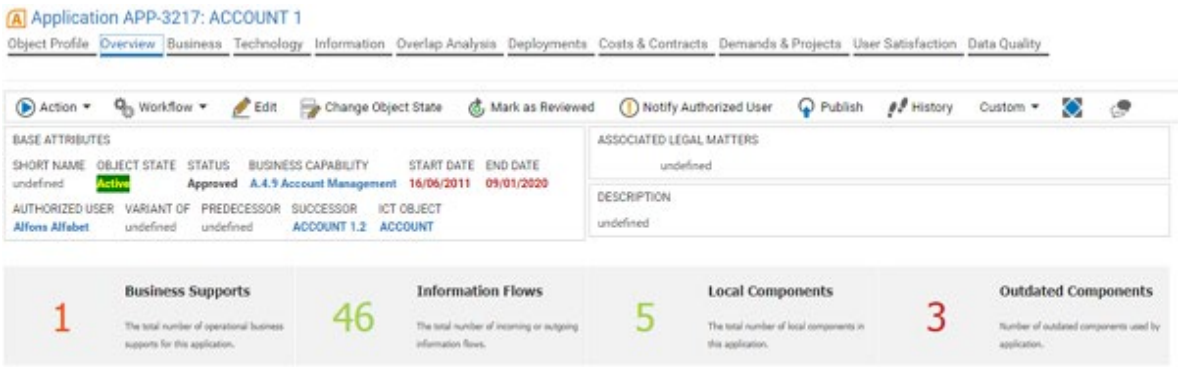
You can integrate HTML table reports in object views in order to provide the user responsible for the object with immediate information about the criticality of the object's indicators.

For a detailed description about how to configure an HTML table report, see Defining HTML Reports.

Widget Reports

A widget report is a small report displaying information in a predefined format. It is designed to fit into object cockpits to provide information in an easy to understand graphical manner.

For example, the object cockpit displayed in the image below contains four widget reports, each informing about the number of assigned objects of a specific subordinate object type (business support, information flow, local component and outdated component) for the current object:



Each information is a different widget report included into the object cockpit. The basic design of all widget reports in the example is identical. To ease configuration of a scenario with different widget reports equal in design, the design is defined in a separate configuration object called widget and reused for all configured widget reports. In the configured widget report, the widget is referenced and either static definitions or queries are defined to add images and text to the underlying text fields and image fields of the widget.

Coloring and font design can also be changed via the configured widget report. In the example above, text is changed from green to red to indicate critical numbers that may require changes to the selected application. The coloring is defined via the query that counts the number of subordinate objects.

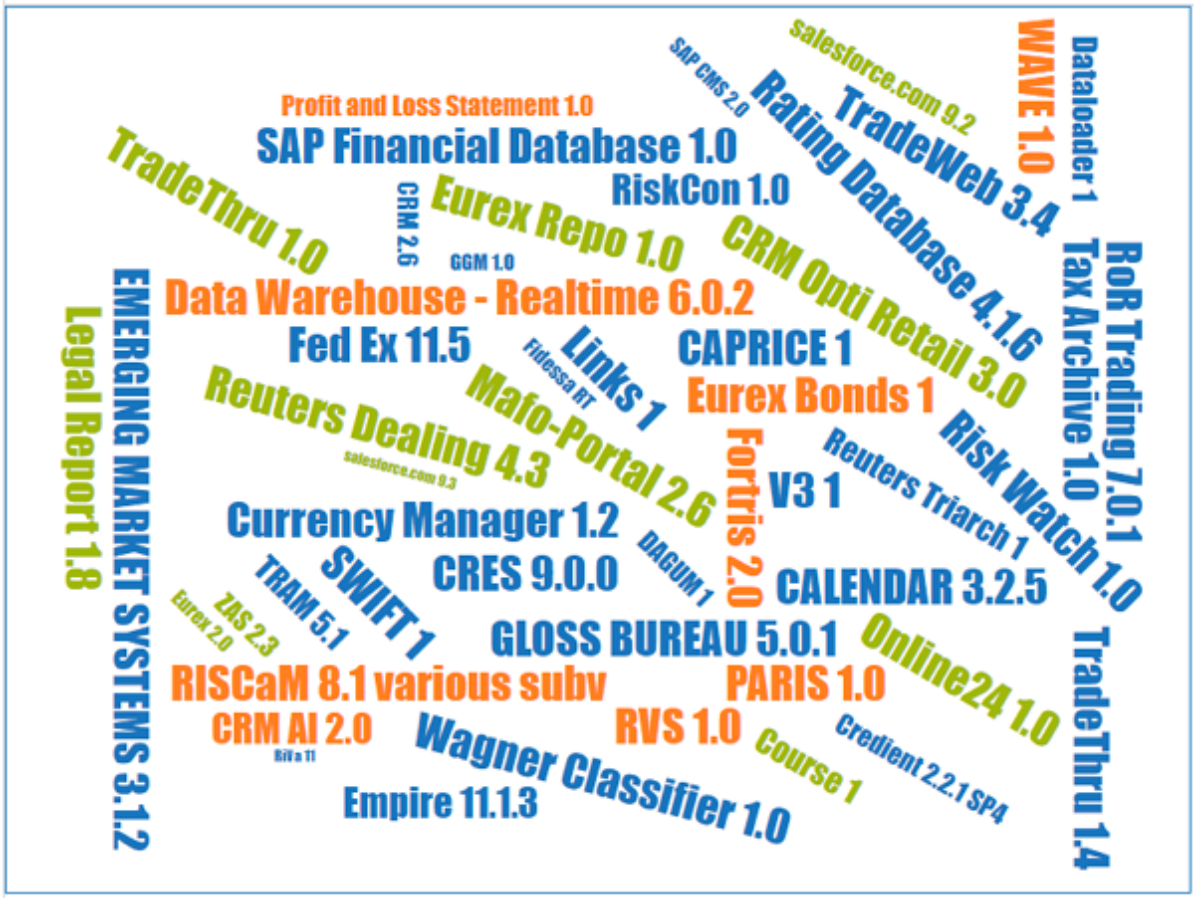
A widget report can contain multiple text and image elements. The following example for a widget report displays the object state of the current object in a graphic that shows the chronological order of object states available for an object as graphic elements and text. An arrow marks the current object state of the object. The location of the arrow is defined via a query. All other elements are static elements defined in the configured widget report for the underlying widget element.

Widget reports can also provide navigation from elements in the widget to Alfabet views both via the preview or via double-click on the element providing the link.

For information about defining widget reports, see Defining Widget Reports.

Words Cloud Report

A words cloud report displays textual information in a cloud where text color, text orientation and text size can depend on a configurable condition. In the example below, application names including version number are displayed in a words cloud with text size, text color and text orientation depending on the setting of three different indicators:



The text font type in words cloud reports cannot be changed. The font type displayed in the word cloud report is optimial for display in different font sizes and text orientations.

The algorithm of the words cloud report evaluates the location of text elements to make maximum use of the available space. Nevertheless, the space might not be sufficient to display all results found for the words cloud report. Above the report, the number of found and displayed objects is shown to inform the user about the completeness of information.

The position of elements in the report is re-calculated each time a user opens the words cloud report. Therefore, the position of each text element can vary even if no changes has been made to the underlying data.

A legend is available for the report as well as navigation to defined Alfabet views from the text strings in the report.

For information about defining words cloud reports, see Defining a Words Cloud Report.

Configured Reports Providing Ability for Mass Data Changes

Software AG provides configured report formats that allow a multi-editor for mass data update to be configured:

- Tabular Configured Report for Udate of Properties for a Selected Object Class
- Data Capture Matrix Reports for Business Supports or Object Relations

- Evaluation Reports

Tabular Configured Report for Udate of Properties for a Selected Object Class

Configured editable class view reports allow property values, indicator values and role responsibilities for selected objects of a single base class to be set. The configured report displays the objects in a tabular view based on a query. The user can select one or multiple objects in the result dataset and open an editor to change the property values.

The solution designer creating the configured report not only defines which properties are displayed in the tabular dataset, but also which properties are available in the editor of the configured report. These properties can be defined as view only to inform the user about data required for the setting of other properties, or as editable.

Edit permissions of the user are only taken into account if the configured report is configured to evaluate edit permissions. In the dataset, this is done via the query that may For example, restrict display to objects for that the current user is the authorized user. If the query of the configured report does not take editability of the object for the current user into account, the table will also display objects that are not editable.

In the editor, access permissions are taken into account if this is activated in the configured report configuration. If permission evaluation for the editor is configured, and the user selects objects in the dataset and opens the editor, only the subset of the selected objects that are editable by the user are displayed in the editor. If the user selects only objects that are not editable for him/her, an empty editor opens. If no objects are selected, the Edit button is deactivated.

A **Configure** button is available to the user that allows columns to be excluded from the configured report for the current user. Excluded columns are also excluded from the editor. The user can also change the number of freeze headers for the configured report, but a restructuring of the column order is not available for this type of configured report.

The toolbar contains two different options for editing objects. Both are only activated if objects are selected in the table:

- The **Edit** button opens an editor that lists the selected objects. If a property is not configured to be view only by the report designer, the user can set the property in the configured report. The way attributes are edited in the editor differs slightly from the processing in standard Alfabet editors. The edit field depends on the data type:
  - Properties of the type `String` are editable in a text edit field.
  - Properties of the type `Text` are editable in a text edit field with a pencil symbol on the right. If the user clicks the pencil, a wider text edit area opens to define long text.
  - Properties of the type `Text` based on an enumeration are editable via a multi-select combo box that lists all enumeration items of the enumeration the property is based on.
  - Properties of the type `String` based on an enumeration are editable via a single-select combo box that lists all enumeration items of the enumeration the property is based on.
  - The standard properties `Status` and `ObjectState` of the type `String` displaying an object state or release status a editable in a drop-down list.
  - Properties of the type `URL` can be displayed in the configured report and in the editor, but they are only available view only.

- Properties of the type `Integer` or `Real` are editable in a text edit field.
- Properties of the type `Date` are editable in a text field with a date picker on the right. The user can either type in the date in the correct format or select a date from the calendar in the date picker.
- Properties of the type `Boolean`: A checkbox is displayed that can be selected or deselected. For existing values, the checkbox is displayed as not checked if the boolean property is either not set or set to `False`.
- Properties of the type `Reference`: The solution designer configuring the report can configure which of the following edit options is available:
  - The reference can be set via a dropdown list displaying either all available objects of the object class or object classes that the reference is targeting, or a subset of these objects defined by the solution designer in the report configuration. The information displayed in the field is by default the object's name and, if applicable, the object's version with a whitespace as delimiter. This can be changed by the solution designer.
  - The reference can be set via an edit field with autocomplete functionality and, optionally, a selector button. When the user setting the reference in the editor types in text in the edit field, a dropdown list opens that displays all applicable objects in the Alfabet database with the name starting with the typed in text. A selection can then be made from the dropdown list. Objects are identified with the object's name and, if applicable, the object's version with a whitespace as delimiter. If the field is configured to provide access to a selector, the user can alternatively select the object from the selector.
- Properties of the type `ReferenceArray`: These properties are neither available in the tabular dataset of the configured report nor available for editing.
- Role Types: The solution designer configuring the report can configure which of the following edit options is available for a role type:
  - The role can be set via a dropdown list displaying either all users in the Alfabet database or a subset of users defined by the solution designer in the report configuration. The information displayed in the field is by default the family name followed by the first name with a whitespace as delimiter. This can be changed by the solution designer.
  - The role can be set via an edit field with autocomplete functionality and, optionally, a selector button. When the user setting the role in the editor types in text in the edit field, a dropdown list opens that displays all users in the Alfabet database with the family name starting with the typed in text. A selection can then be made from the dropdown list. Users are identified with the family name followed by the first name with a whitespace as delimiter. If the field is configured to provide access to a selector, the user can alternatively select the user from the selector. The users displayed in the autocomplete dropdown list are then limited to the users that are available via the selector.
- Indicators: How and if values can be defined for indicators depends on the configuration of the indicator.
  - For indicators with a range definition, the user can select one of the configured values from a dropdown list.

- Indicators that are automatically calculated are not displayed in the editor.
- For all other indicators, the user can set the value in a text field.

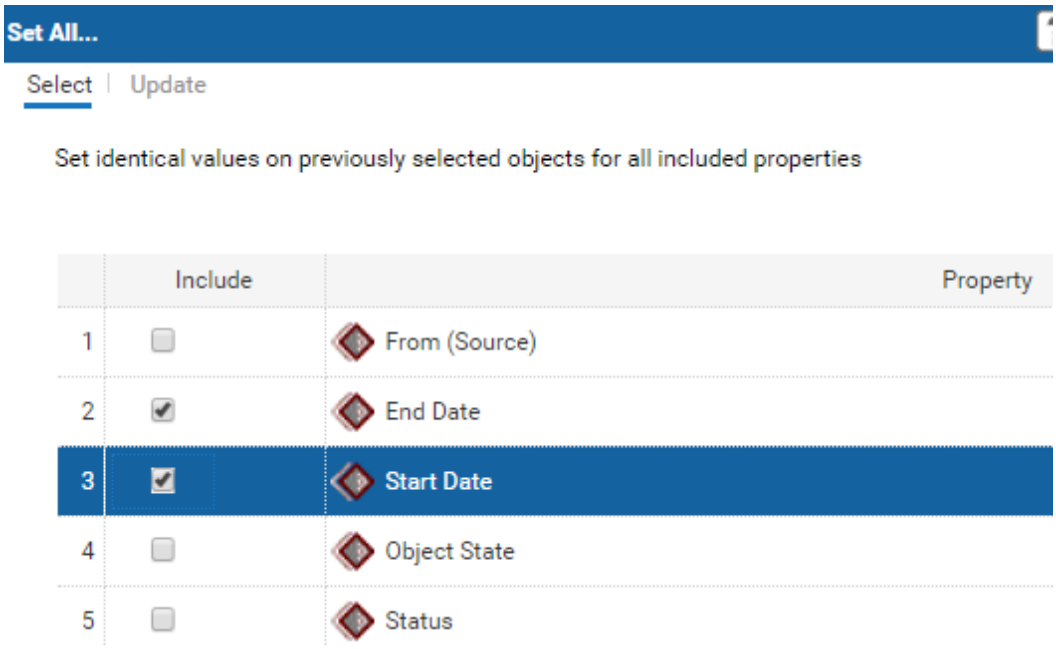
The editable fields in the configured report all display the default placeholder texts for the type of field. It is not possible to re-define placeholders, like For example, for editor fields.

If a value in a dropdown list is not fully displayed because it is longer than the width of the dropdown list field, the user can either change the size of the column in the editor to view a larger dropdown list area or move the cursor over the name that is not fully displayed. the full text is then displayed in a tooltip.

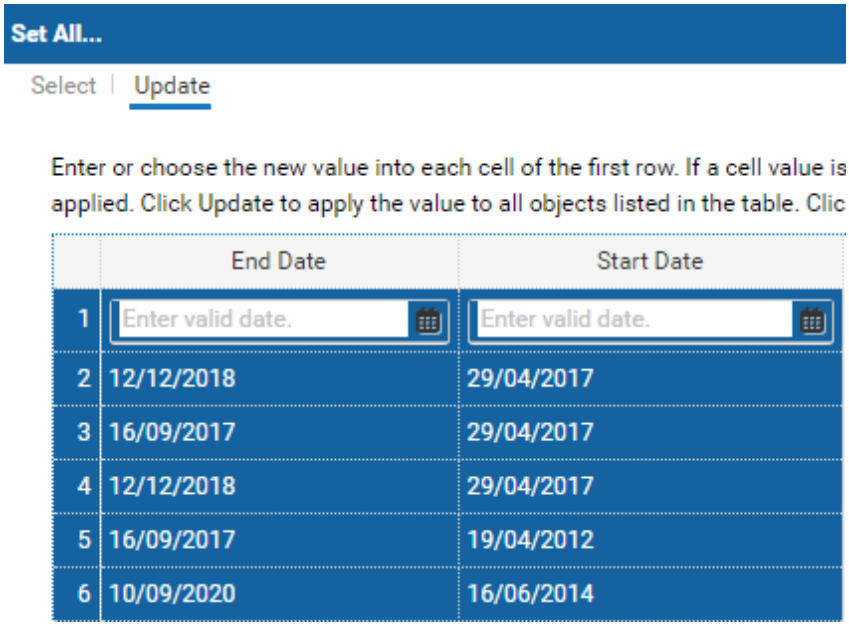
- The **Set all** button provides a dropdown list for selection of a data type. **Properties**, **Indicators** or **Role Types** can only be processed separately. The report designer can decide to remove properties, indicators or role types from the dropdown list. If properties, indicators and role types are all removed from the dropdown list, the button is not shown at all.

If a user selects an option from the dropdown list, an editor with two tabs opens:

- In the **Select** tab, the user must first select one or multiple properties to be defined:



- In the **Update** tab, a table is then displayed with one column for each selected property and one row for display of the current value of the property for each selected object. Above the listed values, there is a row with an editable field for each value. The value that the user sets in the editable field for a property will be applied to all selected objects after the user has clicked **OK** to close the editor. If the user does not provide a value for a selected property and clicks **OK**, the current values are removed from all selected objects.



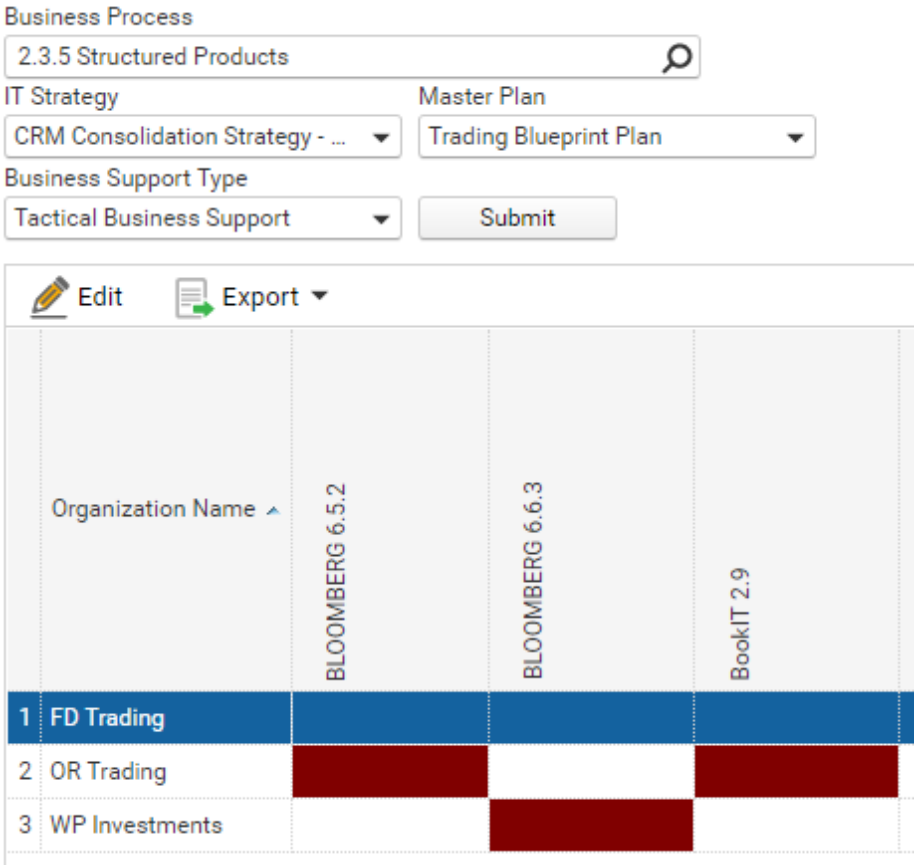
Data Capture Matrix Reports for Business Supports or Object Relations

Data capture matrix reports allow either business supports or relationships between objects to be defined by clicking relevant cells in a matrix. The matrix is customer defined and based on templates provided by Software AG. The objects displayed on the X-axis and Y-axis of the report are specified by the customer either in an Alfabet query or a native SQL query. A **Report Assistant** is available for the specification of rows and columns and for the relationship or business support defined in the matrix.

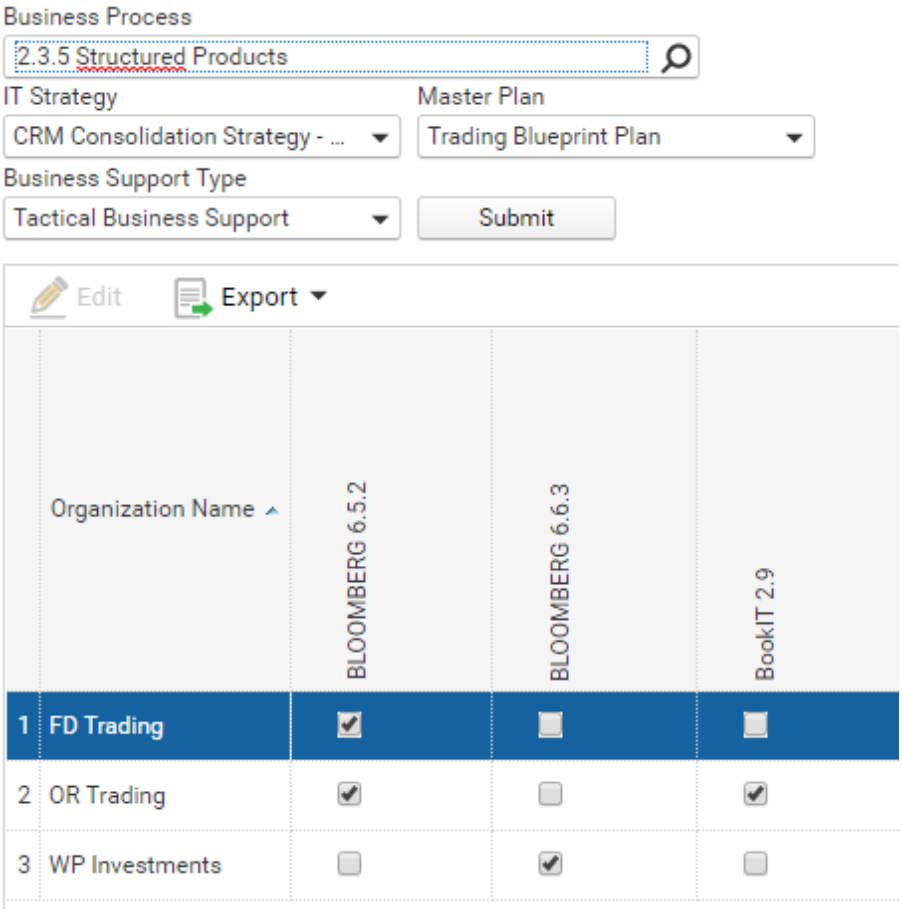
This kind of report can be used to provide multi-editing of relationships either in a separate view, opened For example, via an object view, a guide page or guide view or the Configured Reports functionality, or as part of a wizard.

The editability of the data depends on the configuration of the report.

- Business Support matrices display the existing settings in a matrix as colored bars in the matrix cells:



If the user clicks the **Edit** button in the toolbar, the matrix cells change to editable and display checkbox fields. The user can change the settings according to demands:



If the configured report is added to a wizard, the changes will be submitted when the user changes to the next wizard step. If the configured report is displayed as a separate view, the changes will be submitted when the user clicks the **Submit** button of the report. the view then changes back to display of the settings in a colored view-only matrix.

- For relationship matrices editability depends on the template used for the report. There is one template that permits direct editing within the report view. The matrix opens with all cells displaying editable checkboxes. It is recommended to use this relationship matrix in wizards. If the configured report is added to a wizard, the changes will be submitted when the user changes to the next wizard step. If the configured report is displayed as a separate view, the changes will be submitted when the user clicks the **Submit** button of the report. the view then changes back to display of the settings in a colored view-only matrix.
  - The template `Relationships_EditableTableReport` creates a configured report that permits direct editing within the report view. The matrix opens with all cells displaying editable checkboxes. It is recommended to use this relationship matrix in wizards. If the configured report is added to a wizard, the changes will be submitted when the user changes to the next wizard step. If the configured report is displayed as a separate view, the changes will be submitted when the user clicks the **Submit** button of the report. the view then changes back to display of the settings in a colored view-only matrix.
  - The template Relationships_TableReport creates a configured report displaying the current settings as colored cells in a non-editable matrix. To edit the values, the user must click the Edit button in the toolbar to open an editor that displays an editable version of the matrix. Changes are submitted when the user closes the editor.

Detailed information about the properties relevant for matrix-based maintenance reports supporting the multi-editing of objects is provided in the section *Analyzing Your Data in Page Views and Reports* in the reference manual *Getting Started with Alfabet*.

For a detailed description about how to configure a matrix-based maintenance report for multi-editing of objects, see Creating Configured Reports With Editing Capabilities.

Evaluation Reports

Evaluation reports allow indicator values to be edited for multiple objects in a table. An expandable row is displayed for each object in the table. A user can expand a row by clicking the + sign in front of a row in order to display the relevant evaluation types, indicator types and indicator values.

A separate row is displayed for each indicator type. The indicator types are sorted by evaluation type. The evaluation type is displayed in the **Evaluation** row. The evaluation type is displayed in the row of the first indicator type configured for that evaluation type.

Comments or icons defined for the indicator can also be displayed in the report. The report can either be configured to display existing indicator values only or to allow indicator values to be edited and comments to be written directly in the report table.

There are four ways to change an indicator in the report:

- Indicator values and comments assigned to the indicator can be written directly in the respective table cell if the report is configured to display the **Value** and/or **Comments** row. Depending on the configuration of the indicator type, indicator values can either be written into the **Value** cell or selected from a drop-down menu in the **Value** cell.
- Indicator values can be defined by selecting an indicator in the table, clicking the **Edit** button in the toolbar of the report, and editing the indicator value or writing a comment in the editor.
- If the same evaluation type is assigned to the parent and the child objects of a report and the report is configured to allow the same indicators to be set on both the parent and child level, the indicator values of the parent object can be batch copied to all child objects. If the row representing the parent object is selected, the toolbar option **Action** > **Copy Indicators to All Child Objects** can be used to take over the values for all relevant indicators on the child level.
- All computed indicators in the report can be recalculated using the toolbar option **Action** > **Compute Indicators for All Objects in the Report**.

  Please note the following about recalculation of computed indicators in the report:

  - The evaluation report can be configured to recalculate computed indicators when the user clicks the **Save** button. The option **Action** > **Compute Indicators for All Objects in the Report** is not available in reports that are configured to recalculate computed indicators each time indicator values are saved.

After changing one or multiple indicators, changes must be saved by clicking the **Save** button.

The object class Indicator has a property Name combining the indicator type and evaluation type names for the indicator. This property is not filled for new indicators created via an evaluation report, because the property Name of an indicator is obsolete and only available for backward compatibility reasons.

Computed indicators that are calculated automatically and cannot be set by the user are displayed in colored lines.

When the user selects an object in the report, an overview over the object data is displayed in a preview pane.

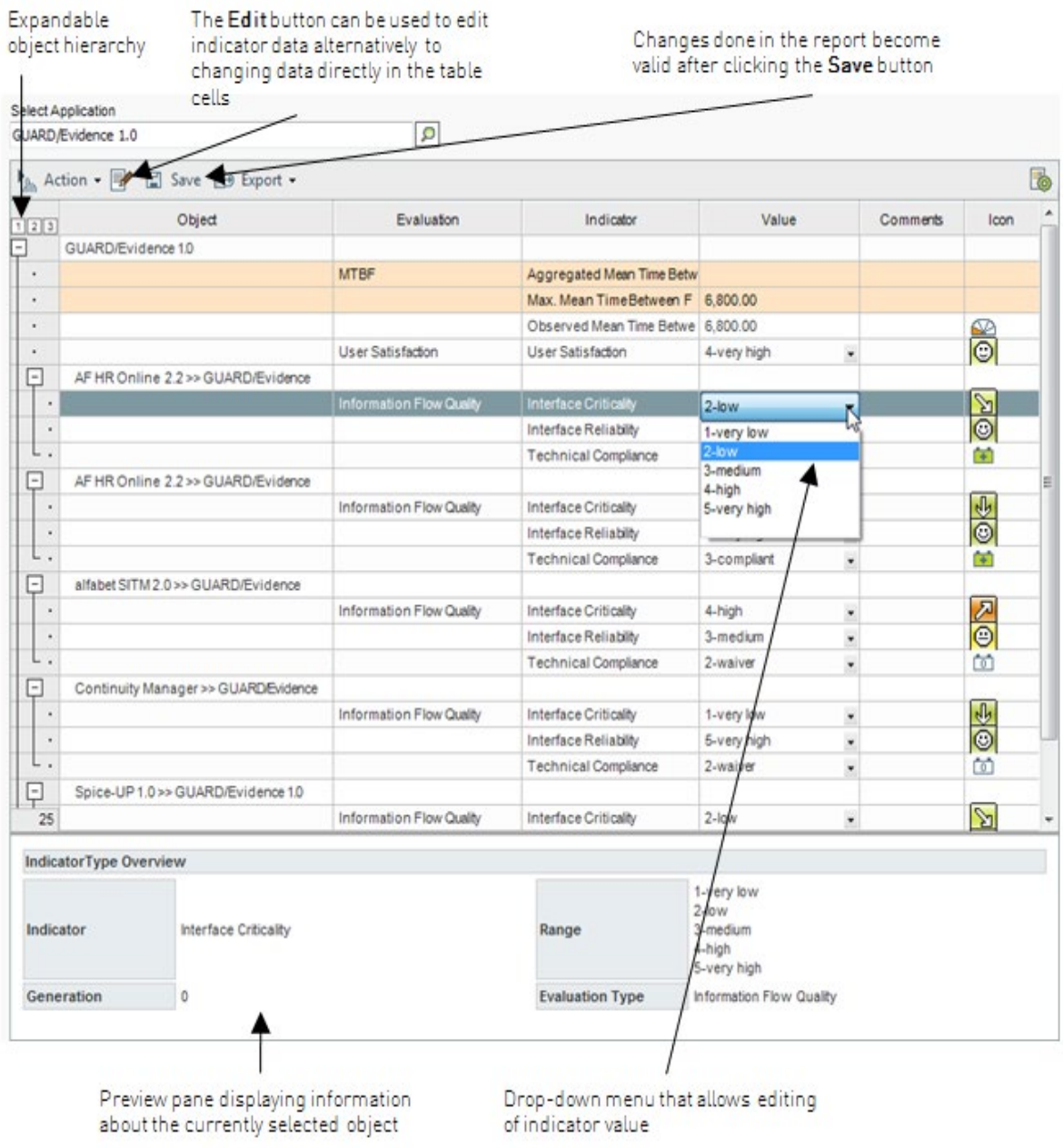


FIGURE: Evaluation report for the evaluation of a selected application and its incoming information flows

If the configured evaluation report is embedded into an object cockpit, editing is deactivated.

For a detailed description about how to configure an evaluation report, see the section Configuring a Report for Multi-Editing of Indicators.

Presenting Object Views or Object Cockpits in Configured Reports

Configured reports can be configured to display object views or object cockpits to provide access to data for an object class that the user doesn't have access to by means of the functionalities assigned to his/her user profile. While an object view is a workspace that provides access to a number of subordinate views for evaluating and editing of data, an object cockpit features a specified set of reports that the user can view directly in the cockpit.

Configured reports can provide access to one of the following:

- The object view of an object class configured in the ***Presentation*** tab of Alfabet Expand.
- An object cockpit configured for an object view in the ***Presentation*** tab of Alfabet Expand.
- An object cockpit defined directly in the configured report. In contrast to object cockpits defined for object views, an object cockpit defined in a configured report can have a filter area on top of the report and filter settings can be applied to all reports displayed in the object cockpit.

Configured Reports opening Existing Object Views or Object Cockpits

Configured reports of the type `ObjectView` can display object views or object cockpits configured for custom object views that feature a specified set of reports available for the object. In the report configuration, the object class must be defined as well as the relevant object view or object cockpit that you want to display. The user opening the configured report must select an object of the specified object class in a selector in order to view either the relevant object view or object cockpit.

If multiple object cockpits are defined in a custom object view, the availability of object cockpits via the configured report is controlled by the view scheme configuration of the user profile used when opening the configured report.

Configured reports of the type `ObjectView` can be used, For example, to provide access to data for an object class that the user doesn't have access to by means of the functionalities assigned to his/her user profile. For a detailed description about how to configure a report of the type `ObjectView`, see Creating an Alfabet Configured Report That Opens an External Report.

Configuring Console Reports

Console reports are configured reports of the type `Custom` that include an object cockpit definition that is directly defined within the configured report and an optional filter area. The filter area can be defined to apply filter settings to all configured reports that are either included into the object cockpit definition or opened via navigation from one of the configured reports in the object cockpit.

Please note that console reports are not suitable for the use in:

- Publications

- The Mobile Portfolio Manager
- Object Cockpits
- Object Views
- Wizards

If a console report is added to a publication or opened via the Mobile Portfolio Manager, the first subordinate report added to the console report is displayed instead of the console report.

For a detailed description about how to configure a console report see Creating Console Reports.

Combining Multiple, Cascading Reports in One View

Multiple reports can be combined in one view and configured to be cascading. When a user clicks an object in the report that is configured to be the master report, the data in the subordinate report changes and displays information related to the selected object.

In the example below, a graphic representation of domains that are sub-domains of a selected domain is displayed. As subordinate report, a tabular report displaying information about the applications assigned to the domain currently selected in the master report.

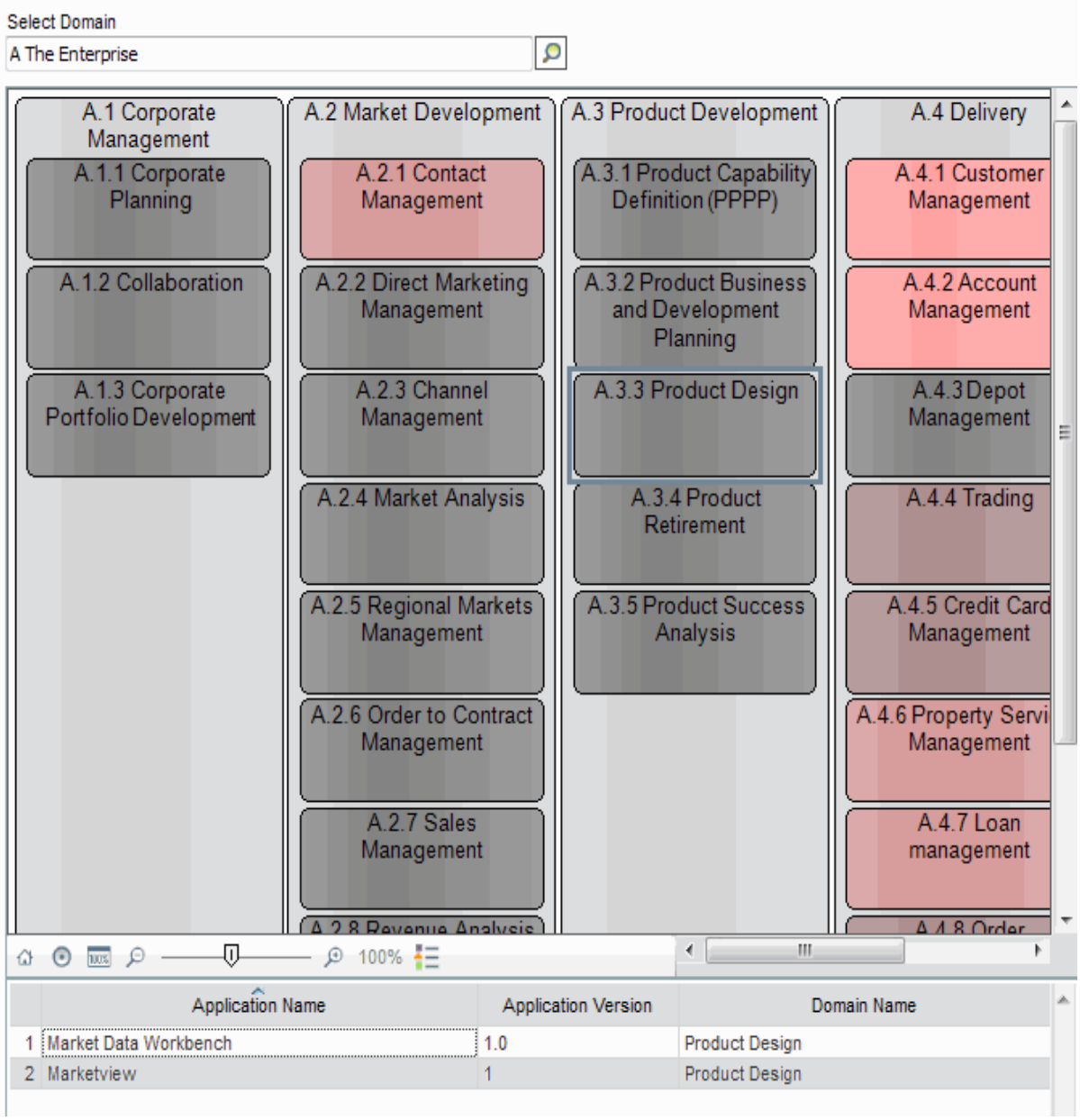


FIGURE: Cascading report with a master and a subordinate report

For a detailed description about how to configure a cascading report see Creating Cascading Reports.

Presenting Geographically Relevant Data in Maps

For international companies it might be useful to present the geographic relevance of data in geographic maps. For example, to display which locations depend on support from applications from other locations or how many deployments of an applications are available in which region.

In Alfabet, geo map reports displaying geographic maps based on FusionMaps® from FusionCharts that can be configured to display data from the Alfabet database. Regions or countries in the map can be colored according to their relevance with regard to specific data in the Alfabet database. Markers can be set at customer defined locations. The markers can differ in coloring and size according to their relevance with

regard to specific data in the Alfabet database. Connecting lines can be configured to display dependencies between markers. When a user moves the mouse over an element in the map, a tooltip is displayed. The information displayed in the tooltip is configurable in the report configuration. For color codings, a legend can be added to the report.

Multiple geo map reports can be connected with each other. For example, a world map can be configured to open a map of the respective country if a user clicks on the country in the world map.

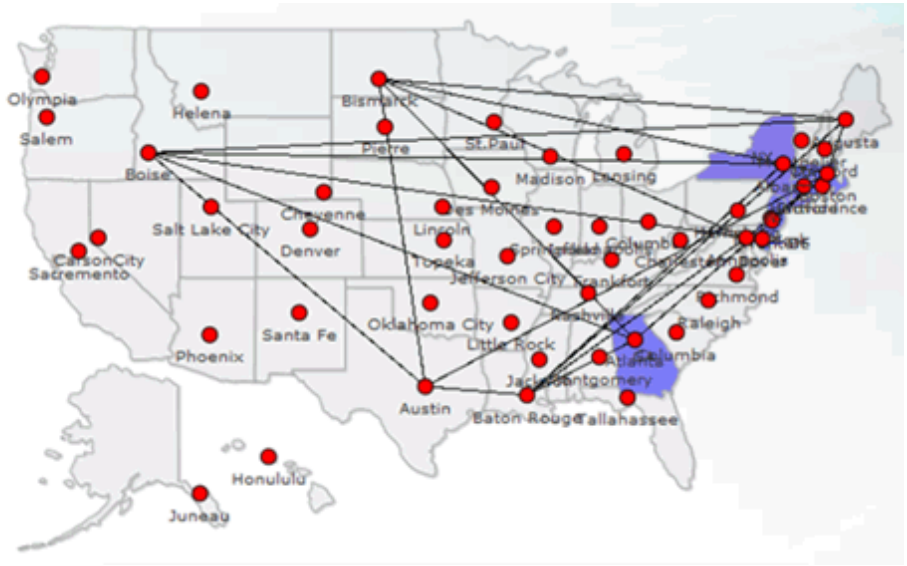


FIGURE: World map displaying interconnected markers and colored regions

For a detailed description about how to configure a geo map report, see Creating Geo Map Reports.

Integration of External Reports

You can include any report generated by third-party components (For example, reporting tools) in the Alfabet interface. The external reports are generated independently of any Alfabet components. An external report is accessible in the Alfabet interface via a configured report of the type `Extern` that basically consists of a link to its URL. When the user accesses the configured report from the Alfabet interface, the external report will open in a new window and the user will work outside of the Alfabet interface as long as he/she works with the external report.

It is possible to scan the Alfabet database with standard SQL in order to present the data in the external report. Software AG also provides a link syntax that is required in order to link from an external report to the Alfabet interface.

You can include either static external reports or external reports that are executed at runtime.

For more information about configuring the interfaces for external reporting, see the section *Accessing the Alfabet Database with External Applications* in the reference manual *System Administration*.

The following table describes the advantages and disadvantages of Software AG 's internal mechanism of reporting versus the use of external reporting tools:

Type	Data Collection and Presentation	Advantages	Limits

`Query`
 	
`Custom`
 	
`ObjectView`
	You can use the Alfabet query language or native SQL to find objects in the database and display the results of the query either in a configurable table or in a treemap report or layered diagram report in the Alfabet interface. Alternatively, you can configure reports that display object views of objects in the database or display an object cockpit showing a specified set of reports available for the object. The query-based reports are executed by the user at runtime and display the current results.
	
	You can use the tool Alfabet Expand to define the configured report. The query-based configured report allows you to customize the information available in the Alfabet interface.
		Knowledge about SQL query language or a scripting language or other programming skills are not required.
		
		The access rights defined for objects in Alfabet apply. This includes authorized access and mandate-specific access rights.
		
		It is possible to navigate to the object views of object's in the report during the same Alfabet session.
			Data from data sources other than the Alfabet database cannot be included in the results.
			
			The display of results is limited to the formats designed by Software AG.

`Extern`	You can include any report generated by third-party components (For example, reporting tools) in the Alfabet interface by referencing the URL of the report. These external reports of the type `Extern` are generated independently of any Alfabet components. An external report is accessible in the Alfabet interface via a report that consists of a link to its URL. When the user accesses the report from the Alfabet interface, the report will open in a new window and the user will work outside of the Alfabet interface as long as he/she works with the report. It is possible to scan the Alfabet database with standard SQL for the presentation of data in external reports. Alfabet also provides a link syntax in order to link from an external report to the Alfabet interface. You can include either static reports or reports that are executed at runtime.	The means to visualize the results via graphics are enhanced, as they depend exclusively on the external tool used to generate the report.	Knowledge about the SQL query language and the handling of the external reporting tool or scripting language is required for the report. The access rights to objects configured in Alfabet (authorized access and mandates concept) do NOT apply to the results displayed in reports. When a user accesses the Alfabet interface via a link from an external report, a new Alfabet session will open and the user might see only the view or the view based on a different user profile.

For a detailed description about how to integrate an external report into the Alfabet interface, see Creating an Alfabet Configured Report That Opens an External Report.

General Guidelines for Creating Configured Reports

Customer configured reports can be created using the following tools:

- Alfabet Expand
- The **Reports Administration** functionality of Alfabet provides a subset of the functionalities for configured report creation and editing for demonstration purpose. Only configured reports of the type `Extern`, `Query` or `NativeSql` can be created in the **Reports Administration** functionality. It is recommended that you use Alfabet Expand in stand-alone mode for configured report creation.

Configured reports can be created and edited during runtime of the Alfabet Web Application. To avoid conflicts resulting from a user executing a configured report at the same time that the configured report is edited by a solution designer, the configured report is either editable or can be executed depending on the setting of the **Report State** attribute of the configured report. A configured report can have the following **Report State**:

- ***Plan***: A configured report that is in a `Plan` state may have modifications made to the configuration and is not accessible in the Alfabet interface:
  - The configured report is no longer available in the list of configured reports that can be added to the **Configured Reports** view.
  - If a user already added the configured report to his/her table in the **Configured Reports** view, the configured report remains in the table, but the button **Open Report** is deactivated for the configured report.
  - If the configured report is added as a view to an object profile, the user opening the configured report is informed that the configured report is currently under construction.
- ***Active***: A configured report that is in an `Active` state can no longer be edited and is visible to authorized users in the **Configured Reports** functionality and optionally on **Configured Reports** page views of object profiles. Configured reports assigned to object profiles or used in the **Compliance Management** functionality are visible in the Alfabet interface independent of the configured report's **Report State**.

  If the report view for a configured report is opened in the central editor pane of Alfabet Expand and the **Report State** of the configured report is set to `Active`, the report view editor is not closed and the content of the report view is still editable, and changes can be saved for the configured report as long as the editor pane is not closed.

- ***Retired***: Once a configured report is set to `Retired`, it can no longer be accessed in the **Configured Reports** functionality and optionally on **Configured Reports** page views of object profiles. Configured reports assigned to object profiles or used in **Compliance Management** functionality are visible in the Alfabet interface independent of the configured report's **Report State**.

In Alfabet Expand, the attribute **Report State** is not editable in the attribute window for the configured report.

Do the following to change the **Report State** attribute of a single configured report:

1) In the explorer of the **Reports** tab of Alfabet Expand, right-click the configured report for which you want to alter the **Report State** attribute and select **Set Report State to 'Plan'**, **Set Report State to 'Active'**, or **Set Report State to 'Retired'**.

The **Report State** attribute of the configured report is changed according to your selection. It is not required to save this change via the **Save** button.

To change the **Report State** attribute of all configured reports in a report folder:

- In the explorer of the **Reports** tab of Alfabet Expand, right-click the report folder for which you want to alter the **Report State** attribute of all subordinate configured reports and select **Set Report State to 'Plan'**, **Set Report State to 'Active'**, or **Set Report State to 'Retired'**.

The **Report State** attribute of all configured report that are subordinate to the report folder is changed according to your selection. This includes all configured reports directly located in the folder and all configured reports located in sub-folders of the report folder. It is not required to save this change via the **Save** button.

Please consider the following when editing configured reports during runtime of the Alfabet Web Application:

- When configured reports that are already available to the user community are changed during runtime of the Alfabet Web Application, changes may become visible to the users with a delay of about 10 minutes. Configured reports are stored in a cache that is updated in regular intervals. Change to a configured report will become valid with the next cache update. Alternatively, restart the Web server to load the configured report to the cache of the Alfabet Web Application.

It is recommended to follow the best practice procedure described below for configuring reports. Next to best practice procedure, this section describes configurations with general relevance for configured reports:

- Best Practice Procedure for Configuring Reports
- Using Server Variables in Web Link and Database Server-Related Specifications
- Restriction of Database Access Permissions on Execution of Configured Reports
  - Using an Alfabet Internal ReadOnly User for Report Execution
  - Configuring Database Users With Explicit Access Rights for Individual Configured Reports
- Restricting Access to Administrative User Profiles
- Activating Navigation From Reports to Object Classes Not Automatically Offering Navigation
- Specifying Custom Help for a Configured Report
- Assigning a Category for Specific Functional Use to a Configured Report
- Handling Long Running Reports
  - Defining an Execution Timeout for Long Running Reports
  - Open Tabular Reports with Filters Without Immediate Execution
  - Configuring Offline Execution for Long Running Tabular Reports with Filters
- Analysis of Reports via the Semantic Analyser
  - Adding an Object Class to the Semantic Analysis
  - Adding an Object Class Property to the Semantic Analysis

- Removing Manually Added Entries From the Semantic Analysis

Best Practice Procedure for Configuring Reports

Configured reports shall be created in a design environment and tested prior to use in the production environment. The report configuration of the test environment can then be saved together with other configurations to an updater file or separately into an XML file and read or merged into the production environment during a short, planned server outage.

The advantages of this procedure are:

- Avoidance of complaints about unavailability of page views in object profiles because users cannot distinguish between standard Alfabet views or configured reports in the customized object profiles and would regard unavailability because of configuration maintenance as software malfunction.
- Maximum security of data integrity in the Alfabet database. Configured reports can also be configured to allow objects to be edited directly in configured reports. Therefore, report configurations should be thoroughly tested in a test environment before they are implemented in the production database to make sure that no access permission violations are incurred by the report configuration.
- Ad-hoc reports required to answer one-time questions about data in the Alfabet database can be configured and executed in the test environment without the need to specify access permissions that inhibit visibility to users other than the executing administrator.

Normally, Ad Hoc reports should not be part of the configuration that is later used in the production environment. If you create reports for one-time execution only at the root level directly below the **Reports** folder and store all other configured reports in sub-folders of the **Reports** folder, you will be able to batch delete all Ad Hoc reports without losing relevant reports. To do so, right-click the **Reports** folder and select **Delete All Root Reports**. Configured reports on the root level will be deleted while the configured reports in the sub-folders are still available.

The configuration of all available types of configured reports is described in detail in the following sections.

After completion of the report configuration in the test environment, you can use one of the following methods to import the report configuration to the production environment:

- The configuration of reports can be saved to an AMM update file either separately or in combination with other configurations specified in Alfabet Expand. The AMM update file can then be used to replace or merge the configuration in the updater file with the configuration of a target database. This method requires that you are using Alfabet Expand in stand-alone mode. It is part of the overall database update methods and described in the chapter Applying Configuration Changes to Other Databases.
- The configuration of all configured reports, the content of a report folder or a single configured report can be saved to an XML file. This file can be merged with the report configuration in a target database or replace the report configuration in the target database. The method is available in stand-alone and remote mode. The procedure is described below.
- The configuration of a single configured report can be directly copied from one environment to another if both Alfabet Expand tools connecting to the test and the production environment are run

on the same machine using the copy and paste functionality. The method is available in stand-alone and remote mode. The procedure is described below.

To update the report configuration of the production environment with all or part of a configuration from a test environment stored in an XML file:

1) In the test environment, open Alfabet Expand.
2) In the **Reports** tab, right-click the folder **Reports** and select **Save As**.
3) In the explorer window that opens, select a location for the XML file that is accessible via Alfabet Expand in the production environment and click **Save** to save the file.
4) In the production environment, open Alfabet Expand.
5) In the **Reports** tab, right-click the folder **Reports** and select one of the following:
   - **Replace from File** to overwrite the current configuration with the configuration in the XML file. The whole report configuration is deleted and then the configuration stored in the XML file is applied. That means that all configured reports and report folder in the XML file will overwrite corresponding objects in the database. Database objects that have no corresponding object in the XML file will be deleted.
   - **Merge from File** to merge the current configuration with the configuration in the XML file. All configured reports in the XML file will overwrite corresponding configured reports in the database. Database objects that have no corresponding object in the XML file will remain unchanged. Configured reports that are only available in the XML file will be added.
6) In the explorer window that opens, select the XML file with your new report configuration.
7) Confirm the warning message. The report configuration in the production environment is changed.

To update a single report configuration of the production environment or add a single report from the test environment:

1) In the test environment, open Alfabet Expand.
2) In the **Reports** tab, right-click the configured report that you want to copy to the production environment and select **Copy**.
3) In the production environment, open Alfabet Expand.
4) In the **Reports** tab, right-click the root folder **Reports** or the report folder in which the configured report shall be updated or added, and select **Paste**. The configured report is added to the report configuration in the production environment or, if a configured report with the same technical name existed in the production environment, that configured report is updated.

After replacing or merging a report configuration and after upgrade to another Alfabet release, it is recommended that you test whether changes in the meta-model require you to adapt configured reports to the new model (For example, to change a `JOIN` or a `WHERE` statement in an Alfabet query) A test mechanism is provided that controls whether the Alfabet query of a configured report matches the current meta-model. The test can be conducted for all configured reports, all configured reports in a report folder, or for a single configured report.

You can define queries for Alfabet configurations in native SQL instead of the Alfabet query language, but the test mechanism described here only tests queries defined in the Alfabet query language.

To test the configuration of configured reports, carry out one of the following steps:

- In the **Reports** tab, right-click the folder **Reports** (or a report folder or a configured report) and select **Check and Update AQL Based Reports**. An error report is generated in Microsoft® Word® format.
- In the Alfabet Expand menu bar, select **Meta-Model** > **Check All AQL Queries**. An error report is displayed directly in Alfabet Expand and allows you to navigate to incorrect Alfabet queries directly from the report. For more information about locating and correcting errors with this functionality, see Testing Queries for Compliance with the Current Release in the section Defining Queries.

The procedure to define a configured report will depend on the type of configured report you plan to create. Therefore, the definition of configured reports of different types is described separately in the following sections.

The attributes that can be defined for a configured report are specific to the type of configured report that is being created. The following descriptions list all attributes that can be defined for the specific type of configured report you are defining. If an attribute is not mentioned, it is not possible to set it for the selected report type.

Using Server Variables in Web Link and Database Server-Related Specifications

The definition of server variables allows information For example, about the target of web links to be defined in the server alias configuration. Defining the information about the URL in the server alias configuration instead of directly defining it in the configured report definitions eases the propagation of changes.

For example, the configuration of external reporting for Alfabet is done first in a test environment, and the test environment is an exact copy of the production environment except that the components are installed on different servers. Therefore, all URLs defined will be identical in the test and the production environment except for the Web server name. When migrating to the production environment, the Web server name must be changed in all URL definitions in Alfabet Expand. But if the server name is defined as server variable in the server alias, the configurations done in Alfabet Expand reference the Web server name as variable in the URL definition and can be reused without changes in the production environment. Only the variable definition in the server alias of the production environment must be set to the current Web server.

Server variables are defined in the Alfabet Administrator:

1) In the Alfabet Administrator, click the **Alfabet Aliases** node in the **Administrator** explorer.
2) In the table on the right, select the server alias that you want to define a server variable for and click the **Edit**  button. The alias editor opens.
3) Go to the **Variables** tab and click the **New** button. A dialog box opens.
4) In the **Variable Name** field, enter a unique name for the server variable.

   The server variable name may only contain letters (English alphabet), numbers, and the underscore symbol.

5) In the **Variable Value** field, enter all or part of the URL that should be the target of the Web link.

   If the Web link contains a special character according to XML standards (For example,: &, %, ;,<,>), these characters must be replaced by their XML conformant variant (For example,: `&` for &)

6) Click **OK** to save your changes. The server variable definition appears in the list of server variables.

To edit or delete the server variable, select the server variable in the table and click the **Edit** or **Delete** button below the table.

7) Click **OK** to save your changes and close the editor. The server variable definition is now available in the server alias configuration and can be used for the URL definition specified in Alfabet Expand.

You can use the server variable definitions in the following configurations for configured reports:

- in the definition of the **URL** property of external reports of the type `Extern`.
- in the definition of the **Help Index** attribute and **Custom Help Index** attribute of any configured report. For more information about configuring custom help, see the section Specifying Custom Help for a Configured Report or the chapter Providing Custom Online Help to the User Community.
- in the definition of database user names and passwords in the XML object ***DatabaseUsers*** that allows database access permissions to be restricted for configured reports of the type `Query` and `NativeSQL`.
- in any query results the content of a cell can be replaced with the value of a server variable using the instruction ReplaceServerVariable. For more information, see Adding the Value of a Server Variable to the Dataset.

A variable is referenced as `$`<server variable name>. For example, a server variable called `SQLSERVER` is referenced as `$SQLSERVER`.

In the definition of the **URL** property of external reports and in the definition of the **Help Index** attribute and **Custom Help Index** attribute of any configured report, the variable definition can either substitute the whole connection string or only part of the connection string. It is also possible to build the URL from a number of concatenated server variables. For the definition of database users in the XML object ***DatabaseUsers***, the value of a database user name and password can only be substituted completely with a server variable.

For example, an external report hosted on the company Web server is first tested on a test Web server. Therefore, the following Web server variables are defined in the server alias:

- `WEBSERVER`, specifying the Web server used
- `APPLICATION`, specifying the virtual directory for the referenced external reporting functionality on the Web server

The URL definition in the configuration of the dynamic Web link contains the variables instead of the current Web server name and virtual directory thereof:

```
HRef=http://$WEBSERVER/$APPLICATION/OrganizationReport.aspx
```

The definition is automatically converted into a valid link containing the values of the server variables. For example,:

```
HRef=http://localhost/Reports/OrganizationReport.aspx
```

Restriction of Database Access Permissions on Execution of Configured Reports

On the database server, multiple users can be created for a database with restricted rights. Users can have restricted rights for the complete database, like For example, read only rights or even different access rights to different database tables, like For example, read only access to one table only while access to other database tables is completely denied.

The database user that is used for connections from the Alfabet Web Application to the database server for access of the Alfabet database has very extended rights. You can configure all or individual reports to be executed with a database user different from the standard database user for connections from the Alfabet Web Application to enhance security. There are two levels of restrictions:

- The Alfabet Web Application can be configured to use a user with restricted access permissions for executing all configured reports. The database user is automatically created on the database server during installation and has read only access permissions to the database This option should not be selected if data cube based configured reports shall be executed. Access permissions of the automatically created database user with ReadOnly access do not allow cube processing.
- SQL users with password having specific access permission rights can be defined by the customer on the database server. The configured reports can then be configured to be executed with one of the customer defined database users. If a user opens a configured report and the database user defined for the execution of the configured report does not allow report execution, an error message informing the user about the inadequate access permissions for report execution and asking him/her to contact the system administrator for resolution of the issue. This method is currently only available for configured reports of the type `Query` and `NativeSql`. For configured reports of the type `Query` and `NativeSql`, the user of custom configured database users supersedes the configuration to use the Alfabet internal ReadOnly user.

The implementation of both security mechanisms includes system administration tasks that may not be executed by a solution designer. In the following all required configuration steps for both methods are described, including the tasks not performed with Alfabet Expand.

Using an Alfabet Internal ReadOnly User for Report Execution

During installation of the Alfabet components, a ReadOnly database user is automatically created on the database server for the Alfabet database. This ReadOnly user is re-created after each meta-model update. The Alfabet Web Application can be configured to connect to the Alfabet database with the database user with ReadOnly access rights when executing configured reports.

Please note that cube based reports cannot be executed when the database connection is set up with the ReadOnly database user.

Configuration is done with the tool Alfabet Administrator:

For information about how to access and work with the Alfabet database, see the chapter *Working with the Alfabet Administrator* in the reference manual *System Administration*.

1) In the explorer of the Alfabet Administrator, click the **Alfabet Aliases** node.
2) In the table, click the server alias that you want to configure.
3) In the toolbar, click the **Edit**  button. An editor opens.

4) In the editor, go to the **Server Settings** tab and open the tab **General**
5) Select the checkmark **Use ReadOnly User for Report Execution**.
6) Click **OK** to save your changes.

Configuring Database Users With Explicit Access Rights for Individual Configured Reports

Customer defined SQL users with password that are available at the database server can be used for connections of the Alfabet Web Application to the Alfabet database when executing configured reports of the type `Query` or `NativeSQL`.

One of the SQL users with password can be used as default user for all configured reports of the type `Query` or `NativeSQL`. Additional SQL users with password can be specified for execution of one or multiple individual configured reports.

The setting **Use ReadOnly User for Report Execution** in the server alias of the Alfabet Web Application is ignored if a definition of customer defined database users for report execution is available. The customer defined database users are only applied on configured reports of the type `Query` or `NativeSQL`. All other configured reports are then executed with the database user with extended rights that is the standard database user the Alfabet Web Application is connecting with to the database server.

The configuration includes three steps:

First step: Definition of SQL users with password on the database server host

An SQL user with password must be created and the access permissions configured on the database server. This is typically done by a database administrator. For details see the documentation of the database server used.

The access permissions to the Alfabet database for SQL users with password on the database server might be overwritten on update of the meta-model, For example, when migrating to another release or when taking over a configuration to the production environment via AMM file. It is recommended to use auto-run ADIFschemes for re-constitution of the permissions for SQL users with password after migration. For more information about ADIF schemes, see the reference manual *Alfabet Data Integration Framework*.

Second Step: Configuring the Alfabet Web Application to find the SQL users with password

The login credentials of all SQL users with password that shall be used for execution of configured reports must be defined in the configuration of the meta-model of the current database in Alfabet Expand. The SQL users with password are defined as database users in the Alfabet configuration.

1) Open Alfabet Expand and go to the **Presentation** tab.
2) Expand the node **XML Objects** > ***Administration***.
3) Right-click on ***DatabaseUsers*** and select **Edit XML** from the context menu. The XML object opens in the middle pane.
4) In the XML object, add one or multiple XML elements ***DatabaseUser*** to the XML root element ***DatabaseUsers*** and set the following XML attributes for each XML element ***DatabaseUser***:

- `Name`: Define a unique name for the database user. The name will be displayed in the drop-down list in the attribute **Database Restricted User** of configured reports of the type `Query` or `NativeSQL` that allows a database user to be assigned to the configured report.
- `DBUserName`: Enter the name of the SQL user with password defined in the database server that shall be used for authentication against the database server. Server variables can be used to define the name. For more information on server variables, see Using Server Variables in Web Link and Database Server-Related Specifications.
- `DBUserPassword`: Enter the password of the SQL user with password defined in the database server that shall be used for authentication against the database server. Server variables can be used to define the password. For more information on server variables, see Using Server Variables in Web Link and Database Server-Related Specifications.
- `IsDefaultForReports`: Set to `true`, if the database user shall be used for execution of all configured reports of the type `Query` or `NativeSQL` that do not have a database user explicitly defined. If the XML attribute is `false` or not set, the database user is not the default user.

  Only one of the defined database users can be defined as default for reports.

5) In the toolbar, click the **Save**  button to save your changes.

Step 3: Configuring reports to be executed with one of the defined database users

If the database user specification in the XML object ***DatabaseUsers*** includes the definition of a default user, all configured reports of the type Query or NativeSql are executed with the default database user without any further configuration required on the level of report configuration. If no database user is defined as default or if a default is defined, but a report shall be executed with another database user than the default database user, the database user for execution must be explicitly defined in the configured report's attributes:

1) Open Alfabet Expand and go to the **Reports** tab.
2) In the explorer, click on the node of the report for that you want to define a database user.
3) In the attribute window, select a database user from the drop-down list in the field **Database Restricted User**.

Restricting Access to Administrative User Profiles

User profiles can be defined as administrative user profiles with the **Is Administrative User Profile** attribute of the user profile. Access to a configured report can be limited to users currently logged in with an administrative user profile. Please note however, that this only applies to access via the **Configured Reports** functionality or page view. If a configured report is embedded into a guide page or an object cockpit, or object view, it is accessible to all users with access permissions to the guide page, object cockpit, or object view.

To restrict access to a configured report via the **Configured Reports** functionality or page view to users logged in with an administrative user profile:

1) If a report view has not yet been defined for the configured report, right-click the configured report and select **Create SQL Report View**. In the explorer, the SQL Report View is displayed as a subordinate object of the configured report.
2) Click the SQL Report View node and set the **Restrict to Administrative User Profiles** attribute to `True`.

Activating Navigation From Reports to Object Classes Not Automatically Offering Navigation

The user can navigate to the object view of an object displayed in configured reports of the type `Query`, `NativeSQL` or `Custom`.

Navigation to a base class of the configured report is activated per default for a preconfigured subset of object classes. For these object classes, navigation is available without further configuration for configured reports based on an Alfabet query and for configured reports based on a native SQL query, if the first argument in the `SELECT` clause returns the `REFSTR` of the base object class.

- For an overview of the navigable object classes for that navigation is activated by default, see *Overview of Configurable Features for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.
- For configured reports of the type Custom that are based on the template CustomPivotTable, navigation is not available by default. To activate navigation, the **Component** attribute of the report assistant for the configured report must be set to `PivotGrid` and navigation must be activated for all object classes, including the object classes that are automatically navigable from other reports, with the mechanism described below. Please note that activation of navigation for the object class `Artifact` will batch activate navigation to the object classes for that navigation is automatically available for other report types.
- For configured reports of the type `Custom` that display a chart report, like For example, a bar or line chart, navigation is not available by default. The required configuration is described in the section Configuring Navigation from a Chart Report.

If an object class is not offering navigation by default, navigation can be activated for that object class per configured report:

1) In the explorer of the **Reports** tab, expand the node of the configured report that shall provide navigation to the object class.

   The configured report must have a configured report view defined. If the node is not expandable, right-click the configured report's node and select **Create Sql Report View** for a configured report of the type `NativeSql` or `Query` or `Create Custom Report View` for a configured report of the type `Custom`.

2) Expand the report view node that is sub-node of the configured report's node.
3) Right-click the presentation object sub-node under the report view node and select **Create Item** from the context menu.
4) In the class selector, select the object class for that you want to activate navigation and click **OK**.

5) Make sure that the **Can Navigate** attribute of the new item is set to `True`.

For graphic configured reports of the **Type** `Custom`, the presentation object contains an item `Artifact` by default. If you would like to restrict navigation to a specific object class, you can delete the Artifact item that batch activates navigation to the object classes for that navigation is automatically available by default and add a new item for the object class that would like to navigate to.

Specifying Custom Help for a Configured Report

You can provide a link to custom help for any configured report created by your solution designer. You can specify only one custom online help per configured report. Users will be able to access the online help for the configured report that is currently displayed in the Alfabet user interface.

The custom help that you provide for your user community must be available via a URL that can be viewed in a browser. For each configured report, you can decide whether custom online help should be displayed and whether the standard online help should or should not be displayed. If you specify that both the standard online help and the custom online help are to be displayed, the link for the standard online help will be listed first in the **Help Selector**, followed by the custom help link. The online help entry will be displayed in the **Help Selector** with the syntax **Custom Help on<Configured Report Caption>**. For general information about the standard and custom context-sensitive help, see the section Understanding the Context-Sensitive Help.

Furthermore, you can specify content for the automated help assistant and make it available to a selected configured report. The automated help assistant will be displayed in a fly-in element in the upper-right corner of the user interface when the user accesses the configured report in the Alfabet user interface. The automated help assistant can be closed, whereby it will drop into the slide-in toolbar and can be opened for the current view at another time. For general information about the automated help assistant capability, see the section Understanding the Automated Help Assistant.

The custom help that you assign to a configured report will be available for every instance of that configured report. For example, a custom help link assigned to the configured report **Data Quality Checklist** in the context of a custom object view configured for the class `Application` will also be available in the configured report **Data Quality Checklist** in the context the object view for the class `Component`.

Server variables can be used in the **Custom Context-Sensitive Help URL** attribute and the **Automated Assistant URL** attribute to specify the custom help links. Server variables allow you to define all or part of the URL definition in the alias server setting instead of directly defining it in the configuration in Alfabet Expand. Use of server variables is, For example, useful in order to change a Web server name defined in multiple URL definitions in Alfabet Expand. Instead of changing all URL definitions, only the variable value in the alias server setting must be updated. For more information about defining server variables for custom online help, see the section Configuring Server Variables for the Custom Help.

To specify custom help for a configured report:

1) Go to the **Reports** tab and navigate to the configured report that you want to define a custom help for.
2) Ensure that the configured report has **Report State** set to **Plan**. To change the report state, right-click the configured report and select **Set Report State to 'Plan'**.

3) Click the report view node below the configured report to open the attribute grid. Define the following attributes, as needed:

- **Automated Assistant URL**: Enter the URL or server variable that targets the content to display in the automated help assistant assigned to the configured report.

  The size of the automated help assistant, its header color and position in the Alfabet user interface, and the notch in the slide-in toolbar are configured for each configuration object type in the GUI scheme. For more information about defining GUI schemes, see the section Configuring GUI Scheme Definitions for the Alfabet Interface. For details about the individual GUI scheme attributes that are relevant for the automated help assistant, see the section *Overview of GUI Scheme Attributes* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

- **Custom Context-Sensitive Help URL**: Enter the URL or server variable that targets the custom context-sensitive help.
- **Standard Context-Sensitive Help Index**: This field will be empty for a configured report.

  The **Standard Context-Sensitive Help Index** attribute on the configured report node (not the configured report view node) can be edited for purposes of backward compatibility. Prior to release 10.1, users specified the location of the external help file using the following syntax: `CUSTOM_HELP:<URL>` (For example,: `CUSTOM_HELP:http://helphost/report1.html`). The link to the external help file will be displayed in the **Help Selector**. Server variables can be used in the help links. It is recommended that you specify your custom help by means of the **Custom Context-Sensitive Help URL** attribute on the report view node.

- **Standard Context-Sensitive Help Visible**: Select `True` to display both the standard and custom context-sensitive help file in the **Help Selector** Select `False` if the link to the standard context-sensitive help should not be available and should not be displayed in the **Help Selector**.

4) In the toolbar, click the **Save** button to save your changes.

Assigning a Category for Specific Functional Use to a Configured Report

For the following Alfabet functionality, configured reports either must or can optionally provide required data in a defined structure:

- Compliance policies
- Compliance prioritization for compliance projects
- Questionnaire policies
- ADIF schemes
- Constraints
- Events
- Data capture templates
- Conditions

- Rules for custom diagram definitions
- Chart view reports

Reports can only be used for these functionalities if they are assigned to a category related with the functionality. The name of the categorie related to each functionality can be assigned to the functionality by the customer in the XML object ***UseCaseCategories***. After having assigned a name for the category in the XML object ***UseCaseCategories***, the category must then be assigned to the configured report via the attribute **Category** of the configured report.

To set the category name in the XML object ***UseCaseCategories***:

1) In the **Presentation** tab of Alfabet Expand, expand the explorer node **XML Objects**.
2) Right-click on the XML object ***UseCaseCategories*** in the explorer and select **Edit XML** from the context menu. The XML object opens in the middle pane.
3) In the XML object, add an XML element `UseCaseInfo` as child element of the root XML element `UseCaseCategories` with the following XML attributes and child elements:

```
<UseCaseInfo UseCase="UseCaseName">
<ScopeInfo Scope="Report" Categories="CommaSeparatedListOfCategories" />
</UseCaseInfo>
...
```

4) Set the XML attributes `UseCase` and `Categories` as follows:
   - `UseCase`: Enter the value corresponding to the functionality you want to implement:
     - `Compliance`: Configuring compliance policies for compliance management. For more information about the functionality, see Configuring Queries for Compliance Policies.
     - `QuestionnairePolicies`: Defining users to answer questions in a questionnaire and objects about which questions shall be answered. For more information about the functionality, see Configuring Reports Finding Objects Or Users for the Questionnaire Policy.
     - `Diagrams`: Defining rules to automatically add objects to custom diagrams. For more information about the functionality, see Defining Rules to Automatically Add Objects to New Diagrams.
     - `Conditions`: Configuring conditional constraints for object profiles and object cockpits. For more information about the functionality, see Configuring Conditional Constraints for an Object Profile and Configuring Conditional Restraints in the Object Cockpit.
     - `JobScheduler`: Limiting the rescanning of indicators via the Job Schedule functionality to a subset of objects of a defined object class. For more information about the functionality, see Configurations for Scheduling Rescan of Indicators.
     - `Events`: Configuring events for triggering a RESTful service call with a dynamic method path or with a JSON payload. For more information about this functionality, see Defining a Dynamic Method Path for the RESTful Service Call and Defining the JSON Payload for the Body of the Service Call.
     - `CustomChartViews`: Generating a configured report that will be used in a report collection for an object class or object class stereotype. The configured report will be

available via a tab in all tabular configured reports about the object class or object class stereotype and provides additional information about all objects listed in the tabular report. For more information about this functionality, see Integration of Configured Reports as Report Collection Into Tabular Configured Reports.

- GenericOASIntegration:
- `DataCaptureTemplates`: Providing data for the generation of data capture templates. For more information about this functionality, see *Capturing Data with Data Capture Templates*.

- `Categories`: Enter a category name. If you enter more than one name comma separated, all names will be valid as category name for the functionality.

5) In the toolbar, click the **Save** button.

Handling Long Running Reports

While a configured report is loading, the user cannot perform any other activity on the Alfabet user interface of the current session. This can lead to problems with long running reports. If report execution requires For example, 10 minutes, the user will have to wait for the report execution to finish prior to doing other work in Alfabet without the ability to interrupt report execution. There are several mechanisms available for dealing with long running reports:

- Defining an Execution Timeout for Long Running Reports
- Open Tabular Reports with Filters Without Immediate Execution
- Configuring Offline Execution for Long Running Tabular Reports with Filters

Defining an Execution Timeout for Long Running Reports

You can configure a maximum time that the Alfabet Web Application shall try to execute a report that requires an exceedingly long execution time. The report execution will be terminated with an error message after the defined time has elapsed. This mechanism prevents a user session from being blocked for an excessively long time because of a long running report execution.

Report execution timeout is configured in the serer alias of the Alfabet Web Application. Server alias configuration is done in the tool Alfabet Administrator:

1) In the explorer of the Alfabet Administrator, click the **Alfabet Aliases** node.
2) In the table, click the server alias that you want to configure.
3) In the toolbar, click the **Edit** button. An editor opens.
4) In the editor, go to the **Command** tab.
5) Define the following attribute:

- **Report Execution Timeout**: Define the time in seconds after which the report execution will be terminated with an error message. By default, the Report Execution Timeout attribute is set to -1 and there is no restriction in report execution time.

Please note that **Report Execution Timeout** should have a lower value than the HTTP request timeout defined for the Alfabet Web Application. The definition of the HTTP request timeout will supersede the specification of the **Report Execution Timeout** attribute. The default HTTP request timeout of the web server is 110 seconds, if not otherwise specified in the `web.config` file of the Alfabet Web Application. If the value of the **Report Execution Timeout** attribute is higher than the HTTP request timeout, the number of seconds of the **Report Execution Timeout** attribute will be reduced to the HTTP request timeout minus ten seconds. The example `web.config` files delivered with Alfabet will contain a specification of an HTTP request timeout of 60 seconds with the entry: `<httpRuntime executionTimeout="60" />`

6) Click **OK** to save your changes.

Open Tabular Reports with Filters Without Immediate Execution

By default, configured reports are executed directly when opened by a user. This behavior can be changed for tabular reports of the type `Query` or `NativeSql` to avoid exceedingly large result datasets on execution with empty filter fields. The report will then only show results when the user clicks the **Submit** button for the filter.

Do the following to prevent a tabular configured report from being executed immediately on access:

1) Open Alfabet Expand and go to the **Reports** tab.
2) In the explorer, click on the node of the report for that you want to execute on submit only.
3) In the attribute window, set the **Execute on Enter** attribute to `False`.

When you set **Execute on Enter** to `False`, you must make sure that the **Submit** button is available in the SQL report view of the configured report. If the **Submit** button is deleted from the SQL report view, the configured report cannot be displayed. The setting and execution of the **Execute on Enter** attribute is independent from the definition of filters for a configured report. A configured report without filters must nevertheless have a **Submit** button when **Execute on Enter** is set to `False`.

Configuring Offline Execution for Long Running Tabular Reports with Filters

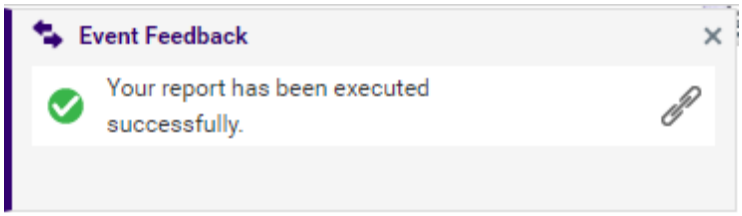
An offline execution mechanism is available for long running tabular reports that allows users to work in the Alfabet user interface while the configured report is being executed. Instead of requiring the user to wait for the report to load, the user will be prompted to define a name for the resulting report output when he/she opens the report or submits changed filter definitions for the report.

The user can then continue working on other Alfabet views until the report is fully loaded. He/she will be notified via an asynchronous response window which provides a link to open the report results in a new browser tab. This enables the user to view the report results while at the same time keeping the view open that he/she is currently working in when report results are available. Filters are displayed view only in the report results.

It is recommended not to execute configured reports with filter fields storing the filter settings into the `UserGlobalData` object class offline. On rendering of the offline executed report results, the values stored in the `UserGlobalData` object class will be overwritten with the filter values defined on triggering the report execution of the current report results. The opening of offline executed report results in a new tab might change a filter setting depending on the same object of the `UserGlobalData` object class for a report opened in parallel in the original tab for working with Alfabet without further notice.

For more information about filter field values stored in the `UserGlobalData` object class, see Configuring Reusability of Filter Settings Across Reports.

The event feedback window displays a text that is configurable for each offline executed report and a link symbol. Moving the cursor over the link symbol will show the name the user defined for the execution of the report in a tooltip.



The asynchronous response window opens for a few seconds and then moves into the slide-in toolbar where it is available as long as the user is working with the same view that was open when the message appeared. If the user changes to another view or logs out and logs in again, the asynchronous response cannot be opened separately any longer, but report results are still available for a configurable period of time and the user can access the report results during this time period via one of the following mechanisms:

- The **My Last Event Feedback** notch in the slide-in toolbar provides a link to the report results similar to the one in the **Event Feedback** window.
- If the user re-opens the long-running configured report while report results are available, he/she is offered to view the existing report results instead of triggering a new report execution.

Offline execution limits the availability of the configured report for other functionalities:

- Offline executed reports cannot be embedded in object cockpits. They are Therefore, not displayed in the drop-down list in the **Source** attribute to select a configured report as the source of a presentation object control in an object cockpit. If offline execution is configured for a configured report already embedded in an object cockpit, a user opening the object cockpit will see a message that the report cannot be displayed because of offline execution.
- Offline executed reports must not be used for REST API calls to the objects endpoint of the Alfabet RESTful services.

Offline execution requires the following preconditions:

- Offline execution is only available for configured reports of the type `Query` or `NativeSql` with a filter definition.
- Offline execution must be activated and configured in the attributes of the configured report. The configured report is then executed offline for all users having access to the configured report.
- An Alfabet Server must be running that is connected to the same Alfabet database than the Alfabet Web Application.

To execute a report offline, the following configuration is required:

1) Open Alfabet Expand and go to the **Reports** tab.
2) In the explorer, click on the node of the report for that you want to execute offline.
3) In the attribute window, set the **Execute Offline** attribute to `True`.
4) In the attribute window, set the following attributes:
   - **Estimated Execution Time**: The maximum time in minutes allowed for report execution. This value overwrites a general report execution timeout on the Alfabet Web Application for this report.
   - **Offline Completion Message**: A message that will be displayed to the user in the slide-in **Event Feedback** window that opens when the report execution has finished. If no text is provided, the message window will not contain any text but only the symbol for execution success and the link to the report results.
   - **Offline Result Retention Time**: The time in hours that a generated result for the configured report will be available for the user. A regular process is in place to remove results from offline executed reports once their retention time has elapsed.
5) In the toolbar, click the **Save**  button to save your changes.

Analysis of Reports via the Semantic Analyser

The semantic analysis functionality for configured reports scans all queries in a configured report for references to Alfabet object classes and object class properties thereof.

The information is used for the following functionalities:

- The **Show Usage** functionality in the context menu of object class properties evaluates the use of the object class property in configured reports on basis of the semantic analysis results.
- The AlfaBot uses the semantic analysis to fine tune the search results for the `Analyze` intent finding configured reports displaying information about defined object class properties and values thereof.

The results are listed in a node **Semantic Analysis** beneath the node of each configured report in a hierarchical structure going down from object classes to object class properties to values of object class properties. The Name property of object classes and object class properties is used in the explorer.

If object class properties restrict the display of objects in the report via a `WHERE` clause definition, the values will be included into the semantic analysis for object class properties returning one of the following:

- The value of a standard `Stereotype` object class property.

The semantic analysis ignores value definitions including a wildcard and checks whether the object class stereotype name equals a stereotype name defined for the object class. Only existing object class stereotypes are listed.

- The value of a the `Name` of an object.

The semantic analysis ignores value definitions including a wildcard.

- A Boolean value.
- A value from an enumeration.

The semantic analysis ignores value definitions including a wildcard and checks whether the enumeration value is defined for the enumeration defined for the object class property. Only existing enumeration items are listed.

Scanning is done automatically when the report state is set to `Active`.

Please note the following:

- If the **Semantic Analysis** node is empty after setting the configured report to `Active`, this is an idicator for an error in the underlying query, For example, a typo in an object class name.
- If you are importing configured reports stored in an XML file or AMM file from a database version Alfabet 10.6.x or lower, the semantic analysis node will not be available for the imported reports. The status of the imported configured reports must be re-set to `Plan` and set back to `Active` to start the analysis.

The analysis scans both Alfabet queries and native SQL queries. It comprises object class properties referenced by any part of the query including joins and the object class properties of the type `ReferenceArray` which are included in native SQL queries via the `PROPERTY` column of the `RELATIONS` database table.

Alfabet queries and native SQL queries are handled differently:

- The queries in a report based exclusively on Alfabet queries will be re-scanned each time the report state is set to `Active`. The solution designer cannot edit the list of results.

Indicator and role definitions added via the show properties of the type `Indicator` or `Role` are not included into the semantic analysis results.

The REFSTR object class property of the FIND class of the Alfabet Alfabet query is added as object class property to the semantic analysis.

- Native SQL queries can be very complex and in some cases, the scanning mechanism may not find all relevant object classes and object class properties. Therefore, the solution designer can add missing information to the semantic analysis of any report including at least one native SQL query via the **Add Properties** option in the context menu of the **Semantic Analysis** node. Object class property values included in the semantic analysis are ReadOnly and excluded from editing. The semantic analysis is executed each time the report state is set to Active. The manually-added semantic analysis results will be left unchanged during a subsequent semantic analysis.

Information about object classes and object class properties may be missing in the analysis for one of the following reasons:

- The query definition is incorrect. For example, an object class or object class property name may have been added with a typo. It is recommended to check the query definition for correctness if the semantic analysis is incomplete.
- Native SQL queries can be very complex and the scanning mechanism for semantic analysis may fail to identify object class or object class property specification.

If the semantic analysis of a correct native SQL query is incomplete, missing information can be added manually to complete the analysis:

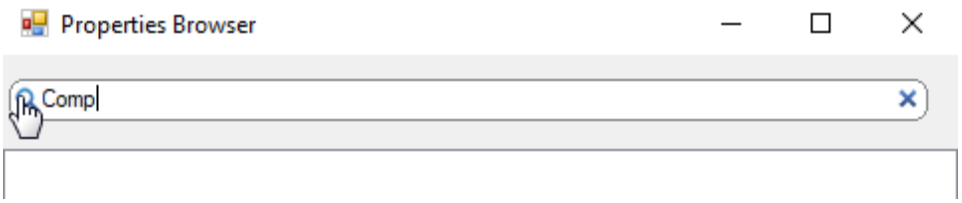
- Adding an Object Class to the Semantic Analysis
- Adding an Object Class Property to the Semantic Analysis
- Removing Manually Added Entries From the Semantic Analysis

If you encounter similar problems with the semantic analysis for various different native SQL queries, please inform Software AG Support. this will enable Software AG to futher enhance the automatic scanning of native SQL queries.

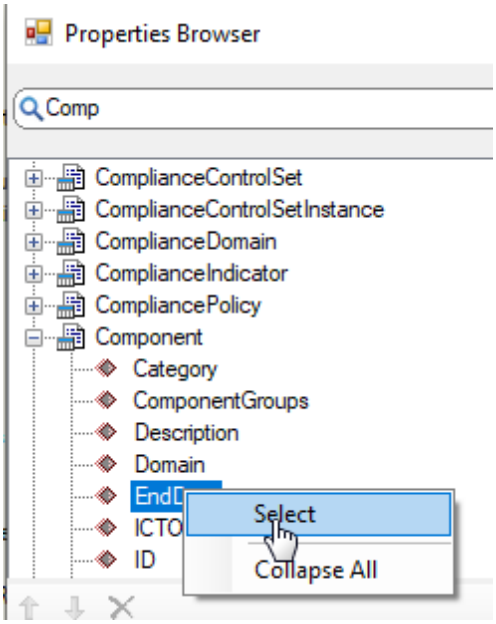
Adding an Object Class to the Semantic Analysis

To add information about an object class missing in the automatic semantic analysis output:

1) Right-click the **Semantic Analysis** node and select **Add Properties**.
2) In the selector window, type at least the first three letters of the object class name into the search field and click the search icon on the left of the field.



3) All object classes matching your search are listed in the middle field of the selector. Expand the relevant object class node, right-click the relevant property and select **Select** from the context menu.



4) Click OK. The object class and the selected object class property is added to the semantic analysis explorer persistently.

Adding an Object Class Property to the Semantic Analysis

To add information about an object class property missing in the automatic semantic analysis output for an object class already included in the semantic analysis:

1) Right-click the object class node and select **Add Properties**.
2) In the selector window, right-click the relevant property and select **Select** from the context menu.
3) Click **OK**. The object class property is added to the semantic analysis explorer persistently.

Removing Manually Added Entries From the Semantic Analysis

If you added an object class property manually to the semantic analysis of a configured report, it will not be removed the next time the configured report status will be set to `Active`, even if it is no longer used in the query. Manually added entries in the semantic analysis can only be removed manually:

- To remove a manually added object class property from the semantic analysis, right-click the object class property node and select **Remove Property**.
- To remove a manually added object class including all object class properties thereof from the semantic analysis, right-click the object class node and select **Remove Class**

Creating a Tabular Configured Report of the Type Query

To generate a configured report of the type `Query`:

- In the **Reports** tab of Alfabet Expand, create a new configured report and define the general attributes for the configured report and the attributes specific for Alfabet query-based configured reports. For more information about creating a configured report and defining the configured report's attributes, see Creating a New Alfabet Query-Based Configured Report.
- Configure an Alfabet query for the configured report. For more information, see Configuring an Alfabet Query for a Configured Report. If the configured report result shall be an expandable table, the Alfabet query must be configured to contain instructions. This is described separately in the section Grouping Query Results in Expandable Reports in the chapter Defining Queries.
- If the configured report has filters defined, create an SQL report view, check the functionality of the filters, and optionally change the captions and look of the filter fields. For more information, see Defining Filters for Configured Reports and Selectors in the chapter Defining Queries.
- Define access rights for the configured report in the **Reports Administration** functionality of Alfabet. By default, access is possible for all users. For more information, see the chapter *Defining and Managing User Access to Configured Reports* in the *User and Solution Administration* reference manual.
- Set the **Report State** of the configured report to `Active`. For information on how to change theattribute of the configured report, see.

The following information is available:

- Creating a New Alfabet Query-Based Configured Report
- Configuring an Alfabet Query for a Configured Report
- Defining Filters for an Alfabet Query Based Configured Report
  - Defining a Where Clause that Causes the Generation of a Filter Field in the Configured Report
  - Creating SQL Report Views to Configure Filters
- Defining a Data Capture Environment for a Configured Report of the Type Query
- Defining a User Management Functionality via a Configured Report of the Type Query
- Allowing the User to Change the Number and Order of Columns in the Table
- Configuring the Analysis of the Tabular Report in a Pivot Grid
- Configuring a Data Table Report with Restructuring Options for the End User

Creating a New Alfabet Query-Based Configured Report

To create a new Alfabet query-based configured report in Alfabet Expand:

1) In the **Reports** tab, right-click the **Reports** folder or a sub-folder thereof and select **Create New Report**. You will see the new configured report listed below the selected report folder.

As an alternative to defining all attributes for a new configured report, you can copy and paste an existing configured report and make modifications, as needed. To do so, right-click the existing configured report that you want to copy and select **Copy**. Right-click the new configured report and select **Paste**. Modify the necessary attributes described below.

2) In the property window, define the following attributes for the configured report:

- **Name**: Optionally, change the technical name of the configured report. The technical name of the configured report must be unique.

  The technical name is used for identification of the configured report in the context of technical processes such as merging the report configuration with a report configuration saved as an XML file. Therefore, the following is required when defining the name of a configured report:

  - The name must be unique. The name must be unique regardless of the report folder it is structured in.
  - The name may not be changed once a configured report has been created.
  - A validation mechanism checks for correct syntax when defining a technical name. Please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.
  - If the technical name of the configured report is changed, the name will be correctly updated in other configuration objects referencing the changed report during design time.

  Please note that the name of a changed configuration object will not be updated in guide pages or guide views. If you plan to change the name of a configuration object, the reference in the navigation page should be changed prior to changing the configuration object name. You can use the **Show Usage** functionality in the context menu of the configured report to evaluate whether the configured report has been added to a guide view or guide page.

- **Caption**: Enter a caption for the configured report. The caption you define here will be displayed in the **Reports Administration** functionality in the **Administration** module and the **Configured Reports** views of the Alfabet interface. If the configured report is assigned to an object view as a page view, the text will be displayed as the page view caption in the object view. The caption of the configured report may exceed the conventional 64 character limit.
- **Description**: Provide a short information about the configured report that is useful to Alfabet users. This comment will be displayed on the Alfabet user interface in the header of the configured report in a single line under the report caption and the **Description** field for the configured report in the table of all views listing configured reports, like the **Configured Reports** views of the **Search** functionalities. If the configured report is assigned to an object view as a page view, the text will be displayed under the page view caption as short description in the object view.

In the configured report, the description will be truncated if it is longer than one line on the screen. Therefore, the description should be short to ensure complete display in the configured report header.

- **Usage Guideline**: Provide information that helps users to execute and interpret the configured report. This information will not be displayed directly on the user interface. The user can access the information using the following mechanisms:
  - If the user moves the cursor over the description provided for the configured report with the attribute **Description**, a tooltip opens that includes both the description and the usage guideline.
  - The **Options**  button will be displayed on the upper right with an option **Help On Filter Fields** to open a new window displaying the usage guideline in addition to any filter field hints, if configured.

    For views that have filters defined, the **Options**  button will only be displayed if the filter panel either allows batch clearing of filter fields or collapsing and expanding of the filter panel. For more information about the configuration of these functionalities, see Configuring the Complete Filter Area to be Collapsible and Allowing the User to Clear the Filter Area.

- **Technical Comment**: Provide information that is useful for solution designers working with Alfabet Expand in order to better understand the configured report from a technical perspective. The **Technical Comment** will not be displayed in the user interface. Nevertheless, you can configure the interface to display the information in, For example, the attribute section of the configured report's object profile in the **Reports Administration** functionality.
- **Type**: Select `Query`. This is the default for new configured reports.
- **Report State**: The **Report State** attribute is view only and is set to `Plan` for new configured reports. The configured report can only be edited when the **Report State** attribute is set to `Plan`. After finishing all configuration steps described in the following, the **Report State** attribute must be set to `Active` as described in the section General Guidelines for Creating Configured Reports. The configured report is then visible to users in the Alfabet user interface.
- **Alfabet Query**: Click the **Browse**  button to open the **Alfabet Query Builder** and define the Alfabet query for the configured report.

  For information about how to work with the **Alfabet Query Builder**, see the chapter Defining Queries.

  For special hints for building an Alfabet query for configured reports, see Configuring an Alfabet Query for a Configured Report in this section.

  For a description of how to define report filters in an Alfabet query, see Defining Filters for an Alfabet Query Based Configured Report.

  For a description of how to define Alfabet queries for expandable report tables, see the section Grouping Query Results in Expandable Reports in the chapter Defining Queries.

- **Execute on Enter**: Set to `True` to execute the configured report with empty filter settings when the user opens the configured report on the Alfabet interface. Set to `False` to open the configured report without execution of the Alfabet query with empty filter settings. By default, this attribute is set to `True`. It should only be set to `False` when a configured report would result in an exceedingly big dataset when executed with empty filter settings.

  When you set **Execute on Enter** to `False`, you must make sure that the **Submit** button is available in the SQL report view of the configured report. If the **Submit** button is deleted from the SQL report view, the configured report cannot be displayed. The setting and execution of the **Execute on Enter** attribute is independent from the definition of filters for a configured report. A configured report without filters must nevertheless have a **Submit** button when **Execute on Enter** is set to `False`.

- **Help Index**: Specify the location of the external Help file using the following syntax:(For example,:). The external help file will be displayed when the user clicks the context-sensitive Help link in the standardHelp menu.

  Server variables can be used in the Help links. Server variables allow you to define all or part of the URL definition in the server alias configuration of the Alfabet Server instead of directly defining it in the configuration in Alfabet Expand. Use of server variables is useful, For example, in order to change a Web server name defined in multiple URL definitions in Alfabet Expand. Instead of changing all URL definitions, only the variable value in the server alias configuration must be updated. The use of server variables is described in the section Using Server Variables in Web Link and Database Server-Related Specifications in the chapter Configuring Reports.

- **Apply to Class**: To make the configured report available on the object profile of objects of one specified object class, click the **Browse**  button to open a dialog box and select the object class or object class stereotype to which you want to apply the configured report and click **OK**. If the attribute **Apply to Class** is set, the Alfabet query for the configured report must include a `WHERE` clause using the Alfabet parameter:`BASE` to refer to the current object. When the user opens the configured report, results related to the current object are displayed. A selector that is automatically added to the top of the configured report allows the user to select another object of the object class or object class stereotype that the configured report is applied to and view the results related to the selected object. If the configured report is added to a custom object profile or an object cockpit of the selected object class or object class stereotype, it opens automatically for the current object.

- **Database Restricted User**: Optionally select a database user that shall be used by the Alfabet Web Application for connecting to the Alfabet database instead of the standard database user with extended access permission when executing the configured report. The drop-down list is only filled if the use of different database users for configured report is specified in the XML object ***DatabaseUsers***. For more information about the concept of restricting access permissions for configured reports and the implementation of execution of configured reports with restricted database user rights, see the section Restriction of Database Access Permissions on Execution of Configured Reports.

- **Selector Behavior**: The attribute can be used to exclude configured reports that are defined for special purposes, like For example, triggering of data export via the RESTful API, from being selectable in the **Configured Reports** table. By default, the attribute is set to `Visible` and the configured report is visible in the selector for adding configured reports to the **Configured Reports** table if all other conditions for access to the configured report apply. If the attribute is set to `NotVisible`, the configured report is excluded from the selector for

adding configured reports to the **Configured Reports** functionality. Please note however, that this setting is not excluding the configured report from any standard views or customer configured views and selectors, but exclusively from the standard selector for configured reports implemented in Alfabet.

The attribute cannot be edited in the attribute window directly. To change the setting, right-click the configured report and select **Set Report Selector Behavior to 'Not Visible'** or **Set Report Selector Behavior to 'Visible'** respectively.

- **Applicable for AlfaBot**: Specify whether the configured report is accessible via the AlfaBot. By default, the attribute will be set to `True` for new reports. Set the attribute to `False` if the configured report should not be opened outside of a specific context. For example, this may be relevant for reports that are specifically designed to be embedded in an object cockpit.

  This attribute is only relevant if the AlfaBot is implemented. For information about the implementation of the AlfaBot, see Implementing the AlfaBot for Navigation via a Full-Text Command.

- **Business Problem Statement**: Optionally enter a description that will help users to search for the configured report in the AlfaBot. The text is not displayed in the user interface, but it is of special relevance for the search mechanisms implemented for the AlfaBot. If the report caption entered by the user in the AlfaBot for a request to open a configured report does not match the caption of one of the available configured reports, the AlfaBot will split the caption entered by the user into keywords and will perform a keyword search on the **Caption**, **Description** and the **Business Problem Statement** attributes of the configured reports that are accessible via the AlfaBot. In addition to a keyword search, a synonym search is performed on the text provided in the **Business Problem Statement** attribute.

  This attribute is only relevant if the AlfaBot is implemented. For information about the implementation of the AlfaBot, see Implementing the AlfaBot for Navigation via a Full-Text Command.

- **Base Classes**: Define the object classes that must be available to run the configured report. If this attribute is set, the Alfabet Server checks whether the object classes specified as base classes are excluded from the view scheme used to access the configured report. If the user does not have access permissions to one of the base classes, the configured report will be disabled. To set the attribute, click the **Browse** button to select the object classes that are required to run the configured report and click **OK**.

- **Can Create Express View**: Select `True` if an express view may be created for the report. Select `False` if an express view may not be created for the report. Please note that for reports that have a custom report view, the setting must be consistent with the setting of the attribute **Can Create Express View** of the custom report view.

  An express view is a link to a specific Alfabet view displaying data that allows individuals inside or outside of the user community to view information in Alfabet. When the express view is created, an email notification is automatically mailed to a specified recipient. The recipient receives a URL that allows him/her to access the current page view in Alfabet. For more information, see the section *Sending and Receiving Express Views* in the reference manual *Getting Started with Alfabet*.

- **Chart Views Mode**: This attribute defines whether additional information about the objects found in the configured report will be available via standard page views or configured reports directly accessible via links from the configured report view. Select one of the following:
  - `Standard`: A button **Chart Views** for opening a standard lifecycle chart, business support map, information flow diagram or portfolio for an object selected in the result dataset of the configured report is available automatically in the toolbar of the configured report if the following applies:
    - At least one of these standard reports is available for the object class the report finds objects for.
    - No Alfabet query language instruction is defined in the Alfabet query of the configured report.
    - The **Enable Chart Views** attribute is set to **True**.
  - `Custom`: All configured reports defined in the **Supplementary Reports** attribute of the class settings of the object class found via this configured tabular report are available via tabs on top of the toolbar of the tabular dataset. If a filter is available, the tabs are located between the filter area and the toolbar of the dataset. For information about the configuration of configured reports suitable to be opened via tabs in tabular configured reports and the configuration of the class setting to provide the configured reports, see Integration of Configured Reports as Report Collection Into Tabular Configured Reports.
  - `None`: Neither standard page views nor configured reports are directly accessible via links from the configured report view.
- **Custom Chart View Base Class**: This attribute is only visible if the **Chart Views Mode** attribute is set to `Custom`. Select the object class or object class stereotype for which the tabs opening the configured reports for the report collection shall be displayed. The configured reports defined in the **Supplementary Reports** attribute of the class settings of the object class or object class stereotype will be available via tabs. Please note that the selected object class must be identical to the object class found in the tabular dataset of the configured report you are currently configuring. If the objects found in the tabular dataset do not match the setting, the report will not show the tabular dataset, but an error message about the incorrect setting in the configuration.

3) In the toolbar, click the **Save** button to save your changes.

Configuring an Alfabet Query for a Configured Report

To define or edit the Alfabet query of a configured report:

1) In the table, select the configured report whose Alfabet query you want to define or edit.
2) Click the **Browse** button in the **Alfabet Query** attribute. In the dialog box that opens, select the base class for the Alfabet query. This class is the object class a user can navigate to when clicking a result in the configured report. The **Alfabet Query Builder** opens.
3) In the **Query** tab of the **Alfabet Query Builder**, define the Alfabet query for the selection of objects for the configured report.

For information about how to work with the **Alfabet Query Builder**, see the section Defining Queries.

The following is important for the generation of an Alfabet query for a configured report:

- If the configured report is applied to an object class with the attribute **Apply to Class**, you must configure the Alfabet query to show results for the current object that is selected by the user when opening the configured report only. To refer to the current object, use the Alfabet parameter "`:BASE`". For example, a configured report with the base class `Application` with a `WHERE` clause "`WHERE Application.REFSTR=:BASE`" will show results for the application that the user is currently working with only. A selector is added automatically to each configured report that is applied to an object class by means of the attribute **Apply to Class** . The selector allows the user to display other base objects in the configured report. If you do not specify a `WHERE` clause with the Alfabet parameter:`BASE`, the selector will still be present, but it will not have any impact on the resulting configured report.
- You can define filters for a configured report to allow the users to specify the range of results in the configured report. You can generate a filter from any `WHERE` clause of the configured report by clicking the checkbox **Parameter** in the condition value field. For more information about configuring filters, see Defining Filters for Configured Reports and Selectors in the chapter Defining Queries.

4) In the **Show Properties** pane, define the object class properties that are displayed in the columns of the configured report. In the **Sort Properties** pane, you can optionally select object class properties to sort the results sets in alpha-numeric order when the configured report is executed. The user can change the sort order of datasets in the report table by clicking the header of any column to display results in alpha-numeric order of the selected column's values.
5) Click **OK** to save the Alfabet query.
6) In the toolbar, click the **Save** button to save your changes.
7) In the explorer, right-click the configured report and select **Review Report**. The configured report opens in a web browser. If the results are not as expected, you can edit the Alfabet query until the output of the configured report meets your expectations.

The **Review Report** functionality also allows the visibility of toolbar buttons of the configured report to be defined for different view schemes. For more information, see Refining Visibility Issues in the View Scheme.

The amount of data that can be displayed in a configured report is restricted in order to avoid exceeding memory size usage. If a configured report results in an error message informing you that the data set is too large, refine the query to return only a subset of the results. If you do not want to exclude objects from the configured report but rather would like to limit the set of data that is displayed, you can define filters that allow the user to restrict the data display as needed (For example, by displaying only objects with a name starting with a specific character or objects assigned to a specific ascendant object).

Defining Filters for an Alfabet Query Based Configured Report

You can allow users to select the output of a configured report at runtime by means of filters that you configure.

The following steps are required to define filters for a configured report based on an Alfabet query:

- Define one or more `Where` clauses in the Alfabet query that are configured to generate a filter field. For more information, see Defining a Where Clause that Causes the Generation of a Filter Field in the Configured Report.
- Create an **SQL Report View** for the configured report and edit the attributes of the filter fields in the SQL report view, if applicable. For more information, see Creating SQL Report Views to Configure Filters.
- Test the functionality of the filter fields in the configured report or custom selector.

Defining a Where Clause that Causes the Generation of a Filter Field in the Configured Report

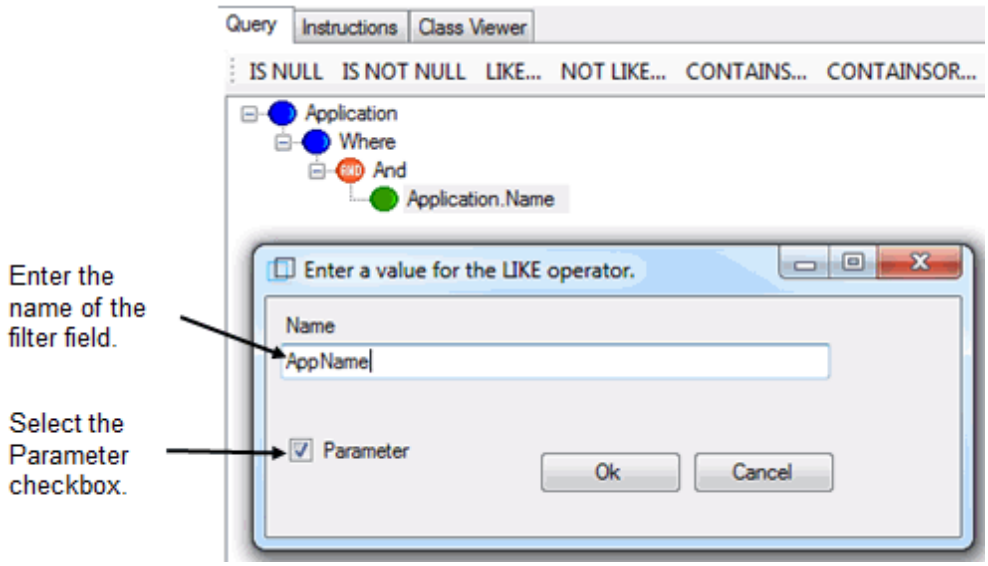
To set a filter for a configured report, you must configure a `Where` clause for the object class property you want the user to define in the filter field. The `Where` clause must be defined as follows:

- When defining the value for the condition, enter `@` followed by a technical name of the filter field. The value is not used to search in the property, but rather only to assign a technical name to the filter field.

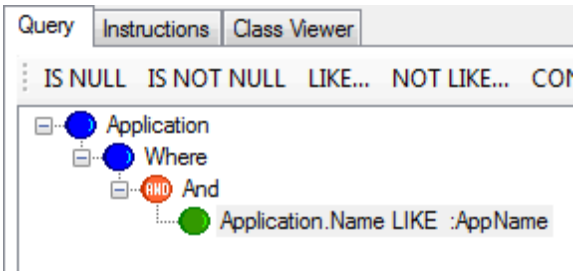
An example `Where` clause for the generation of a filter field:

```
WHERE APPLICATION.NAME LIKE (@AppName)
```

When defining the value for the condition in the **Alfabet Query Builder**, select the checkbox **Parameter** in the dialog box, shown below. In the **Name** field, enter a technical name of the filter field. The value in the **Name** field is not used to search in the object class property, but rather only to assign a technical name to the filter field.



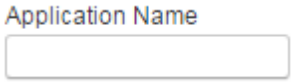
After clicking **OK**, you will see the `Where` clause in the **Alfabet Query Builder** with a colon written before the value. This is an alternative syntax to the definition of the parameter with an `@` prefix.



- Select a condition operator that allows the specification of a value. For example, `IS NULL` cannot be used for a filter because it does not require the specification of a value whereas, For example, the condition `LIKE` allow the specification of filter fields.

Depending on the data type of the property and condition that you have configured, different type of filter fields may be defined:

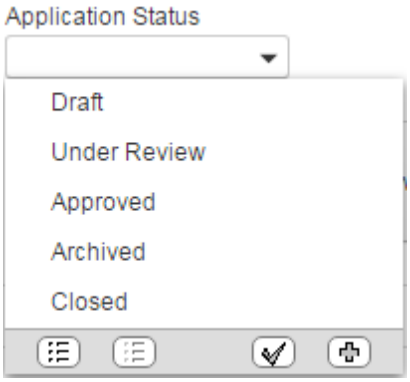
- Editable fields:



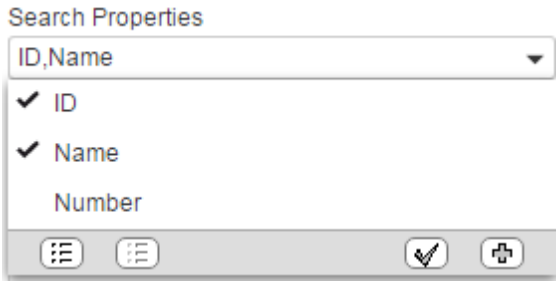
Editable fields can either allow wildcards in the search string or to search for a completely matching string only.

- Editable fields with a selector that allows to select an object from an object selector:

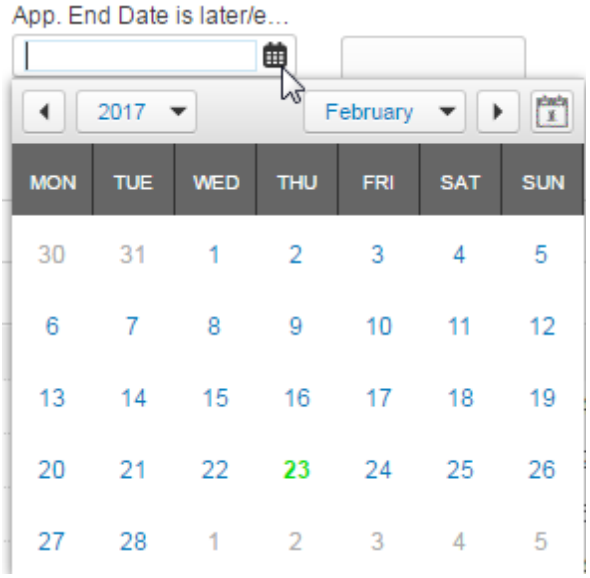
- Combo boxes allowing one value to be selected from a set of values



- Checked combo boxes allowing multiple values to be selected from a set of values



- Calendar fields with date pickers



The following tables lists all condition operators that allow the generation of filter fields for a specific data type and the corresponding type of filter field. Some of the filter fields require configuration of the attributes of the filter fields after creation of the SQL report view for the configured report. If this is required, it is also mentioned in the table.

In Alfabet query language the following combinations of conditions and data types can be used for filter fields:

Property Data Type	Condition	Resulting Filter Field Type	Required Configuration of Filter Field Attributes	Remark
`String`	=,<>	Edit field without wildcards	If the ***AutoWildCard*** attribute of the XML object ***SearchManager*** is set to `true`, the attribute **Allow Wildcards** of the filter field must be set to `False`. For more information, see Automatically Adding Wildcards to Search Strings	<> means different from, = means identical to. In both cases, the complete string in the search field is compared with the complete string of the property value. Wildcards are not allowed.
`String`	<,>,<=,>=	Edit field without wildcards	If the ***AutoWildCard*** attribute of the XML object ***SearchManager*** is set to `true`, the attribute **Allow Wildcards** of the filter field must be set to `False`. For more information, see Automatically Adding Wildcards to Search Strings	The less than and greater than operators are interpreted alphanumerically when applied on a string property. For example, if the greater than operator is used for a filter for application name and you enter "`alfa`" into the filter field, all applications with names in the alphabetical order before or identical to "`alfa`" are not displayed. The application with the name "`account`" will not be displayed, while the application with the name "`Alfabet`" For example, is displayed, because it is coming after `alfa` in the alphabetical order.
`String`	`LIKE, NOT LIKE, CONTAINS, CONTAINSOR`	Edit field with wildcards		In the resulting edit field, part of a string value can be entered with an asterisk (*) as wildcard to search for property values including a defined string.
`StringArray`	`LIKE, NOT LIKE, CONTAINS, CONTAINSOR`	Edit field with wildcards		In the resulting edit field, part of a string value can be entered with an asterisk (*) as wildcard to search for property values including a defined string.

Property Data Type	Condition	Resulting Filter Field Type	Required Configuration of Filter Field Attributes	Remark
`String` based on an enumeration or a definition in an XML object (release status, object state, lifecycle status, stereotype definitions with a hierarchical order)	`LIKE, NOT LIKE`	Edit field with wildcards		
`String` based on an enumeration or a definition in an XML object (release status, object state, lifecycle status, stereotype definitions with a hierarchical order)	`CONTAINS, =`	Combo box	If the release status definition is different for different stereotypes of a class, a filter field can only display the release status definitions of one of the stereotypes and the stereotype name must be defined with the attribute **Custom Property** of the filter field.	Stereotype definitions that do not have a hierarchical order defined will generate a normal string filter field, that means an edit field with (`CONTAINS`) or without (=) allowing wildcard settings.
`String` based on an enumeration or a definition in an XML object (release status, object state, lifecycle status, stereotype definitions with a hierarchical order)	`CONTAINSOR`	Checked combo box	If the release status definition is different for different stereotypes of a class, a filter field can only display the release status definitions of one of the stereotypes and the stereotype name must be defined with the attribute **Custom Attribute** of the filter field.	Stereotype definitions that do not have a hierarchical order defined will generate a normal string filter field, that means an edit field allowing wildcard settings.
`String` based on an enumeration or a definition in an XML object (release status, object state, lifecycle status, stereotype definitions with a hierarchical order)	<,>,<=,>=	Edit field without wildcards	If the ***AutoWildCard*** attribute of the XML object ***SearchManager*** is set to `true`, the attribute **Allow Wildcards** of the filter field must be set to `False`. For more information, see Automatically Adding Wildcards to Search Strings	The less than and greater than operators are interpreted alphanumerically when applied on a string property. For example, if the greater than operator is used for a filter for application name and you enter "`alfa`" into the filter field, all applications with names in the alphabetical order before or identical to "`alfa`" are not displayed. The

Property Data Type	Condition	Resulting Filter Field Type	Required Configuration of Filter Field Attributes	Remark
				application with the name "account" will not be displayed, while the application with the name "Alfabet" For example, is displayed, because it is coming after alfa in the alphabetical order.
Text	LIKE, NOT LIKE	Edit field with wildcards		
Date	=,<>,<=, >=,<,>	Date picker		
DateTime	=,<>,<=, >=,<,>	Edit field without wildcards		If the operator = is used, the complete date and time definition in the correct culture format must be specified in the filter field to find a match.
Integer / Real	=,<>,<=, >=,<,>	Edit without wildcards		
Boolean	=	Combo box		
Reference / ReferenceArray	CONTAINS	Edit search (selector)		
Reference	CONTAINSOR, IN, NOT IN	Checked combo box	The content of the checked combo box must be defined with an Alfabet query or native SQL query in the attribute **Alfabet Query/Query As Text** of the filter field.	

Property Data Type	Condition	Resulting Filter Field Type	Required Configuration of Filter Field Attributes	Remark
`ReferenceArray`	`CONTAINSOR`	Checked combo box	The content of the combo box must be defined with an Alfabet query or native SQL query in the attribute **Alfabet Query/Query As Text** of the filter field.	

Note the following when defining filter field name parameters in `Where` clauses:

- A filter field value can be used in multiple `Where` clauses by defining the same Alfabet parameter name.
- When a filter field is not filled, the query processing mechanisms of the Alfabet components delete the condition with the filter parameter from the `Where` statement before executing the query. For example, the following `Where` statement of an Alfabet query

```
WHERE
AND (
    APPLICATION.STATUS = 'Plan'
    APPLICATION.NAME LIKE @PARAM
)
```

is shortened to the following statement before executing the query if the filter field with the name `@PARAM` is not filled:

```
WHERE APPLICATION.STATUS = 'Plan'
```

- In standard Alfabet views and configured reports based on Alfabet queries entries in filter fields are interpreted case insensitive.
- The search behaviour in edit fields is controlled by the setting of the ***AutoWildCard*** attribute of the XML object ***SearchManager***. When ***AutowildCard*** is set to `true`, a wildcard is added automatically to search strings entered by the user. Therefore, the `=` condition cannot be used for the definition of edit fields because the condition does not allow wildcards in the argument. It is also recommended to inform users with a static text in the filter section whether wildcards are automatically added to strings.

For more information about the ***AutoWildCard*** setting in the XML object ***SearchManager***, see Configuring the Wildcard for Standard and Custom Search Functionalities.

Creating SQL Report Views to Configure Filters

Once the filter fields are defined in the query, an SQL report view must be created to generate filter fields automatically. In the SQL report view, you can change the display of the filters by, For example, specifying new captions for the filter fields or rearranging the fields within the filter area. For checked combo-boxes, combo-boxes, and edit fields not allowing wildcard settings, the filters may require additional specification of the content.

This is done in an SQL report view. The SQL report view must be created for a configured report after all filters have been defined in the Alfabet query.

Make sure that all required filters for the configured report are specified before you generate an SQL report view. When adding a new filter to a configured report with an already tested and configured SQL report view, you have to delete and re-create the SQL report view. Then you will lose the configurations previously defined for the filter fields.

To create an SQL report view:

1) In the explorer, right-click the configured report and select **Create SQL Report View**. The view editor opens and displays the SQL report view with the automatically generated filter fields. In the explorer, the SQL report view is displayed as a subordinate object of the configured report.
2) For combo boxes, checked combo boxes and edit fields not allowing wildcard settings, click the filter field and edit the following attributes in the attribute window:
   - **For edit fields not allowing wildcard settings**: Set the attribute **Allow Wildcards** to `False`.
   - **For (checked) combo boxes for the selection of a lifecycle status**: The lifecycle status definition is different for each object class. You must specify the name of the object class in order to display lifecycle definitions for an object class. For more information about lifecycle status definitions, see the section Configuring Lifecycle Definitions for Object Classes. in the chapter Configuring the Class Model.
   - **For (checked) combo boxes for the selection of a release status for a class with stereotype definitions**: If the release status of a class is different for different stereotypes of an object class, the name of the stereotype must be defined in the attribute **Custom Attribute** of the filter field.
   - **For (checked) combo boxes for the selection of a reference target**: When the filter field allows to select objects referenced by a property of the type `Reference` or `ReferenceArray`, define an Alfabet query or a native SQL query in the attribute **Alfabet Query/Query As Text** to refine the filter field. The output of the query specified here defines the values that will be displayed in the (checked) combo-box.

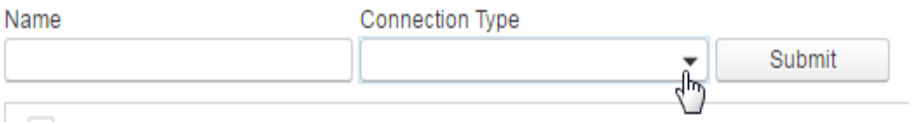
     Note the following about the display of the properties in the (checked) combo-box:
     - When defining filter field content with a native SQL query, the first `SELECT` statement must be the `REFSTR` of the selected object and is not displayed in the combo box.
     - When you specify more than one show property in an Alfabet query or more than one result column in the `SELECT` clause of a native SQL query, the values of the properties are displayed in the combo box in one row per object separated with a whitespace.

- If you want two property values to be displayed with a fixed string between them, For example, the properties for Name and Version with a "v." in between, you can use the `JoinColumns` instruction to define the string. For information about the `JoinColumns` instruction, see Aggregating Multiple Query Results in One Column of a Configured Report in the chapter Defining Queries.
- You can define `SORT` properties to define the order of values in the combo box or checked combo box.

A configured report shall allow users to select applications with a specific name and show all incoming and outgoing information flows of a specific connection type. The Alfabet query for the configured report joins the object class `Information-Flow` to the base class `Application` and defines the search conditions as filters:

```
ALFABET_QUERY_500

FIND Application

InnerJoin

   InformationFlow ON (Or InformationFlow.From =
   Application.REFSTR InformationFlow.To =
   Application.REFSTR)

WHERE

   (AND

      Application.Name LIKE:AppName

      InformationFlow.ConnectionType
      CONTAINSOR:ConnectionType)

QUERY_XML

<QueryDef>

   <ShowProperty Type="Property" ClassName="Application"
   Name="Name" />

   <ShowProperty Type="Property"
   ClassName="InformationFlow" Name="ConnectionType" />

</QueryDef>
```

The test of the configured report shows a filter with a text input field for the application name and a combo box to select one or multiple connection types. The checked combo box for the connection types is empty and does not open when clicking the arrow at the right of the field:



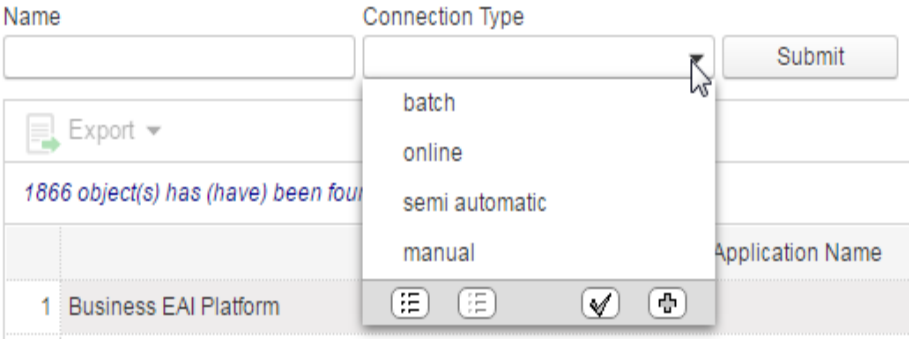
To display the available connection types in the checked combo box, you must specify an Alfabet query for the **Connection Type** checked combo box. The Alfabet query must find connection types and defines the attribute **Name** of the connection type as a Show property:

```
ALFABET_QUERY_500

FIND ConnectionType
```



```
QUERY_XML

<QueryDef>

    <ShowProperty Type="Property"
    ClassName="ConnectionType" Name="Name" />

</QueryDef>
```

The checked combo box will then allow the user to select between the available connection types:



3) Optionally, you can further configure the display and behaviour of the filter. The options for the configuration of filter panels and filter fields are described in the section Defining Filters for Configured Reports and Selectors in the chapter Defining Queries. the following configurations can be performed:

- Grouping Filters on the Filter Panel
- Adding Static Information to the Filter Panel
- Configuring the Complete Filter Area to be Collapsible
- Defining Mandatory Filter Fields
- Automatically Adding Wildcards to Search Strings
- Configuring Cascading Filters
- Excluding Filters From Storage in Bookmarks

4) In the toolbar, click the **Save** button to save your changes.

Defining a Data Capture Environment for a Configured Report of the Type Query

The buttons that are available in standard data capture functionalities in Alfabet can also be added to a configured report that displays data about the respective class in a table. Standard data capture functionalities are available for the object classes application, component, device, peripheral, ICT object and service product.

Note the following about the configuration:

- The attribute **Template** of the configured report must be set prior to creating the custom report view. If a configured report already has a custom report view defined, setting of the Template attribute will not add the buttons unless the custom report view is deleted and re-created.
- The objects found be the query the configured report is based on must be of the object class or object class stereotype for that the data capture environment is defined. Please note that for the `CaptureProjects` template, the report must be limited to a defined object class stereotype to deliver valid results.
- If the configured data capture report shall be limited to editing of objects of a specific stereotype only, the following is required:
  - All objects that are displayed in the report can be edited regardless of the stereotype. If you want to limit editing to data with a specific steretype, you must limit display of objects in the report table to objects of that stereotype via the query definition of the configured report.
  - To limit creation of new objects to object of a specific stereotype, enter the name of the stereotype into the attribute **Parameter** of the configured report. This setting must be performed prior to creating the report view.

To add the buttons available in a standard data capture environment to the configured report, the following configuration is required in addition to the configuration described above for general configuration of the configured report:

1) In the attributes window of the configured report, set the attribute **Template** to one of the following values:
   - `CaptureApplications`
   - `CaptureComponents`
   - `CaptureContracts`
   - `CaptureDemands`
   - `CaptureDevices`
   - `CaptureIssues`
   - `CapturePeripherals`
   - `CaptureProjects`
   - `CaptureICTObjects`
   - `CaptureServiceProducts`
   - `CaptureVendorProducts`
   - `CaptureVendors`
2) If you have selected the template `CaptureIssues`,set the **Apply to Class** attribute of the configured report to the object class that the issues shall be created for.
3) If creating of objects shall be limited to a defined stereotype, enter the name of the stereotype into the attribute **Parameter** of the report. If you have selected the template `CaptureIssues`, you can add a comma separated list of stereotype names in the **Parameter** attribute to restrict the creation of issues to issues of all specified stereotypes. The sub-menu of the **New** button in the toolbar of the report will be adapted to inform the user which issue stereotypes can be selected.

4) Right-click the configured report and select **Create Report View**.

5) Optionally, you can define which buttons will be displayed in the toolbar:

- You can hide standard buttons from the toolbar. For information about the required settings, see Hiding Functionalities in a Page View or Configured Report.

- You can substitute the standard mass update functionalities in the data capture functionality with the editor available from a configured report for multi-editing of object class properties. The editor will be available via a button **Bulk Edit** that will be automatically added to the toolbar for all data capture environment report that have a configured report for multi-editing of object class properties assigned. The button will have submenu options for opening the editor or, if available via the configured report for multi-editing of object class properties, the options for setting the same value for all selected indicators, properties or roles. The editor will open for the objects selected in the table and the rules for visibility and editability of properties defined in the configured report for multi-editing of object class properties will be applied. Do the following to enable mass update in an editor from a configured report for multi-editing of object class properties.

  - Create a configured report for multi-editing of object class properties based on the template `EditableClassViewReport`. Make sure that the report is targeting the same object class as your data capture report and that the edit rights are in accordance with the edit rights that you would like users to have for objects when working in your configured data capture environment. For information about the definition of a report for multi-editing of object class properties based on the template `EditableClassViewReport`, see Creating Configured Reports With Editing Capabilities.

  - Right-click the data capture report and select **Start Alfabet Report Assistant**. In the assistant, select the configured report for multi-editing of object class properties in the dropdown list of the field **Editable View Report**. Click **OK**.

Defining a User Management Functionality via a Configured Report of the Type Query

The functionalities for user management that are available in the standard Alfabet views **User Administration** (`ADMIN_UsersOverview`) and **Contact Administration** (`ADMIN_ContactsOverview`) can also be added to a configured report that displays user data in a table. The administrative functionalities available in the configured report are limited to the functionalities that apply to selected users in the table. All functionalities that apply changes to all users in the database are not included in the configured report.

Via the configured reports, users can be provided with administrative access to the user data for a selected sub-set of users in the Alfabet database, For example, to all users that are in a defined user group. In addition, the view can be adapted to the specific tasks of the user that will administrate users for the configured report by adding filter fields and configuring the tabular dataset of the report to include information about a user that is not available in the standard **User Administration** view.

Note the following about the configuration:

- The attribute **Template** of the configured report must be set prior to creating the SQL report view. If a configured report already has an SQL report view defined, setting of the **Template** attribute will not add the buttons unless the custom report view is deleted and re-created.

- The objects found be the query the configured report is based on must be of the object class `PERSON` for that the data capture environment is defined.

- You can limit the availability of the configured report to access from user profiles with administrative rights. If you set the attribute **Restrict to Administrative User Profiles** of the SQL report view is set to `True`, the configured report can only be opened if the user is logged in with a user profiles for that the attribute **Is Administrative User Profile** is set to `True`.

To add the the functionality available in the standard **User Administration** view for administration of selected users to the configured report, the following configuration is required in addition to the configuration described above for general configuration of the configured report:

1) In the attributes window of the configured report, set the attribute **Template** to `UserManagementReport` or `ContactManagementReport`.
2) Right-click the configured report and select **Create SQL Report View**.
3) Optionally, click the SQL report view and set the attribute **Restrict to Administrative User Profiles** of the SQL report view to `True`.

Allowing the User to Change the Number and Order of Columns in the Table

A tabular configured report can be configured to display a **Configure** button in the toolbar of the configured report. The user can do the following in the editor opened via the button:

- Change the order of columns in the table.
- Hide columns from the table.

The following configuration is required to display the **Configure** button:

1) If a report view has not yet been defined for the configured report, right-click the configured report and select **Create SQL Report View** . In the explorer, the SQL Report View is displayed as a subordinate object of the configured report.
2) Expand the SQL Report View node under the explorer node of the configured report and click the Presentation Object node under the SQL Report View node. The attribute window displays the attributes of the presentation object.
3) Set the attribute **Configurable** of the Presentation Object to `True`.

Configuring the Analysis of the Tabular Report in a Pivot Grid

A tabular configured report can be analysed in a Pivot grid and the graphical representation thereof. By default, new configured tabular reports have a button **Pivot Grid** that opens a new view with the Pivot functionality. This default behaviour can be changed to other analysis modes or deactivated.

The third party component DevExpress® is used to render results in a Pivot table.

DevExpress® must be activated in the web.config file of the Alfabet Web Application to be used in configured reports. For more information about the activation, see *Prerequisits for Using DevExpress®* in the *Technical Requirements*.

If you would like to offer Pivot grid analysis for a tabular report, you should note the following when creating the query for the report:

- The analysis is performed on basis of the result dataset. Columns are identified by their caption. Therefore, the caption of the columns must be unique. In addition, the user cannot see the data table when working with the Pivot Table Analysis. Captions should be designed to provide information about the kind of data returned in the respective column to enable the user to decide where to place the data in the Pivot grid.
- At least one column in the result dataset must return a numerical value. Only numerical values can be added to the cells in the matrix of the Pivot Grid.
- If navigation from the table to the object profile or object cockpit of a base object for each result dataset is provided, it is also identically available in the Pivot Grid.
- If you would like the user to analyze multiple numerical values that are having a different aspect, For example, costs that are having a different cost type, the aspects can only be summed up if you list the aspect, For example, the cost type, in one column and the value in a second column. If you add one column for each aspect, that each contain the respective values for this aspect, the data can only be analysed next to each other, but not merged.

For example, if a configured report shall return for applications in application groups the current costs over the years separately for each cost type. The way the data can be analyzed in a Pivot Grid depends on the structure of the tabular output of the report:

- The tabular output of the configured report may display one line returning the cost type name and one row returning the budget value for all cost types:

	Application Group Name	Application Name	Cost Type Name	Budget Value Value	Budget Value Year
1	2. Trade Entry	EMERGING MARKET SYSTEMS v. 3.1.2	Subscription	25.00	2015
2	2. Trade Entry	EMERGING MARKET SYSTEMS v. 3.1.2	Expense	115.00	2015
3	2. Trade Entry	EMERGING MARKET SYSTEMS v. 3.1.2	Consulting	31.00	2015
4	2. Trade Entry	EMERGING MARKET SYSTEMS v. 3.1.2	Other	14.00	2015
5	2. Trade Entry	EMERGING MARKET SYSTEMS v. 3.1.2	Subscription	25.00	2016
6	2. Trade Entry	EMERGING MARKET SYSTEMS v. 3.1.2	Expense	129.00	2016
7	2. Trade Entry	EMERGING MARKET SYSTEMS v. 3.1.2	Consulting	37.00	2016
8	2. Trade Entry	EMERGING MARKET SYSTEMS v. 3.1.2	Other	15.00	2016
9	2. Trade Entry	EMERGING MARKET SYSTEMS v. 3.1.2	Subscription	25.00	2017
10	2. Trade Entry	EMERGING MARKET SYSTEMS v. 3.1.2	Expense	132.00	2017
11	2. Trade Entry	EMERGING MARKET SYSTEMS v. 3.1.2	Consulting	37.00	2017

If the report is analyzed in a Pivot Grid for application group costs per year. The cost values are all from the same column and Therefore, the Pivot Grid can be configured to display costs independent from the cost type. The cells in the matrix then display a sum of the costs for all cost types belonging to the same year and application group:

	Budget Value Year				
Application Group Name	2015	2016	2017	2018	Grand Total
1. Market Data	5857	6611	9485	9697	31650
2. Trade Entry	13615	14990	14803	15346	58754
3. Front Office	12174	14412	15439	15396	57421
4. Back Office	18231	20295	20683	377891	437100
5. Customer Relations	1340	2199	7013	9599	20151
6. Risk Reporting	4241	5566	9366	10139	29312
7. Finance Reporting	6083	6957	7612	1705340	1725992
8. Accounting	3408	4284	4484	1511832	1524008
9. Payments	3561	4168	4444	4861	17034
Grand Total	68510	79482	93329	3660101	3901422

Alternatively, the Pivot Grid can display costs per cost type by adding the cost types as column header:

	Budget Value Year						
	⊟ 2015					⊟ 2016	
Application Group Name	Consulting	Expense	Other	Subscription	2015 Total	Consulting	Ex
1. Market Data	920	3178	544	1215	5857	1048	
2. Trade Entry	2405	8507	1013	1690	13615	2671	
3. Front Office	2150	7265	974	1785	12174	2518	
4. Back Office	3223	10974	1414	2620	18231	3555	
5. Customer Relations	95	342	198	705	1340	270	
6. Risk Reporting	604	2125	417	1095	4241	876	
7. Finance Reporting	978	3300	545	1260	6083	1099	
8. Accounting	593	2144	251	420	3408	675	
9. Payments	640	2087	289	545	3561	765	
Grand Total	11608	39922	5645	11335	68510	13477	

A filter field is then displayed on top of the column headers that allows to remove costs for singel cost types from the dataset.

For each year, a total sum is calculated, displaying the total costs independent from the cost type. On the right of the grid, the **Grand Total** column calculates the total costs per application group and year:

⊟ 2018				2018 Total	Grand Total
Consulting	Expense	Other	Subscription		
1731	5946	735	1285	9697	31650
2692	10021	1093	1540	15346	58754
2670	9805	1111	1810	15396	57421
3109	370402	1545	2835	377891	437100
1689	5689	760	1461	9599	20151
1892	5986	821	1440	10139	29312
957	342963	645	1360775	1705340	1725992
983	209988	441	1300420	1511832	1524008
871	3097	348	545	4861	17034

- The tabular output may alternatively display one line per cost type with the cost type name given as column header and the cost type value listed in the respective row:

	Application Group	Application	Subscription	Expense	Consulting	Other	Year
1	OptiRetail CRM As-Is Situation	Business EAI Platform 2.2	110.00	4.00	1.00	28.00	2015
2	Corporate CRM AS-IS Situation	Business EAI Platform 2.2	110.00	4.00	1.00	28.00	2015
3	Corporate CRM TO-BE Situation	Business EAI Platform 2.2	110.00	4.00	1.00	28.00	2015
4	Applications Using UDB	Groupware Services 2.2	60.00	29.00	7.00	16.00	2015
5	Corporate CRM AS-IS Situation	Groupware Services 2.2	60.00	29.00	7.00	16.00	2015
6	Corporate CRM TO-BE Situation	Groupware Services 2.2	60.00	29.00	7.00	16.00	2015
7	5. Customer Relations	Mafo-Portal 2.6	500.00	63.00	17.00	129.00	2015
8	CRM Analysis	Mafo-Portal 2.6	500.00	63.00	17.00	129.00	2015
9	Applications Using UDB	Mafo-Portal 2.6	500.00	63.00	17.00	129.00	2015
10	SAP Landscape	Mafo-Portal 2.6	500.00	63.00	17.00	129.00	2015
11	Customer Management	Mafo-Portal 2.6	500.00	63.00	17.00	129.00	2015
12	OptiRetail CRM As-Is Situation	Mafo-Portal 2.6	500.00	63.00	17.00	129.00	2015

To analyze the data in a Pivot Grid including all cost types, the columns containing cost type values must all be added to the matrix cells. The cost types are then added as sub-headers to the columns, but there is neither a filter nor a total per year calculated, because the columns are regarded ans independent from each other:

Application Group	2015				2016	
	Consulting	Subscription	Expense	Other	Consulting	Subscription
1. Market Data	920	1215	3178	544	1048	1280
2. Trade Entry	2405	1690	8507	1013	2671	1690
3. Front Office	2150	1785	7265	974	2518	1785
4. Back Office	3223	2620	10974	1414	3555	2620
5. Customer Relations	95	705	342	198	270	705
6. Risk Reporting	604	1095	2125	417	876	1095
7. Finance Reporting	978	1260	3300	545	1099	1345
8. Accounting	593	420	2144	251	675	770
9. Payments	640	545	2087	289	765	545
Grand Total	11608	11335	39922	5645	13477	11835

The **Grand Total** column is also splitted into the sub-columns:

2018				Grand Total			
ting	Expense	Subscription	Other	Consulting	Expense	Subscription	Other
1731	5946	1260	729	5305	18357	5285	2672
2665	10021	1540	1032	10499	37360	6610	4197
2670	9805	1785	1105	10144	35845	7140	4261
3109	370402	2835	1545	13553	406800	10695	6052
1689	5689	1390	743	3140	10792	4295	1836
1804	5986	1380	785	4835	16847	5010	2436
957	342960	1360660	605	4246	354481	1364675	2432
983	209988	1300420	441	3028	217175	1302380	1425
871	3097	545	348	3080	10487	2180	1287
16479	963894	2671815	7333	57830	1108144	2708270	26598

The report can be configured to display the table with a button in the toolbar that changes the view to the Pivot table analysis. The configured report can alternatively be configured to open directly in the Pivot table analysis view.

Do the following to enable Pivot table analysis:

1) In the explorer of the **Reports** tab, click the configured report.
2) In the attribute window, select one of the following options in the **Analysis Mode** attribute:

- `Optional`: A button **Pivot Grid**  is displayed in the tabular report to open a Pivot analysis. This is the default setting for new reports.
- `Always`: The report directly opens in the Pivot table analysis view.
- `Never`: A Pivot table analysis is not provided.

3) In the toolbar, click the **Save**  button to save your changes.

Please note that changing the **Analysis Mode** for existing configured reports might require a re-creation of the report view. It is recommended that the configured report be tested and the report view recreated only if changes were not applied.

If you would like the user to be informed about the functionality of the Pivot grid analysis view via the context sensitive help, you can do the following:

- An Alfabet specific standard help file is available for Pivot grid analysis. The standard help is only displayed if it is activated for the configured report:
  - If the **Analysis Mode** is changed to `Optional` or `Always`, the name of the standard help file is automatically written into the attribute **Help Index** of the report. The file is then offered to the user as context sensitive help for the configured report. Once the **Help Index** is set, it is not automatically removed when the Analysis Mode is changed again, but it can be removed manually.
  - The default setting for the report is `Optional`. Nevertheless the **Help Index** attribute is empty by default. To activate the standard help while keeping the default setting, you must either set the **Analysis Mode** to `Never` and then back to `Optional`, or manually enter `PivotTableAnalysis.html` in the **Help Index** attribute.
- A customer specific help can be opened. The method for defining customer specific help is described in the section Specifying Custom Help for a Configured Report.

Configuring a Data Table Report with Restructuring Options for the End User

End user design option can be added to a tabular configured report based on an Alfabet query. The configured report will then be displayed in the design defined via the Alfabet query it is based on, but it can be reconfigured by the user. The end user can select properties of any of the object classes that are added to the Alfabet query either as `FIND` class or via a `JOIN` as additional show properties and deselect show properties from the default layout. Furthermore, the report design can be changed from tabular to matrix and pictures and colors can be added to the design.

Any tabular configured report can be defined as data table report with end user design capabilities by adding the following configuration to the report:

1) In the explorer of the **Reports** tab, click the configured report.
2) In the attribute window, select `DataTableReport` from the dropdown list of the **Template** attribute.

   The attribute **Standard Context-Sensitive Help Index** is filled automatically with the correct link to the end user help file in the standard Alfabet online help on selection of the `DataTableReport` template. The standard help can be deactivated by deleting the value from the **Standard Context-Sensitive Help Index** attribute. A customer specific help can be opened. The method for defining customer specific help is described in the section Specifying Custom Help for a Configured Report.

3) In the toolbar, click the **Save**  button to save your changes.

Note the following about the definition of the Alfabet query for the report:

- The report must have a default layout defined via the **Show Properties**. If no default layout is defined, the report will not be displayed to the user and the design options will also be missing.
- The user will be able to select all properties of the object classes included into the Alfabet query except for the properties of the type `Reference` or `Reference Array`. References to other object classes are only available if a `JOIN` to the respective object class is defined.
- If you would like to restrict the properties available via the report to a subset of the properties of the object classes involved, you can define a database view containing the relevant data and create the report about the database view instead of the object class tables.
- If you would like the user to add role assignments or indicators to the configured report, you must either include all indicators and roles as show properties of the type `Indicator` or `Role` to the Alfabet query, or add the involved object classes via `JOINS`. The indicator and role definitions that are available in the property overview of the Alfabet Query Builder for an object class will not be part of the properties that are only selectable by the user, but not available via the standard layout.
- **Sort Properties** definitions are ignored.
- In the **Data Source** editor that allows the user to select the data for display in the report, the object class properties are listed as ObjectClassCaption.PropertyCaption. If an alias is defined for the object class or object class property, the alias name is displayed instead of the object class caption or property caption. The definition of a `SetColumnShowName` instruction supersedes an alias definition for the object class property. The end user can change the caption of the columns that are defined via the Alfabet query of the report directly in the report.
- Navigation from the report to the object view or object profile of objects that are accessible via the configured report is not available by default. You can define columns in the tabular report to provide navigation. Navigation can be defined for any data and any number of columns. For example, for a configured report that displays information flows and the source and target applications thereof, the report can be configured to provide navigation to the information flows if the user double clicks on the information flow name, to the source applications if the user double clicks the name of a source application and to the target applications if the user double clicks the name of the target application.

Navigation is provided via the Alfabet query language instruction `CreateRefImage`, that merges a column returning the `REFSTR` of an object with any other column in the report. As a result, a single, navigable column is displayed at the position of the column returning the `REFSTR`, but showing the information from the other merged column. For information about the definition of the instruction, see Configuring Cells in a Data Table Report to Provide Navigation to an Object Profile in the section Using Instructions to Format the Results of an Alfabet or Native SQL Query.

- In general, Alfabet query language instructions can be used in the Alfabet query, with the limitation that the result must not involve dependencies between cells in different columns of the resulting dataset. For example,:
  - You can color cells using a `ColorAssignment` instruction, because the coloring depends on the results in the current column only, but a `RowColorAssignment` instruction will be ignored, because cell coloring of cells of one column would depend on the result of cells in another column.
  - You can use a `JoinColumns` instruction. The result is a single, independent column in that will be presented as selectable or deselectable data to the end user and the results in the joined column will be displayed wherever the end user selects it to be displayed in a tabular or matrix

structure. In contrast, you cannot use a `GroupBy_Ex` instruction, because the instruction creates a structure in the result dataset that involves more than one column to be created.

- You cannot use a `LinkAssignment` instruction because the link displayed in one column depends on an object specification in another column of the dataset.

- Coloring and picture assignment defined in the Alfabet query for data in a column will only be available in a tabular layout. If the user changes the design to a matrix layout, the default picture and color assignment will be removed. The end user can define own color and picture rules that supersede the default color and picture rules. Color rules and picture rules defined in the Alfabet query are not added to the legend of the data table report. The legend is limited to user defined coloring and picture assignment.

Creating a Tabular Configured Report of the Type NativeSQL

To generate a configured report of the type `NativeSQL`:

- In the **Reports** tab of Alfabet Expand, create a new configured report and define the general attributes for the configured report and the attributes specific for native SQL query-based configured reports. For more information about creating a configured report and defining the configured report's attributes, see Creating a New Native SQL Query-Based Configured Report.

- Configure a native SQL query for the configured report. For more information, see Configuring a Native SQL Query for a Configured Report. If the report result shall be an expandable table, the native SQL query must be configured to contain instructions. This is described separately in the section Grouping Query Results in Expandable Reports in the chapter Defining Queries.

- If the configured report has filters defined, create an SQL report view, check the functionality of the filters, and optionally change the captions and look of the filter fields. For more information, see Defining Filters for Configured Reports and Selectors in the chapter Defining Queries.

- Define access rights for the configured report in the **Reports Administration** functionality of Alfabet. By default, access is possible for all users. For more information, see the chapter *Defining and Managing User Access to Configured Reports* in the reference manual *User and Solution Administration*.

- Set the **Report State** attribute of the configured report to `Active`. For information on how to change theattribute of the configured report, see.

The following information is available:

- Creating a New Native SQL Query-Based Configured Report
- Configuring a Native SQL Query for a Configured Report
- Defining a Data Capture Environment for a Configured Report of the type NativeSQL
- Defining a User Management Functionality via a Configured Report of the Type NativeSQL
- Allowing the User to Change the Number and Order of Columns in the Table

- Configuring the Analysis of the Tabular Report in a Pivot Grid
- Configuring a Data Table Report with Restructuring Options for the End User
- Defining Audit Management Related Configured Reports

Creating a New Native SQL Query-Based Configured Report

To create a new native SQL query-based configured report in Alfabet Expand:

1) In the **Reports** tab, right-click the **Reports** folder or a sub-folder thereof and select **Create New Report**. You will see the new configured report listed below the selected folder.

As an alternative to defining all attributes for a new configured report, you can copy and paste an existing configured report and make modifications, as needed. To do so, right-click the existing configured report that you want to copy and select **Copy**. Right-click the new configured report and select **Paste**. Modify the necessary attributes described below.

2) In the attribute window, select `NativeSQL` in the **Type** field.
3) In the attribute window, define the following attributes for the configured report:
   - **Name**: Optionally, change the technical name of the configured report. The technical name of the configured report must be unique.

The technical name is used for identification of the configured report in the context of technical processes such as merging the report configuration with a report configuration saved as an XML file. Therefore, the following is required when defining the name of a configured report:

- The name must be unique. The name must be unique regardless of the report folder it is structured in.
- The name may not be changed once a configured report has been created.
- A validation mechanism checks for correct syntax when defining a technical name. Please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.
- If the technical name of the configured report is changed, the name will be correctly updated in other configuration objects referencing the changed report during design time.

Please note that the name of a changed configuration object will not be updated in guide pages or guide views. If you plan to change the name of a configuration object, the reference in the navigation page should be changed prior to changing the configuration object name. You can use the **Show Usage** functionality in the context menu of the configured report to evaluate whether the configured report has been added to a guide view or guide page.

   - **Caption**: Enter a caption for the configured report. The caption you define here will be displayed in the **Reports Administration** functionality in the **Administration** module and the

Configured Reports views of the Alfabet interface. If the configured report is assigned to an object view as a page view, the text will be displayed as the page view caption in the object view. The caption of the configured report may exceed the conventional 64 character limit.

- **Description**: Provide a short information about the configured report that is useful to Alfabet users. This comment will be displayed on the Alfabet user interface in the header of the configured report in a single line under the report caption and the **Description** field for the configured report in the table of all views listing configured reports, like the **Configured Reports** views of the **Search** functionalities. If the configured report is assigned to an object view as a page view, the text will be displayed under the page view caption as short description in the object view.

  In the configured report, the description will be truncated if it is longer than one line on the screen. Therefore, the description should be short to ensure complete display in the configured report header.

- **Usage Guideline**: Provide information that helps users to execute and interpret the configured report. This information will not be displayed directly on the user interface. The user can access the information using the following mechanisms:

  - If the user moves the cursor over the description provided for the configured report with the attribute **Description**, a tooltip opens that includes both the description and the usage guideline.
  - The **Options**  button will be displayed on the upper right with an option **Help On Filter Fields** to open a new window displaying the usage guideline in addition to any filter field hints, if configured.

    For views that have filters defined, the **Options**  button will only be displayed if the filter panel either allows batch clearing of filter fields or collapsing and expanding of the filter panel. For more information about the configuration of these functionalities, see Configuring the Complete Filter Area to be Collapsible and Allowing the User to Clear the Filter Area.

- **Technical Comment**: Provide information that is useful for solution designers working with Alfabet Expand in order to better understand the configured report from a technical perspective. The **Technical Comment** will not be displayed in the user interface. Nevertheless, you can configure the interface to display the information in, For example, the attribute section of the configured report's object profile in the **Reports Administration** functionality.
- **Report State**: The **Report State** attribute is view only and is set to `Plan` for new configured reports. The configured report can only be edited when the **Report State** attribute is set to `Plan`. After finishing all configuration steps described in the following, the **Report State** attribute must be set to `Active` as described in the section General Guidelines for Creating Configured Reports. The configured report is then visible to users in the Alfabet user interface.
- **Native SQL**: Click the **Browse**  button to open an editor and define the native SQL query for the configured report. Optionally, you can define instructions in Alfabet query language to alter the display of the query results.

For basic information about how to define native SQL queries, consult the relevant literature.

For special information about how to define native SQL for Alfabet, see the section Defining Native SQL Queries in the chapter Defining Queries.

For special hints for building a native SQL query for configured reports, see Configuring a Native SQL Query for a Configured Report in this section.

For a description of how to define report filters in a native SQL query, see Defining Filters for Configured Reports and Selectors in the chapter Defining Queries

For a description of how to define native SQL queries in combination with Alfabet instructions to built an expandable report table, see the section Grouping Query Results in Expandable Reports in the chapter Defining Queries.

- **Execute on Enter**: Set to `True` to execute the configured report with empty filter settings when the user opens the configured report on the Alfabet interface. Set to `False` to open the configured report without execution of the native SQL query with empty filter settings. By default, this attribute is set to `True`. It should only be set to `False` when a configured report would result in an exceedingly big dataset when executed with empty filter settings.

  When you set **Execute on Enter** to `False`, you must make sure that the **Submit** button is available in the SQL report view of the configured report. If the **Submit** button is deleted from the SQL report view, the configured report cannot be displayed. The setting and execution of the **Execute on Enter** attribute is independent from the definition of filters for a configured report. A configured report without filters must nevertheless have a **Submit** button when **Execute on Enter** is set to `False`.

- **Help Index**: Specify the location of the external Help file using the following syntax:(For example,:). The external help file will be displayed when the user clicks the context-sensitive Help link in the standardHelp menu.

  Server variables can be used in the Help links. Server variables allow you to define all or part of the URL definition in the server alias configuration of the Alfabet Server instead of directly defining it in the configuration in Alfabet Expand. Use of server variables is useful, For example, in order to change a Web server name defined in multiple URL definitions in Alfabet Expand. Instead of changing all URL definitions, only the variable value in the server alias configuration must be updated. The use of server variables is described in the section Using Server Variables in Web Link and Database Server-Related Specifications in the chapter Configuring Reports.

- **Apply to Class**: When the configured report is applied to an object class or object class stereotype, a base object can be selected for the configured report. A filter for selection of a base object will automatically be set on top of the configured report. When configuring queries for the configured report, you can use the Alfabet parameter `BASE` to refer to the object selected in the filter and define search conditions dependent on the base object. To apply the configured report to a base class, click the **Browse** button to open a dialog box and select the object class to which you want to apply the configured report and click **OK**.

  Note the following:

- If the configured report is assigned to the object view of an object of the class it is applied to, the configured report will automatically open with the current object selected as base object, even if **Apply to Class** is not set.
- If the configured report is applied to an object class stereotype, the filter field for selection of the base object will use the caption of the object class stereotype as caption of the filter field and the selector defined in the class settings for the object class stereotype for selection of the base object.
- Alternatively, the base object of a configured report can be set with the attribute **Base Query**. While the attribute **Apply to Class** enables the user to determine the current base object by setting a filter, the **Base Query** evaluates the base object by execution of a query when opening the configured report. A filter field is not available and the user cannot change the base object.
- The parameter **Apply to Class** also influences the availability of the configured report on the Alfabet interface. Select a class to make the configured report available in the **Configured Reports** page view on the object view of objects of the selected object class. If you do not set the attribute, the configured report is only available in the **Configured Reports** functionality. The configured report can be assigned to object views and wizards independent from the parameter.

- **Base Object Query**: When a base object query is defined for the configured report, a base object is evaluated for the configured report at runtime. When configuring the other queries for the configured report, you can use the Alfabet parameter `BASE` to refer to the object returned by the base object query and define search conditions dependent on the base object. To apply the configured report to a base class, click the **Browse** button to open a text editor, define either a native SQL query or an Alfabet query that returns a single object, and click **OK**.

  Note the following:

  - Alternatively, the base object of a configured report can be set with the attribute **Apply To Class**. While the setting of **Base Object Query** triggers the evaluation of the base object by execution of a query when opening the configured report, the attribute **Apply to Class** enables the user to determine the current base object by setting of an automatically generated filter.
  - The query defined for base object query is executed when the user opens the configured report to evaluate the base object at runtime. Parameters referring to the current working environment can be used in the query to find a base object dependent For example, on the user opening the configured report or the user profile used for login. A base object query may also find For example, the ICT object that a specific application is currently assigned to.
  - If **Base Object Query** is set and the configured report is assigned as a view to an object profile, the Alfabet parameter `BASE` referring to the `REFSTR` of the object the user is working with when opening the configured report can be used in the base object query.

- **Database Restricted User**: Optionally select a database user that shall be used by the Alfabet Web Application for connecting to the Alfabet database instead of the standard

database user with extended access permission when executing the configured report. The drop-down list is only filled if the use of different database users for configured report is specified in the XML object ***DatabaseUsers***. For more information about the concept of restricting access permissions for configured reports and the implementation of execution of configured reports with restricted database user rights, see the section Restriction of Database Access Permissions on Execution of Configured Reports.

- **Selector Behavior**: The attribute can be used to exclude configured reports that are defined for special purposes, like For example, triggering of data export via the RESTful API, from being selectable in the **Configured Reports** table. By default, the attribute is set to `Visible` and the configured report is visible in the selector for adding configured reports to the **Configured Reports** table if all other conditions for access to the configured report apply. If the attribute is set to `NotVisible`, the configured report is excluded from the selector for adding configured reports to the **Configured Reports** functionality. Please note however, that this setting is not excluding the configured report from any standard views or customer configured views and selectors, but exclusively from the standard selector for configured reports implemented in Alfabet.

The attribute cannot be edited in the attribute window directly. To change the setting, right-click the configured report and select **Set Report Selector Behavior to 'Not Visible'** or **Set Report Selector Behavior to 'Visible'** respectively.

- **Applicable for AlfaBot**: Specify whether the configured report is accessible via the AlfaBot. By default, the attribute will be set to `True` for new reports. Set the attribute to `False` if the configured report should not be opened outside of a specific context. For example, this may be relevant for reports that are specifically designed to be embedded in an object cockpit.

  This attribute is only relevant if the AlfaBot is implemented. For information about the implementation of the AlfaBot, see Implementing the AlfaBot for Navigation via a Full-Text Command.

- **Business Problem Statement**: Optionally enter a description that will help users to search for the configured report in the AlfaBot. The text is not displayed in the user interface, but it is of special relevance for the search mechanisms implemented for the AlfaBot. If the report caption entered by the user in the AlfaBot for a request to open a configured report does not match the caption of one of the available configured reports, the AlfaBot will split the caption entered by the user into keywords and will perform a keyword search on the **Caption**, **Description** and the **Business Problem Statement** attributes of the configured reports that are accessible via the AlfaBot. In addition to a keyword search, a synonym search is performed on the text provided in the **Business Problem Statement** attribute.

  This attribute is only relevant if the AlfaBot is implemented. For information about the implementation of the AlfaBot, see Implementing the AlfaBot for Navigation via a Full-Text Command.

- **Base Classes**: Define the object classes that must be available to run the configured report. If this attribute is set, the Alfabet Server checks whether the object classes specified as base classes are excluded from the view scheme used to access the configured report. If the user does not have access permissions to one of the base classes, the configured report will be disabled. To set the attribute, click the **Browse** button to select the object classes that are required to run the configured report and click **OK**.

- **Can Create Express View**: Select `True` if an express view may be created for the report. Select `False` if an express view may not be created for the report. Please note that for reports that have a custom report view, the setting must be consistent with the setting of the attribute **Can Create Express View** of the custom report view.

  An express view is a link to a specific Alfabet view displaying data that allows individuals inside or outside of the user community to view information in Alfabet. When the express view is created, an email notification is automatically mailed to a specified recipient. The recipient receives a URL that allows him/her to access the current page view in Alfabet. For more information, see the section *Sending and Receiving Express Views* in the reference manual *Getting Started with Alfabet*.

- **Chart Views Mode**: This attribute defines whether additional information about the objects found in the configured report will be available via standard page views or configured reports directly accessible via links from the configured report view. Select one of the following:

  - `Standard`: A button **Chart Views** for opening a standard lifecycle chart, business support map, information flow diagram or portfolio for an object selected in the result dataset of the configured report is available automatically in the toolbar of the configured report if the following applies:

    - At least one of these standard reports is available for the object class the report finds objects for.
    - No Alfabet query language instruction is defined in the Alfabet query of the configured report.
    - The **Enable Chart Views** attribute is set to **True**.

  - `Custom`: All configured reports defined in the **Supplementary Reports** attribute of the class settings of the object class found via this configured tabular report are available via tabs on top of the toolbar of the tabular dataset. If a filter is available, the tabs are located between the filter area and the toolbar of the dataset. For information about the configuration of configured reports suitable to be opened via tabs in tabular configured reports and the configuration of the class setting to provide the configured reports, see Integration of Configured Reports as Report Collection Into Tabular Configured Reports.

  - `None`: Neither standard page views nor configured reports are directly accessible via links from the configured report view.

- **Custom Chart View Base Class**: This attribute is only visible if the **Chart Views Mode** attribute is set to `Custom`. Select the object class or object class stereotype for which the tabs opening the configured reports for the report collection shall be displayed. The configured reports defined in the **Supplementary Reports** attribute of the class settings of the object class or object class stereotype will be available via tabs. Please note that the selected object class must be identical to the object class found in the tabular dataset of the configured report you are currently configuring. If the objects found in the tabular dataset do not match the setting, the report will not show the tabular dataset, but an error message about the incorrect setting in the configuration.

4) In the toolbar, click the **Save**  button to save your changes.

Configuring a Native SQL Query for a Configured Report

To define or edit the native SQL query of a configured report:

1) In the table, select the configured report whose native SQL query you want to define or edit.
2) Click the **Browse** button in the **Native SQL** attribute. A text editor opens.
3) In the **Native Query** tab of the text editor, define the native SQL query for the selection of objects for the configured report.

   The following is important for the generation of a native SQL query for a configured report:

   - If the configured report is applied to an object class with the attribute **Apply to Class**, you must configure the native SQL query to show results for the current object that is selected by the user when opening the configured report only. To refer to the current object, use the Alfabet parameter "`@BASE`". For example, a configured report with the base class `Application` with a `WHERE` clause "`WHERE Application.REFSTR=@BASE`" will show results for the application that the user is currently working with only. A selector is added automatically to each configured report that is applied to an object class by means of the attribute **Apply to Class** . The selector allows the user to display other base objects in the configured report. If you do not specify a `WHERE` clause with the Alfabet parameter `@BASE`, the selector will still be present, but it will not have any impact on the resulting configured report.
   - You can define filters for a configured report to allow the users to specify the range of results in the configured report. You can generate a filter from any `WHERE` clause of the configured report by defining the value of the `WHERE` condition as Alfabet parameter. For more information about configuring filters, see Defining Filters for Configured Reports and Selectors in the chapter Defining Queries.
   - You can combine the native SQL query with Alfabet instructions written in Alfabet query language. The instructions are defined separately in the tab **Instructions**. Instructions allow you For example, to define the output format for date and time information and to generate expandable report tables. For information about configuring Alfabet instructions, see Using Instructions to Format the Results of an Alfabet or Native SQL Query in the chapter Defining Queries .

     To check whether the syntax and the use of Alfabet parameters is correct in the native SQL query, click **Check**. The **Check** button checks the query for Alfabet conformance. Database vendor specific SQL constructs, requirements, or dialects are not taken into account for this check.

4) Click **OK** to save the native SQL query.
5) If the query contains `WHERE` clauses that trigger the generation of filter fields, right-click the configured report and select **Create SQL Report View**. The view editor opens and displays the SQL report view with the automatically generated filter fields. In the explorer, the SQL report view is displayed as a subordinate object of the configured report.

   In the SQL report view, a filter field is automatically generated for each Alfabet parameter specification in a `WHERE` clause of the query of the configured report. Each combination of property data type and condition in the `WHERE` clause lead to the generation of a specific filter field.

   For some data types, the generated fields need further configuration by the report designer to work properly. In addition, filter fields can be removed and substituted by other filter field types.

For example, a combo box can be substituted by a checked combo box to allow the user to select multiple values.

The following table lists the filter fields that result from automatic generation of filter fields and whether processing is required. Detailed information about the types of filter fields available for configured reports and the configurations that are required to generate and design filters is available in the section Defining Filters for Configured Reports and Selectors in the chapter Defining Queries.

Condition	Property Data Type	Resulting Filter Field Type	Required additional configuration of filter field
=,!=	`String`	Edit (one-line text field; no wildcards allowed in text entered into the field)	If automatic setting of wild cards is defined in the XML object `SearchManager`, the attribute **Allow Wildcards** of the edit fields must be set to `False`. For more information, see Configuring Edit Fields in the section Defining Filters for Configured Reports and Selectors.
LIKE, NOT LIKE, IN	`String`	Edit (one-line text field; wildcards allowed in text entered into the field)	
=, <>, LIKE, NOT LIKE, IN	`String (Property based on configuration in XML object or enumeration)`	Combo box	Some of the combo boxes require further setting of attributes to read the data from the correct configuration objects. For more information, see Configuring Radio Button Groups, Combo-Boxes, Checked Combo-Boxes, List Boxes or Checked List Boxes in the section Defining Filters for Configured Reports and Selectors.
LIKE, NOT LIKE	`StringArray`	Combo box	The content of the combo box must be defined in the attributes of the combo box. For more information, see Configuring Radio Button Groups, Combo-Boxes, Checked Combo-Boxes, List Boxes or Checked List Boxes in the section Defining Filters for Configured Reports and Selectors.
=,!=, <>,<=, >=,<,>	`Date`	Date picker	

Condition	Property Data Type	Resulting Filter Field Type	Required additional configuration of filter field
=,!=, <>,<=, >=,<,>	`Integer / Real`	Edit	
=, !=, LIKE, NOT LIKE, IN	`Boolean`	Combo box	Filter fields for boolean properties cannot be generated automatically for technical reasons. The automatically generated combo box will not work and must be substituted with a check box. For more information, see Configuring Check Boxes in the section Defining Filters for Configured Reports and Selectors.
=, !=, LIKE, NOT LIKE, IN	`Reference / ReferenceArray`	Combo box	The content of the combo box must be defined in the attributes of the combo box. For more information, see Configuring Radio Button Groups, Combo-Boxes, Checked Combo-Boxes, List Boxes or Checked List Boxes in the section Defining Filters for Configured Reports and Selectors.

6) In the explorer, right-click the configured report and select **Review Report**. The configured report opens in a web browser. If the results are not as expected, you can edit the native SQL query until the output of the configured report meets your expectations.

The **Review Report** functionality also allows the visibility of toolbar buttons of the configured report to be defined for different view schemes. For more information, see Refining Visibility Issues in the View Scheme.

7) In the toolbar, click the **Save** button to save your changes.

The amount of data that can be displayed in a configured report is restricted in order to avoid exceeding memory size usage. If a configured report results in an error message informing you that the data set is too large, refine the query to return only a subset of the results. If you do not want to exclude objects from the configured report but rather would like to limit the set of data that is displayed, you can define filters that allow the user to restrict the data display as needed (For example, by displaying only objects with a name starting with a specific character or objects assigned to a specific ascendant object).

Defining a Data Capture Environment for a Configured Report of the type NativeSQL

The buttons that are available in standard data capture functionalities in Alfabet can also be added to a configured report that displays data about the respective class in a table. Standard data capture functionalities are available for the object classes application, component, device, peripheral, ICT object and service product.

Note the following about the configuration:

- The attribute **Template** of the configured report must be set prior to creating the custom report view. If a configured report already has a custom report view defined, setting of the Template attribute will not add the buttons unless the custom report view is deleted and re-created.
- The objects found be the query the configured report is based on must be of the object class or object class stereotype for that the data capture environment is defined. Please note that for the `CaptureProjects` template, the report must be limited to a defined object class stereotype to deliver valid results.
- If the configured data capture report shall be limited to editing of objects of a specific stereotype only, the following is required:
  - All objects that are displayed in the report can be edited regardless of the stereotype. If you want to limit editing to data with a specific steretype, you must limit display of objects in the report table to objects of that stereotype via the query definition of the configured report.
  - To limit creation of new objects to object of a specific stereotype, enter the name of the stereotype into the attribute **Parameter** of the configured report. This setting must be performed prior to creating the report view.

To add the buttons available in a standard data capture environment to the configured report, the following configuration is required in addition to the configuration described above for general configuration of the configured report:

1) In the attributes window of the configured report, set the attribute **Template** to one of the following values:
   - `CaptureApplications`
   - `CaptureComponents`
   - `CaptureContracts`
   - `CaptureDemands`
   - `CaptureDevices`
   - `CaptureIssues`
   - `CapturePeripherals`
   - `CaptureProjects`
   - `CaptureICTObjects`
   - `CaptureServiceProducts`
   - `CaptureVendorProducts`
   - `CaptureVendors`
2) If you have selected the template `CaptureIssues`,set the **Apply to Class** attribute of the configured report to the object class that the issues shall be created for.
3) If creating of objects shall be limited to a defined stereotype, enter the name of the stereotype into the attribute **Parameter** of the report. If you have selected the template `CaptureIssues`, you can add a comma separated list of stereotype names in the **Parameter** attribute to restrict the

creation of issues to issues of all specified stereotypes. The sub-menu of the **New** button in the toolbar of the report will be adapted to inform the user which issue stereotypes can be selected.

4) Right-click the configured report and select **Create Report View**.

5) Optionally, you can define which buttons will be displayed in the toolbar:

- You can hide standard buttons from the toolbar. For information about the required settings, see Hiding Functionalities in a Page View or Configured Report.

- You can substitute the standard mass update functionalities in the data capture functionality with the editor available from a configured report for multi-editing of object class properties. The editor will be available via a button **Bulk Edit** that will be automatically added to the toolbar for all data capture environment report that have a configured report for multi-editing of object class properties assigned. The button will have submenu options for opening the editor or, if available via the configured report for multi-editing of object class properties, the options for setting the same value for all selected indicators, properties or roles. The editor will open for the objects selected in the table and the rules for visibility and editability of properties defined in the configured report for multi-editing of object class properties will be applied. Do the following to enable mass update in an editor from a configured report for multi-editing of object class properties.

  - Create a configured report for multi-editing of object class properties based on the template `EditableClassViewReport`. Make sure that the report is targeting the same object class as your data capture report and that the edit rights are in accordance with the edit rights that you would like users to have for objects when working in your configured data capture environment. For information about the definition of a report for multi-editing of object class properties based on the template `EditableClassViewReport`, see Creating Configured Reports With Editing Capabilities.

  - Right-click the data capture report and select **Start Alfabet Report Assistant**. In the assistant, select the configured report for multi-editing of object class properties in the dropdown list of the field **Editable View Report**. Click **OK**.

Defining a User Management Functionality via a Configured Report of the Type NativeSQL

The functionalities for user management that are available in the standard Alfabet views **User Administration** (`ADMIN_UsersOverview`) and **Contact Administration** (`ADMIN_ContactsOverview`) can also be added to a configured report that displays user data in a table. The administrative functionalities available in the configured report are limited to the functionalities that apply to selected users in the table. All functionalities that apply changes to all users in the database are not included in the configured report.

Via the configured reports, users can be provided with administrative access to the user data for a selected sub-set of users in the Alfabet database, For example, to all users that are in a defined user group. In addition, the view can be adapted to the specific tasks of the user that will administrate users for the configured report by adding filter fields and configuring the tabular dataset of the report to include information about a user that is not available in the standard **User Administration** view.

Note the following about the configuration:

- The attribute **Template** of the configured report must be set prior to creating the SQL report view. If a configured report already has an SQL report view defined, setting of the **Template** attribute will not add the buttons unless the custom report view is deleted and re-created.
- The objects found be the query the configured report is based on must be of the object class `PERSON` for that the data capture environment is defined.
- You can limit the availability of the configured report to access from user profiles with administrative rights. If you set the attribute **Restrict to Administrative User Profiles** of the SQL report view is set to `True`, the configured report can only be opened if the user is logged in with a user profiles for that the attribute **Is Administrative User Profile** is set to `True`.

To add the the functionality available in the standard **User Administration** view for administration of selected users to the configured report, the following configuration is required in addition to the configuration described above for general configuration of the configured report:

1) In the attributes window of the configured report, set the attribute **Template** to `UserManagementReport` or `ContactManagementReport`.
2) Right-click the configured report and select **Create SQL Report View**.
3) Optionally, click the SQL report view and set the attribute **Restrict to Administrative User Profiles** of the SQL report view to `True`.

Allowing the User to Change the Number and Order of Columns in the Table

A tabular configured report can be configured to display a **Configure** button in the toolbar of the configured report. The user can do the following in the editor opened via the button:

- Change the order of columns in the table.
- Hide columns from the table.

The following configuration is required to display the **Configure** button:

1) If a report view has not yet been defined for the configured report, right-click the configured report and select **Create SQL Report View** . In the explorer, the SQL Report View is displayed as a subordinate object of the configured report.
2) Expand the SQL Report View node under the explorer node of the configured report and click the Presentation Object node under the SQL Report View node. The attribute window displays the attributes of the presentation object.
3) Set the attribute **Configurable** of the Presentation Object to `True`.

Configuring the Analysis of the Tabular Report in a Pivot Grid

A tabular configured report can be analysed in a Pivot grid and the graphical representation thereof. By default, new configured tabular reports have a button **Pivot Grid** that opens a new view with the Pivot functionality. This default behaviour can be changed to other analysis modes or deactivated.

The third party component DevExpress® is used to render results in a Pivot table.

DevExpress® must be activated in the web.config file of the Alfabet Web Application to be used in configured reports. For more information about the activation, see *Prerequisits for Using DevExpress®* in the *Technical Requirements*.

If you would like to offer Pivot grid analysis for a tabular report, you should note the following when creating the query for the report:

- The analysis is performed on basis of the result dataset. Columns are identified by their caption. Therefore, the caption of the columns must be unique. In addition, the user cannot see the data table when working with the Pivot Table Analysis. Captions should be designed to provide information about the kind of data returned in the respective column to enable the user to decide where to place the data in the Pivot grid.
- At least one column in the result dataset must return a numerical value. Only numerical values can be added to the cells in the matrix of the Pivot Grid.
- If navigation from the table to the object profile or object cockpit of a base object for each result dataset is provided, it is also identically available in the Pivot Grid.
- If you would like the user to analyze multiple numerical values that are having a different aspect, For example, costs that are having a different cost type, the aspects can only be summed up if you list the aspect, For example, the cost type, in one column and the value in a second column. If you add one column for each aspect, that each contain the respective values for this aspect, the data can only be analysed next to each other, but not merged.

  For example, if a configured report shall return for applications in application groups the current costs over the years separately for each cost type. The way the data can be analyzed in a Pivot Grid depends on the structure of the tabular output of the report:

  - The tabular output of the configured report may display one line returning the cost type name and one row returning the budget value for all cost types:

	Application Group Name	Application Name	Cost Type Name	Budget Value Value	Budget Value Year
1	2. Trade Entry	EMERGING MARKET SYSTEMS v. 3.1.2	Subscription	25.00	2015
2	2. Trade Entry	EMERGING MARKET SYSTEMS v. 3.1.2	Expense	115.00	2015
3	2. Trade Entry	EMERGING MARKET SYSTEMS v. 3.1.2	Consulting	31.00	2015
4	2. Trade Entry	EMERGING MARKET SYSTEMS v. 3.1.2	Other	14.00	2015
5	2. Trade Entry	EMERGING MARKET SYSTEMS v. 3.1.2	Subscription	25.00	2016
6	2. Trade Entry	EMERGING MARKET SYSTEMS v. 3.1.2	Expense	129.00	2016
7	2. Trade Entry	EMERGING MARKET SYSTEMS v. 3.1.2	Consulting	37.00	2016
8	2. Trade Entry	EMERGING MARKET SYSTEMS v. 3.1.2	Other	15.00	2016
9	2. Trade Entry	EMERGING MARKET SYSTEMS v. 3.1.2	Subscription	25.00	2017
10	2. Trade Entry	EMERGING MARKET SYSTEMS v. 3.1.2	Expense	132.00	2017
11	2. Trade Entry	EMERGING MARKET SYSTEMS v. 3.1.2	Consulting	37.00	2017

  If the report is analyzed in a Pivot Grid for application group costs per year. The cost values are all from the same column and Therefore, the Pivot Grid can be configured to display costs independent from the cost type. The cells in the matrix then display a sum of the costs for all cost types belonging to the same year and application group:

	Budget Value Year				
Application Group Name	2015	2016	2017	2018	Grand Total
1. Market Data	5857	6611	9485	9697	31650
2. Trade Entry	13615	14990	14803	15346	58754
3. Front Office	12174	14412	15439	15396	57421
4. Back Office	18231	20295	20683	377891	437100
5. Customer Relations	1340	2199	7013	9599	20151
6. Risk Reporting	4241	5566	9366	10139	29312
7. Finance Reporting	6083	6957	7612	1705340	1725992
8. Accounting	3408	4284	4484	1511832	1524008
9. Payments	3561	4168	4444	4861	17034
Grand Total	68510	79482	93329	3660101	3901422

Alternatively, the Pivot Grid can display costs per cost type by adding the cost types as column header:

	Budget Value Year		Cost Type Name				
Application Group Name	2015				2015 Total	2016	
	Consulting	Expense	Other	Subscription		Consulting	Ex
1. Market Data	920	3178	544	1215	5857	1048	
2. Trade Entry	2405	8507	1013	1690	13615	2671	
3. Front Office	2150	7265	974	1785	12174	2518	
4. Back Office	3223	10974	1414	2620	18231	3555	
5. Customer Relations	95	342	198	705	1340	270	
6. Risk Reporting	604	2125	417	1095	4241	876	
7. Finance Reporting	978	3300	545	1260	6083	1099	
8. Accounting	593	2144	251	420	3408	675	
9. Payments	640	2087	289	545	3561	765	
Grand Total	11608	39922	5645	11335	68510	13477	

A filter field is then displayed on top of the column headers that allows to remove costs for singel cost types from the dataset.

For each year, a total sum is calculated, displaying the total costs independent from the cost type. On the right of the grid, the **Grand Total** column calculates the total costs per application group and year:

⊟ 2018				2018 Total	Grand Total
Consulting	Expense	Other	Subscription		
1731	5946	735	1285	9697	31650
2692	10021	1093	1540	15346	58754
2670	9805	1111	1810	15396	57421
3109	370402	1545	2835	377891	437100
1689	5689	760	1461	9599	20151
1892	5986	821	1440	10139	29312
957	342963	645	1360775	1705340	1725992
983	209988	441	1300420	1511832	1524008
871	3097	348	545	4861	17034

- The tabular output may alternatively display one line per cost type with the cost type name given as column header and the cost type value listed in the respective row:

	Application Group	Application	Subscription	Expense	Consulting	Other	Year
1	OptiRetail CRM As-Is Situation	Business EAI Platform 2.2	110.00	4.00	1.00	28.00	2015
2	Corporate CRM AS-IS Situation	Business EAI Platform 2.2	110.00	4.00	1.00	28.00	2015
3	Corporate CRM TO-BE Situation	Business EAI Platform 2.2	110.00	4.00	1.00	28.00	2015
4	Applications Using UDB	Groupware Services 2.2	60.00	29.00	7.00	16.00	2015
5	Corporate CRM AS-IS Situation	Groupware Services 2.2	60.00	29.00	7.00	16.00	2015
6	Corporate CRM TO-BE Situation	Groupware Services 2.2	60.00	29.00	7.00	16.00	2015
7	5. Customer Relations	Mafo-Portal 2.6	500.00	63.00	17.00	129.00	2015
8	CRM Analysis	Mafo-Portal 2.6	500.00	63.00	17.00	129.00	2015
9	Applications Using UDB	Mafo-Portal 2.6	500.00	63.00	17.00	129.00	2015
10	SAP Landscape	Mafo-Portal 2.6	500.00	63.00	17.00	129.00	2015
11	Customer Management	Mafo-Portal 2.6	500.00	63.00	17.00	129.00	2015
12	OptiRetail CRM As-Is Situation	Mafo-Portal 2.6	500.00	63.00	17.00	129.00	2015

To analyze the data in a Pivot Grid including all cost types, the columns containing cost type values must all be added to the matrix cells. The cost types are then added as sub-headers to the columns, but there is neither a filter nor a total per year calculated, because the columns are regarded ans independent from each other:

Application Group	2015				2016		
	Consulting	Subscription	Expense	Other	Consulting	Subscription	E
1. Market Data	920	1215	3178	544	1048	1280	
2. Trade Entry	2405	1690	8507	1013	2671	1690	
3. Front Office	2150	1785	7265	974	2518	1785	
4. Back Office	3223	2620	10974	1414	3555	2620	
5. Customer Relations	95	705	342	198	270	705	
6. Risk Reporting	604	1095	2125	417	876	1095	
7. Finance Reporting	978	1260	3300	545	1099	1345	
8. Accounting	593	420	2144	251	675	770	
9. Payments	640	545	2087	289	765	545	
Grand Total	11608	11335	39922	5645	13477	11835	

The **Grand Total** column is also splitted into the sub-columns:

2018				Grand Total			
ting	Expense	Subscription	Other	Consulting	Expense	Subscription	Other
1731	5946	1260	729	5305	18357	5285	2672
2665	10021	1540	1032	10499	37360	6610	4197
2670	9805	1785	1105	10144	35845	7140	4261
3109	370402	2835	1545	13553	406800	10695	6052
1689	5689	1390	743	3140	10792	4295	1836
1804	5986	1380	785	4835	16847	5010	2436
957	342960	1360660	605	4246	354481	1364675	2432
983	209988	1300420	441	3028	217175	1302380	1425
871	3097	545	348	3080	10487	2180	1287
16479	963894	2671815	7333	57830	1108144	2708270	26598

The report can be configured to display the table with a button in the toolbar that changes the view to the Pivot table analysis. The configured report can alternatively be configured to open directly in the Pivot table analysis view.

Do the following to enable Pivot table analysis:

1) In the explorer of the **Reports** tab, click the configured report.
2) In the attribute window, select one of the following options in the **Analysis Mode** attribute:

- `Optional`: A button **Pivot Grid**  is displayed in the tabular report to open a Pivot analysis. This is the default setting for new reports.
- `Always`: The report directly opens in the Pivot table analysis view.
- `Never`: A Pivot table analysis is not provided.

3) In the toolbar, click the **Save**  button to save your changes.

Please note that changing the **Analysis Mode** for existing configured reports might require a re-creation of the report view. It is recommended that the configured report be tested and the report view recreated only if changes were not applied.

If you would like the user to be informed about the functionality of the Pivot grid analysis view via the context sensitive help, you can do the following:

- An Alfabet specific standard help file is available for Pivot grid analysis. The standard help is only displayed if it is activated for the configured report:
  - If the **Analysis Mode** is changed to `Optional` or `Always`, the name of the standard help file is automatically written into the attribute **Help Index** of the report. The file is then offered to the user as context sensitive help for the configured report. Once the **Help Index** is set, it is not automatically removed when the Analysis Mode is changed again, but it can be removed manually.
  - The default setting for the report is `Optional`. Nevertheless the **Help Index** attribute is empty by default. To activate the standard help while keeping the default setting, you must either set the **Analysis Mode** to `Never` and then back to `Optional`, or manually enter `PivotTableAnalysis.html` in the **Help Index** attribute.
- A customer specific help can be opened. The method for defining customer specific help is described in the section Specifying Custom Help for a Configured Report.

Configuring a Data Table Report with Restructuring Options for the End User

End user design option can be added to a tabular configured report based on a native SQL query. The configured report will then be displayed in the design defined via the `SELECT` statement of the native SQL query it is based on, but it can be reconfigured by the user. The user can deselect columns from display, change the sort order of columns and change the report design from tabular to matrix. Furthermore, pictures and colors can be added to the design.

Any tabular configured report can be defined as data table report with end user design capabilities by adding the following configuration to the report:

1) In the explorer of the **Reports** tab, click the configured report.
2) In the attribute window, select `DataTableReport` from the dropdown list of the **Template** attribute.

The attribute **Standard Context-Sensitive Help Index** is filled automatically with the correct link to the end user help file in the standard Alfabet online help on selection of the `DataTableReport` template. The standard help can be deactivated by deleting the value from the **Standard Context-Sensitive Help Index** attribute. A customer specific help can be opened. The method for defining customer specific help is described in the section Specifying Custom Help for a Configured Report.

3) In the toolbar, click the **Save**  button to save your changes.

Note the following about the definition of the native SQL query for the report:

- In the **Data Source** editor that allows the user to select the data for display in the report, the object class properties are listed as defined in the column names of the `SELECT` statement. The end user can change the caption of the columns that are defined via the native SQL query of the report directly in the report.
- If the `DataTableReport` template is used, the first argument of the `SELECT` statement is not hidden in the tabular output of the configured repor and is not automatically providing navigation capabilities if returning the REFSTR values of objects.
- Navigation from the report to the object view or object profile of objects that are accessible via the configured report is not available by default. You can define columns in the tabular report to provide navigation. Navigation can be defined for any data and any number of columns. For example, for a configured report that displays information flows and the source and target applications thereof, the report can be configured to provide navigation to the information flows if the user double clicks on the information flow name, to the source applications if the user double clicks the name of a source application and to the target applications if the user double clicks the name of the target application.

Navigation is provided via the Alfabet query language instruction `CreateRefImage`, that merges a column returning the `REFSTR` of an object with any other column in the report. As a result, a single, navigable column is displayed at the position of the column returning the `REFSTR`, but showing the information from the other merged column. For information about the definition of the instruction, see Configuring Cells in a Data Table Report to Provide Navigation to an Object Profile in the section Using Instructions to Format the Results of an Alfabet or Native SQL Query.

- In general, Alfabet query language instructions can be used with the native SQL query, with the limitation that the result must not involve dependencies between cells in different columns of the resulting dataset. For example,:
  - You can color cells using a `ColorAssignment` instruction, because the coloring depends on the results in the current column only, but a `RowColorAssignment` instruction will be ignored, because cell coloring of cells of one column would depend on the result of cells in another column.
  - You can use a `JoinColumns` instruction. The result is a single, independent column in that will be presented as selectable or deselectable data to the end user and the results in the joined column will be displayed wherever the end user selects it to be displayed in a tabular or matrix structure. In contrast, you cannot use a `GroupBy_Ex` instruction, because the instruction creates a structure in the result dataset that involves more than one column to be created.
  - You cannot use a `LinkAssignment` instruction because the link displayed in one column depends on an object specification in another column of the dataset.
- Coloring and picture assignment defined in the native SQL query for data in a column will only be available in a tabular layout. If the user changes the design to a matrix layout, the default picture and color assignment will be removed. The end user can define own color and picture rules that supersede the default color and picture rules. Color rules and picture rules defined in the native SQL query are not added to the legend of the data table report. The legend is limited to user defined coloring and picture assignment.

Defining Audit Management Related Configured Reports

The audit history is available for an object in an object class for which the **Audit** attribute is set to "`true`".

The complete audit history is available for an object when the user clicks the **Audit Trail** button in the toolbar of the object profile.

Nevertheless it might be required to define a customized audit history via a configured report of the type `NativeSQL` for the following reasons:

- Because the **Object Audit** view displayed for a selected object displays all information about the object independent of the access permission available to the user viewing the audit information, it is recommended that you customize the display of audit information for users. To this end, you can create native SQL-based configured reports to extract specific information from the audit history such as the changes made to the object by a specific user or specific types of changes made to the object (For example, new information flows added to an application).
- Although property values based on an enumeration, indicator range values, object states and statuses are translatable, the translations are not considered in the audit trail displayed in the **Object Audit** view. If you want translated values to be displayed, you must define a native SQL-based configured report that displays the translated values. For information about the required query configuration to display the translated values in the configured report, see the section Translating Configured Reports.

Please note the following:

- If you want to define filters for configured reports displaying audit history information, you must add the filters manually. For more information, see the section Defining Filters for Configured Reports and Selectors in the chapter Defining Queries.
- Once the report is configured, you must make it available in the toolbar of the relevant custom object view by configuring a custom button for the object view. For more information about how to configure a custom button, see thesection Configuring Custom Buttons for the Toolbar of a Custom Object View in the chapter Configuring Object Views.
- For general information about how to make the auditing capability available for an object class, see the section Specifying History Tracking for an Object Class in the chapter Configuring the Class Model.

This chapter provides information about the audit database table that is required to build native SQL queries to display audit information.

Audit information is stored in a separate audit table for each object class. For each object class for which the property **Audit** is set to "`true`", a table named <TechnicalNameOfObjectClass>`_AU` is created. Whenever an object is changed, a new line is written in the audit table. The audit table has the following columns:

- ***AUDIT_TIME1***: The start date of the validity of the row. In other words, the date when the change was performed.
- ***AUDIT_TIME2***: End data of the validity of the row. This is set to `NULL` during creation of the row. When the next change to the object occurs, a new row is added for the object and the time is set to the ***AUDIT_TIME1*** of the next change. In other words, if the ***AUDIT_TIME2*** is set, the object was changed again after the change written to the ***AUDIT_TIME1*** row. For the last change to the object, the ***AUDIT_TIME2*** is `NULL`.

The ***AUDIT_TIME2*** is not necessarily identical with the ***AUDIT_TIME1*** of the next row in the audit table. For example, data import via the Alfabet Data Integration Framework (ADIF) can result in irrelevant audit table entries that document a change to the object that is not relevant to any of the object class properties listed in the audit table. As a result, audit table rows identical in all object properties are created. ADIF allows a mechanism to be activated that deletes the irrelevant entries from the audit table after data import is performed. Only the first of a number of identical audit table entries is kept. For audit table cleanup performed prior to Alfabet 10.6, the ***AUDIT_TIME2*** of that entry was not identical to the ***AUDIT_TIME1*** of the next relevant entry in the table. For audit table cleanup performed with Alfabet 10.6 and higher, a mechanism has been implemented to close the time gap between the previous and next meaningful entry in the table. Entries are not only removed, but the audit times are aligned. ***AUDIT_TIME2*** (end time) of the previous row is set to ***AUDIT_TIME1*** (start time) of the next row to ensure a gapless coverage of the history timeline.

- ***AUDIT_USER***: The user name of the user that performed the change.

  The storage of the user name in the database table depends on the setting of the parameter **Update History User Name** of the alias configuration of the Alfabet Web Application. Either the user name or the technical name of the user can be stored as user name in the audit tables.

- ***DELETE_USER***: The user name of the user that deleted the object. This row is only filled on deletion of an object.

  The storage of the user name in the database table depends on the setting of the parameter **Update History User Name** of the alias configuration of the Alfabet Web Application. Either the user name or the technical name of the user can be stored as user name in the audit tables.

- ***AUDIT_TR***: The unique GUID of the transaction.
- ***REFSTR***: The `REFSTR` of the object that was changed. This column specifies the object that has been changed.
- <OtherTableProperties>: For each column in the object class' database table, a column with an identical column header exists in the audit table. All properties of the objects are written in the respective row of the audit table when one of the properties is changed. Therefore, each row of the audit table is a complete image of the object for the time period between ***AUDIT_TIME1*** and ***AUDIT_TIME2***.

	AUDIT_TIME1	AUDIT_TIME2	AUDIT_USER	DELETE_USER	AUDIT_TR	REFSTR	INSTGUID	ID	NAME
1	1993-02-06 15:00:00.000	2010-03-25 11:24:00.420	PICARD	NULL	68EA5CA135C911	76-2934-0	3383726EB8614(	APP-2934	GAA-POS
2	2010-03-25 11:24:00.420	2010-03-25 11:24:25.130	CUSTOMER	NULL	27460FEC312D40	76-2934-0	3383726EB8614(	APP-2934	GAA-POS
3	2010-03-25 11:24:25.130	2010-03-25 16:24:03.160	CUSTOMER	NULL	0B968D9217574E	76-2934-0	3383726EB8614(	APP-2934	GAA-POS
4	2010-03-25 16:24:03.160	NULL	PICARD	NULL	63FE6B7535C911	76-2934-0	3383726EB8614(	APP-2934	GAA-POS

FIGURE: Example for the change history of an object in the object table

It is recommended that you create an index for the audit table when a configured report is implemented for audit information. An index is automatically created for the audit table if you set an audit key for the object class for which the audit table is assigned. For more information about creating an index, see *Creating an Index to Improve Performance for Query Searches in Database Tables* in the chapter Defining Queries.

Creating a Card View Report

To generate a configured report of the type **Custom** that displays graphic containers for each row of data from search results:

- In the **Reports** tab of Alfabet Expand, create a new configured report and define the attributes for the configured report. For more information about creating a configured report and defining the configured report's attributes, see Creating a New Card View Report.
- Define a custom report view for the configured report. For more information, seeCreating a New Card View Report.
- Open the **Report Assistant** and configure the configured report. For more information, see Configuring the Card View Report with the Report Assistant.
- If the configured report has filters defined, check the functionality of the filters and optionally change the captions and look of the filter fields. For more information, see Defining Filters for Configured Reports and Selectors in the chapter Defining Alfabet Queries.
- If the configured report has no filters defined, you can optionally edit or delete the parameter panel of the configured report.
- Set the **Report State** of the report to **Active**. For information on how to change the attribute of the configured report, see General Guidelines for Creating Configured Reports.
- You can define the visibility of toolbar buttons of the configured report for different view schemes using the **Configure Report** functionality in the context menu of the report. For more information, see Refining Visibility Issues in the View Scheme.
- Define access rights for the configured report in the **Reports Administration** functionality of Alfabet. By default, access is possible for all users. For more information, see the chapter *Defining and Managing User Access to Configured Reports* in the reference manual .

The following information is available:

Creating a New Card View Report

To create a new configured report of the type `Custom` in Alfabet Expand:

1) In the **Reports** tab, right-click the **Reports** folder or a sub-folder thereof and select **Create New Report**. You will see the new configured report listed below the selected folder.

As an alternative to defining all attributes for a new configured report, you can copy and paste an existing configured report and make modifications, as needed. To do so, right-click the existing configured report that you want to copy and select **Copy**. Right-click

the new configured report and select **Paste**. Modify the necessary attributes described below.

2) In the attribute window, select `Custom` in the **Type** field.

3) In the attribute window, define the following attributes for the configured report.

- **Name**: Optionally, change the technical name of the configured report. The technical name of the configured report must be unique.

  The technical name is used for identification of the configured report in the context of technical processes such as merging the report configuration with a report configuration saved as an XML file. Therefore, the following is required when defining the name of a configured report:

  - The name must be unique. The name must be unique regardless of the report folder it is structured in.
  - The name may not be changed once a configured report has been created.
  - A validation mechanism checks for correct syntax when defining a technical name. Please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.
  - If the technical name of the configured report is changed, the name will be correctly updated in other configuration objects referencing the changed report during design time.

  Please note that the name of a changed configuration object will not be updated in guide pages or guide views. If you plan to change the name of a configuration object, the reference in the navigation page should be changed prior to changing the configuration object name. You can use the **Show Usage** functionality in the context menu of the configured report to evaluate whether the configured report has been added to a guide view or guide page.

- **Caption**: Enter a caption for the configured report. The caption you define here will be displayed in the **Reports Administration** functionality in the **Administration** module and the **Configured Reports** views of the Alfabet interface. If the configured report is assigned to an object view as a page view, the text will be displayed as the page view caption in the object view. The caption of the configured report may exceed the conventional 64-character limit.
- **Description**: Provide short information about the configured report that is useful to Alfabet users. This comment will be displayed on the Alfabet user interface in the header of the configured report in a single line under the report caption and in the **Description** field for the configured report in the table of all views listing configured reports, like the **Configured Reports** views of the **Search** functionalities. If the configured report is assigned to an object view as a page view, the text will be displayed under the page view caption as short description in the object view.

  In the configured report, the description will be truncated if it is longer than one line on the screen. Therefore, the description should be short to ensure complete display in the configured report header.

- **Usage Guideline**: Provide information that helps users to execute and interpret the configured report. This information will not be displayed directly on the user interface.

The user can access the information using the following mechanisms:

- If the user moves the cursor over the description provided for the configured report with the attribute **Description**, a tooltip opens that includes both the description and the usage guideline.
- The **Options**  button will be displayed on the upper right with an option **Help On Filter Fields** to open a new window displaying the usage guideline in addition to any filter field hints, if configured.

  For views that have filters defined, the **Options**  button will only be displayed if the filter panel either allows batch clearing of filter fields or collapsing and expanding of the filter panel. For more information about the configuration of these functionalities, see Configuring the Complete Filter Area to be Collapsible and Allowing the User to Clear the Filter Area.

- **Technical Comment**: Provide information that is useful for solution designers working with Alfabet Expand in order to better understand the configured report from a technical perspective. The **Technical Comment** will not be displayed in the user interface. Nevertheless, you can configure the interface to display the information in, For example, the attribute section of the configured report's object profile in the **Reports Administration** functionality.
- **Template**: Select `CardsViewReport` to create a configured report that displays information for each row of search results in card containers.
- **Report State**: The **Report State** attribute is view-only and is set to `Plan` for new configured reports. The configured report can only be edited when the **Report State** attribute is set to `Plan`. After finishing all configuration steps described in the following, the **Report State** attribute must be set to `Active` as described in the section General Guidelines for Creating Configured Reports. The configured report is then visible to users in the Alfabet user interface.
- **Help Index**: Specify the location of the external Help file using the following syntax:`CUSTOM_HELP:<URL>`(For example,:`CUSTOM_HELP:http://helphost/report1.html`). The external help file will be displayed when the user clicks the context-sensitive Help link in the standard Help menu.

  Server variables can be used in the Help links. Server variables allow you to define all or part of the URL definition in the server alias configuration of the Alfabet Server instead of directly defining it in the configuration in Alfabet Expand. Use of server variables is useful, For example, in order to change a Web server name defined in multiple URL definitions in Alfabet Expand. Instead of changing all URL definitions, only the variable value in the server alias configuration must be updated. The use of server variables is described in the section Using Server Variables in Web Link and Database Server-Related Specifications in the chapter Configuring Reports.

- **Apply to Class**: When the configured report is applied to an object class or object class stereotype, a base object can be selected for the configured report. A filter for selection of a base object will automatically be set on top of the configured report. When configuring queries for the configured report, you can use the Alfabet parameter `BASE` to refer to the

object selected in the filter and define search conditions dependent on the base object. To apply the configured report to a base class, click the **Browse** button to open a dialog box and select the object class or object class stereotype to which you want to apply the configured report and click **OK**.

Please note the following:

- If the configured report is assigned to the object view for the relevant object class, the configured report will automatically open with the current object selected as the base object even if the **Apply to Class** attribute is not set.
- If the configured report is applied to an object class stereotype, the filter field for selection of the base object will use the caption of the object class stereotype as caption of the filter field and the selector defined in the class settings for the object class stereotype for selection of the base object.
- Alternatively, the base object of a configured report can be set with the **Base Object Query** attribute. Whereas the **Apply to Class** attribute enables the user to determine the current base object by setting a filter, the **Base Object Query** attribute evaluates the base object by executing a query when the configured report is opened. A filter field is not available and the user cannot change the base object.
- The **Apply to Class** attribute also influences the availability of the configured report in the Alfabet interface. Select a class to make the configured report available in the **Configured Reports** page view in the object view for the selected object class. If you do not set the attribute, the configured report will only be available in the **Configured Reports** functionality. The configured report can be assigned to object views and wizards independent of the attribute.

- **Base Object Query**: If a base object query is defined for the configured report, a base object will be evaluated for the configured report at runtime. When configuring the other queries for the configured report, you can use the Alfabet parameter `BASE` to refer to the object returned by the base object query and define search conditions dependent on the base object. To apply the configured report to a base class, click the **Browse** button to open a text editor, define either a native SQL query or an Alfabet query that returns a single object, and click **OK**.

  Note the following:

  - Alternatively, the base object of a configured report can be set by means of the **Apply To Class** attribute. Whereas the setting of the **Base Object Query** attribute triggers the evaluation of the base object by executing a query when the configured report is opened, the **Apply to Class** attribute enables the user to determine the current base object by setting of an automatically-generated filter.
  - The query defined for the base object query is executed when the user opens the configured report to evaluate the base object at runtime. Parameters referring to the current working environment can be used in the query to find a base object dependent on the user opening the configured report or the user profile used for login, For example,. A base object query may also find,

For example, the ICT object that a specific application is currently assigned to.

- If the **Base Object Query** attribute is set and the configured report is assigned as a view to an object profile, the Alfabet parameter `BASE` referring to the `REFSTR` of the object that the user is working with when the configured report is opened can be used in the base object query.

- **Execute on Enter**: Set to `True` to execute the configured report with empty filter settings when the user opens the configured report on the Alfabet interface. Set to `False` to open the configured report without execution of the Alfabet query with empty filter settings. By default, this attribute is set to `True`. It should only be set to `False` when a configured report would result in an exceedingly big dataset when executed with empty filter settings.

  When you set **Execute on Enter** to `False`, you must make sure that the **Submit** button is available in the custom report view of the configured report. If the **Submit** button is deleted from the custom report view, the configured report cannot be displayed. The setting and execution of the **Execute on Enter** attribute is independent from the definition of filters for a configured report. A configured report without filters must nevertheless have a **Submit** button when **Execute on Enter** is set to `False`.

- Selector Behavior: The attribute can be used to exclude configured reports that are defined for special purposes, like For example, triggering of data export via the RESTful API, from being selectable in the **Configured Reports** table. By default, the attribute is set to `Visible` and the configured report is visible in the selector for adding configured reports to the **Configured Reports** table if all other conditions for access to the configured report apply. If the attribute is set to `NotVisible`, the configured report is excluded from the selector for adding configured reports to the **Configured Reports** functionality. Please note however, that this setting is not excluding the configured report from any standard views or customer configured views and selectors, but exclusively from the standard selector for configured reports implemented in Alfabet. The attribute cannot be edited in the attribute window directly. To change the setting, right-click the configured report and select **Set Report Selector Behavior to 'Not Visible'** or **Set Report Selector Behavior to 'Visible'** respectively.

- **Base Classes**: Define the object classes that must be available to run the configured report. If this attribute is set, the Alfabet Server checks whether the object classes specified as base classes are excluded from the view scheme used to access the configured report. If the user does not have access permissions to one of the base classes, the configured report will be disabled. To set the attribute, click the **Browse**  button to select the object classes that are required to run the configured report and click **OK**.

- **Can Create Express View**: Select `True` if an express view may be created for the report. Select `False` if an express view may not be created for the report. Please note that if the report has a custom report view, the setting must be consistent with the setting of the attribute **Can Create Express View** of the custom report view.

  An express view is a link to a specific Alfabet view displaying data that allows individuals inside or outside of the user community to view information in Alfabet. When the express view is created, an email notification is automatically mailed to a specified recipient. The recipient receives a URL that allows him/her to access the current page view in Alfabet. For more information, see the section *Sending and Receiving Express Views* in the reference manual Getting Started with Alfabet Expand.

4) In the explorer, right-click the configured report and select **Create Configured Report View**. The view editor opens. In the explorer, the custom report view is displayed as a subordinate object of the configured report. The attributes for the custom report view are displayed in the attribute window. You can optionally edit the following attributes:

   - **Name**: Optionally you can change the name for the custom report view. The **Name** must be unique for custom report views.
   - **Can Create Bookmark**: Select `True` if a bookmark may be created for the configured report. Select `False` if a bookmark may not be created for the configured report.
   - **Can Create Express View**: Select `True` if an express view may be created for the configured report. Select `False` if an express view may not be created for the configured report. Please note that the setting must be consistent with the setting of the attribute **Can Create Express View** of the configured report.

     The custom report view initiates the **Report Assistant**. A custom report view shall never be deleted from the configured report after the **Report Assistant** was already used to configure the configured report. When you delete the custom report view of an existing template-based configured report and create a new one, the **Report Assistant** is reset to the example specified by default and your configuration is lost.

5) In the explorer, right-click the configured report and select **Start Alfabet Report Assistant**. The **Report Assistant** opens.
6) Card View Reports are designed in the **Report Assistant**. Configure the report as described below in the section Configuring the Card View Report with the Report Assistant.
7) Click **OK** to save the configuration.
8) In the toolbar, click the **Save** button to save your changes.
9) In the explorer, right-click the configured report and select **Review Report**. The configured report opens in a web browser. If the results are not as expected, you can edit the configured report until the output of the configured report meets your expectations.

Configuring the Card View Report with the Report Assistant

The **Report Assistant** allows you to define the content of the **Card View Report** and to design the graphic display. For information about how to open the **Report Assistant**, see Creating a New Card View Report. For general information about working with the **Report Assistant**, see the section Configuring the Report with the Report Assistant in the chapterCreating a Graphic Report.

The content and layout of the Card View report are defined and designed via the attributes of the node elements of the explorer in the **Report Assistant**.

The following table lists the types of explorer node elements that can be added to the report configuration to specify the content of the Card View report. Information is provided in the table about the purpose of each explorer node element:

Explorer Node Element	Required to Configure:
Root Node	General layout of the card view report and mapping of the information provided by the queries of the card view report to the relevant graphic display in the card view report.
Root Query	Definition of a query that defines the display of card view elements depending on data in the Alfabet database.
Picture rendering definitions	Design of specific picture data columns of be displayed. For each picture column to be displayed, use the **Add Picture Definition** context menu option to create a new rendering definition. You must then name the column according to how it is named in the query and define the image height desired.

Defining the Content of the Card View Report

The content of the card view report and all object specific design and information are defined in a native SQL or Alfabet query in the **Query** section of the **Root Query** node element. For general information about how to query a dataset for a configured report, see Defining Query Attributes in the chapter Creating a Graphic Report.

The following settings can be applied:

Attribute	Description
Name	Defines the name of the **Query** element displayed in the explorer of the **Report Assistant**.
Alfabet Query/Query as Text/Native SQL	Defines a query to find the objects that are displayed in the report and the instructions used to format the boxes in the card view.

The query defined for the Card View Report must find the objects that shall be displayed in the report and the relevant information about the objects which shall be displayed in the form of card boxes. For each row in the result dataset, a box is added to the report.

For native SQL queries, the first argument of the `SELECT` statement must return the `REFSTR` of the object to be displayed in the report in addition to defining the following information in the `SELECT` statement.

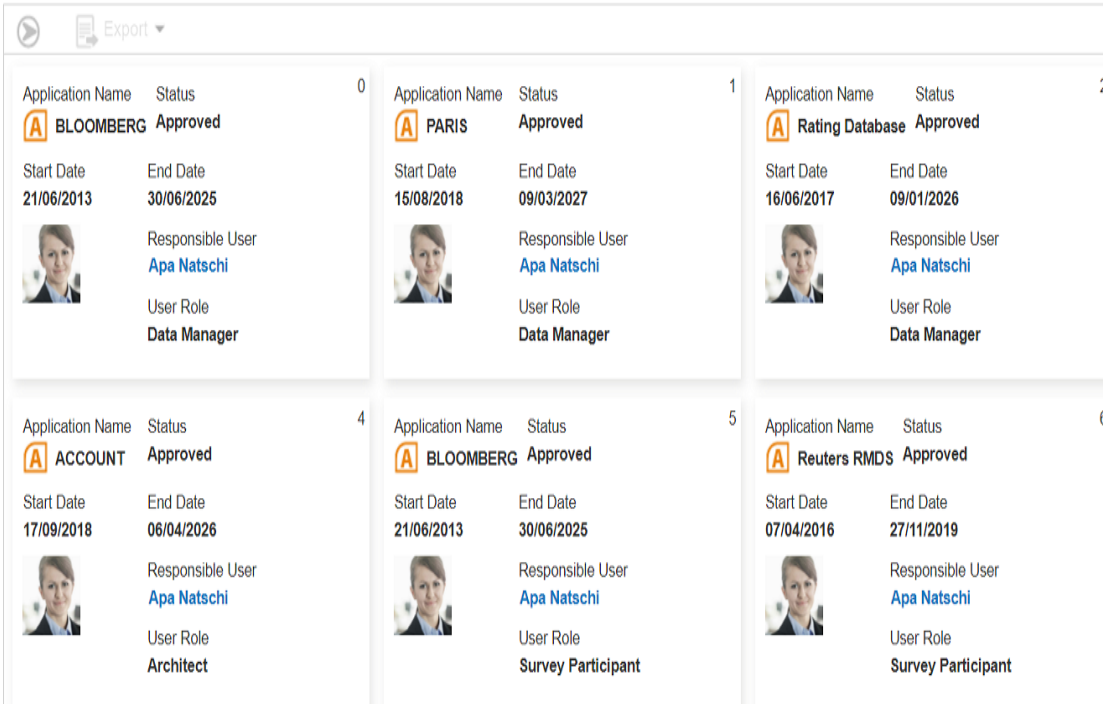
For information about defining Alfabet queries, see the chapter Defining Queries. For information about the rules that apply to the definition of native SQL queries when used in Alfabet configurations, see Defining Native SQL Queries in the chapter Defining Queries.

Alfabet instructions can be used in configuring the output format of the card view or to rename and remove columns. For example, the `MakeReferenceValue` instruction allows a string value to be presented as a link that enables navigation to an object. The `CreateDSInfo` instruction is useful to change the the caption of columns displayed as headings for data in the cards. The Alfabet instruction `SetObjectIcon` can also be used to display the icon assigned to a class. For more information about configuring Alfabet instructions, see Using Instructions to Format the Results of an Alfabet or Native SQL Query in the chapter Defining Queries.

For example, a configured card view report shall display information about applications, the users assigned to those applications and the role of the users. A sample query is defined in the SQL query below:

```
SELECT app.REFSTR, app.NAME AS 'ApplicationName', app.STARTDATE AS
'StartDate', app.ENDDATE AS 'EndDate', app.STATUS AS 'Status',
per.PICTURE AS 'Picture', per.REFSTR AS 'UserResp', per.TECH_NAME AS
'PName', rtp.NAME AS 'Role'

FROM APPLICATION app

LEFT JOIN ROLE AS rol ON rol.OBJECT = app.REFSTR

LEFT JOIN PERSON AS per ON rol.RESPONSIBLE = per.REFSTR

LEFT JOIN ROLETYPE AS rtp ON rol.ROLETYPE = rtp.REFSTR

WHERE per.PICTURE IS NOT NULL

Instructions

CreateDSInfo("ApplicationName,Status,StartDate,EndDate,Picture,UserR
esp,Role", "Application Name,Status,Start Date,End
Date,Picture,Responsible User,User Role");

MakeReferenceValue(UserResp, PName);

RemoveColumns("PName");

SetObjectIcon("ApplicationName","IconText");

EndOfInstructions
```

The resulting card view report displays a card box for each row in the result dataset of the query. In each card box, the column names are displayed as captions above each value and the row number of the resulting dataset is at the top-right corner of the box. The icon assigned to the application is displayed alongside the name of the application. The image of the user responsible for the application is also displayed, while the name of the user has been converted into a link using the `MakeReferenceValue` instruction to provide navigation to the object cockpit of the user to display further information about the user. The order of columns and and the column names have been edited using the `CreateDSInfo` instruction:



This example report can also be generated using a simple tabular report, however, without the picture rendering and other graphic elements:

SQL Test Report

Export

233 object(s) has (have) been found.

ApplicationName	StartDate	EndDate	Status	Picture
20 CRM CSS	23/06/2015	04/02/2023	Approved	AlfaBuffer: size = 16341
21 GL Trade	14/04/2012	12/10/2018	Approved	AlfaBuffer: size = 16341
22 SUNGARD TREASURY TRADER	14/09/2012	12/12/2024	Approved	AlfaBuffer: size = 16341
23 Financial Times	16/05/2016	31/05/2023	Approved	AlfaBuffer: size = 16341
24 Corporate FI-CO	27/03/2017	05/04/2021	Draft	AlfaBuffer: size = 16341
25 CRM AI	08/04/2018	18/06/2021	Approved	AlfaBuffer: size = 16341
26 OptiRetail Marketing Solution	23/05/2017	04/03/2020	Approved	AlfaBuffer: size = 16341
27 SAP International	19/01/2019	05/10/2020	Approved	AlfaBuffer: size = 16341
28 Trade*Net	20/01/2015	20/01/2022	Approved	AlfaBuffer: size = 16341
29 TradeWeb	06/08/2011	25/05/2020	Approved	AlfaBuffer: size = 16341
30 Reuters Triarch	17/06/2013	09/01/2022	Approved	AlfaBuffer: size = 16341
31 SAP CMS	16/06/2016	10/10/2021	Approved	AlfaBuffer: size = 16341
32 Wagner Classifier	30/01/2019	20/05/2021	Approved	AlfaBuffer: size = 16341
33 Legal Report	10/10/2021	25/03/2027	Approved	AlfaBuffer: size = 16341
34 Demo Reporting	01/11/2020	03/01/2024	Approved	AlfaBuffer: size = 16341
35 SAP CMS	16/06/2016	10/10/2021	Approved	AlfaBuffer: size = 16341
36 Trade*Net	20/01/2015	20/01/2022	Approved	AlfaBuffer: size = 16341
37 TradeThru	01/04/2020	31/03/2026	Approved	AlfaBuffer: size = 16341
38 Cust. Survey Tool	01/07/2018	12/12/2023	Approved	AlfaBuffer: size = 16341

Defining the Basic Design of the Card View Report

The basic design of the card view report, like For example, any design that applies to all content independent from the displayed data is defined in the section **Layout**:

Attribute	Description
Apply Common Container Skin	Set to `True` if you want to display all object boxes with the container skin defined in the GUI scheme used for the Alfabet user interface when rendering the report. If you set the attribute to `False`, the cards will be displayed with white background and black, thin border.
Control Layout	Select the desired design of the card boxes to be displayed in the window. • Set to `Stack` if you want each card box to occupy an entire row of the window. • Set to `Flow` if you want as many cards placed side-by-side in a row as the responsive display window permits.
Card Width	Define the width of the individual object boxes in pixel when **Control Layout** is set to `Flow`.
Max Value Item Width	Define the maximum width in pixels that the items in each box can occupy.
Min Value Item Width	Define the minimum width in pixels that the items in each box can occupy.
Show Data-Set Row Numbers	Set to `True` if you want to display the row number of an object box. Please note: • The row number can only be displayed in the top right-hand corner of each object box.
Show Undefined Values	Set to `True` if you want to show values that are not defined to be displayed in the card box.
Values Followed by new lines	Specify where you would like to force a line break in each card box.
Values with no captions	Specify which values in a card box should be displayed without their captions.

When dealing with data that contains images, the attributes of each image column to be displayed must be defined in the **Picture Rendering Definitions** node element:

Attribute	Description
Column Name	Define the name of the specific image column.
Height	Define the height of the image in pixel to be displayed in the card box.

Creating a Graphic Report

To generate a configured report of the type `Custom` that displays graphical representations of search results:

- In the **Reports** tab of Alfabet Expand, create a new configured report and define the attributes for the configured report. For more information about creating a configured report and defining the configured report's attributes, see Creating a New Graphic Report.
- Define a custom report view for the configured report. For more information, see Creating a New Graphic Report.
- Open the **Report Assistant** and configure the configured report. For more information, see Configuring the Report with the Report Assistant.
- If the configured report has filters defined, check the functionality of the filters and optionally change the captions and look of the filter fields. For more information, see Defining Filters for Configured Reports and Selectors in the chapter Defining Queries.
- If the configured report has no filters defined, you can optionally edit or delete the parameter panel of the configured report.
- Set the **Report State** of the report to `Active`. For information on how to change theattribute of the configured report, see.
- You can define the visibility of toolbar buttons of the configured report for different view schemes using the **Configure Report** functionality in the context menu of the report. For more information, see Refining Visibility Issues in the View Scheme.
- Define access rights for the configured report in the **Reports Administration** functionality of Alfabet. By default, access is possible for all users. For more information, see the chapter *Defining and Managing User Access to Configured Reports* in the reference manual *User and Solution Administration*.

The following information is available:

- Creating a New Graphic Report
- Configuring the Report with the Report Assistant
  - Defining Elements of the Report
  - Configuring the Attributes of Report Elements

Creating a New Graphic Report

To create a new configured report of the type `Custom` in Alfabet Expand:

1) In the **Reports** tab, right-click the **Reports** folder or a sub-folder thereof and select **Create New Report**. You will see the new configured report listed below the selected folder.

As an alternative to defining all attributes for a new configured report, you can copy and paste an existing configured report and make modifications, as needed. To do so, right-click the existing configured report that you want to copy and select **Copy**. Right-click

the new configured report and select **Paste**. Modify the necessary attributes described below.

2) In the attribute window, select `Custom` in the **Type** field.

3) In the attribute window, define the following attributes for the configured report.

- **Name**: Optionally, change the technical name of the configured report. The technical name of the configured report must be unique.

  The technical name is used for identification of the configured report in the context of technical processes such as merging the report configuration with a report configuration saved as an XML file. Therefore, the following is required when defining the name of a configured report:

  - The name must be unique. The name must be unique regardless of the report folder it is structured in.
  - The name may not be changed once a configured report has been created.
  - A validation mechanism checks for correct syntax when defining a technical name. Please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.
  - If the technical name of the configured report is changed, the name will be correctly updated in other configuration objects referencing the changed report during design time.

  Please note that the name of a changed configuration object will not be updated in guide pages or guide views. If you plan to change the name of a configuration object, the reference in the navigation page should be changed prior to changing the configuration object name. You can use the **Show Usage** functionality in the context menu of the configured report to evaluate whether the configured report has been added to a guide view or guide page.

- **Caption**: Enter a caption for the configured report. The caption you define here will be displayed in the **Reports Administration** functionality in the **Administration** module and the **Configured Reports** views of the Alfabet interface. If the configured report is assigned to an object view as a page view, the text will be displayed as the page view caption in the object view. The caption of the configured report may exceed the conventional 64 character limit.
- **Description**: Provide a short information about the configured report that is useful to Alfabet users. This comment will be displayed on the Alfabet user interface in the header of the configured report in a single line under the report caption and the **Description** field for the configured report in the table of all views listing configured reports, like the **Configured Reports** views of the **Search** functionalities. If the configured report is assigned to an object view as a page view, the text will be displayed under the page view caption as short description in the object view.

  In the configured report, the description will be truncated if it is longer than one line on the screen. Therefore, the description should be short to ensure complete display in the configured report header.

- **Usage Guideline**: Provide information that helps users to execute and interpret the configured report. This information will not be displayed directly on the user interface. The user can access the information using the following mechanisms:
  - If the user moves the cursor over the description provided for the configured report with the attribute **Description**, a tooltip opens that includes both the description and the usage guideline.
  - The **Options** button will be displayed on the upper right with an option **Help On Filter Fields** to open a new window displaying the usage guideline in addition to any filter field hints, if configured.

    For views that have filters defined, the **Options** button will only be displayed if the filter panel either allows batch clearing of filter fields or collapsing and expanding of the filter panel. For more information about the configuration of these functionalities, see Configuring the Complete Filter Area to be Collapsible and Allowing the User to Clear the Filter Area.

- **Technical Comment**: Provide information that is useful for solution designers working with Alfabet Expand in order to better understand the configured report from a technical perspective. The **Technical Comment** will not be displayed in the user interface. Nevertheless, you can configure the interface to display the information in, For example, the attribute section of the configured report's object profile in the **Reports Administration** functionality.
- **Template**: Select one of the following:
  - `BranchingDiagramReport` to create a report with multiple levels of objects connected via lines.
  - `BusinessChartReport` to create a pie chart, bar chart, radar chart or line chart.
  - `CircularRoadMap` to create a report that displays object icons distributed in a circle divided into shells and circles.
  - `CustomPivotTable` to create a configured report that allows to analyse a cube in a Pivot table layout.
  - `DataTableReport` to create a configured report that allows users to sort, filter and layout a dataset at runtime.
  - `DynamicLaneReport` to create a dynamic lane report displaying relations between objects as connections between the object in different columns, whereby the number of columns depends on the results in the dataset.
  - `GalleryReport` to create a configured report displaying a gallery of object boxes with informations about objects.
  - `GanttReport` to create a Gantt chart.
  - `GaugeReport` to create a linear or circular gauge report.
  - `GridReport` to create a grid report displaying one or multiple configured reports in a cluster or tree structure.

- `LaneReport` to create a lane report displaying relations between objects as connections between the objects in different columns, whereby the number of columns is fixed.
- `MatrixMapReport` to create a matrix report.
- `MatrixMapReport_Ex` to create a matrix with enhanced functionalities.
- `NodeArcReport` to create a configured report that displays objects and connections between objects in the diagram styles available for standard Alfabet diagrams designed with the Alfabet Diagram Designer.
- `NodeArcReportWithBundling` to create a configured report that displays objects and connections between objects in the `Spring Layout` of standard Alfabet diagrams designed with the Alfabet Diagram Designer with the additional ability to bundle connections to provide a better overview in diagrams with a high number of objects and connections.
- `PercentageDistributionReport` to create a configured report that displays percentage distributions of objects in a bar or half-circle diagram.
- `PortfolioReport` to create a portfolio.
- `AugmentedAIReport` to create a portfolio diagnostics report.
- `TreeMapReport` to create a tree map or layered diagram report.

For HTML reports displaying indicator values in an HTML based table, no template is available and the attribute must be empty to define the configured report.

The templates `CaptureApplications`, `CaptureComponents`, `CaptureDevices`, `CaptureICTObjects`, `CapturePeripherals` and `CaptureServiceProducts` cannot be used in configured reports of the **Type** `Custom`. They are for exclusive use with tabular datasets defined in configured reports of the **Type** `Query` or `NativeSQL`.

- **Report State**: The **Report State** attribute is view only and is set to `Plan` for new configured reports. The configured report can only be edited when the **Report State** attribute is set to `Plan`. After finishing all configuration steps described in the following, the **Report State** attribute must be set to `Active` as described in the section General Guidelines for Creating Configured Reports. The configured report is then visible to users in the Alfabet user interface.
- **Help Index**: Specify the location of the external Help file using the following syntax:(For example,:). The external help file will be displayed when the user clicks the context-sensitive Help link in the standardHelp menu.

Server variables can be used in the Help links. Server variables allow you to define all or part of the URL definition in the server alias configuration of the Alfabet Server instead of directly defining it in the configuration in Alfabet Expand. Use of server variables is useful, For example, in order to change a Web server name defined in multiple URL definitions in Alfabet Expand. Instead of changing all URL definitions, only the variable value in the server alias configuration must be updated. The use of server variables is described in the section Using Server Variables in Web Link and Database Server-Related Specifications in the chapter Configuring Reports.

- **Apply to Class**: When the configured report is applied to a class, a base object can be selected for the configured report. A filter for selection of a base object will automatically be set on top of the configured report. When configuring queries for the configured report, you can use the Alfabet parameter `BASE` to refer to the object selected in the filter and define search conditions dependent on the base object. To apply the configured report to a base object class or object class stereotype, click the **Browse** button to open a dialog box and select the object class or object class stereotype to which you want to apply the configured report and click **OK**.

  Note the following:

  - This attribute is mandatory for configured reports displaying an HTML report about indicators. It is optional for all other configured reports of the type `Custom`.
  - If the configured report is applied to an object class stereotype, the filter field for selection of the base object will use the caption of the object class stereotype as caption of the filter field and the selector defined in the class settings for the object class stereotype for selection of the base object.
  - If the configured report is assigned to the object view for the relevant object class, the configured report will automatically open with the current object selected as the base object even if the **Apply to Class** attribute is not set.
  - Alternatively, the base object of a configured report can be set with the **Base Query** attribute. Whereas the **Apply to Class** attribute enables the user to determine the current base object by setting a filter, the **Base Query** attribute evaluates the base object by executing a query when the configured report is opened. A filter field is not available and the user cannot change the base object.
  - The **Apply to Class** attribute also influences the availability of the configured report in the Alfabet interface. Select a class to make the configured report available in the **Configured Reports** page view in the object view for the selected object class. If you do not set the attribute, the configured report will only be available in the **Configured Reports** functionality. The configured report can be assigned to object views and wizards independent of the attribute.

- **Base Object Query**: If a base object query is defined for the configured report, a base object will be evaluated for the configured report at runtime. When configuring the other queries for the configured report, you can use the Alfabet parameter `BASE` to refer to the object returned by the base object query and define search conditions dependent on the base object. To apply the configured report to a base class, click the **Browse** button to open a text editor, define either a native SQL query or an Alfabet query that returns a single object, and click **OK**.

  Note the following:

  - Alternatively, the base object of a configured report can be set by means of the **Apply To Class** attribute. Whereas the setting of the **Base Object Query** attribute triggers the evaluation of the base object by executing a query when the configured report t is opened, the **Apply to Class** attribute enables

the user to determine the current base object by setting of an automatically-generated filter.

- The query defined for the base object query is executed when the user opens the configured report to evaluate the base object at runtime. Parameters referring to the current working environment can be used in the query to find a base object dependent on the user opening the configured report or the user profile used for login, For example,. A base object query may also find, For example, the ICT object that a specific application is currently assigned to.
- If the **Base Object Query** attribute is set and the configured report is assigned as a view to an object profile, the Alfabet parameter `BASE` referring to the `REFSTR` of the object that the user is working with when the configured report is opened can be used in the base object query.

- **Execute on Enter**: Set to `True` to execute the configured report with empty filter settings when the user opens the configured report on the Alfabet interface. Set to `False` to open the configured report without execution of the Alfabet query with empty filter settings. By default, this attribute is set to `True`. It should only be set to `False` when a configured report would result in an exceedingly big dataset when executed with empty filter settings.

  When you set **Execute on Enter** to `False`, you must make sure that the **Submit** button is available in the custom report view of the configured report. If the **Submit** button is deleted from the custom report view, the configured report cannot be displayed. The setting and execution of the **Execute on Enter** attribute is independent from the definition of filters for a configured report. A configured report without filters must nevertheless have a **Submit** button when **Execute on Enter** is set to `False`.

- **Selector Behavior**: The attribute can be used to exclude configured reports that are defined for special purposes, like For example, triggering of data export via the RESTful API, from being selectable in the **Configured Reports** table. By default, the attribute is set to `Visible` and the configured report is visible in the selector for adding configured reports to the **Configured Reports** table if all other conditions for access to the configured report apply. If the attribute is set to `NotVisible`, the configured report is excluded from the selector for adding configured reports to the **Configured Reports** functionality. Please note however, that this setting is not excluding the configured report from any standard views or customer configured views and selectors, but exclusively from the standard selector for configured reports implemented in Alfabet.

The attribute cannot be edited in the attribute window directly. To change the setting, right-click the configured report and select **Set Report Selector Behavior to 'Not Visible'** or **Set Report Selector Behavior to 'Visible'** respectively.

- **Applicable for AlfaBot**: Specify whether the configured report is accessible via the AlfaBot. By default, the attribute will be set to `True` for new reports. Set the attribute to `False` if the configured report should not be opened outside of a specific context. For example, this may be relevant for reports that are specifically designed to be embedded in an object cockpit.

  This attribute is only relevant if the AlfaBot is implemented. For information about the implementation of the AlfaBot, see Implementing the AlfaBot for Navigation via a Full-Text Command.

- **Business Problem Statement**: Optionally enter a description that will help users to search for the configured report in the AlfaBot. The text is not displayed in the user interface, but it is of special relevance for the search mechanisms implemented for the AlfaBot. If the report caption entered by the user in the AlfaBot for a request to open a configured report does not match the caption of one of the available configured reports, the AlfaBot will split the caption entered by the user into keywords and will perform a keyword search on the **Caption**, **Description** and the **Business Problem Statement** attributes of the configured reports that are accessible via the AlfaBot. In addition to a keyword search, a synonym search is performed on the text provided in the **Business Problem Statement** attribute.

  This attribute is only relevant if the AlfaBot is implemented. For information about the implementation of the AlfaBot, see Implementing the AlfaBot for Navigation via a Full-Text Command.

- **Base Classes**: Define the object classes that must be available to run the configured report. If this attribute is set, the Alfabet Server checks whether the object classes specified as base classes are excluded from the view scheme used to access the configured report. If the user does not have access permissions to one of the base classes, the configured report will be disabled. To set the attribute, click the **Browse** button to select the object classes that are required to run the configured report and click **OK**.

- **Can Create Express View**: Select `True` if an express view may be created for the report. Select `False` if an express view may not be created for the report. Please note that for reports that have a custom report view, the setting must be consistent with the setting of the attribute **Can Create Express View** of the custom report view.

  An express view is a link to a specific Alfabet view displaying data that allows individuals inside or outside of the user community to view information in Alfabet. When the express view is created, an email notification is automatically mailed to a specified recipient. The recipient receives a URL that allows him/her to access the current page view in Alfabet. For more information, see the section *Sending and Receiving Express Views* in the reference manual *Getting Started with Alfabet*.

4) In the explorer, right-click the configured report and select **Create Configured Report View**. The view editor opens. In the explorer, the custom report view is displayed as a subordinate object of the configured report. The attributes for the custom report view are displayed in the attribute window. You can optionally edit the following attributes:

   - **Name**: Optionally you can change the name for the custom report view. The **Name** must be unique for custom report views.

   - **Can Create Bookmark**: Select `True` if a bookmark may be created for the configured report. Select `False` if a bookmark may not be created for the configured report.

   - **Can Create Express View**: Select `True` if an express view may be created for the configured report. Select `False` if an express view may not be created for the configured report. Please note that the setting must be consistent with the setting of the attribute **Can Create Express View** of the configured report.

     An express view is a link to a specific Alfabet view displaying data that allows individuals inside or outside of the user community to view Alfabet information. When the express view is created, an e-mail notification is automatically mailed to the specified recipient who receives a URL that allows him/her to access the current

page view in Alfabet. For more information, see *Sending and Receiving Express Views* in the reference manual *Getting Started with Alfabet*.

The custom report view initiates the **Report Assistant**. A custom report view shall never be deleted from the configured report after the **Report Assistant** was already used to configure the configured report. When you delete the custom report view of an existing template-based configured report and create a new one, the **Report Assistant** is reset to the example specified by default and your configuration is lost.

5) In the explorer, right-click the configured report and select **Start Alfabet Report Assistant**. The **Report Assistant** opens.

The **Report Assistant** is not available for HTML reports displaying indicator values in an HTML-based table. The configured report is defined in an XML object in the attributes of the configured report. For more information, see Defining HTML Reports. After defining the configured report in the XML object, proceed with step 7.

6) Define the configured report as described below in the sections for the configuration of the different configured report types. For general information about working with the **Report Assistant**, see Configuring the Report with the Report Assistant.

- Defining Branching Diagram Reports
- Defining Chart Reports
- Defining a Circular Roadmap Report
- Defining Reports to Analyse Data Cubes
- Defining a Gallery Report
- Defining Gantt Chart Reports
- Defining a Grid Report
- Defining Lane Reports
- Defining a Matrix Report
- Defining Node Arc Reports or Node Arc Reports With Edge Bundling
- Defining Portfolio Reports
- Defining Portfolio Diagnostics Reports
- Defining a Treemap Report
- Defining a Rectangular Treemap Report
- Defining a Layered Diagram Report
- Defining Widget Reports

7) Delete or edit the parameter panel for the configured report. For more information, see Defining Filters for Configured Reports and Selectors.

For most configured reports, the parameter panel only contains a **Submit** button. If you do not define own filters for the configured report, the filter panel can then be deleted. For configured reports based on the template `NodeArcReport`, filter fields for the

definition of a diagram layout are automatically added. If these filter fields are deleted, the default filter settings defined in the configured report are applied to the node arc report and the user cannot select a different layout. Customer configured filters can be added to the node arc report on demand in addition to the automatically created layout filters.

8) Click **OK** to save the configuration.

9) In the toolbar, click the **Save**  button to save your changes.

10) In the explorer, right-click the configured report and select **Review Report**. The configured report opens in a web browser. If the results are not as expected, you can edit the configured report until the output of the configured report meets your expectations.

Configuring the Report with the Report Assistant

The **Report Assistant** is available to help you to define a template-based report. The **Report Assistant** allows you to define the content of the report and to design the graphic display. Most graphic reports show object dependencies in a structured layout. .

The **Report Assistant** for these reports is similar and the general method to add and configure layers is described in this section. This knowledge is assumed in the sections providing information about the configurations required for each individual report. If the use of the **Report Assistant** deviates for a report type, this will be described separately in the section addressing the report's configuration.

The **Report Assistant** displays an explorer and an attribute window on the right of the explorer. The name of the explorer root node is identical to the name of the report you are currently designing. The attribute window displays the attributes of the selected element in the explorer. The attributes allow to define the look and content of elements of the report.

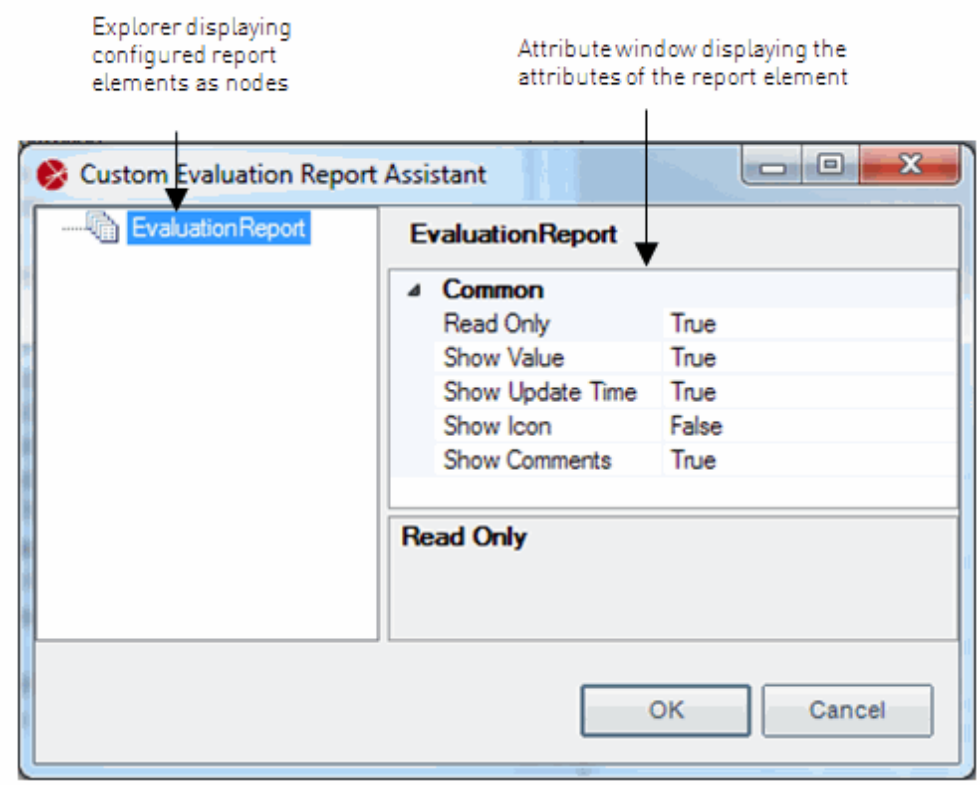


FIGURE: Interface of the Report Assistant for an evaluation report

Defining Elements of the Report

When you right-click a node in the explorer, the context menu of the selected element opens and allows you to add new elements as child elements of the current element, if applicable.

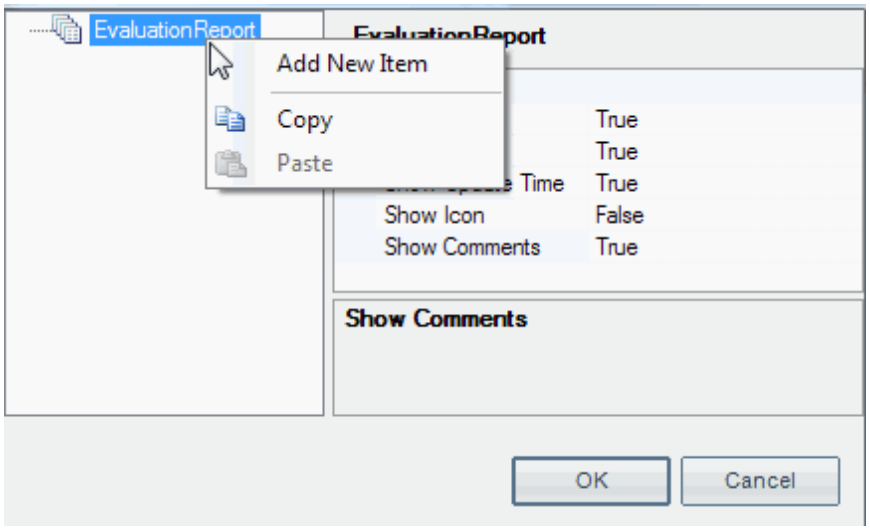


FIGURE: The context menu of the explorer root node of an evaluation report

The context menu also offers a copy and paste functionality to support the re-use of elements within a report or between reports.

If you copy an element, the element, its attributes and all child elements will be copied. You can paste the copied element into another element of the same report or a parallel report using the paste functionality.

Configuring the Attributes of Report Elements

The attribute window of the **Report Assistant** allows you to define the layout of the report. When you click an element node in the explorer, the attribute window of the element opens and displays the default settings for the configuration. Depending on the attribute you want to edit, you can either directly overwrite the default setting with a string, select a new value from a drop-down menu, or open a selector or text editor to define your settings. If a **Browse** button is displayed when you click the value field of an attribute, you cannot change settings directly within the value field but instead must open the text editor or selector via the **Browse** button to edit the attribute.

Defining Color Attributes

Colors can be defined either by writing the RGB or hexadecimal color code into the attribute or by using a color selector.

To define the color directly in the field, delete the text for the default color definition and enter the color either as hexadecimal code (For example, `#336699`) or as RGB values (For example, `52,104,168`). Click into another attribute field to submit the value. Hexadecimal values will be changed to the corresponding RGB values automatically and the colored field on the left of the attribute will show the new color.

When you click the **Drop-Down** button in an attribute field in order to define a color, a color selector opens that allows you to select colors from a **Custom**, **Web** or **System** palette.

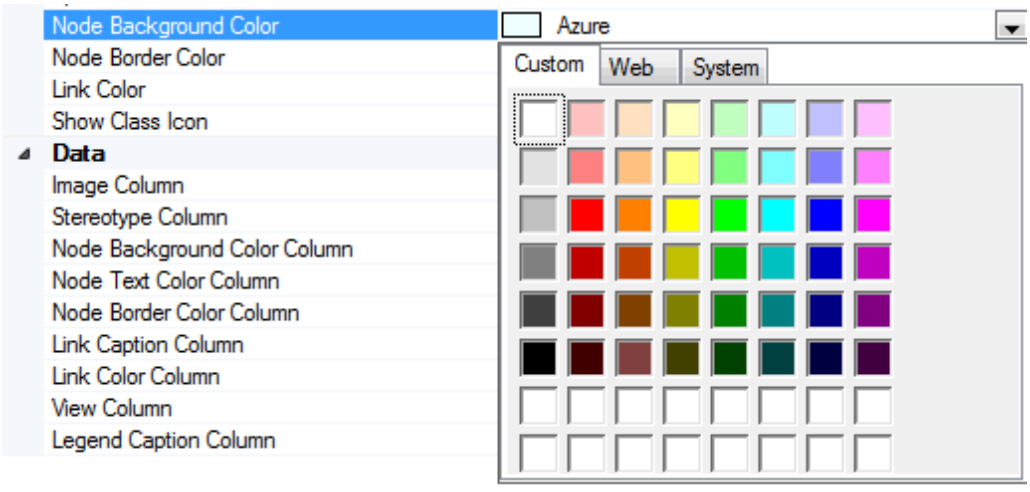


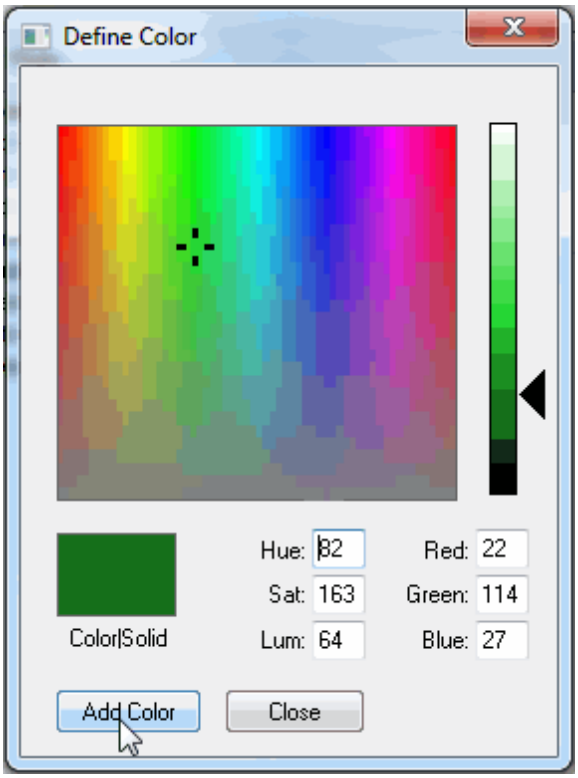
FIGURE: The color selector of the Report Assistant

You can either click a color in the palette in one of the tabs and click **OK** to select it, or you can add a new color to the **Custom** tab.

To add a new color to the **Custom** tab of the color selector:

1) In the **Custom** tab of the color selector, right-click one of the empty white fields. A new window opens.

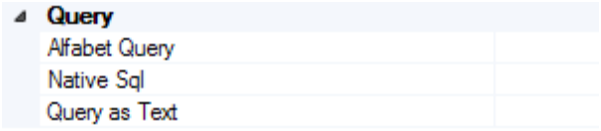
2) Select a color either from the palette fields in the upper part of the selector window or by defining the Red/Green/Blue and Hue/Saturation/Luminescence values for the color.



3) Click **Add Color** to add the new color to the palette. The color will be available in the **Custom** palette of all color selectors for all users working with the **Report Assistant** in Alfabet Expand.

Defining Query Attributes

If a query needs to be specified for a report element, the attribute window will display a separate section labeled **Query** with different attributes for the specification of a native SQL query or Alfabet query:



To define an Alfabet query in the **Alfabet Query Builder**, click the **Browse** button in the **Alfabet Query** field to select a `FIND` class and open the **Alfabet Query Builder**.

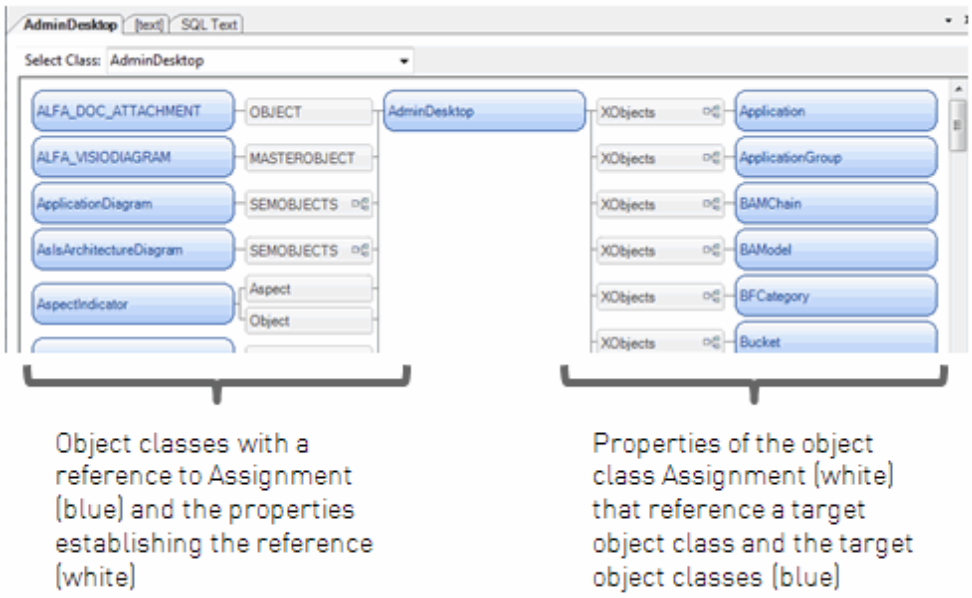
To define an Alfabet query in a text editor, click the the **Browse** button in the **Query as Text** field to open a text editor.

To define a native SQL query, click the **Browse** button in the **Native Sql** field to open a text editor for the definition of native SQL. The text editor has a separate tab for the definition of instructions. The text editor for the definition of SQL queries offers help for the definition of the SQL query in separate tabs:

- References to other object classes:

When you first open the text editor, the **AdminDesktop** tab is displayed. The object class `AdminDesktop` is the first class in alphabetical order in the Alfabet meta-model. In the **Select Class** field, you can select the object class whose references to other object classes that you want to see. A new tab with the name of the selected object class will open that displays the relation of the selected object class to other object classes in a graphic.

The selected object class is displayed in the upper middle of the graphic. On the left, all object classes referencing the selected object class via one or multiple of their object class properties are listed in blue boxes. The object class property that establishes the reference is displayed in a white box between the classes. On the right, all object class properties of the selected object class that store references to other object classes are listed. The target classes are listed in the blue boxes on the right.



If you double-click a referenced object class in the graphic, a new tab will open that displays the references of this class to other object classes.

- Object class documentation

When you open the **[text]** tab and select an object class in the **Select Class** field, the help document provided for the object class is displayed. The documentation includes a description of the purpose of the object class and all the object class properties of the object class and lists all relevant attributes in a table. Additionally, information is provided about class dependencies based on database triggers. All rules that define which dependent objects are deleted when an object of the selected class is deleted and vice versa are described in the documentation.

Once you have defined either a native SQL query or an Alfabet query, the attribute fields that are not used will no longer be displayed.

For information about defining Alfabet queries, see the chapter Defining Queries.

For information about the rules that apply to the definition of native SQL queries when used in Alfabet configurations, see Defining Native SQL Queries in the chapter Defining Queries.

Note the following when defining the query:

- If the configured report is applied to an object class with the attribute **Apply to Class**, you must configure the query to show results for the current object that is selected by the user when opening the configured report only. To refer to the current object, use the Alfabet parameter "`BASE`". For example, a configured report with the base class `Application` with a `WHERE` clause "`WHERE Application.REFSTR=@BASE`" will show results for the application that the user is currently working with only. A selector is added automatically to each configured report that is applied to an object class by means of the attribute **Apply to Class** . The selector allows the user to display other base objects in the configured report. If you do not specify a `WHERE` clause with the Alfabet parameter `BASE`, the selector will still be present, but it will not have any impact on the resulting configured report.
- You can define filters for a configured report to allow the users to specify the range of results in the configured report. You can generate a filter from any `WHERE` clause of the configured report by defining the value of the `WHERE` condition as Alfabet parameter. For more information about configuring filters, see Defining Filters for Configured Reports and Selectors in the chapter Defining Queries.
- You can combine the native SQL query as well as Alfabet query with Alfabet instructions written in Alfabet query language. For native SQL, the Alfabet instructions are defined separately in the tab **Instructions**. Alfabet instructions allow you For example, to define the output format for date and time information or to rename columns. For information about configuring Alfabet instructions, see Using Instructions to Format the Results of an Alfabet or Native SQL Query in the chapter Defining Queries .

  To check whether the syntax and the use of Alfabet parameters is correct in the native SQL query, click **Check**. The **Check** button checks the query for Alfabet conformance. Database-vendor specific SQL constructs, requirements, or dialects are not taken into account for this check.

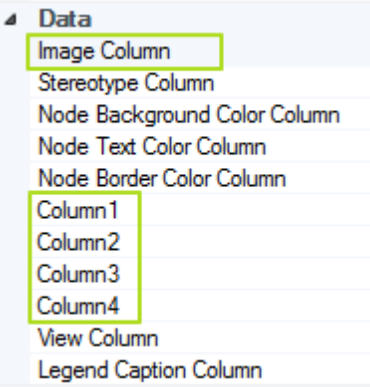
Mapping of Query Data to Attributes

Many of the data that is displayed in graphic reports is taken over directly from the result dataset of the query defined for the configured report or for a specific part of the configured report. For example, the definition of a tooltip that is displayed in a graphic report when the user moves the mouse over a box representing an object can be defined per object in the query results. With this mechanism it is possible to display individual tooltips for each object box.

The column name of the result dataset column containing the data must be defined in an attribute of the configured report in the report assistant to enable the use of the data for building of the graphic. Attributes that refer to data in the result dataset are displayed in the section **Data** of the root node of the report assistant or of a subnode thereof, depending on the report type.

For detailed information about the way column names are defined in Alfabet queries and native SQL queries, see Defining Column Names and Captions.

For example, for configured gallery reports, the central text displayed in the graphic for an object is defined with an attribute **Image Column** in the section **Data** of the attributes of the root node of the report while text displayed in the corners of the object boxes can be defined with the attributes **Column 1** - **Column 4**:



In a configured gallery report, applications in a selected application group shall be displayed together with the start date, end date, release status and object state of the application. In the middle of the object box, the name and version number of the application shall be displayed. All information about the applications is added to the Alfabet query defined for the configured gallery report:

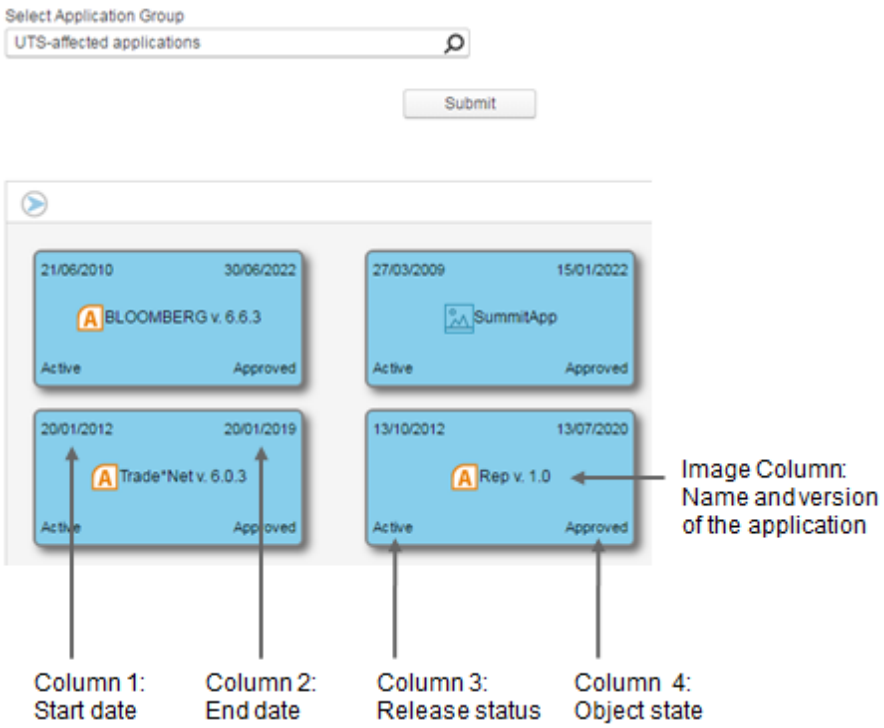
```
ALFABET_QUERY_500

FIND Application

WHERE Application.ApplicationGroups CONTAINS:BASE

Instructions

   JoinColumns("Application.Name,Application.Version","Application.N
   ame", " v. ");

EndOfInstructions

QUERY_XML

<QueryDef>

   <ShowProperty Type="Property" ClassName="Application" Name="Name"
   />

   <ShowProperty Type="Property" ClassName="Application"
   Name="Version" />

   <ShowProperty Type="Property" ClassName="Application"
   Name="StartDate" />

   <ShowProperty Type="Property" ClassName="Application"
   Name="EndDate" />

   <ShowProperty Type="Property" ClassName="Application"
   Name="ObjectState" />

   <ShowProperty Type="Property" ClassName="Application"
   Name="Status" />

</QueryDef>
```

Name and version of the applications shall be displayed as text in the gallery report. Therefore, a `JoinColumns` instruction was used to write the information about the name and version number of the applications into one column that can than be referenced in the attributes of the root node of the configured report.

In the attribute section **Data** of the root node of the report, the names of the columns in the query are filled into the respective attributes:

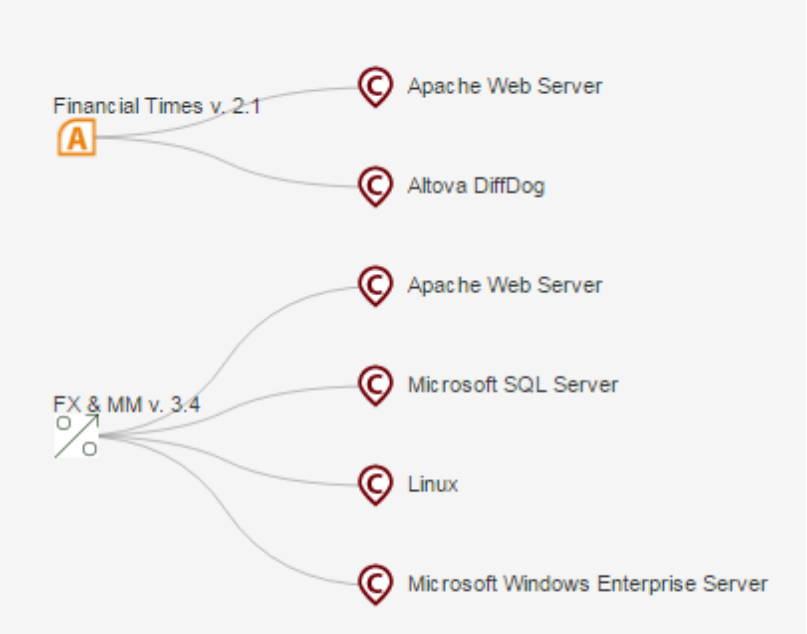
Data	
Image Column	Application.Name
Stereotype Column	
Node Background Color Column	
Node Text Color Column	
Node Border Color Column	
Column1	Application.StartDate
Column2	Application.EndDate
Column3	Application.Status
Column4	Application.ObjectState
View Column	

As a result, the report displays the object boxes with the defined data:



In some reports, multiple queries can be defined but only the root node includes the mapping in the attribute section **Data**. Other reports only include one query but the query defines a grouped dataset with multiple levels of objects that shall all refer to the same attribute of the section **Data**. In these cases the columns that define the same property, For example, the background color of object boxes or the text to be displayed in the object boxes must have the same name. In grouped dataset this requires the definition of a grouped dataset joining information about objects from different levels in a single query.

For example, a configured branching diagram report displays applications and local components thereof and the name of the applications and local components shall be displayed on the nodes of the report:

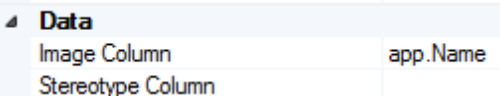


The text to be displayed on the nodes of the configured report is defined with one single attribute in the section **Data** of the root node attributes: **Image Column**. Therefore, only one column name can be defined for display of the text. The query the configured report is based on must not only group the dataset to display application on the first and local component on the second level, but it must also contain a `JoinColumns` instruction that joins the columns into one single column.

In the example this is the column `app.Name`. First the information about the application name and application version is joined in this column to display all required information about the application. In a second step, the information to display for applications is joined with the information to display about the local components:

```
SELECT app.REFSTR, app.REFSTR AS 'app.REFSTR', app.NAME AS
'app.Name', app.VERSION AS 'app.Version', lcom.REFSTR AS
'lcom.REFSTR', lcom.NAME AS 'lcom.Name'

FROM APPLICATION app, LOCALCOMPONENT lcom

WHERE app.REFSTR = lcom.OWNER

   AND app.NAME LIKE 'F%'

/* Alfabet Instructions */

GroupBy_Ex("app.REFSTR","lcom.REFSTR","app",0);

JoinColumns("app.Name,app.Version","app.Name", " v. ");

JoinColumns("app.Name,lcom.Name","app.Name", "");
```

The column `app.Name` is then defined in the **Image Column** attribute in the root node of the configured report:



A column in the result dataset can be defined in multiple attributes in the section **Data**. If the configured report has an attribute for the definition of a label text and one for the definition of a legend item text and you want the label text to be identical to the text that shall be displayed in the legend, you can enter the same column name in both attributes.

The design of the configured report via data from the result dataset of a query is limited for Alfabet queries. Alfabet queries can only return data from the Alfabet database. For example, if you want to include a color definition in a result dataset, the color definition can only be taken over from a color definition stored for an object in the Alfabet database. It is not possible to define the color as string directly in the Alfabet query.

Some features of configured reports are limited to configured reports defined with native SQL. This applies For example, to the definition of a link target to open an Alfabet view when the user clicks the object in the report. It is Therefore, recommended to define configured graphic reports via native SQL queries.

Defining Navigation from the Report to Alfabet Views

By default, the object profile of the object represented by the node opens when the user double-clicks a node or object box in the report or uses the **Navigate** button in the toolbar of the report. Alternatively, many configured graphic reports can be configured to open another configured report, an object profile or a standard Alfabet view with the object represented by the node or object box as base object. For configured reports with multiple levels of nodes or object boxes, you can define an alternative navigation target for all or only for a subset of levels in the configured report. If no navigation target is defined for one level, the default is used and the object profile of the object opens.

Navigation to a defined view from the configured report can only be configured via native SQL queries and requires the following configurations:

1) In the native SQL query defined for the report, a column must be added for the definition of the navigation target. If the configured report is based on a grouped dataset, and you want to define the navigation target for all grouping levels of the report, you must specify one column for each level, containing the navigation target definition for the objects of the respective level, and then combine the columns into one column with a `JoinColumns` instruction.

   The syntax required for definition of the view is the following:

   `View=`*`ViewType`*`:`*`ViewName`*

   The `ViewType` can be one of the following:

   - ***Report*** for a configured report
   - ***GraphicView*** for a standard Alfabet view
   - ***ObjectView*** for an object profile

   `ViewName` is the name of the view.

   If the navigation target is a configured report, parameters values can be handed over to the view that opens.

   The query of the report must contain each parameter definition as a variable defined as

   `@`*`ParameterName`*

   Please note that the old syntax of the Alfabet query language defining parameters as:`ParameterName` is not supported.

   The parameters can be used in `WHERE` clauses of the query without defining a filter for the report, because the parameter values are already defined in the navigation link.

Optionally, filters can be defined for the report to allow the user opening the report to alter the parameter values. In this case the parameter values in the navigation link are used as default values for the filter when opening the report. The filter settings are also stored in the user context settings for the report. That means, when the user first opens the report via drilldown from a chart report and then afterwards opens the report in any other context, For example, via the **Reports** functionality, the report will open with the last filter settings derived from the drill-down.

For each parameter the following string must be added to the navigation target definition:

```
&ParameterName=ParameterValue
```

The setting of three parameters would result in the following navigation link:

```
View=ViewType:ViewName&ParameterName1=ParameterValue1&ParameterName2=Para
meterValue2&ParameterName1=ParameterValue2
```

2) In the attributes of the root node of the report assistant, enter the name of the column containing the navigation target specification into the attribute defined to store the navigation target view definition.

For all configured reports that allow specification of the navigation target via a query, a base object for the view that opens has to be specified. For some configured reports the base object for the link is automatically identical with the object represented by the graphic element containing the link. For other configured reports, the navigation target object must be defined in a separate column of the configured report. If the navigation target is defined separately, or the graphic element is not representing a single object anyway, there are two ways to define the navigation target object:

- To define the link to open a configured report independent from a base object via the following specification of the link target:
  - Configured reports are objects of the object class `ALFA_REPORT`. Define the configured report's REFSTR as REFSTR of the navigation target object.
  - Define the navigation link to open the report defined as navigation target object.
  - If the titel of the configured report shows the name instead of the caption of the configured report when opening the configured report via the link, define class settings for the class `ALFA_REPORT` with a format string returning the caption. For more information about defining class settings, see Configuring Class Settings for Object Classes and Object Class Stereotypes.
- To define the link to open for multiple objects of the same object class:
- Define the object of the object class `ALFA_MM_CLASS_INFO` representing the object class the objects belong to as navigation target object. If the graphic element containing the link is styled via the class settings of an object class, the class settings of the object class the `ALFA_MM_CLASS_INFO` object represents will be used for styling of the graphic element.

Defining Rules for Display of Objects In Reports Dependent on Object Property Values

To enhance the visibility of important aspects for the objects displayed in a report, customer graphic reports for Alfabet can be configured to change the size, color or icons dependent on the value of a defined object property.

In many reports this can be achived by mapping data from the result dataset of the query finding the objects to specific attributes in the report assistant. For more information see Mapping of Query Data to Attributes.

For other reports, rules can be defined that find objects with a defined property value or property value range via a query and apply the formatting defined in the rule to the object. For example, objects with a critical value for an indicator can be marked in red while all other objects can be displayed in green.

The following design can be configured to be dependent on object property values via rules:

Rule	Report Type	Description
Color Rules	Treemap Report Layered Diagram Grid Report Matrix Map Lane Report Gantt Charts	Color rules allow colors to be assigned to object boxes in a report dependent on the object class or on the availability or value of a property of the object class. For more information, see Defining Color Rules.
Indicator Rule	Treemap Report Rectangular Treemap Report Layered Diagram Grid Report Matrix Map Lane Report Dynamic Lane Report Node Arc Reports	Indicator rules allow an indicator icon to be displayed in the object boxes of all objects specified in the indicator rule. For more information, see Defining Indicator Rules.
Size Rule	Treemap Report Layered Diagram	Size rules can be defined to display the object boxes in a report with different heights dependent on an object property values. For more information, see Displaying Evaluation Criteria By Size or Color Variation or Addition of Indicator Icons in the description of how to define a treemap report.

Defining Color Rules

The colors of graphic reports are normally specified per object class. Either the colors defined in the **Background Color** and **Foreground Color** attributes for the relevant class setting are used or the color of an object class can be defined in the attributes of the element nodes defined for the report.

For a subset of reports, coloring can be defined via color rules. Color rules allow coloring to be defined not only per object class, but also on the basis of a query that specifies a subset of objects of a class the color shall be applied to.

You can specify various color rules for a report that allow you to display objects with different attribute values in different colors. For example, you can display objects that are in the object state **Plan** in one color and objects in the object state **Active** in another color.

Color rules are available for the following reports:

- TreeMap Reports
- Layered Diagram Reports
- Grid Reports
- Matrix Maps
- Lane Reports

Color rules are valid for all objects found by the query defined for the color rule independent of the level configuration of the report. For example, if you define a layered diagram report that displays domains in the first layer and sub-domains in the second layer, you can define a color rule that colors all domains in all layers that have a defined indicator value below a specified value.

For tree maps, layered diagrams, lane reports and matrix maps, color rules are defined as sub-nodes of the report's root node. For grid reports, color rules are defined as sub-nodes of the **Cell** nodes of the report. Different coloring can be applied to all cells of the report.

The specification of color rules is optional. If no color rules are specified and no other attributes of the report allow definition of a default color, all objects will be displayed in the background and foreground color specified in the class settings defined for the respective object class and the border color defined in the attributes of the element nodes defined for the report.

Depending on the values you want to analyze, you can define as many color rules as required. When defining the color rules, you must ensure that each object can be explicitly assigned to one of the color rules. Objects that cannot be assigned to any of the color rules are displayed in the default color. Objects that can be assigned to more than one color rule are displayed in the color of the last rule they apply to.

You must take this behaviour into account if you want to use For example, the foreground color and the border color to highlight different object properties in a report. this scenario requires to define a different rule for each combination of property values.

A report shall highlight an indicator value of the object via the foreground color while the border color changes according to the release status of the objects. The objects can be in the release status **Plan**, **Active**, or **Retired**. The indicator value can be either low or high. Therefore, 6 color rules must be defined, one for each combination of release status and indicator value. The rules must find objects matching the following conditions and apply the appropriate combination of background color and border color:

Rule Number	Indicator Value	Release Status	Coloring
1	high	Plan	

2	high	Activ	
3	high	Retired	
4	low	Plan	
5	low	Active	
6	low	Retired	

To change the default color in a report, two different types of color rules can be defined:

- `ObjectQuery`: The query defined for the color rule returns the `REFSTR` of the object that the color shall be applied to. The colors that shall be applied to the object boxes are defined separately via attributes of the color rule.

  When defining an Alfabet query, no Show property definition is required. The `REFSTR` of the `FIND` object class is returned.

  When defining a native SQL query, the `SELECT` statement of the query must have the `REFSTR` of the relevant object class defined as the first argument. No further arguments are required in the `SELECT` statement.

- `ColorQuery`: The query defined for the color rule not only returns the `REFSTR` of the objects that the color shall be applied to but also the HTML-compliant specification of the colors to be applied to the object boxes.

  Color queries require the specification of a native SQL query.

  The `SELECT` statement of the SQL query must return the following in the given order:

  - the `REFSTR` of the current object
  - a string that is used as the caption for the color assignment in the legend of the report
  - a string that defines the background color of the object boxes of objects matching the query conditions in HTML-compliant color coding
  - a string that defines the foreground color of the object boxes of objects matching the query conditions in HTML-compliant color coding

- a string that defines the border color of the object boxes of objects matching the query conditions in HTML-compiant color coding

For example,:

```
SELECT DOMAIN.REFSTR, "Legend text", "#336699", "white",
"black"
```

The definition of a color query in a color rule has the following advantages as compared to an object query:

- The number of overall color rules required for a report can be reduced. For example, if you want to apply a different color to an application in a different object state, you can define the color coding for all object states in one query while a separate color rule for each object state would be required when using object queries.
- Dynamic color assignment can be applied to a report. You can, For example, define a query that colors object boxes according to the indicator values assigned to the objects. In the report, you can define a filter that allows the user viewing the report to choose an indicator type. The coloring of object boxes is then reassigned according to the values for the selected indicator (see the example below).

For example, in a report displaying applications a filter is defined that allows the user to select an indicator type for which the indicator values shall be used for coloring the object boxes of the applications in the report. Indicators are selected in a filter field named `@AppColorType`. The query defined for the color rule finds the indicator for the selected object that is based on the selected indicator type and assigns different colors for each value that may be defined for the indicator. The values for all indicators are in the range of 0 - 4.

```
SELECT obj.REFSTR,'Very Low','#80ff80','Black'
FROM APPLICATION obj, INDICATOR ind
WHERE (ind.INDICATORTYPE = @AppColorType OR 1 = 2)
AND ind.OBJECT = obj.REFSTR
AND ind.VALUE = 0
UNION ALL
SELECT obj.REFSTR,'Low','#b0ff80','Black'
FROM APPLICATION obj, INDICATOR ind
WHERE (ind.INDICATORTYPE = @AppColorType OR 1 = 2)
AND ind.OBJECT = obj.REFSTR
AND ind.VALUE = 1
UNION ALL
SELECT obj.REFSTR,'Moderate','#ffff80','Black', 'Black'
FROM APPLICATION obj, INDICATOR ind
WHERE (ind.INDICATORTYPE = @AppColorType OR 1 = 2)
AND ind.OBJECT = obj.REFSTR
```



```
AND ind.VALUE = 2
UNION ALL
SELECT obj.REFSTR,'High','#ffb080','Black', 'Black'
FROM APPLICATION obj, INDICATOR ind
WHERE (ind.INDICATORTYPE = @AppColorType OR 1 = 2)
AND ind.OBJECT = obj.REFSTR
AND ind.VALUE = 3
UNION ALL
SELECT obj.REFSTR,'Very High','#ff8080','Black', 'Black'
FROM APPLICATION obj, INDICATOR ind
WHERE (ind.INDICATORTYPE = @AppColorType OR 1 = 2)
AND ind.OBJECT = obj.REFSTR
AND ind.VALUE = 4
```

The condition for the indicator type in the query above takes into account that a user may submit the report without selecting an indicator type. In this case, the condition will be deleted from the query before execution. This would result in a query applying colors according to the first setting of an indicator of any type for the application because only the indicator object and value are then defined in the `WHERE` clause. To deactivate the color rule when no indicator type is defined, a condition that cannot be fulfilled (like 1 = 2) is added as alternative text if no indicator type is defined.

If a legend is displayed for the report, all color rules of the type `ObjectQuery` form one legend group with the caption **Colors** . Each color rule of the type `ColorQuery` forms a separate legend group with the **Name** of the color rule displayed as a caption of the legend group.

The legend lists all coloring options defined as small boxes with a background color and border color as defined in the color rule and the character "x" displayed in the box in the color defined as foreground color.

The table below lists all available attributes in the **Color Rule** element.

Attribute	Description
Name	Defines the title displayed for the color in the legend. The **Name** is also displayed as the name of the color rule node in the explorer of the **Report Assistant**. **NOTE**: The attribute **Show Legend** of the report's root node element must be set to `True` to display the legend.
Type	Select `ObjectQuery` to define the objects that the color rule shall apply to in either a native SQL query or Alfabet query and define the colors assigned to the objects with the attributes **Background Color** and **Foreground Color** of the color rule. Select `ColorQuery` to define both the objects and the colors assigned to the object in a native SQL query.

Attribute	Description
Background Color	Only if the attribute **Type** is set to `ObjectQuery`: Defines the color of the boxes representing the objects in the report.
Foreground Color	Only if the attribute **Type** is set to `ObjectQuery`: Defines the color of the text in the boxes representing the objects in the report. **NOTE**: In color rules defining coloring of links in a lane report, the **Foreground Color** defines the color of the text in the link label.
Border Color	Only if the attribute **Type** is set to `ObjectQuery`: Defines the color of the border of the boxes representing the objects in the report.
Native Sql/ Alfabet Query/ Query as Text	Define the objects to which this color rule applies. A valid Alfabet query or native SQL query has to be written into the parameter that finds the relevant objects. If the **Type** of the color rule is `ObjectQuery`, define a native SQL query or Alfabet query that returns the `REFSTR` of the objects that the color rule shall apply to. If the **Type** of the color rule is `ColorQuery`, define a native SQL query that returns: • the `REFSTR` of the current object • a string that is used as the caption for the color assignment in the legend of the report • a string that defines the background color of the object boxes of objects matching the query conditions in HTML-compliant color coding • a string that defines the foreground color of the object boxes of objects matching the query conditions in HTML-compliant color coding
Legend Row Item Count	Defines the number of color rules displayed in one row of the legend of the report on export of the report via the **Export** button. This setting is only evaluated on export of the report. In the Alfabet user interface, the legend displays one entry per row independent of this setting. For color rules of the type `ObjectQuery`, only the value defined in the first color rule of the type `ObjectQuery` of the report is applied to the group. By default, 4 items are displayed in a row of the legend.

Defining Indicator Rules

You can define that an indicator should be displayed in the report for each object of an object class. This allows an additional dimension of evaluation for the objects in the report. Indicator rules add icons to the boxes representing the object. Multiple icons can be assigned to each object box via multiple different indicator rules, or single rules of a subtype that permit definition of multiple indicators.

You can define one of the following:

- Indicators defined for the object that are represented by an icon can be displayed to visualize the object evaluation. Multiple icons can be placed into one object box.

  Note the following:

  - An indicator can only be displayed in the report if the indicator type has been configured to display icons rather than numerical values. Indicator types are specified in the **Evaluations and Portfolios** functionality of Alfabet. For information about the configuration of indicator types and the icon gallery used to visualize their values, see the section *Configuring Indicator Types for an Evaluation Type* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.
  - The indicator icon is only displayed in the object boxes of objects that the indicator is set for.

- Icons can be assigned independent form indicator settings via a query. For example, the object boxes representing applications can contain an icon for each domain the application is assigned to or associated with. The user can navigate from the icon to the object view of the object represented by the icon. Multiple icons can be assigned to an object box.

You can specify various indicator rules for a report that allow you to display a different indicator per object class or for objects matching defined conditions. It is also possible to display multiple indicators per object.

Indicator rules are available for the following reports:

- TreeMap Reports
- Rectangular Treemap Reports
- Layered Diagram Reports
- Grid Reports
- Matrix Maps
- Lane Reports
- Dynamic Lane Reports
- Node Arc Reports (only indicator rules of the type `IconQuery` are supported)

Indicator rules are valid for all objects defined within the attributes of the indicator rule independent of the level of configuration of the report. For example, if you define a layered diagram report that displays domains in the first layer and sub-domains in the second layer, you can define an indicator rule that displays an indicator for all domains in all layers.

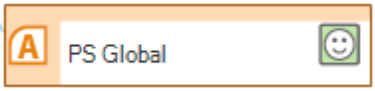
For tree maps, layered diagrams, node arc reports, lane reports, dynamic lane reports and matrix maps, indicator rules are defined as sub-nodes of the report's root node. For grid reports, indicator rules are defined as sub-nodes of the **Cell** nodes of the report. Different indicator rules can be applied to cells of the report.

The specification of indicator rules is optional. If no indicator rules are specified, all objects will be displayed without an indicator icon in the object box. Depending on the values you want to analyze, you can define as many indicator rules as required. The configured report can then be configured to display one or multiple icons per object box: Whether one or multiple indicators are displayed per object is defined with the

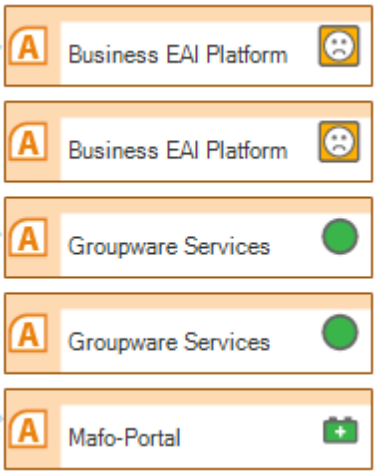
attribute **Show Multiple Indicators** of the configured report. The attribute is available in the root node of the configured reports displaying a treemap, layered diagram, rectangular treemap, matrix map, or lane report, in the **Item** element of grid reports, and in the **Node** element of lane reports.

- **Displaying one icon** (**Show Multiple Indicators** is set to `False`):

The icon is displayed on the right of the object box:



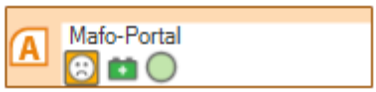
When defining the indicator rules, you must ensure that each object can be explicitly assigned to one of the indicator rules. Objects that cannot be assigned to any of the indicator rules are displayed without indicator icons. Objects that can be assigned to more than one indicator rule are displayed with the indicator icon of the last rule they apply to. If a single indicator rule is defined for the assignment of multiple icons, the last icon returned in the result dataset of the indicator rule query is displayed. For example, if three indicator rules all assign an icon to the object boxes of applications, and none of the indicators represented by the icon is set for all applications, the icon from the last indicator rule returning an indicator for that application is displayed for each application, returning a configured report:



This method is not available for node arc reports.

- **Displaying multipe icons** (**Show Multiple Indicators** is set to `True`):

The icons are displayed on the bottom of the white sector of the object box from left to right:

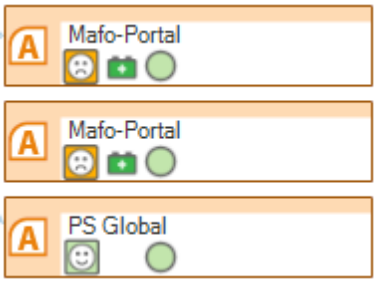


If the space is not sufficient to display all defined indicators, the last indicators will be dropped. The object box width is not changed if more indicators are displayed.

The position of the indicators depends on the setting of the attribute **Indicator Fill Policy**. The attribute is available in the root node of the configured reports displaying a treemap, layered diagram, rectangular treemap, matrix map, node arc or lane report, in the **Cell** element of grid reports, and in the **Node** element of lane reports. The **Indicator Fill Policy** can be one of the following:

- `ByIndex`: This setting shall be used if multiple indicators are defined via multiple indicator rules, each returning a single icon. The position of the icon resulting from each indicator rule

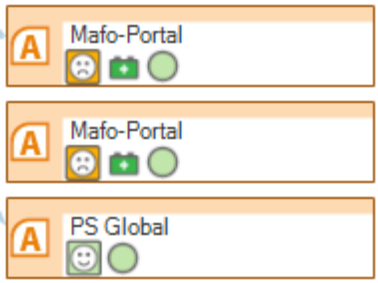
is defined by means of the attribute **Indicator Index** of the respective indicator rule that must be set to an unsigned integer starting with 0 for the left most position. If an **Indicator Index** attribute has been defined for each indicator rule and an indicator is not set for a specific object, the position of this indicator icon in the box will be left empty and the position of all other indicator icons will not be changed:



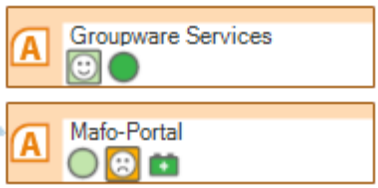
If the order of the **Indicator Index** attribute is not defined, the placement of indicators will be arbitrary.

This method is designed for positioning only one icon per indicator rule. If an indicator rule returns multiple icons, only one of the icons is displayed.

- `LeftAlign`: This setting shall be used if a single indicator rule is defined that returns multiple icons. It can also be applied when multiple indicator rules each return a single icon, but the position and order of icons is not important to understand the report. The Icons are displayed in the order of occurence within the indicator rule(s) from left to right. If a different number of indicator icons is returned, the remaining icons are moved to the left:



If the indicator icons are all found by a single indicator rule, the order of icons depends on the order of results in the result dataset of the query of the indicator rule and may be different for different objects:



To display indicators in object boxes of a report, you can define four different types of indicator rules:

- `ClassName`: This indicator rule type returns a single icon representing an indicator. An indicator icon is assigned to all objects of a defined object class for that the indicator is set. All indicator icons represent the same indicator type.
- `ObjectQuery`: This indicator rule type returns a single icon representing an indicator. An indicator icon is assigned to all objects found via a query. All indicator icons represent the same indicator type.

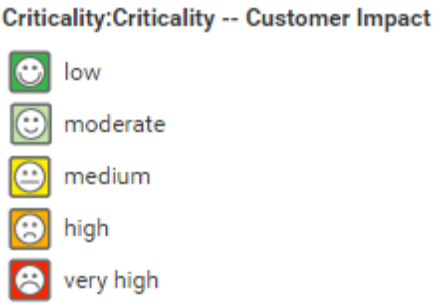
- `TypeQuery`: This indicator rule type returns one or multiple icons representing indicators. One or multiple indicator icons are assigned to all objects found via a query. The indicator icons can represent different indicator types.

  The definition of a `TypeQuery` in an indicator rule has the following advantages as compared to an object query:

  - The number of overall indicator rules required for a report can be reduced. For example, if you want to display a different indicator for objects in a different object state, you can define the indicators for all object states in one query while a separate indicator rule for each object state would be required when using object queries.
  - Dynamic indicator assignment can be applied to a report. For example, you can define a filter for the report that allows the user viewing the report to choose an indicator type. The indicator icon of the selected value is then displayed in the report (see the example below).

- `IconQuery`: This indicator rule type returns one or multiple icons that represent information not related to an indicator. One or multiple icons are assigned to all objects found via a query. The icons can For example, represent related objecs or highlight the setting of a property of the object, like the release status or object state. Navigation to any Alfabet view related to the information can be activated.

If icons are added to object boxes, the icons are explained in a legend. For icons found via an indicator rule of the type `IconQuery` or `TypeQuery`, the legend header and legend texts for the icons are defined via the query. For icons representing a fixed indicator via an icon rule of the type `ClassName` or `ObjectQuery`, the heading of the legend group displaying the indicator icons returns the name of the evaluation type and indicator type of the indicator by default. If a legend caption is defined in the indicator rule, the default caption is substituted with the defined caption. For each icon, the semantic value of the indicator is displayed:



The following sections inform about the attribute settings in the indicator rule node for the different types of indicator rules:

- Creating an Indicator Rule of the Type ClassName
- Creating an Indicator Rule of the Type ObjectQuery
- Creating an Indicator Rule of the Type TypeQuery
- Creating an Indicator Rule of the Type IconQuery

Creating an Indicator Rule of the Type ClassName

This indicator rule type assignes an indicator icon for a defined indicator type and evaluation type to all objects of a defined class.

Set the following attributes of the indicator rule node:

Attribute	Description
Name	Defines the name of the indicator rule node in the explorer of the **Report Assistant**.
Type	Select `ClassName`.
Class Name	Defines the object class for which the indicator shall be displayed. For the object classes that allow stereotype specifications (For example, `Project` and `Domain`), the XML attribute `ClassName` can specify an object of a specific stereotype. To define an XML element ***Query*** for a stereotype, the XML attribute `ClassName` must be *`ClassName`*`:`*`StereotypeName`* For example,: *`Project`*`:`*`StatementOfWork`*
Evaluation Type	Defines the name of the evaluation type for which the indicator type is specified. **NOTE:** The evaluation type must be assigned to the object class specified with the attribute **ClassName**. This is done in the **Class Configuration** functionality in the **Configuration** module. For more information, see the section *Assigning Evaluations Types to an Object Class* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.
Indicator Type	Defines the name of the indicator type that shall be displayed in the report. **Note**: The indicator type must be configured to use an icon gallery to display its value in evaluations. This is done in the **Evaluations and Portfolios** functionality in Alfabet. For more information, see the section *Configuring Indicator Types for an Evaluation Type* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.
Legend Row Item Count	Defines the number of indicator values displayed in one row of the legend of the report. By default, 4 items are displayed in a row of the legend.
Indicator Index	If multiple indicator rules are defined, and the attribute **Indicator Fill Policy** of the configured report is set to `ByIndex`, define the position of the icon displayed for the current indicator rule. Enter 0 for the first position on the left and ascending integers for the next positions from left to right.

Attribute	Description
Legend Caption	Defines a heading for the legend group that explains the indicator icons for the indicator found by this indicator rule. If no Legend Caption is defined, the name of the evaluation type and indicator type of the indicator are displayed as heading of hte legend group.

Creating an Indicator Rule of the Type ObjectQuery

This indicator rule type assignes an indicator icon for a defined indicator type and evaluation type to a sub-set of objects of a defined class. The objects are found by a query.

Set the following attributes of the indicator rule node:

Attribute	Description
Name	Defines the name of the indicator rule node in the explorer of the **Report Assistant**.
Type	Select `ObjectQuery`.
Class Name	Defines the object class for which the indicator shall be displayed. The query defined in the attributes **Native Sql** / **Alfabet Query** / **Query as Text** must be defined to return objects of the specified class only. For the object classes that allow stereotype specifications (For example, `Project` and `Domain`), the XML attribute `ClassName` can specify an object of a specific stereotype. To define an XML element ***Query*** for a stereotype, the XML attribute `ClassName` must be *`ClassName:StereotypeName`* For example,: *`Project:StatementOfWork`*
Evaluation Type	Defines the name of the evaluation type for which the indicator type is specified. **NOTE:** The evaluation type must be assigned to the object class specified with the attribute **ClassName**. This is done in the **Class Configuration** functionality in the **Configuration** module. For more information, see the section *Assigning Evaluations Types to an Object Class* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.
Indicator Type	Defines the name of the indicator type that shall be displayed in the report. **Note**: The indicator type must be configured to use an icon gallery to display its value in evaluations. This is done in the **Evaluations and Portfolios** functionality in Alfabet. For

Attribute	Description
	more information, see the section *Configuring Indicator Types for an Evaluation Type* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.
Native Sql / Alfabet Query / Query as Text	Defines the objects to which this indicator rule applies. A valid Alfabet query or native SQL query has to be written in the parameter that finds the relevant objects. Define a native SQL query or Alfabet query that returns the `REFSTR` of the objects that the indicator rule shall apply to. Objects must belong to the object class defined with the attribute **Class Name**. When defining an Alfabet query, no Show property definition is required. The `REFSTR` of the `FIND` object class is returned. When defining a native SQL query, the `SELECT` statement of the query must have the `REFSTR` of the relevant object class defined as first argument. No further arguments are required in the `SELECT` statement.
Legend Row Item Count	Defines the number of indicator values displayed in one row of the legend of the report. By default, 4 items are displayed in a row of the legend.
Indicator Index	If multiple indicator rules are defined, and the attribute **Indicator Fill Policy** of the configured report is set to `ByIndex`, define the position of the icon displayed for the current indicator rule. Enter 0 for the first position on the left and ascending integers for the next positions from left to right.
Legend Caption	Defines a heading for the legend group that explains the indicator icons for the indicator found by this indicator rule. If no Legend Caption is defined, the name of the evaluation type and indicator type of the indicator are displayed as heading of hte legend group.

Creating an Indicator Rule of the Type TypeQuery

The query defined for the indicator rule not only returns the `REFSTR` of the objects that the indicator shall be displayed for but also a reference to the indicator.

Set the following attributes of the indicator rule node:

Attribute	Description
Name	Defines the name of the indicator rule node in the explorer of the **Report Assistant**.
Type	Select `TypeQuery`.

Attribute	Description
Native Sql/ Alfabet Query/ Query as Text	Defines the objects to which this indicator rule applies. A valid Alfabet query or native SQL query has to be written in the parameter that finds the relevant objects. Define a native SQL query or Alfabet query. When defining an Alfabet query, the `REFSTR` of the `FIND` object class is returned to find the relevant objects. The **Show Properties** must return the following: • the `REFSTR` property of the indicator type of the indicator that shall be displayed. • the `Value` property of the indicator. This specification is optional. When defining a native SQL query, the `SELECT` statement of the SQL query must return the following: • the `REFSTR` of the current object (as first SELECT argument not visible in the result dataset when displayed as table) • the `REFSTR` property of the indicator type of the indicator that shall be displayed. • the `Value` property of the indicator. This specification is optional. The return dataset must either return the values listed above as first columns in the result dataset in the order specified above, or, if the position of columns is changed, the name of the columns defining the `REFSTR` of the indicator type and the value of the indicator must be entered into the attributes **Indicator Reference Column** and **Indicator Value Column**.
Legend Caption Column	Enter the name of the column in the query result dataset returning a string to be used as headline of the legend group for the indicators icons in the legend of the configured report. If not defined, the name of the evaluation type and indicator type of the indicator are used as group caption. Please note that the number of groups is identical to the number of indicators and not derived from the text returned via the dataset. If For example, the same text is returned for each indicator, the icons are still listed with one group per indicator with all groups returning the same name.
Indicator Reference Column	Enter the name of the column in the query results returning the `REFSTR` property of the indicator type of the indicator that shall be displayed. This specification is optional. If the `REFSTR` property of the indicator type is returned as first column in the tabular result dataset of the query, the information is read from the query without defining the column in the **Indicator Reference Column**.
Indicator Value Column	Enter the name of the column in the query results returning the `Value` property of the indicator.

Attribute	Description
	This specification is optional. If the `Value` property of the indicator is returned as second column in the tabular result dataset of the query, the information is read from the query without defining the column in the **Indicator Value Column** attribute.
Legend Row Item Count	Defines the number of indicator values displayed in one row of the legend of the report. By default, 4 items are displayed in a row of the legend.
Indicator Index	If multiple indicator rules are defined, and the attribute **Indicator Fill Policy** of the configured report is set to `ByIndex`, define the position of the icon displayed for the current indicator rule. Enter 0 for the first position on the left and ascending integers for the next positions from left to right. Please note that positioning `ByIndex` only displays one icon per indicator rule. If you defined a query returning multiple icons per object box, the **Indicator Fill Policy** of the configured report should be set to `LeftAlign`. This attribute is then ignored for positioning.

For example, in a report displaying applications a filter is defined that allows the user to select an indicator type that the indicator icons shall be displayed for in the object boxes of the applications in the report. Indicators are selected in a filter field named:`AppIndicatorType`. The Alfabet query defined for the indicator rule finds the indicator for the selected object that is based on the indicator type selected in the filter.

```
ALFABET_QUERY_500

FIND

Application

INNERJOIN Indicator ON Indicator.Object=Application.REFSTR

INNERJOIN IndicatorType ON
Indicator.IndicatorType=IndicatorType.REFSTR

WHERE

    (OR IndicatorType.REFSTR =:AppIndicatorType

    IndicatorType.REFSTR = 'xxx')

QUERY_XML

<QueryDef>

    <ShowProperty Type="Property" ClassName="IndicatorType"
    Name="REFSTR" />

    <ShowProperty Type="Property" ClassName="Indicator" Name="Value"
    />

</QueryDef>
```

The condition for the indicator type in the query above takes into account that a user may submit the report without selecting an indicator type. In this case, the condition is deleted from the query before execution. This would result in a query applying indicator icons according to the

first setting of an indicator of any type for the application. To deactivate the display of indicator icons when no indicator type is defined, a condition that cannot be fulfilled (like `IndicatorType.REFSTR` = 'xxx') is added as an alternative text if no indicator type is defined.

Creating an Indicator Rule of the Type IconQuery

For indicator rules of the the type `IconQuery`, all relevant information has to be derived from a query.

Although it is possible to define an Alfabet query for an indicator rule of the type `IconQuery`, the possibilities to define the query in Alfabet query language are very limited. It is recommended to use native SQL to define the query.

The query must return at least the the name of an icon from the icon gallery and the REFSTR of the object represented by the object box the icon shall be placed in. Optionally, the query can also couple the icon to an object in the Alfabet database, which enables navigation from the icon to the object view of the assigned object or to any other view related to that object if the view is defined in the query as navigation target. Tooltip and legend texts are also defined via the query.

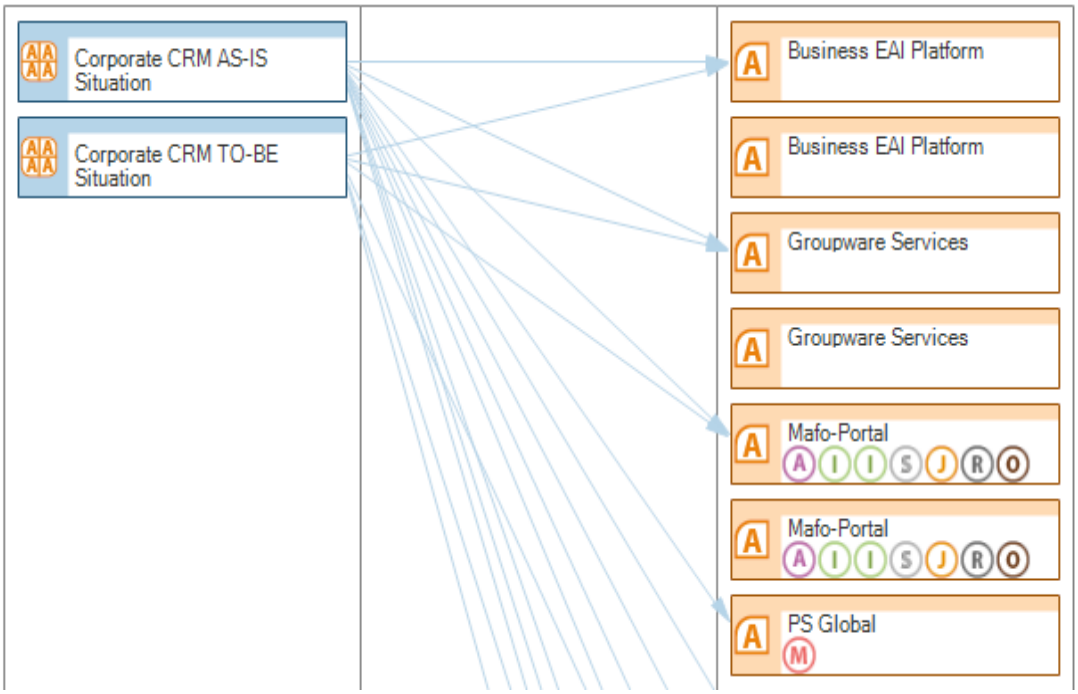
Set the following attributes of the indicator rule node:

Attribute	Description
Name	Defines the name of the indicator rule node in the explorer of the **Report Assistant**.
Type	Select `IconQuery`.
Native Sql/ Alfabet Query/ Query as Text	Defines all information about the icon assignment. Define a query that returns at least: • The `REFSTR` of the current object. This definition must be included in the `SELECT` statement of native SQL query in a visible column, that means that the first argument of the `SELECT` statement cannot be used to define this information. For Alfabet queries, the `REFSTR` of the objects found by the query must be defined in the Show properties. • The name of the icon that shall be displayed. Only icons uploaded as 22 x 22 icon to the icon gallery can be used for display via an icon query. For information about uploading icons to the icon gallery, see the section Adding and Maintaining Icons for the Alfabet Interface. Optionally, the query can contain columns returning the following information: • A string to be displayed as tooltip for the icon. • A string to be displayed as legend text for the icon.

Attribute	Description
	• A string to be displayed as legend group header for the legend group the icon shall be explained in. This information is required to add the icon explanation to the legend. • The REFSTR of an object that shall be coupled with the icon as navigation target. If this information is provided, and a view definition is not given, the object view of the object opens when the user double-clicks the icon. • A view definition that specifies a view related to the object represented by the icon. The view opens instead of the object view if the user double-clicks the icon. Navigation is only activated if the REFSTR of a navigation target object is also defined in the query. The navigation target must be specified in the syntax described in the section Defining Navigation from the Report to Alfabet Views.
Icon Column	Enter the name of the column in the query results returning the name of the icon that shall be displayed.
Object Column	Enter the name of the column in the query results returning the REFSTR of the object the icons shall be assigned to in the configured report.
Tooltip Column	Enter the name of the column in the query results returning the string that shall be displayed as tooltip if the user moves the mouse over the icon.
View Column	Enter the name of the column in the query results returning the view that shall open as a navigation target instead of the object view of the object assigned to the icon with the column defined in the attribute **Navigation Object Column**. The view opens for the assigned object. Navigation is not provided if the attribute **Navigation Object Column** is not defined. The navigation target must be specified in the query using the syntax described in the section Defining Navigation from the Report to Alfabet Views.
Navigation Object Column	Enter the name of the column in the query results returning the REFSTR of the object that shall be assigned to the icon. If an object is assigned to an icon, navigation to the object view of the object or to the alternative navigation view defined with the attribute **View Column** is enabled.
Legend Group Column	Enter the name of the column in the query results returning a string to be used as headline of the legend icon group the icons in the legend of the configured report. If no **Legend Group Column** is defined, the icon is not added to the legend, even if a **Legend Entry Column** is defined.
Legend Entry Column	Enter the name of the column in the query results returning a string that shall be displayed next to the icon in the legend of the configured report. If no **Legend Entry Column** is defined, but **Legend Group Column** is defined, the icons are displayed in the legend with the icon name as legend text.

Attribute	Description
Legend Row Item Count	Defines the number of indicator values displayed in one row of the legend of the report. By default, 4 items are displayed in a row of the legend.
Indicator Index	If multiple indicator rules are defined, and the attribute **Indicator Fill Policy** of the configured report is set to `ByIndex`, define the position of the icon displayed for the current indicator rule. Enter 0 for the first position on the left and ascending integers for the next positions from left to right. Please note that positioning `ByIndex` only displays one icon per indicator rule. If you defined a query returning multiple icons per object box, the **Indicator Fill Policy** of the configured report should be set to `LeftAlign`. This attribute is then ignored for positioning.

For example, a lane report displaying applications in application groups and sub-groups shall display local components defined for the applications as icons in the application. The icons used for this example are bubbles with a letter that shall display the start letter of the name of the local component.



If a user double-clicks an icon, the Evaluations page view shall open for the selected local component. The tooltip text and the legend text shall both return the name of the local component.

The following query is defined for the indicator rule:

```
SELECT NULL AS 'REFSTR', app.REFSTR AS 'Object',
'Contoured_Circle_Color_'+UPPER(LEFT(loco.NAME,1)) AS 'Icon',
loco.NAME AS 'Tooltip', 'View=GraphicView:ObjectEvaluation' AS
```

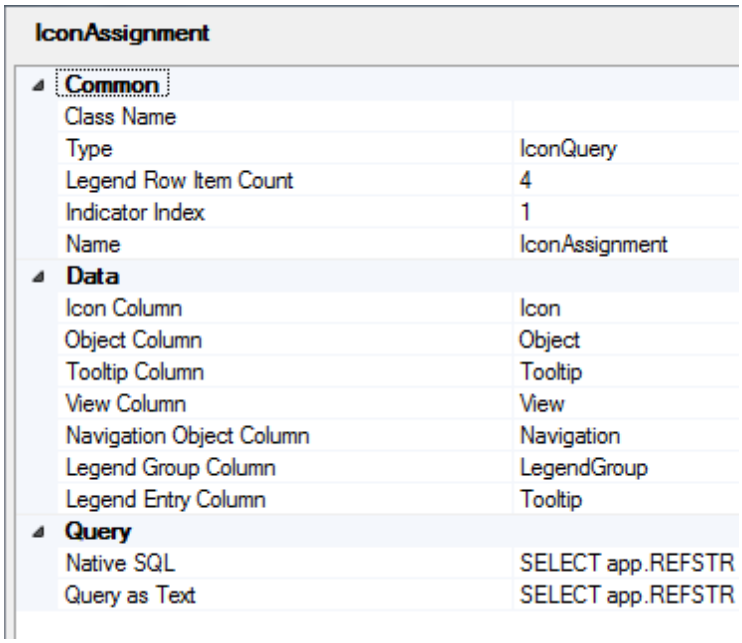


```
'View', loco.REFSTR AS 'Navigation', 'Local Components' AS
'LegendGroup'

FROM APPLICATION app

INNER JOIN LOCALCOMPONENT loco ON loco.OWNER = app.REFSTR

INNER JOIN RELATIONS rel ON rel.FROMREF = app.REFSTR

INNER JOIN APPLICATIONGROUP appg ON appg.REFSTR = rel.TOREF

WHERE rel.PROPERTY = 'ApplicationGroups'

AND appg.BELONGSTO = @BASE
```

The mandatory information is added to the query with the following arguments of the SELECT statement:

- The first argument of the SELECT statement is not required, because a base object is not evaluated for the result dataset. It is set to NULL. The object the icon shall be assigned to is defined in the column `Object`, that returns the REFSTR of the application the local component is assigned to. The `JOIN`s and `WHERE` conditions in the query reduce the number of applications to applications that are displayed in the lane report.
- The name of the icon is defined in the column `Icon` of the result dataset. The name of the icons are identical except for the letter they display. This enables the query of the configured report to return the icon name on basis of a fixed string combined with an object related information, which is the first letter of the local component's name.

The columns in the report dataset are then mapped to the respective attributes of the indicator rule of the type `IconQuery`:



The attributes **Tooltip Column** and **Legend Entry Column** are both mapped to the same column in the query result dataset, because the information shall be identical in the tooltip and the legend.

Defining the General Display of Object Boxes in Treemap, Diagram, Lane and Layered Diagram Reports

By default, object boxes in treemap, diagram matrix, lane and layered diagram reports are displayed as a colored box with a white inner field containing the text:

This design can be changed to fully colored object boxes with the text displayed in the colored area:

To change the coloring of object boxes in the XML object ***SolutionOptions***:

1) Click the **Presentation** tab in the tab bar and expand the **XML Objects** folder by clicking the + symbol. You will see all XML objects that can be edited.
2) Right-click the XML object ***SolutionOptions*** and select **Edit XML**. The **XML Editor** is displayed in the center pane. For general information about editing XML objects in Alfabet Expand, see Working with XML Objects.
3) If not already included, add the XML attribute `CustomReportUserItemStyle` to the XML element `SolutionOptions`. Enter one of the following values for the XML attribute:
   - `WhiteFace`: The object boxes are rendered in the standard design with text displayed with a white background within the colored box.
   - `FullFace`: The object boxes are rendered with a fully colored background with the text directly displayed in the colored area.
4) In the toolbar, click the **Save** button to save the XML definition.

Defining Branching Diagram Reports

Branching diagram reports are designed in the **Report Assistant**. For information about how to open the **Report Assistant**, see Creating a New Graphic Report. For more information about how to work with the **Report Assistant**, see Configuring the Report with the Report Assistant.

The content and layout of a branching diagram report is defined by adding elements to the explorer of the **Report Assistant** and setting the attributes in the attribute pane of the **Report Assistant** window.

The following table lists the types of explorer node elements that can be added to the report configuration to specify the content of the report. Information is provided in the table about the purpose of each explorer node element:

Explorer Node Element	Required to Configure:
Root Node	General layout of the report
Root Queries	Container for **Query** elements that define the content of the report.
Query	Definition of all or part of the content of the report. This node is a sub-node of the **Root Queries** node. A **Root Queries** node can have multiple **Query** sub-nodes.

The content of the report is defined via one or multiple native SQL queries.

A simple version of the report can also be build using Alfabet query language. But many of the functionalities that are implemented like color definitions and definition of an alternative navigation target for objects in the report are only available via native SQL queries.

Defining the Basic Design of the Branching Diagram Report

The basic design of the report is designed directly in the attributes of the root node element of the **Report Assistant** explorer:

Attribute	Description
Rendering Type	Set to `Label_as_Node` to render nodes as a box with the label text in the box. Set to `Circular_Node_with_Separate_Label` to render nodes as bubbles, or class icons with the label text outside the bubble or class icon. Whether nodes are rendered as bubbles or class icons depend on the setting of the **Show Class Icon** attribute.
Show Class Icon	If the **Rendering Type** attribute is set to `Label_as_Node`, and this attribute is set to `True`, object class icons are displayed in front of the label text within the boxes representing the nodes. If the **Rendering Type** attribute is set to `Label As Node`, nodes are rendered as class icons if this attribute is set to `True` and as bubbles if this attribute is set to `False`. The class icon is defined in the class settings of the relevant object class.
Report Background Color	Defines the background color for the branching diagram report.

Attribute	Description
Margin	Defines the space between the branching diagram nodes and the border of the presentation object in pixel.
Node Radius	Defines the radius of each node in pixel. This setting is only relevant if the **Rendering Type** attribute is set to `Circular_Node_with_Separate_Label`.
Node Max Width	Defines the maximum width of boxes representing object nodes. This setting is only relevant if the **Rendering Type** attribute is set to `Label_as_Node`. Note the following about the sizing of object node boxes: The object boxes will all have an identical size. The size is the size required to display the longest label text with the limitation that the boxes will not be bigger than defined with the **Node Max Width** and **Node Max Height** attributes. If the maximum size of the box is not sufficient to display the complete label text, the text will be truncated and will end with three dots. If no tooltip is defined for the object node, the complete label text will be displayed as tooltip if the user moves the cursor over a truncated label text.
Node Max Height	Defines the maximum height of boxes representing object nodes. This setting is only relevant if the **Rendering Type** attribute is set to `Label_as_Node`. Note the following about the sizing of object node boxes: The object boxes will all have an identical size. The size is the size required to display the longest label text with the limitation that the boxes will not be bigger than defined with the **Node Max Width** and **Node Max Height** attributes. If the maximum size of the box is not sufficient to display the complete label text, the text will be truncated and will end with three dots. If no tooltip is defined for the object node, the complete label text will be displayed as tooltip if the user moves the cursor over a truncated label text.
Space Between Nodes Vertical	Defines the vertical spacing between the nodes in pixel.
Space Between Nodes Horizontal	Defines the horizontal spacing between the nodes in pixel.
Node Background Color	Defines the default coloring of the bubbles or boxes representing the objects in the branching diagram. The attribute is not relevant if nodes are rendered as object class icons. The color is applied to all nodes for that no color is defined via the native SQL query for the report.
Node Border Color	Defines the default coloring of the border of bubbles or boxes representing the objects in the branching diagram. The attribute is not relevant if nodes are rendered as object class

Attribute	Description
	icons. The color is applied to all nodes for that no color is defined via the native SQL query for the report.
Node Border Width	Defines the border width for the object nodes in pixel.
Node Border Weight	Defines the border styple for the object nodes. Lines can be solid, dashed or dotted.
Link Color	Defines the default coloring of the links between the object nodes in the branching diagram. The color is applied to all links for that no color is defined via the native SQL query for the report.
Link Weight	Defines the width of the links between the object nodes in pixel.
Link Style	Defines the line style of the links between the object nodes. Lines can be solid, dashed or dotted.
Draw Vertical	Defines the direction of the branching diagram. If set to `True`, the different grouping levels are placed beneath each other in vertical direction with the nodes in the same grouping level displayed next to each other in horizontal direction. If set to `False`, the different grouping levels are displayed horizontally with the nodes in the same grouping displayed next ot each other in vertical direction.
Captions Vertical	Defines the direction of the captions in the reports. This attribute is only available if the attribute **Draw Vertical** is set to `True`. By default, captions are written in horizontal direction. Setting this attribute to `True` changes the display of the caption text. Text is then displayed vertically with the text direction from bottom to top.

Defining Display of Objects and Links via the Query of the Report

The display of object nodes and the links between the different levels of object nodes are specified via a native SQL query or an Alfabet query defined in an element node `Query/Alfabet Query/Query As Text` as sub-node of the `RootQueries` node. The query must define a grouped dataset via definition of Alfabet query language instructions. The grouping levels in the query correspond to the levels in the branching diagram report.

For information about how to define a grouped dataset using Alfabet query language instructions see the section Grouping Query Results in Expandable Reports in the chapter Defining Queries.

The query defined for a configured branching diagram report can be very complex. To make sure that your query corresponds to the above mentioned requirements without taking the configuration of the report into account, you can first define a configured report of the **Type** `NativeSQL` or `Query` and control that the output is a grouped table with the columns joined in the correct way. When the output is as expected, paste the query into the rectangular treemap report and set the required attributes in the report assistant.

Multiple Query elements can be defined within the Root Queries node. If you define multiple Query elements, the objects found by each query are displayed in the same level in the order of queries.

After having defined the query, the name of the column containing the respective information must be defined in the attributes of the root node of the report assistant. The defined column must contain data in a given format. The following table lists the attributes to be filled with the column names and the data format required in the result dataset:

Attribute	Description	Expected Query Output
Image Column	Defines the name of the column containing the text to be displayed in the boxes.	Any string to display under the nodes.
Stereotype Column	Defines the name of the column containing the name of the stereotype of the current object.	The value of the Stereotype attribute of the object.
Node Background Color Column	Defines the name of the column containing a HTML compliant color definition for coloring of the background of the nodes.	An HTML compliant color code, like For example,: `#11A6D0`
Node Text Color Column	Defines the name of the column containing a HTML compliant color definition for coloring of the text under the nodes.	An HTML compliant color code, like For example,: `#11A6D0`
Node Border Color Column	Defines the name of the column containing a HTML compliant color definition for coloring of the border of the nodes.	An HTML compliant color code, like For example,: `#11A6D0`
Link Caption Column	Defines the name of the column containing a text to display under the links connecting the nodes.	Any string to display under the links.
Link Color Column	Defines the name of the column containing a HTML compliant color definition for	An HTML compliant color code, like For example,: `#11A6D0`

Attribute	Description	Expected Query Output
	coloring of the links connecting the nodes.	
View Column	Defines the name of the column containing the definition of an alternative link target for the double click action on the object nodes. By default, the standard object profile of the object opens when a user double clicks on a node or clicks a node and then clicks the **Navigate** button.	`View=`*`ViewType`*`:`*`ViewName`* The `ViewType` can be one of the following: • ***Report*** for a configured report • ***GraphicView*** for a standard Alfabet view • ***ObjectView*** for an object profile `ViewName` is the name of the view. For more information about defining navigation to a target view, see Defining Navigation from the Report to Alfabet Views.
Legend Caption Column	Defines the text to be displayed in the legend for the given node type. The legend gives information about each different node. If the class icons are displayed, each class icon displayed is added to the legend. If bubbles are displayed, each differently colored bubble is explained.	Any string to display in the legend.

In the attributes of the root node of the report assistant, only one column can be defined for a type of information. For example, only one column can be defined to display the text of the object boxes. Therefore, the native SQL query must be defined to display information about all levels in the grouped dataset in the same column. This can be achived with `JoinColumn` instructions of the Alfabet query language.

For detailed information about the mapping of columns in the result dataset to attributes of the root node of the configured report, see Mapping of Query Data to Attribute.

Configuring Bubble Cloud Reports

Bubble cloud reports are designed in the **Report Assistant**. For information about how to open the **Report Assistant**, see Creating a New Graphic Report. For more information about how to work with the **Report Assistant**, see Configuring the Report with the Report Assistant.

The report assistant has a single node in the explorer with the attributes displayed in the area on the right.

The content of the bubble cloud report is defined via a query that returns the data in the required formats. The names of the columns in the dataset resulting from the query must then be mapped to the attributes in the **Report Assistant** to define which information required to build the report is returned in which column of the query.

It is recommended to use native SQL for creating bubble cloud reports. Alfabet query language does not provide the ability to include anything else than data from the tables in the Alfabet database into the search results, and the data is usually not corresponding to meaningful text size or text orientation values for the words cloud report.

The following table lists the attributes available to define the query and the mapping of data from the query to the respective functionality. The description informs about the data type to return in the column mapped to the respective attribute:

Attribute	Description
Native SQL / **Query as Text** / **Alfabet Query**	Define a native SQL query in either the attribute **Native SQL** or **Query as Text**, or define an Alfabet query in either the attribute **Alfabet Query** or **Query as Text**. Details about the required output of the query are given below with the description of the column mapping attributes.
Text Column Name	The name of the database column in the query that returns the text to be displayed. This attribute is mandatory.
Font Size Column Name	The name of the database column in the query that returns the font size for the text in pixel. This attribute is mandatory.
Rotation Column Name	The name of the database column in the query that returns the text orientation. Allowed values are 0 - 180. 0 will display the text in normal horizantal orientation, while 180 will display the text upside down. the text is turned clockwise. This attribute is mandatory.
Tooltip Column Name	The name of the database column in the query that returns a string to be displayed as tooltip on mouseover on the text. This attribute is optional. By default, no tooltip is displayed.
Text Color Column Name	The name of the database column in the query that returns the color of the text in HTML compatible format. This attribute is optional. The default text color is black (#000000).
Legend Group Column Name	If the **Show Legend** attribute is set to `True` and both the **Legend Caption Column Name** attribute and the **Text Color Column Name** attribute are defined, legend will be displayed for the text coloring in the words cloud report. Optionally, the legend can be grouped in sections. This can be useful For example, in case coloring has a different impact for different font sizes. The legend will be grouped in sections if the **Legend Group Column Name** attribute is set to the name of the database column in the query that returns the caption of the legend group that the text color returned in the same row shall be displayed in.

Attribute	Description
	If **Legend Group Column Name** is not set, all coloring will be listed under the heading **Legend** without grouping.
Legend Caption Column Name	A legend is available for the text coloring in the words cloud report. To display the legend, **Legend Caption Column Name** must be set to the name of the database column in the query that returns an explanatory string for the text color returned in the same row.
Reference Column Name	The column in the return dataset returning the REFSTR of the object for that navigation should be activated for the bubble in the report. This attribute is mandatory to define navigation. By default, navigation will open the object profile of the object. This behavior can be changed with the attribute **View Column Name**.
View Column Name	The column in the return dataset returning the navigation target for navigation from bubbles in the report. By default, navigation will open the object profile of the object. the column must return the navigation target in the format <ViewType>:<ViewName>. Allowed view types are `GraphicView`, `Report`, and `ObjectView`. Form ore information about the defintion of a link target, see Defining Navigation from the Report to Alfabet Views.

In addition to the dynamic content, static formatting can be applied to the report with the following attributes. All these attributes are optional.

Attribute	Description
Width	Defines the width of the area that can be filled with text in pixel. The default value is 800 pixel.
Height	Defines the height of the area that can be filled with text in pixel. The default value is 600 pixel.
Background Color	Defines the background color of the area reserved for the report. By default, the background is white. The color can be changed via the color picker. For more information about working with the color picker, see Defining Color Attributes.
Border Color	Defines the color of the border drawn around the report area. By default, the border color is white. As the default for the background color is also white, no border will be visible per default for the report. The color can be changed via the color picker. For more information about working with the color picker, see Defining Color Attributes.

Attribute	Description
Border Thickness	Defines the thickness of the border drawn around the report area. The default value is 1 pixel. Please note that the border is drawn within the area filled with text by the report algorithm. Therefore, if the report shows a very high number of results, the text may overlap the border.
Padding	Defines the space between texts in the report. The default is 2 pixel.

Defining Chart Reports

The **Report Assistant** has four tabs:

- **Data Source Definition**: The **Data Source Definition** tab allows the query to define the objects displayed in the chart to be specified.
  - **To define an Alfabet query**: Write the complete Alfabet query, including instructions, if required, into the **Query for Rows** field or click the Browse  button to open the **Alfabet Query Builder** to specify your Alfabet query.
  - **To define a native SQL query**: Write the native SQL query into the **Query for Rows** field. If you need to define Alfabet query language instructions in combination with the native SQL query, For example, to create expandable rows or columns, define the instructions separately in the **Query Instructions (for native SQL only)** field.

More information about the specification of the data source is given in the section Defining the Display of the Data Source in a Chart.

- **Data Source Result**: The data source result displays the result of the query defined in the **Data Source Definition** tab in a tabular report. The report is generated when clicking the **Check Data Source** button on the upper right corner of the **Data Source Result** tab and on opening of the **Report Assistant**. The query definition should be checked prior to proceeding with the next configuration steps.
- **Business Graphic Definition**: This tab allows you to specify the type and content of the chart on basis of the data source definition.
- **Additional Attributes**: This tab allows you to specify additional formatting options for the chart defined in the **Business Graphic Definition** tab.

Specification of the Data Source

A chart report is a graphic representation of the influence of a property (x value) on a numerical value (y value).

A chart can display For example,

- costs per year

- indicator values per indicator type
- the number of applications per application group
- the number of applications in a specific lifecycle status.

Pie charts, doughnut charts and waterfall charts can only display this two-dimensional relationship, while in line, area, spline, spline area and bar charts, the relationship creates one line, spline, spline area, area or bar and a third dimension can be added by displaying multiple lines, splines, spline areas, areas or bars in the chart (series).

Radar charts are more complex representations of the influence of multiple properties on numeric values. Each radius defines a property with a scale for the numeric value for the property. For each series of data, the value of the object on the respective radius is displayed by a colored bubble on the radius. The bubbles for an object are connected by lines and optionally the net span by the connected bubbles can be colored.

Multi-level pie charts display a two-dimensional relationship on two related levels. For example, the applications in an application group sized by the value of an indicator and on a second level, for each application the local component sized by the same indicator are displayed. The relative sizing of object can either be accurate on the outer or the inner part of the multi-level pie chart.

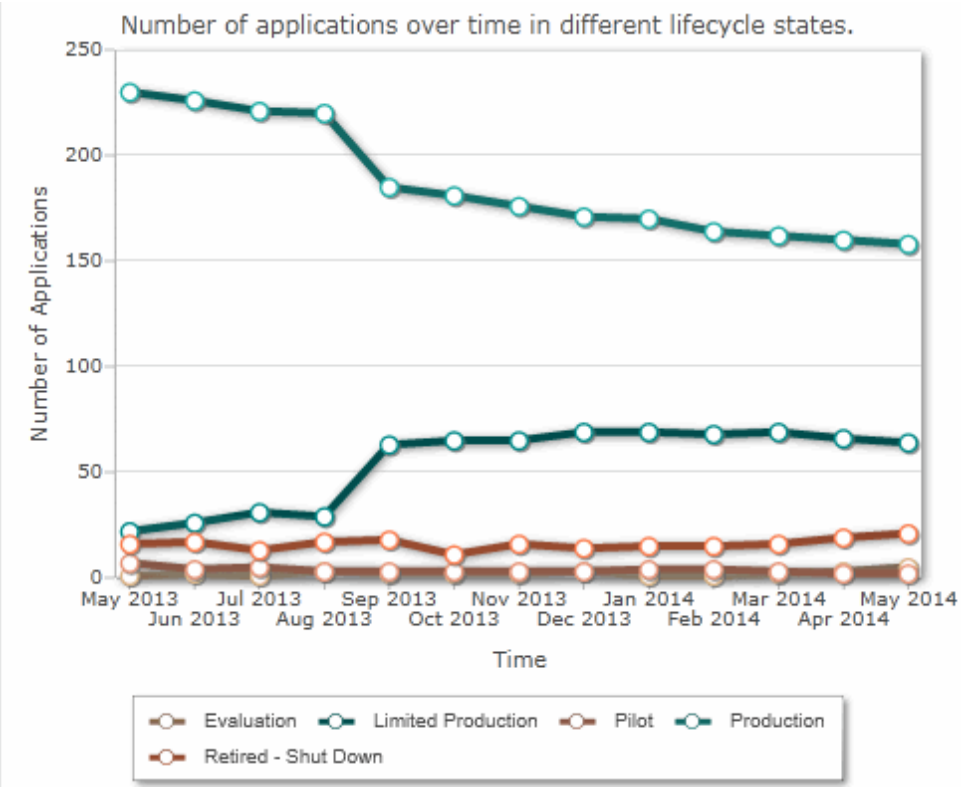
The data source definition specifies which objects and which properties are displayed in the chart report based on a query. Depending on the type of chart that is being designed, the query must result in a table providing the data required to built the respective chart.

The x and y-axis of a report can be defined in two different ways:

- The dataset contains two columns, one for the values on the x-axis and one for the corresponding values on the y-axis. For radar charts, each values in the x-axis column correspond to a radius in the radar chart. The values in the y-axis column must be of the type integer or double. You can either specify an object's property of the type integer, or a `COUNT` on the results for a property of any other data type. For line, spline, bar, area, spline area and radar charts, y values in columns that are identical in the x value and the serie value are summed up and the total of all rows is displayed as y value for the x value and series. For waterfall charts each row in a result dataset is displayed as a separate column, even if the x value and any other value except for the y value is identical. The results are added to the chart in the order that they are specified in the result dataset.

For radar charts, a third column must define the object that the x and y definition in the respective row belongs to. For bar, area, spline area, spline and line charts, the definition of a third row defining which line, spline, spline area, area, or bar series in the chart the data belongs to is optional. For pie charts, multiple pie charts and waterfall charts the third dimension is not applicable.

For example, a line charts displays how many applications are in which lifecycle phase for each month of the next twelve month:



The line chart is based on an SQL query that calculates the time scale and reads the number of applications in each lifecycle status for the calculated time values:

Query for Data Source

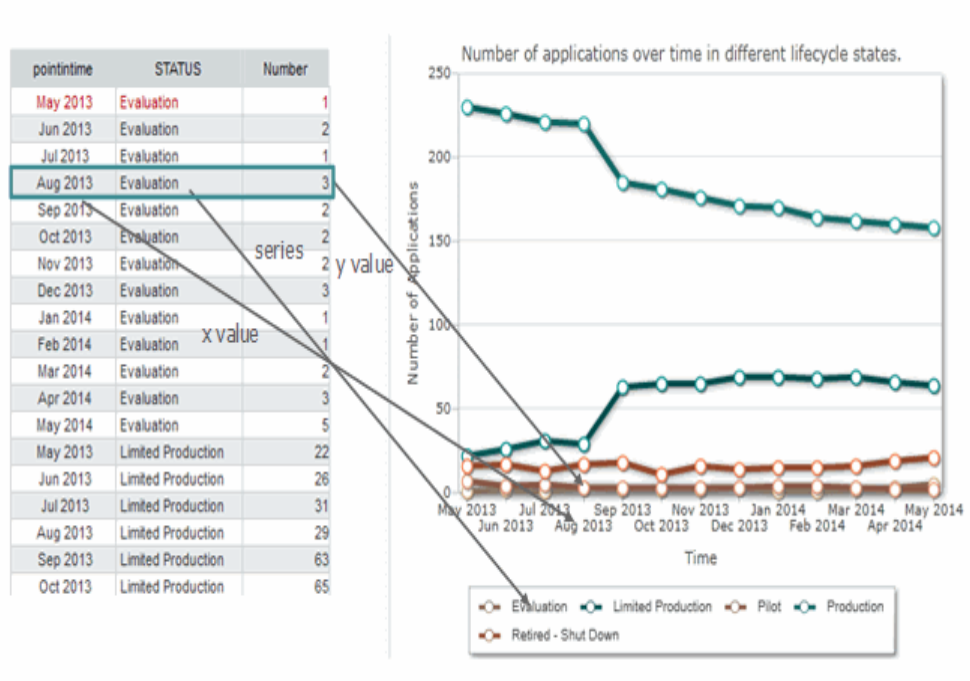
```
WITH tmp(pointintime, monthcount) AS (
  SELECT CURRENT_TIMESTAMP - DAY(CURRENT_TIMESTAMP) + 1, 0
  UNION ALL
  SELECT DATEADD(month, 1, tmp.pointintime), monthcount + 1
  FROM   tmp
  WHERE  tmp.monthcount < 12
)
SELECT NULL, tmp.pointintime, ts.STATUS, COUNT(*) AS Number
FROM   tmp, APPLICATION app, TIMESTATUS ts
WHERE  ts.OWNER = app.REFSTR
  AND  ts.STARTDATE <= tmp.pointintime
  AND  ts.ENDDATE >= tmp.pointintime
GROUP BY tmp.pointintime, ts.STATUS
```

Query Instructions (for native SQL only)

```
SETCOLUMNFORMAT("pointintime", "MMM yyyy");
```

The resulting dataset has three columns. The column `pointintime` contains the x-values, the column `Number` contains the y values and the column `Status` defines wich series a value belongs to. Each row of the dataset defines one point in the line chart. The x

values, y values and the title of the series are all written from the dataset in the respective row.



- The dataset contains multiple columns each corresponding to one x value on the x-axis. The values in the column are the y values for the x value represented by the column. The values in the x-axis column must be of the type integer or double. You can either specify an object's property of the type integer, or a `COUNT` on the results for a property of any other data type.

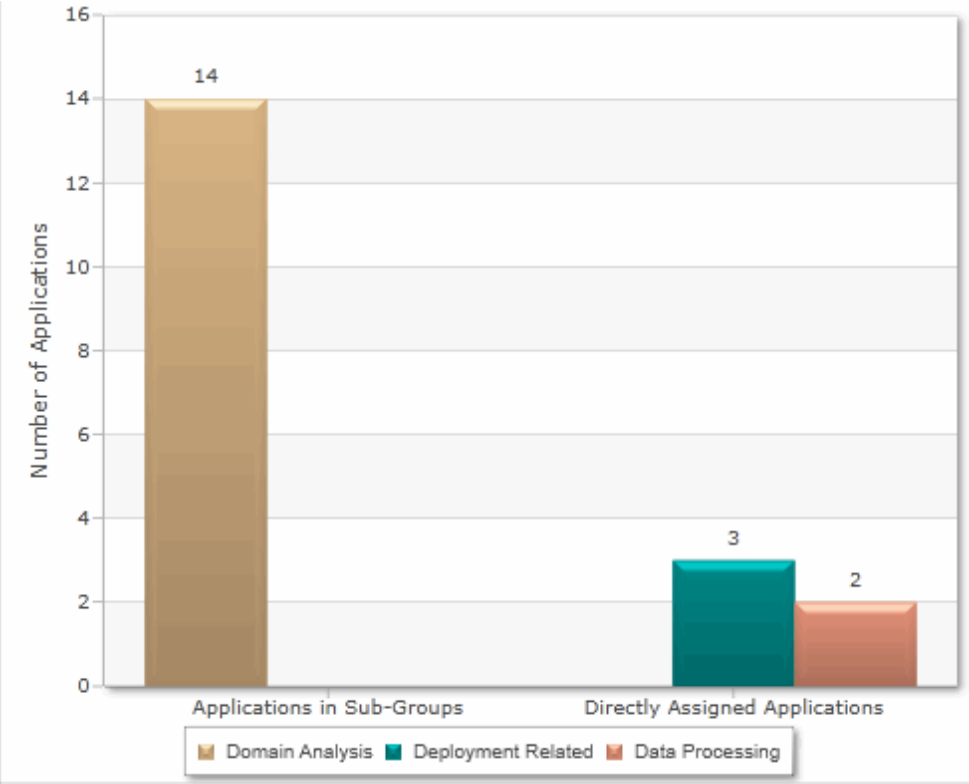
For radar charts, a third column must define the object net that the x and y definition in the respective row belongs to. For bar, area, spline area, waterfall, spline, and line charts, the definition of a third row defining which line or bar series in the chart the data belongs to is optional. For pie, doughnut and multi level pie charts, the definition of a third row is not applicable.

If the dataset contains more than one row and no series is defined, the y value is the total of all values in the rows of the x column. If the dataset contains a column for the series definition, the numerical value in a row is displayed for the series specified in the series column. If multiple rows contain results for the same serie, the total of the numerical values of all rows belonging to the series are displayed as y value for the series.

The text displayed on the x-axis for each x value is identical to the name of the respective column. To change the name of the column, a `RenameColumn` instruction can be used in Alfabet queries and an alias can be defined in native SQL queries. The `SetColumnShowName` instruction and the alias definition in Alfabet queries change the caption of the column without changing the column name and are Therefore, not suitable to define x values in charts. For information about the `RenameColumn` instruction, see the section Defining Column Names and Captions in the chapter Defining Queries.

For example, a bar chart displays the number of applications that are directly assigned to application groups of a first level in the hierarchy and the number of applications that are assigned to the application groups that are assigned as sub-groups to the first level application groups. The number of applications are displayed on the y-axis and the x-axis has two values: directly assigned and indirectly assigned. The first level application groups are defined as series, which means that multiple bars with different coloring are

displayed for each x value and a legend informs about the application group represented by each different color.



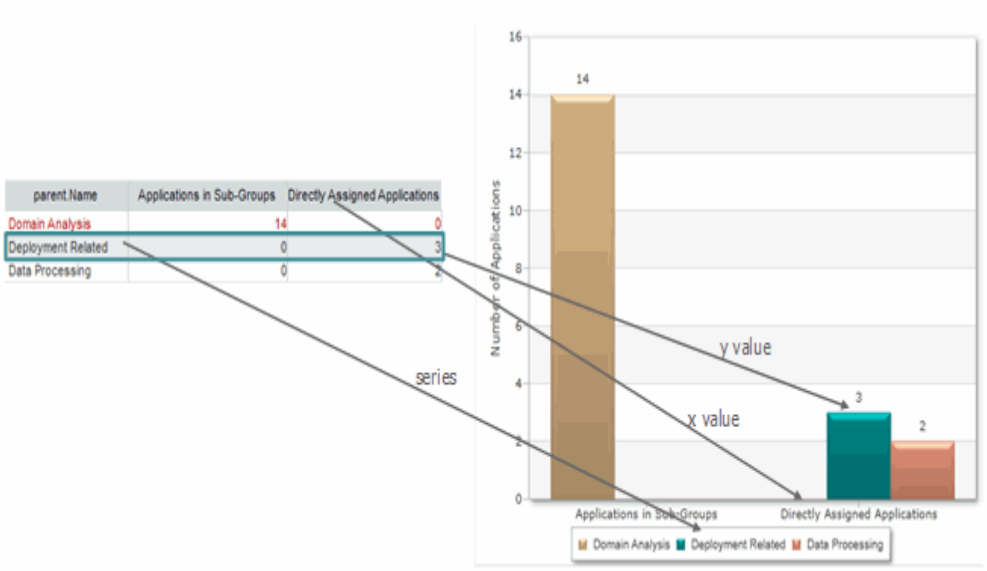
The data source query for the bar chart finds application groups and the directly subordinate application groups thereof as well as the number of applications that are directly assigned to the application group and the number of applications assigned to one of the subordinate application groups:

```
ALFABET_QUERY_500

FIND ApplicationGroup AS parent

LeftJoin ApplicationGroup AS child ON child.BelongsTo =
parent.REFSTR

LeftJoin Application AS capp ON child.Applications =
capp.REFSTR

LeftJoin Application ON parent.Applications =
Application.REFSTR

WHERE parent.Name LIKE 'D%'

Instructions

RenameColumn("Application.REFSTR","Directly Assigned
Applications");

RenameColumn("capp.REFSTR","Applications in Sub-Groups");

EndOfInstructions

QUERY_XML

<QueryDef>

<ShowProperty Type="Property" ClassName="ApplicationGroup"
ClassAlias="parent" Name="Name" />
```



```
<ShowProperty Type="Property" ClassName="Application"
ClassAlias="capp" Name="REFSTR" Function="COUNT" />

<ShowProperty Type="Property" ClassName="Application"
Name="REFSTR" Function="COUNT" />

</QueryDef>
```

The resulting dataset has three columns. The column `parent.Name` is the series column that defines which bar the data in the following two columns belongs to. The other two columns both correspond to one x value, that means that the name of the column is identical to a value on the x-axis. Therefore, a `RenameColumn` instruction was used in the query to give the x values in the graphic a meaningful name. The numerical vlaues in the columns are the y values displayed for the bar for the respective x value of the column.



For multi level pie chart reports, the definition of the x and y-axis must be provided multiple times, that means a separate definition of x and y-axis values must be provided per level. The query must Therefore, return a grouped dataset with each grouping level defining one circle of the multi-level pie chart. The x and y-axis values must be available in a single column. The dataset must Therefore, not only grouped by using a `GroupBy_Ex` instruction, but the columns containing the respective x and y values must be joined using a `JoinColumns` instruction.

For detailed information about how to define a grouped dataset with the **GroupBy_Ex** and **JoinColumns** instructions, see Grouping Query Results in Expandable Reports in the chapter Defining Queries.

A multi-level pie chart shall display applications owned by a selected organization in the first level and the applications that are connected to these applications via information flows in the second level. The sizing of objects shall depend on the value of the indicator "Incidence Rate".

The query defined for the multi-level pie chart returns the name and `REFSTR` of the application and the value of the indicator for both the first level applications (parent) and the second level applications (child).

```
SELECT parent.REFSTR, parent.REFSTR AS 'parent.REFSTR', parent.NAME
AS 'parent.Name', parentind.VALUE As 'parent.Indicator',
child.REFSTR AS 'child.REFSTR', child.NAME AS 'child.Name',
childind.VALUE AS 'child.Indicator'

FROM APPLICATION parent

INNER JOIN RELATIONS rel

   ON rel.PROPERTY = 'ApplicationGroups' AND
   rel.FROMREF=parent.REFSTR

INNER JOIN APPLICATIONGROUP appg

   ON appg.REFSTR = rel.TOREF

INNER JOIN INFORMATIONFLOW inf

   ON inf.A_TO=parent.REFSTR OR inf.A_FROM=parent.REFSTR OR
   inf.FROMOWNER=parent.REFSTR OR inf.TOOWNER=parent.REFSTR

INNER JOIN APPLICATION child

   ON inf.A_TO=child.REFSTR OR inf.A_FROM=child.REFSTR OR
   inf.FROMOWNER=child.REFSTR OR inf.TOOWNER=child.REFSTR

INNER JOIN INDICATOR parentind

   ON parentind.OBJECT=parent.REFSTR

INNER JOIN INDICATOR childind

   ON childind.OBJECT=child.REFSTR

WHERE appg.REFSTR = @BASE

   AND parent.REFSTR != child.REFSTR

   AND parentind.NAME = 'Incidence Rate:Incidence Rate'

   AND parentind.VALUE >= 1

   AND childind.VALUE >= 1

   AND childind.NAME = 'Incidence Rate:Incidence Rate'
```


A `GroupBy_Ex` instruction is used to group the result dataset into the parent and child level. The columns returning the `REFSTR` of the parent and child applications, that were included for grouping only, are removed from the result dataset. The name columns and indicator value columns are joined with `JoinColumns` instructions into one column.

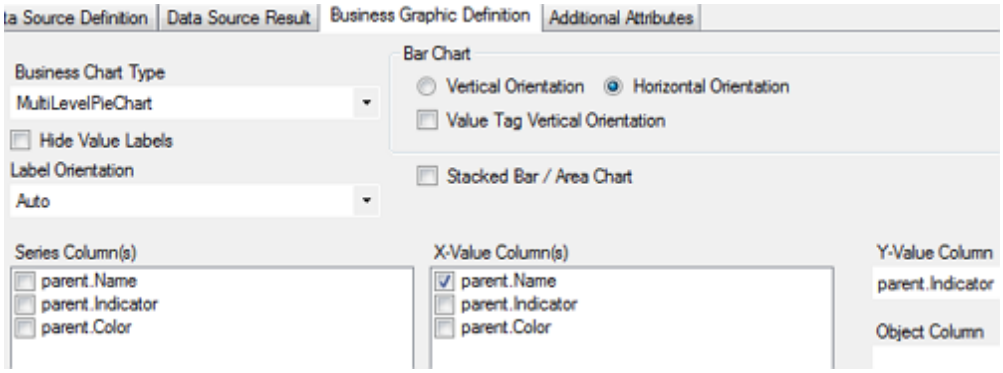
```
GroupBy_Ex("parent.REFSTR","child.REFSTR","parent",0);

RemoveColumns("parent.REFSTR,child.REFSTR");

JoinColumns("parent.Name,child.Name","parent.Name", "");

JoinColumns("parent.Indicator,child.Indicator","parent.Indicator",
"");
```

The resulting dataset displays two columns only: One returning the name of the applications of both levels (`parent.Name`) and one returning the indicator value for both levels (`parent.Indicator`):

	parent.Name	parent.Indicator
1	Eurex Bonds	3.00
2	Summit	3.00
3	Marketview	2.00
4	Eurex Repo	2.00
5	Summit	3.00

First level Second level

In the **Business Graphic Definition** tab, the column `parent.Name` is selected as **X-Value Column(s)** and the column `parent.Indicator` is selected as **Y-Value Column**.

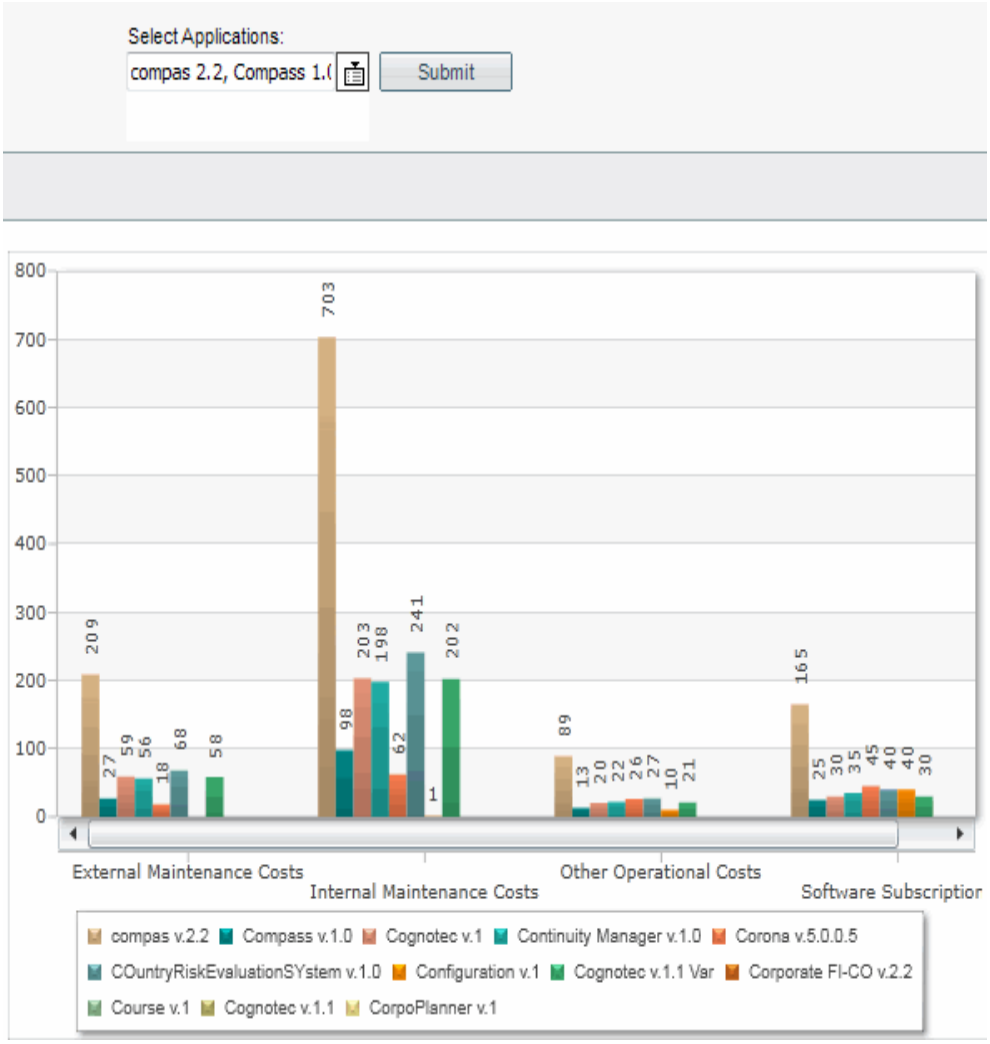


Note the following about the definition of the query the report is based on:

- The attribute `BASE` can be used in the query to generate a report in dependency of the object the user is currently working with. A filter is than generated on top of the report that allows other objects of the base object class to be selected.
- In Alfabet queries, the definition of indicators with a `ShowProperty` element of the type `Indicator` is not supported. Indicators can only be added to the result dataset by joining the respective object classes to the report and adding normal `ShowProperty` elements for property values of the object classes related to indicator definition.

- If a bar, area, spline, spline area or line chart has series defined, a legend is added to the report that relates the series name with the color used for display of the series in the report. If the report contains only one series, the legend is not displayed. For both methods of chart definition described above, the series are read from the values in a column of the report. If a report is defined to display data from multiple series, but at the time the report is used only object data for one of the series is available, the legend will not be shown. If you want to make sure that the report displays a legend even if some of the series are currently not displayed, define a native SQL query that returns at least one empty value for each series.

A report displays budget costs for a group of applications. For each application, all costs are displayed in the same color and a legend is displayed that matches the colors with the application names:



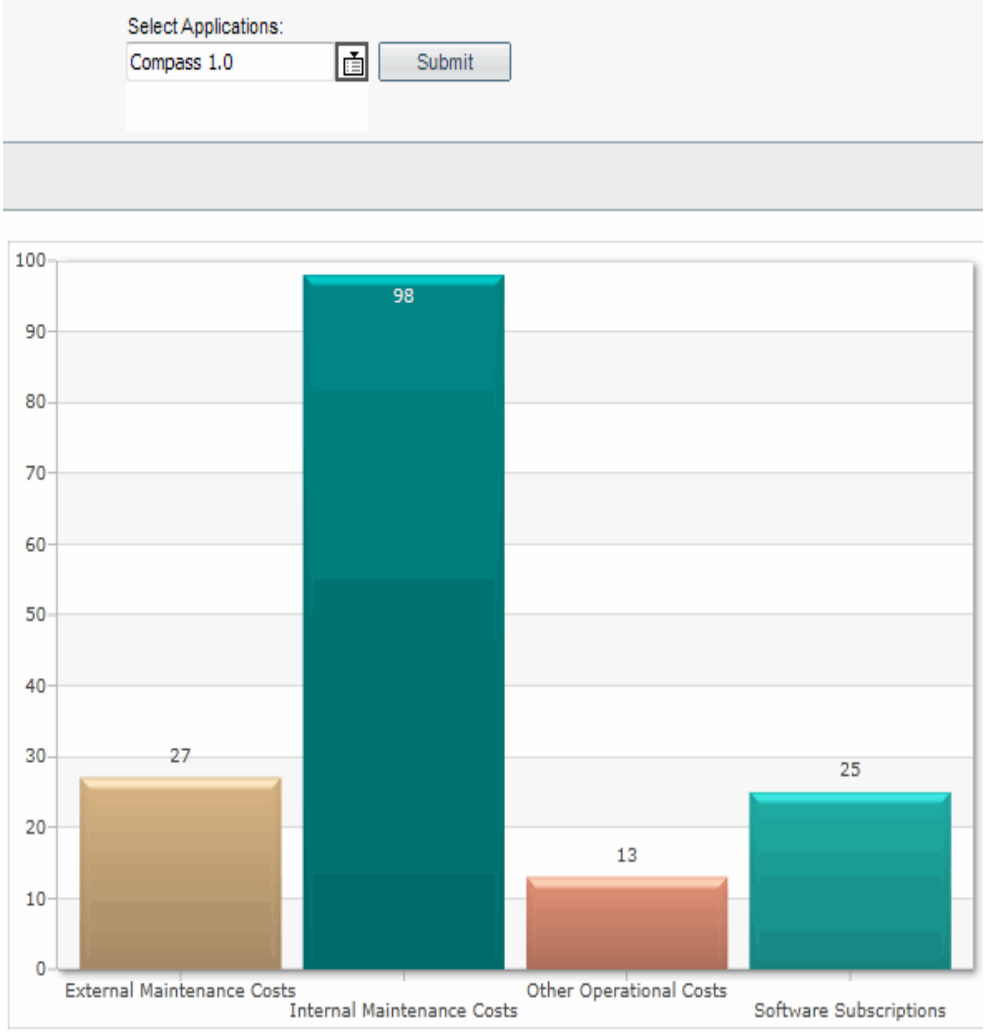
The report is based on the following native SQL query:

```
SELECT NULL As 'REFSTR', app.NAME As 'Application',
app.VERSION As 'Version', COSTTYPE.NAME As 'CostType',
budget.VALUE As 'Budget'

FROM APPLICATION app

INNER JOIN BUDGETVALUE budget

    ON budget.OWNER = app.REFSTR
```



```
INNER JOIN COSTTYPE
    ON COSTTYPE.REFSTR = budget.MONETARYTYPE
WHERE budget.YEAR='2012'
    AND app.NAME LIKE 'Co%'
    AND app.REFSTR IN (@apps)
    AND budget.MONETARYCODEID = 'Current'
```

A filter field on top of the report allows the user to restrict the display to a subset of the applications in the report. If the user selects only one application, the legend is not displayed and the coloring is different for each cost type, although all costs are assigned to the same application:



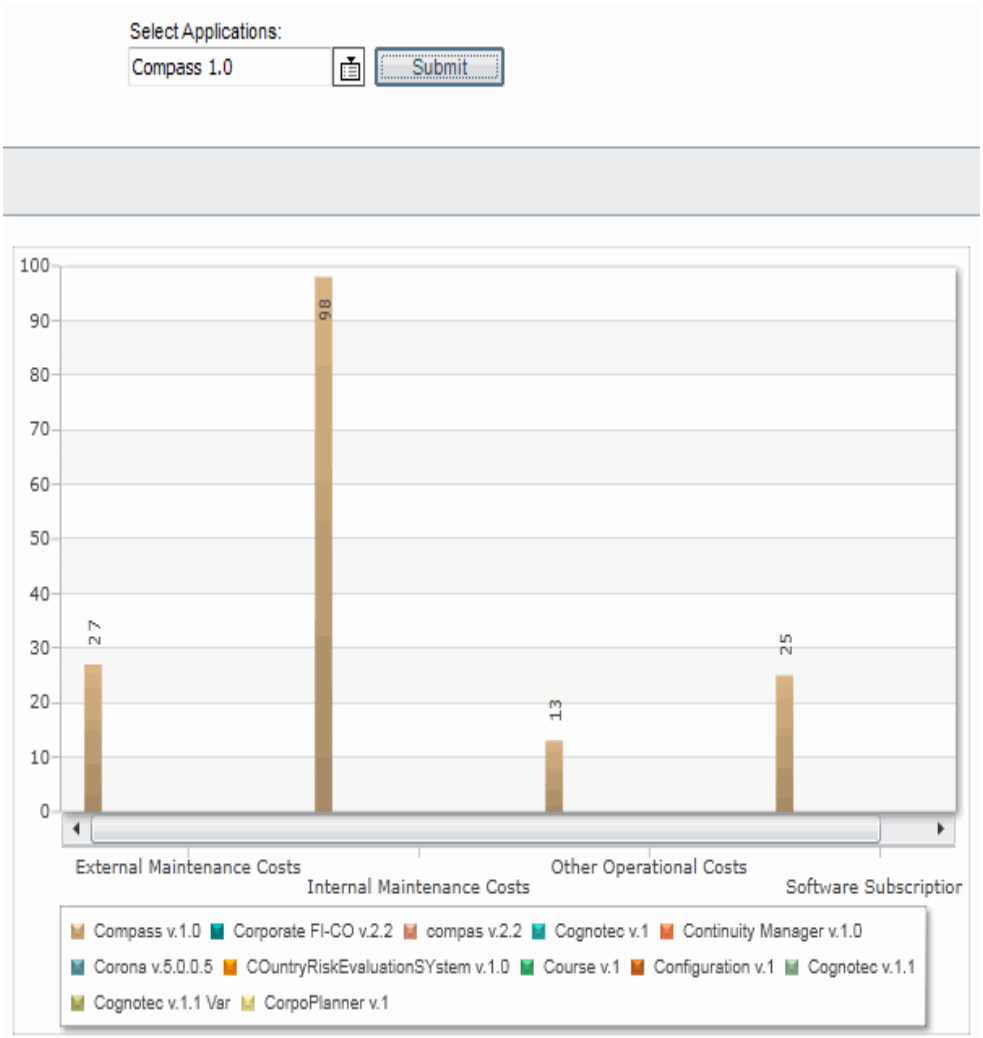
To maintain the constant coloring per application and the display of the language, a UNION ALL is added to the query that adds one row for each application not selected in the filter with the cost type and budget value set to NULL:

```
SELECT NULL As 'REFSTR', app.NAME As 'Application',
app.VERSION As 'Version', COSTTYPE.NAME As 'CostType',
budget.VALUE As 'Budget'
FROM APPLICATION app
INNER JOIN BUDGETVALUE budget
```



```
    ON budget.OWNER = app.REFSTR
INNER JOIN COSTTYPE
    ON COSTTYPE.REFSTR = budget.MONETARYTYPE
WHERE budget.YEAR='2012'
    AND app.NAME LIKE 'Co%'
    AND app.REFSTR IN (@apps)
    AND budget.MONETARYCODEID = 'Current'
UNION ALL
SELECT NULL As 'REFSTR', app.NAME As 'Application',
app.VERSION As 'Version', NULL As 'CostType', NULL As
'Budget'
FROM APPLICATION app
WHERE app.NAME LIKE 'Co%'
    AND app.REFSTR NOT IN (@apps)
```

With this change in the query, the report will look as follows when displaying results for one application only:



Defining the Display of the Data Source in a Chart

The type of chart and the display of data defined in the **Data Source Definition** tab is defined in the **Business Graphic Definition** and **Additional Attributes** tabs:

Do the following to define the chart:

1) Define the data source in the **Data Source Definition** tab.
2) Click the **Check Data Source** button in the **Data Source Result** tab. The result of the test is displayed in the **Data Source Errors** field. The next step can only be performed when the query passes the test without errors.

This step must also be executed if an existing chart is altered, even if the **Data Source Definition** itself is not changed.

3) Define the following fields in the **Business Graphic Definition** tab:
   - **Business Chart Type**: Select the chart type from the drop down list.
   - **Bar Chart**: For bar charts, the following options can also be selected:
     - **Vertical Orientation** / **Horizontal Orientation**: Select the checkbox for the orientation of the bars in the report.
     - **Value Tag Vertical Orientation:** By default, value tags on bars are displayed horizontally. If the required space for complete display of the value tags is not available, you can change the orientation to vertical by selecting the checkbox to allow display of value tags vertically on top of small bars.
   - **Stacked Bar / Area Chart**: For bar and area charts, select the checkbox if you want the results for multiple series in the report to be displayed stacked (each bar or area starting above the previous one with the end of the previous bar or area as starting point). If the checkbox is not selected, the bars or areas of each series will be displayed next to each other starting with the x-axis level.
   - **Hide Value Labels**: The values displayed in chart reports within the graphic can be hidden to enhance readability of charts that contain much data and Therefore, may result in overlap of displayed values. If the checkbox **Hide Value Labels** is selected, values are not displayed within the graphic. To see a value, the user can move the cursor on a point or bar in the graphic to see a tooltip that displays the value.
   - **Label Orientation**: For bar, waterfall, area, spline area, spline and line charts, this parameter defines the text orientation on the x-axis of the report. You can select the following options:
     - `Auto`: The text orientation is automatically determined according to the label text size and the size of the chart.
     - `Vertical`: The text is displayed in a vertical line.
     - `Diagonal`: The text is displayed in a slanted line from bottom left to top right.
   - **Serie Column(s)**: Define the column in the result table of the data source that defines which data belongs to which dataset in a line, spline, radar, area, spline area, bar, or combination chart.

Note the following about the definition of series:

- The number of datasets in a line, spline, waterfall, area, spline area, radar or bar chart should be small to enhance readability. If your data source returns an exceedingly high number of series, you can restrict the display to a maximal number of series by setting the parameter **Max Series Count** in the **Data Source Definition** tab to the maximal allowed number of series. Note however, that this option will hide results from display. Instead of defining the maximal number of series, you should redefine the data source query to return a low number of series values.
- If multiple rows in the dataset match the criteria for the same x-value and serie, the sum of the values in the respective rows are displayed as y-value for the x-value/serie combination.

- **X Value Column(s)**: Select the checkbox of the data source report column(s) that define the x-axis values of the line, spline, spline area, area, or bar chart or the sections in the pie chart, doughnut chart or multi-level pie chart. For radar charts, the x-axes correspond to the radii of the radar chart. For waterfall charts, the x values are used as labels on the x-axis.

Depending on the dataset the report is based on, you can select one or multiple x value columns:

dataset	X value definition	Y value definition
Each combination of x and y value is read from one row in the report. The report contains one column for the x-axis value and one for the y value definition. The data in the y value column must be numerical, the data in the x-axis column can have any data type.	Only one x value column can be selected.	The y value column must be defined with the attribute **Y Value Column**.
For each x-axis value there is a column in the report. The name of the columns are the values displayed on the x-axis. All x-axis columns must contain numerical data.	Multiple x value columns must be selected.	Specification of an y value column is not allowed.

- **Y Value Column**: Select the data source report column that defines the values for the y-axis of the line, spline, area, spline area, waterfall or bar chart or the size of the sectors of the pie chart, doughnut chart or multi-level pie chart. This definition is not required for reports that are based on a dataset where each x value is defined by a separate column in the dataset. For multi-level pie chart, this definition is also not required. All sectors of the pie chart are equally sized if no y-axis is defined.
- **Object Column**: Select the data source report column that define the object that is related to the displayed results. The column must contain the `REFSTR` of the objects. If you select `RowReference`, the `REFSTR` of the `FIND` class of the Alfabet query or the `REFSTR` defined as the first `SELECT` argument in the native SQL query will be used to identify objects. Setting the **Object Column** activates the following functionality:
  - Objects are displayed in different colors and the **Colors** button is activated. The button allows users to change the color for the display of single objects. For more

information about the configuration of coloring based on the **Object Column** attribute see the section Configuring Color Definition per Alfabet Object. If the **Color Column** attribute is set, it supersedes the **Object Column** setting for color definition.

- Navigation from report results to the object profile of the object defined with the **Object Column** attribute is activated. Navigation is only performed if the **Navigate** button is activated for the configured report. If you want to define coloring without defining navigation to the object, you can set the **Object Column** attribute without activating the **Navigate** button. The complete configuration required for navigation based on the setting of the **Object Column** attribute is described in detail in the section Configuring Navigation to the Object's Profile. If the **Navigation Column** attribute is set, it supersedes the **Object Column** setting for color definition.

- **Color Column**: Select the data source column that define the color to be used in the chart for the respective results. The color must be defined in HTML compliant color codes.

  The configuration required to define the color of graphic elements within the dataset is described in detail in the section Configuring Color Definition per Graphic Element.

- **Navigation View Column**: If the **Navigate** button is activated for the report, the user can navigate from graphic parts of the report to an Alfabet view that is defined in the result dataset of the query for the chart report. The link target must be defined in the data source of the report in a special syntax. Select the data source column that defines the Alfabet view that shall open if a user clicks a part of the graphic in the report.

  The required syntax of the link definition and the complete configuration to define the navigation within the dataset is described in detail in the section Configuring Navigation to a Defined View.

- **Use Default Sorting**: Select the check box if you want objects in the report to be sorted by name in alpha-numeric order. If you do not select the checkbox, the sorting order defined in the query in the **Data Source Definition** tab will be used.

- **Graphic Title**: Optionally enter a title that will be displayed on top of the chart.

- **X Title**: For line, spline, waterfall or bar charts, enter the title of the x-axis.

Pie charts and radar charts do not have an x-axis title. The elements in the pie and the radii of the radar chart are labelled with the strings defined as x-axis values.

- **X Value Format**: This field is only relevant if a property of the type date/time is used on the x-axis and a `SetColumnFormat` instruction was used in the data source specification for the property. Enter the format specification specified in the `SetColumnFormat` instruction into the **X-Value Format** field.

- **Y Title**: For line, spline, waterfall, area, spline area and bar charts, enter the title of the y-axis.

Pie charts, multi-level pie charts and radar charts do not have an y-axis title. The size of the elements in the pie and the position of the value bubbles on the radii of the radar chart represent the y values.

- **Width**: Define the width of the graphic by defining the width in pixel. Allowed values are between 1 and 1000. The default value is 500.
- **Height**: Define the height of the graphic by defining the height in pixel. Allowed values are between 1 and 1000. The default value is 400.

  Note the following recommendations for the definition of the **Width** and **Height** attributes:

  - The graphic is displayed in the defined size independent from the size of the client device's screen. If the screen size is not sufficient to display the whole graphic, scroll bars are added to the graphic. In order to ensure that business graphics can be adequately viewed on a variety of devices, it is recommended that graphic reports and object cockpits are designed for a low screen resolution (such as 1280 x 800 or 1280 x 1024).
  - For small display sizes, the graphic views embedded in object cockpits and console reports should be vertically positioned rather than horizontally.
  - For configured reports displaying a pie chart: The dimensions for width to height should be 1:3, whereby the width should be a minimum of 500 pixel. If the pie chart is embedded in an object cockpit, an additional 60 pixel should be added to the height if navigation is supported in the pie chart. An additional 40 pixel should be added to the height if navigation is not supported.
  - For configured reports displaying a radar chart: The dimensions for width to height should be 1:2, whereby the width should be a minimum of 500 pixel. If the pie chart is embedded in an object cockpit, an additional 60 pixel should be added to the height if navigation is supported in the pie chart. An additional 40 pixel should be added to the height if navigation is not supported.
  - For configured reports displaying a bar, stacked bar, waterfall, area, spline area, spline, or line chart: The dimensions will depend on the displayed content. If a bar, stacked bar, waterfall, area, spline area, spline, or line chart is embedded in an object cockpit, an additional 60 pixel should be added to the height if navigation is supported in the chart. An additional 40 pixel should be added to the height if navigation is not supported.

4) In the **Additional Attributes** tab, define the following attributes:

   You will only see the attributes that apply to the type of chart that you selected in the **Business Graphic Definition** tab.

- **Show Tootips**: Defines whether tooltips are displayed when the user moves the mouse over a graphic element in the chart.
- **Show Legend**: Defines whether a legend is available directly beneath the chart. The availability of a legend in the floating toolbar is not affected by this setting. For waterfall charts, a legend is meaningless, and the floating toolbar will not show a legend.
- **Decimals**: An integer that defines the maximum number of decimal values that are displayed for numbers in the chart.

- **Number Prefix**: A string that will be displayed in front of all numbers in the chart. This value can be set to display values For example, with a currency symbol in front without converting the numbers into a string on the SQL layer.
- **Number Suffix**: A string that will be displayed behind all numbers in the chart. This value can be set to display values For example, with a currency symbol behind them without converting the numbers into a string on the SQL layer. If you want number and text to be separated by a whitespace, the whitespace must be added to the string definition.
- **Value Tag Color**: Values can be displayed as label on graphic elements. Depending on the coloring of the graphic, text in the default color for text on the Alfabet interface may not be readable in the graphic elements of the chart. Therefore, for all charts except pie charts and doughnut charts the text color for the labels can be changed by selecting a color with this attribute.
- **Maximum Value**: For all charts except pie charts, doughnut charts and multi-level pie charts, a maximum value can be assigned to the y-axis of the chart. If no maximum value is defined, the maximum value of the y-axis is derived from the highest y value in the dataset. Definition of a fixed maximum value can For example, be used to streamline the y-axis definitions of multiple charts for comparison reasons. If an y value in the result dataset of the report exceeds the defined maximum value, the defined maximum is ignored and the y-axis is extended to display all values in the dataset.
- **Minimum Value**: For all charts except pie charts, doughnut charts and multi-level pie charts, a minimum value can be assigned to the y-axis of the chart. If no minimum value is defined, the minimum value of the y-axis is 0 or, if negative values are displayed, derived from the lowest y value in the dataset. Definition of a fixed minimum value can For example, be used to highlight differences if values are all in a high range and differences would not be clearly visible if values are displayed on a scale starting from 0. If an y value in the result dataset of the report is lower than the defined minimum value, the defined minimum is ignored and the y-axis is extended to display all values in the dataset.

For bar, line, area, spline, spline area and combination reports only:

- **Canvas Background Color**: The color used instead of light grey for coloring of the background of the business chart graphic. The coloring of the canvas is done in differently shaped stripes to highlight the y-axis scale. The selected color will be used in combination with a lighter shade of the same color.
- **Canvas Background Opacity**: The opacity of the background color defined as integer between 0 and 100 to specify the percentage of opacity that shall be applied to the coloring defined with **Canvas Background Color**. By default, **Canvas Background Opacity** is 100%.
- **X-Axis Color**: The color used for display of the x-axis if **X-Axis Thickness** is set to a value higher than zero.
- **X-Axis Thickness**: The width of the line drawn on the x-axis. By default, **X-Axis Thickness** is set to zero and no line is drawn.
- **Y-Axis Color**: The color used for display of the y-axis if **Y-Axis Thickness** is set to a value higher than zero.
- **Y-Axis Thickness**: The width of the line drawn on the y-axis. By default, **Y-Axis Thickness** is set to zero and no line is drawn.

For bar, area, spline area and combination reports only:

- **Plot Border Color**: The color used for display of the border of bars or areas if **Plot Border Thickness** is set to a value higher than zero.
- **Plot Border Thickness**: Define the width of the line drawn around areas or bars. By default, **Plot Border Thickness** is set to 1 and the bars or areas are having a thin border.

For waterfall charts only:

- **Negative Color**: The color used for display of bars representing negative values in the waterfall chart. This attribute is only valid if the attribute **Color Column** in the **Business Graphic Definition** tab is not set.
- **Positive Color**: The color used for display of bars representing positive values in the waterfall chart. This attribute is only valid if the attribute **Color Column** in the **Business Graphic Definition** tab is not set.
- **Show Connectors**: If set to `True`, a dotted line is displayed as connector between the end of a bar and the start of the next bar.
- **Connector Color**: The color used for display of the dotted line between bars.
- **Sum Info Column**: The name of the column in the dataset that contains the information whether this column is a normal column representing results, or a `SUM` column that displays the sum of all columns to the left of it up to the last `SUM` column, or a `CUMSUM` column that displays the sum of al columns representing data left to it no matter how many `SUM` colums are located left of it. The column in the dataset must return the string `SUM` for `SUM` columns, `CUMSUM` for `CUMSUM` columns and must be either empty or set to any other value for result columns. For more information, see Configuring Display of the Sum of Multiple Columns in Waterfall Chart Reports.

For pie charts and doughnut charts only: A sum of the amounts represented by the slices in the pie chart can be calculated and displayed as subheading beneath the graphic title. The subtitle is a string that is concatenated from the text defined with the following attributes and the calculated value in the following way:

<Total Prefix> <calculated value> <Total Unit of Measure> <Total Suffix>

A whitespace is added automatically between the defined parts of the text. The text is only displayed if at least one of the following values is defined:

- **Total Prefix**: A text to be displayed preceeding the calculated sum.
- **Total Suffix**: A text to be displayed following the **Total Unit of Measure**.
- **Total Unit of Measure**: The unit that is applicable to the sum. The **Total Unit of Measure** is displayed following the calculated sum.

For pie charts, doughnut charts and multi-level pie charts:

- **Show Percent Values**: If set to `True`, the percentage of the value for the element on the overall amount of values displayed in the circle is displayed on the element instead of the value the element represents. The value represented by the element is then displayed in the tooltip. If set to `False`, the value that the element represents is displayed on the element and the percentage value is displayed in the tooltip.

- **Pie Radius**: Defines the outer radius of the pie chart, doughnut chart or multi-level pie chart in pixel. If set to zero, the chart is automatically sized to fit into the size defined for the graphic.
- **Doughnut Radius**: For dougnut charts only, the inner radius of the circle, that means the radius of the hole in the center of the chart can also be defined in pixel.

For multi-level pie charts only:

- **Pie Border Color**: The color used for display of the borders of the elements of the multi-level pie chart.
- **Pie Border Thickness**: An integer to define the width of the borders of the elements in the multi-level pie chart. The standard width is 1.

5) Click **OK** to save your settings.

6) In the toolbar, click the **Save** button to save your changes.

An example for the configuration of a bar chart based on Alfabet query language is provided in the section Configuring Color Definition per Alfabet Object. An example for the configuration of a bar chart based on native SQL is provided in the section Configuring Navigation to a Defined View.

An example for the configuration of a pie chart based on native SQL is provided in the section Configuring Color Definition per Graphic Element.

An example for the configuration of a radar chart based on Alfabet query language is provided in the section Configuring Navigation to the Object's Profile.

Configuring Fixed Color Assignment

In configured chart reports, colors are by default assigned randomly to the objects in the report. There are two methods to influence the color coding used in chart reports:

- The report can be defined to use object colors. If a color is assigned to an object, this object is then displayed in the defined color in all standard chart and portfolio reports on the Alfabet interface and in all configured business chart reports that are configured to use object specific coloring. The color of individual objects can be changed by the user if the **Colors** button of the report is enabled.
- Colors for the graphic objects in the chart report can be defined within the result dataset. This method allows to streamline the coloring of all chart reports displayed For example, in the same object cockpit or to assign fixed colors to graphic elements not representing Alfabet objects.

For waterfall reports only, a third method for applying color is available. Two colors can be defined for the waterfall charts, one for positive and one for negative values. The report will then show all negative values, independent from the series they belong to, in the color defined for negative values and all positive values in the color defined for positive values. The color definition is done in the tab **Additional Attributes** with the attributes **Negative Color** and **Positive Color**. The defined positive and negative colors are only used if the **Color Column** attribute is not set.

The color definitions can be activated or deactivated per configured chart report:

Configuring Color Definition per Alfabet Object

A color can be assigned to single objects of any object class with a property ***Color***. If a color is assigned to an object, this color is used for the object in all standard portfolio and chart reports in Alfabet. Whether the object color is used in configured chart reports depend on the configuration of the report.

The assignment of colors to objects can be performed on the Alfabet interface by two different methods:

- The color can be assigned via a color selector in a custom editor for the respective object class. The color may than be assigned to the object by a user with Read/Write access to the object and access to the custom editor.
- The color can be assigned by any user opening a report via the the **Colors** button in the toolbar of the report. Whether a **Colors** button is available in the toolbar of a standard report or configured report and whether the button is active or disabled, depends on the configuration of the Alfabet Server and the view scheme assigned to the user profile used when opening the report.

For information about the configuration required to activate color setting per object, see *Allowing Users to Change the Color Assigned to Objects in Business Graphics* in the reference manual *System Administration*.

When color setting is activated, the colors assigned to objects are only used in a configured chart report if the graphic elements in the report can be assigned to an object and the report is configured to identify the object behind the graphic.

The following configuration is required:

- In the data source of the chart report, each column in the dataset must be related with an object.

When the report is based on an Alfabet query, each column of the dataset is automatically assigned to the respective object of the `FIND` class of the Alfabet query.

For native SQL queries, the first SELECT argument of the query must return the REFSTR property of an object class. This column is removed from display of the result dataset but nevertheless associates the dataset with an object class.

> When configuring chart reports, it is possible to define `NULL` as `REFSTR` in the first `SELECT` argument. For an example, see Configuring Color Definition per Graphic Element. Therefore, in native SQL queries the graphic elements in a chart are not necessarily related with an object.

The assignment of an object to each row in the report can also be performed by adding an additional row to the dataset that returns the `REFSTR` of an object related to the displayed results.

- The attribute **Object Column** is set in the **Business Graphic Definition** tab of the **Report Assistant** and specifies the column containing the `REFSTR` of the differently colored objects. When you select `Row Reference` in the drop-down list of the **Object Column** attribute, the `REFSTR` of the `FIND` class

of the Alfabet query or the first `SELECT` argument in the native SQL query are used to assign an object to the graphic.

- The attribute **Color Column** is NOT set in the **Business Graphic Definition** tab of the **Report Assistant**. This attribute activates coloring according to a color specification in the result dataset of the query. It supersedes an **Object Column** setting.

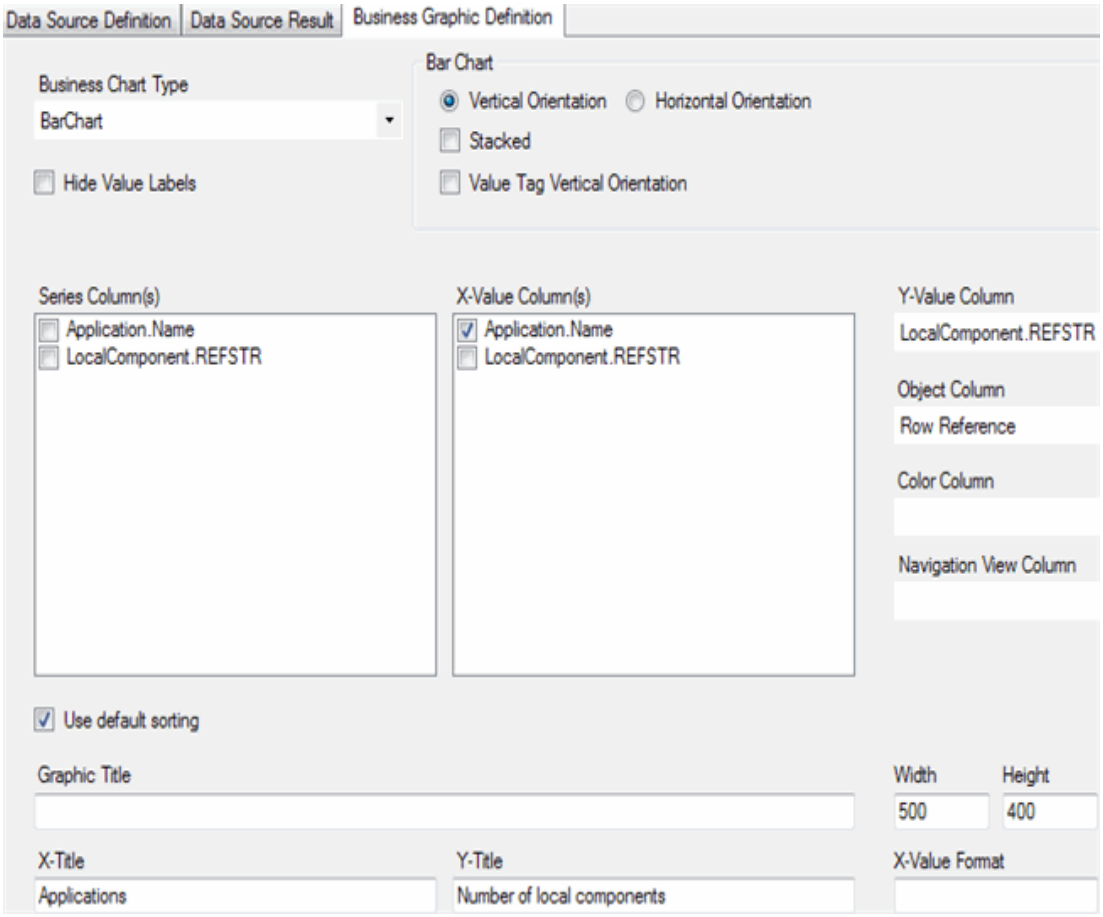
For example, a bar chart is configured to display applications in a selected application group on the x-axis and the number of local components assigned to each application on the y-axis. The configured report is assigned to the object class Application Group with the **Apply To Class** attribute to allow selection of the application group. The following Alfabet query is defined as data source:

```
ALFABET_QUERY_500

FIND Application

InnerJoin LocalComponent ON LocalComponent.Owner = Application.REFSTR

WHERE Application.ApplicationGroups CONTAINS:BASE

Instructions

JoinColumns("Application.Name,Application.Version","Application.Name",
" v.");

EndOfInstructions

QUERY_XML

<QueryDef>

<ShowProperty Type="Property" ClassName="Application" Name="Name" />

<ShowProperty Type="Property" ClassName="Application" Name="Version"
/>

<ShowProperty Type="Property" ClassName="LocalComponent" Name="REFSTR"
Function="COUNT" />

</QueryDef>
```

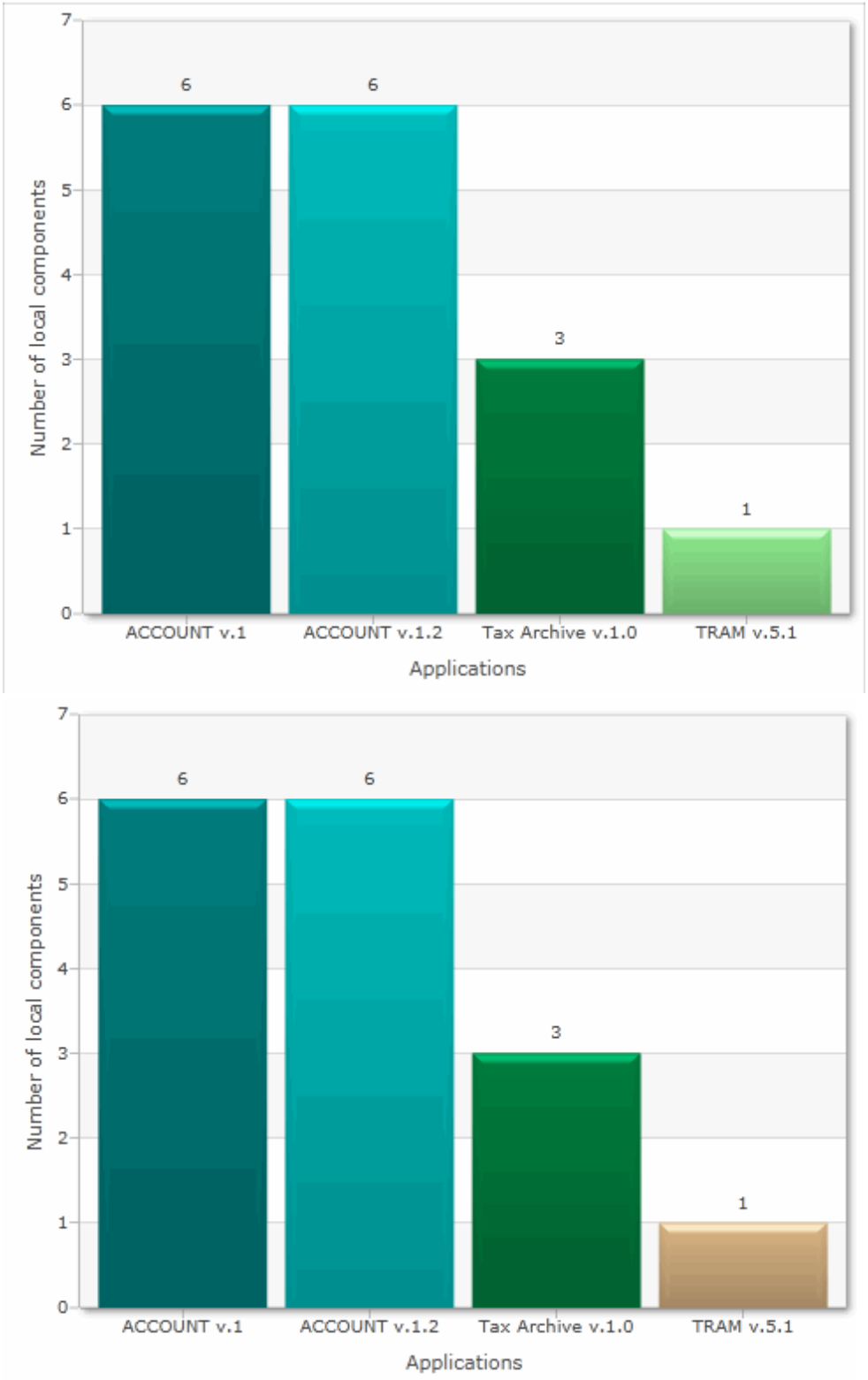
The result dataset has two columns. One column returning the name and version number of the application (column `Application.Name`) and one that returns the number of local components assigned to the respective application (column `LocalComponent.REFSTR`):

Application.Name	LocalComponent.REFSTR
Business EAI Platform v.2.2	1
Groupware Services v.2.2	1
Mafo-Portal v.2.6	12
PS Global v.2.5	1
ALLFinance PISA v.2.9	2
SAP@OptiRetail v.2.0	2
SAP@AI v.4.0	25
BookIT v.2.9	2
Corporate FI-CO v.2.2	29

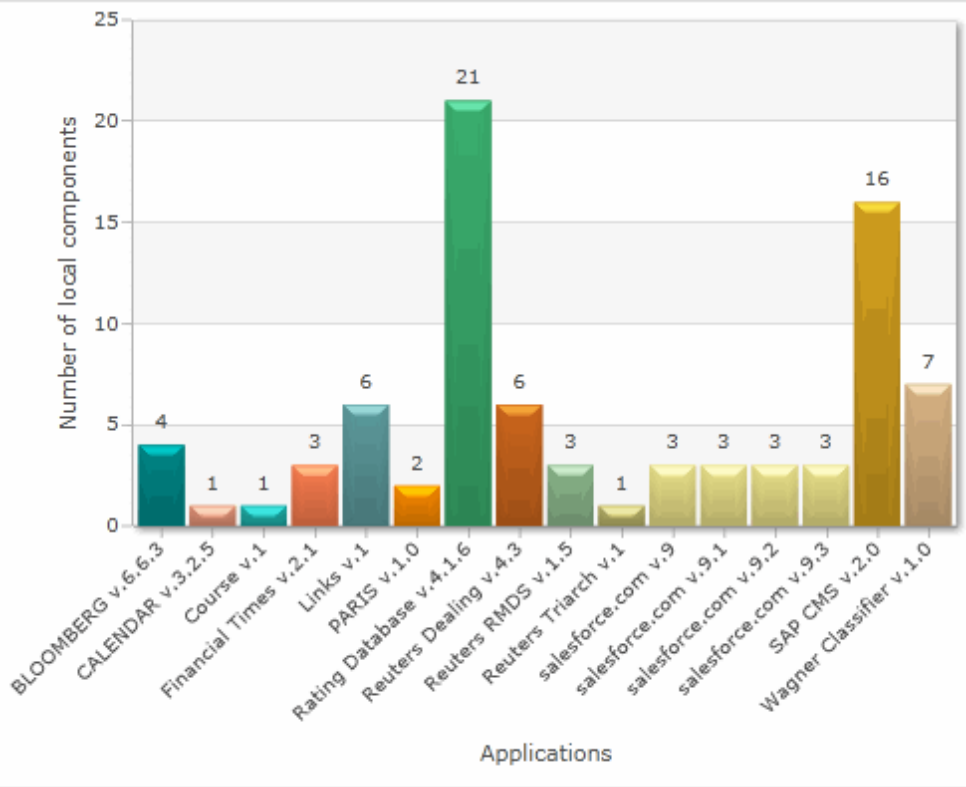
The color defined for single applications shall be applied to the bar chart. The object class Application is the `FIND` class of the data source query and must be set in the **Object Column** definition of the **Business Graphic Definition** of the report to identify the application assigned to each graphic element:



In the resulting report, applications that have a color defined are displayed in the same color in each result dataset while the color of applications with no color defined is assigned randomly at each opening of the report. In the following example, a color is defined for all application except for the application TRAM v. 5.1. Therefore, when the report is opened a second time, the bars representing the applications with a color assigned are displayed with the assigned color while the color of the bar representing the application without color assignment is selected randomly.



Alternatively, a new dimension of information can be added to the report by coloring all applications that are assigned to the same ICT object with the same color. In the example below, all application versions named `salesforce.com` are assigned to the same ICT object and are Therefore, displayed with the same color:

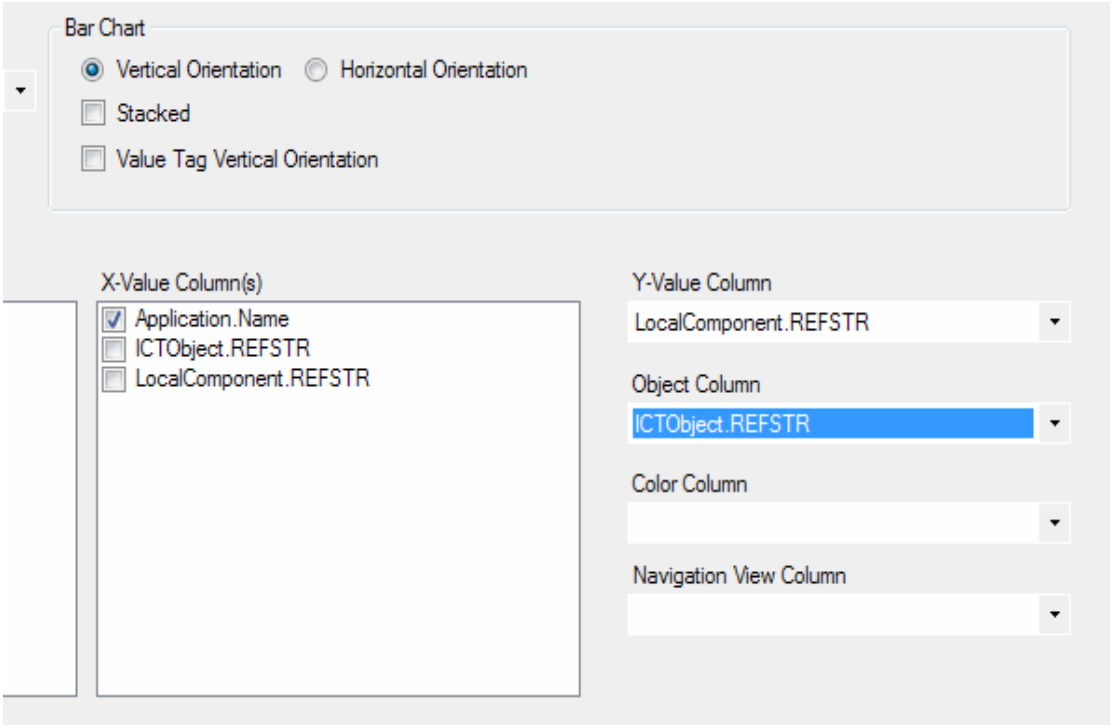


To configure the report to assign the color to the columns depending on the ICT object the applications are assigned to, the query must be extended to find the ICT object the application is assigned to and return the `REFSTR` of the ICT object in the result dataset:

```
ALFABET_QUERY_500

FIND Application

InnerJoin LocalComponent ON LocalComponent.Owner = Application.REFSTR

InnerJoin ICTObject ON Application.ICTObject = ICTObject.REFSTR

WHERE Application.ApplicationGroups CONTAINS:BASE

Instructions

JoinColumns("Application.Name,Application.Version","Application.Name",
" v.");

EndOfInstructions

QUERY_XML

<QueryDef>

<ShowProperty Type="Property" ClassName="Application" Name="Name" />

<ShowProperty Type="Property" ClassName="Application" Name="Version"
/>

<ShowProperty Type="Property" ClassName="ICTObject" Name="REFSTR" />

<ShowProperty Type="Property" ClassName="LocalComponent" Name="REFSTR"
Function="COUNT" />

</QueryDef>
```


Application.Name	ICTObject.REFSTR	LocalComponent.REFSTR
Business EAI Platform v.2.2	26-126-0	1
Groupware Services v.2.2	26-129-0	1
Mafo-Portal v.2.6	26-68-0	12
PS Global v.2.5	26-94-0	1
ALLFinance PISA v.2.9	26-105-0	2
SAP@OptiRetail v.2.0	26-21-0	2
SAP@AI v.4.0	26-23-0	25
BookIT v.2.9	26-27-0	2
Corporate FI-CO v.2.2	26-41-0	29
SAP International v.2.8	26-43-0	11
compas v.2.2	26-14-0	6

The **Object Column** attribute is set to the column containing the `REFSTR` of the ICT object:



Configuring Color Definition per Graphic Element

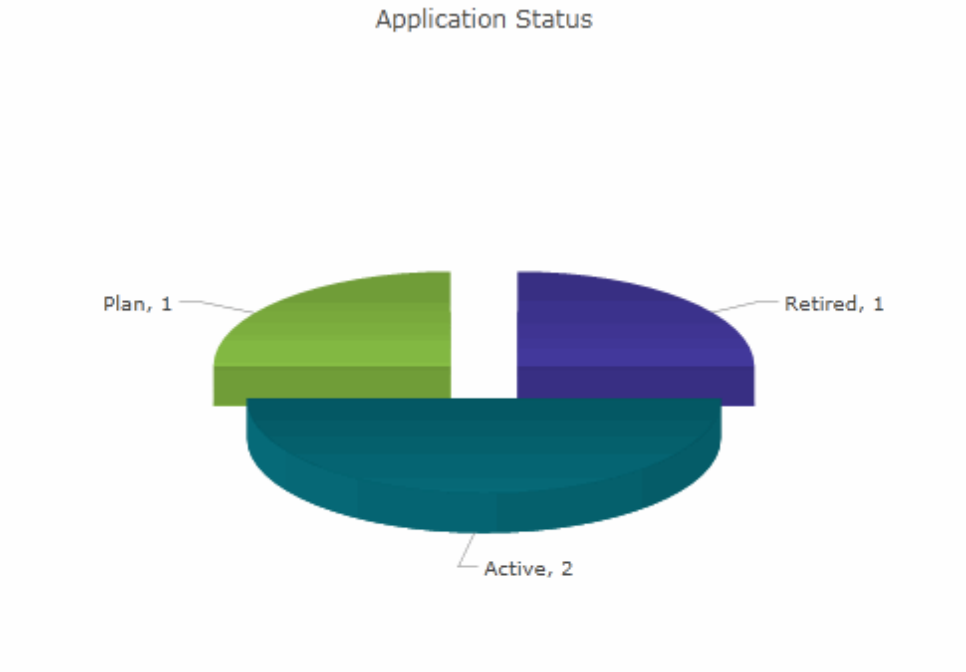
The colors used to display the elements in a configured chart report can be defined within the dataset of the data source query for the chart report.

This requires the following configuration:

- The data source query must be defined to return a dataset with a column containing a color specification in HTML compliant color coding.
- The column containing the color definitions must be selected in the attribute **Color Column** of the **Business Graphic Definition** tab of the **Report Assistant**.

When both a **Color Column** and an **Object Column** are defined, the color coding defined by the **Color Column** definition is used for color definition.

For example, a configured pie chart report report displays the number of applications assigned to an ICT object that are in the object state **Active**, **Plan**, or **Retired** and the coloring of the pie elements shall match a given color scheme.



The configured report is applied to the object class ICT object as base class with the attribute **Apply to Class**.

The data source query of the configured report is a native SQL query that combines three datasets, each counting applications assigned to the base ICT object that are in one of the given object states. Each `SELECT` statement results in one row in the result dataset.

The rows in the result dataset are not related to a single Alfabet object, and Therefore, the first argument in each `SELECT` statement, that has to return the `REFSTR` of the current object, is set to `NULL`.

```
SELECT NULL As 'REFSTR', 'Active' As 'ObjectState', COUNT(*) As
'Applications', '#076D7B' As 'Color'

FROM APPLICATION app

WHERE app.OBJECTSTATE = 'Active'

AND app.ICTOBJECT=@BASE

UNION ALL

SELECT NULL As 'REFSTR', 'Plan' As 'ObjectState', COUNT(*) As
'Applications', '#8AC145' As 'Color'

FROM APPLICATION app

WHERE app.OBJECTSTATE = 'Plan'

AND app.ICTOBJECT=@BASE

UNION ALL
```

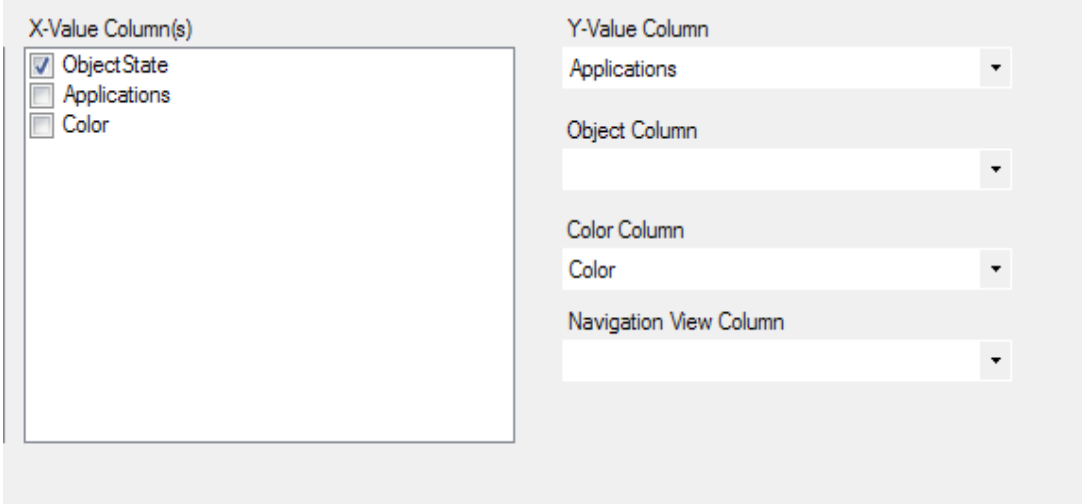


```
SELECT NULL As 'REFSTR', 'Retired' As 'ObjectState', COUNT(*) As
'Applications', '#453AA1' As 'Color'

FROM APPLICATION app

WHERE app.OBJECTSTATE = 'Retired'

AND app.ICTOBJECT=@BASE;
```

The result dataset contains three columns:

ObjectState	Applications	Color
Active	266	#076D7B
Plan	15	#8AC145
Retired	53	#453AA1

- ***ObjectState:*** This column contains the name of the object state, defined as string in the `SELECT` statement. In the **Business Graphic Definition** tab, the column is selected as **X-Value Column** of the pie chart.
- ***Applications:*** This column contains the count of applications in the Alfabet database matching the search criteria. The column is selected as **Y-Value Column** of the pie chart.
- ***Color:*** This column contains a color code, defined as string in the `SELECT` statement. Each element in the graphic shall be colored in the defined color code. The column is selected as **Color Column** of the pie chart.



Configuring Navigation from a Chart Report

Configured busines chart reports can provide navigation from a graphic element in the report. When the user clicks the graphic element, a preview window opens with a button **Show Details**. Clicking on the button can open one of the following, according to the configuration of the configured report:

- If the graphic elements in the chart represent Alfabet objects, the user can navigate to the object profile of the objects represented in the report.

- Navigation targets for the graphic objects in the chart report can be defined within the result dataset. This method allows to open any object profile, configured report or standard Alfabet view from a graphic report. Parameters can be defined in the target definition to pre-set filters in the report that opens.

These functionalities can be activated or deactivated per configured chart report via the following configurations:

- Configuring Navigation to the Object's Profile
- Configuring Navigation to a Defined View

Configuring Navigation to the Object's Profile

Navigation to the object's profile from the report requires the following configurations:

- In the data source of the chart report, each column in the dataset must be related with an object.

When the report is based on an Alfabet query, each column of the dataset is automatically assigned to the respective object of the `FIND` class of the Alfabet query.

For native SQL queries, the first `SELECT` argument of the query must return the `REFSTR` property of an object class. This column is removed from display of the result dataset but nevertheless associates the dataset with an object class.

When configuring chart reports, it is possible to define `NULL` as `REFSTR` in the first `SELECT` argument. For an example, see Configuring Color Definition per Graphic Element. Therefore, in native SQL queries the graphic elements in a chart are not necessarily related with an object.

The assignment of an object to each row in the report can also be performed by adding an additional row to the dataset that returns the `REFSTR` of an object related to the displayed results.

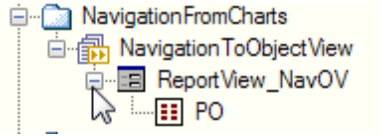
- The attribute **Object Column** is set in the **Business Graphic Definition** tab of the **Report Assistant** and specifies the column containing the `REFSTR` of the objects for which the object profile shall be the navigation target. When you select `Row Reference` in the drop-down list of the **Object Column** attribute, the `REFSTR` of the `FIND` class of the Alfabet query or the first `SELECT` argument in the native SQL query are used to assign an object to the graphic.
- The attribute **Navigation View Column** is NOT set in the **Business Graphic Definition** tab of the **Report Assistant**. This attribute activates navigation to a target view defined in the result dataset of the query. It supersedes an **Object Column** setting.
- The **Navigate**  button is activated for the report. By default, the **Navigate**  button is deactivated for a new configured report. Activation of the button is required for technical reasons to activate the navigation functionality. In the report, the button is not displayed in the toolbar, but navigation can be performed by clicking a graphic element in the chart.

Note the following about the **Navigate**  button:

- If you hide the inactivated **Navigate** button in the toolbar via configuration of the view scheme used for the user's profile, the hyperlinks to navigate to objects are not displayed any longer.
- The setting of the **Object Column** attribute is also required to use coloring defined for single objects in the graphic. If you want to define coloring without defining navigation to the object, you can set the **Object Column** attribute without activating the **Navigate** button.

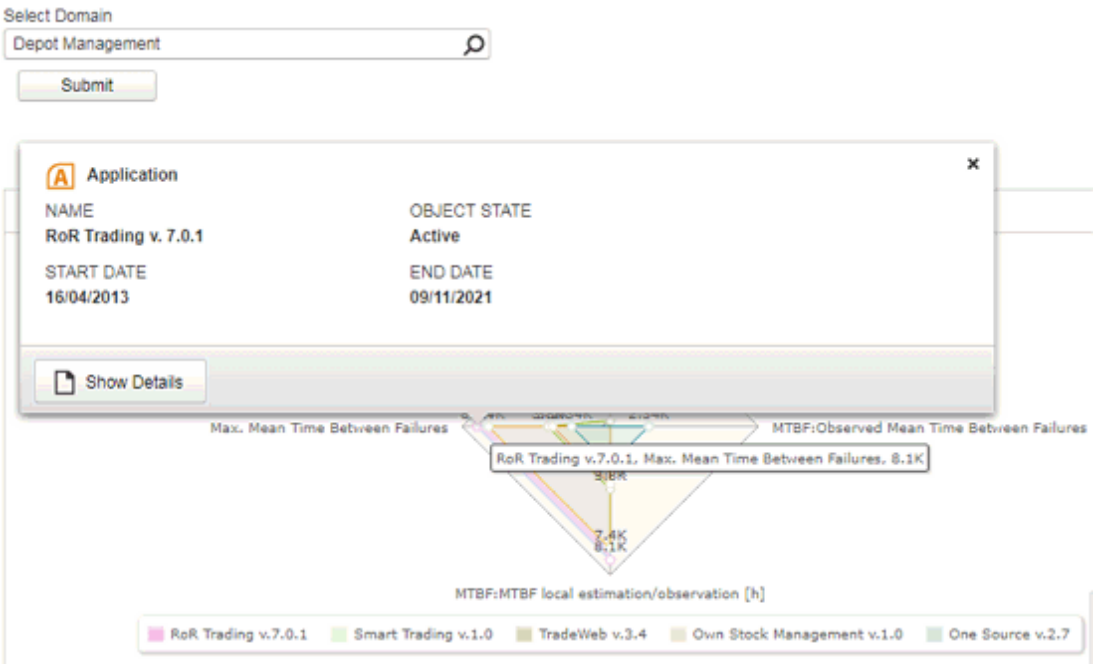
Do the following to activate the navigate button:

1) In the explorer of the Reports tab of Alfabet Expand, expand the Custom Report View's Node of the chart report for that you want to activate the **Navigate** button.



2) Click the `PO` element under the Custom Report View's Node. The attributes of the `PO`-element are displayed in the attribute window.
3) In the **Buttons** section of the attribute window, set the attribute **Has Navigate Button** to `True`.

For example, a radar chart is configured to show the indicators regarding failure times for applications assigned to a domain selected as base class of the report. clicking a node in the report not only shows the value for the selected indicator for the selected application as a tooltip, but also leads to display of a preview window. Clicking on the button **Show Details** in the preview opens the application's object view.



The report is based on the following Alfabet query:

```
ALFABET_QUERY_500

FIND Application

InnerJoin Indicator ON Indicator.Object = Application.REFSTR

InnerJoin EvaluationType ON EvaluationType.REFSTR =
Indicator.EvaluationType

WHERE (AND

Application.Domain CONTAINS:BASE

EvaluationType.Name LIKE 'MTBF%')

Instructions

JoinColumns("Application.Name,Application.Version","Application.Name
", " v.");

EndOfInstructions

QUERY_XML

<QueryDef>

<ShowProperty Type="Property" ClassName="Application" Name="Name" />

<ShowProperty Type="Property" ClassName="Application" Name="Version"
/>

<ShowProperty Type="Property" ClassName="Indicator" Name="Value" />

<ShowProperty Type="Property" ClassName="Indicator" Name="Name" />

<ShowProperty Type="Property" ClassName="EvaluationType" Name="Name"
/>

</QueryDef>
```

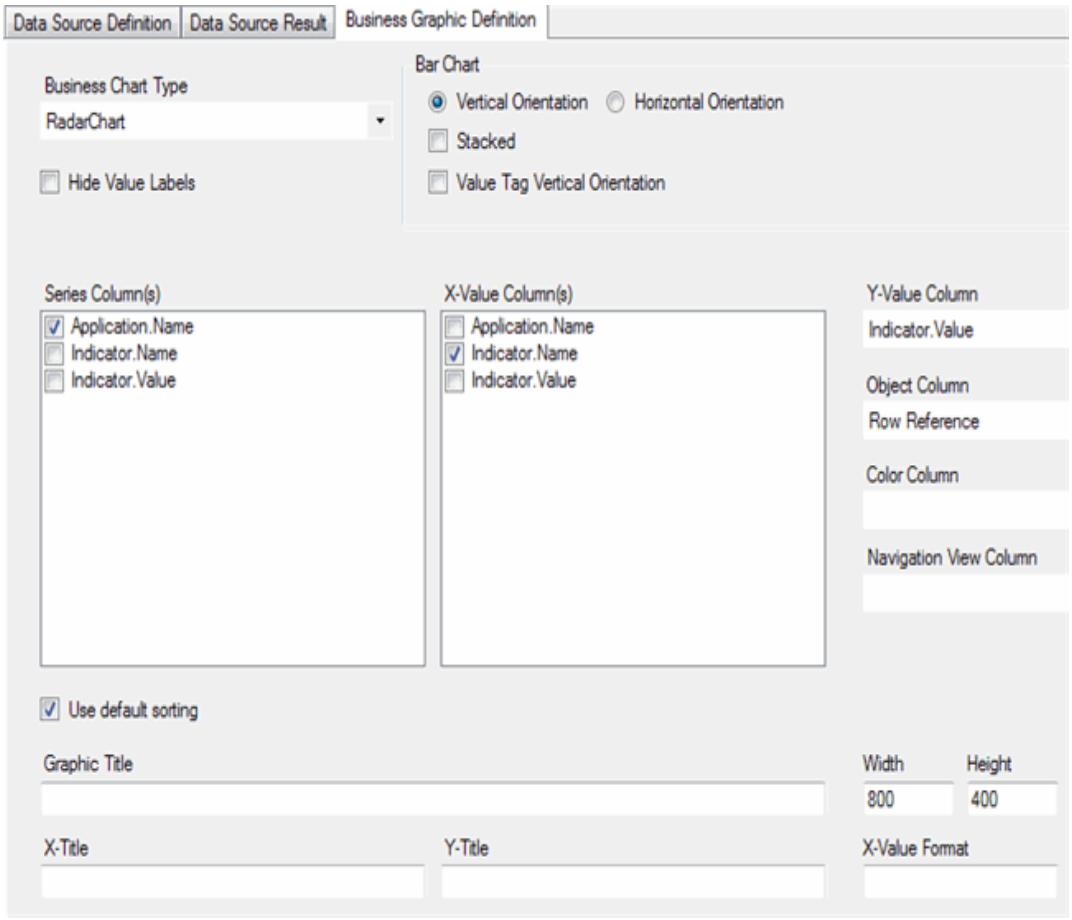
The query results in the following dataset:

Application.Name	Indicator.Name	Indicator.Value
Business EAI Platform v.2.2	Max. Mean Time Between Failures	5,700.00
Business EAI Platform v.2.2	MTBF:MTBF local estimation/observation [h]	5,700.00
Groupware Services v.2.2	Aggregated Mean Time Between Failures [h]	6,300.00
Groupware Services v.2.2	Max. Mean Time Between Failures	6,300.00
Mafo-Portal v.2.6	Aggregated Mean Time Between Failures [h]	1,577.33
Mafo-Portal v.2.6	Max. Mean Time Between Failures	1,577.33
PS Global v.2.5	Max. Mean Time Between Failures	7,600.00
PS Global v.2.5	MTBF:MTBF local estimation/observation [h]	7,600.00
ALLFinance PISA v.2.9	Max. Mean Time Between Failures	8,500.00
ALLFinance PISA v.2.9	MTBF:MTBF local estimation/observation [h]	8,500.00
SAP@OptiRetail v.2.0	Aggregated Mean Time Between Failures [h]	123.53
SAP@OptiRetail v.2.0	Max. Mean Time Between Failures	123.53
SAP@AI v.4.0	Max. Mean Time Between Failures	1,000.00
SAP@AI v.4.0	MTBF:MTBF local estimation/observation [h]	1,000.00
BookIT v.2.9	Max. Mean Time Between Failures	5,600.00
BookIT v.2.9	MTBF:MTBF local estimation/observation [h]	5,600.00

The radar chart has three dimensions.

- The x-axis corresponds to the radii of the radar chart. There is one radius per indicator. The column `Indicator.Name` defines the radii and which result is applicable to which radius.
- The y-axis corresponds to the values on the radii. The values are defined in the column `Indicator.Value`.
- The report displays indicator values for multiple applications. The application are the third dimension that defines which indicator value belongs to which graphic net element within the radar chart. The `Application.Name` column is Therefore, selected as series column.

The report shall also trigger the navigation to the application's profile. Application is the `FIND` class of the Alfabet query the report is based on and Therefore, the attribute **Object Column** of the report can be set to `RowReference` to identify the application that is associated with the value the user clicks on to display the navigation link.



In addition to the configuration in the **Business Graphic Definition** tab of the **Report Assistant**, the navigation to the object's profile is activated in the report by activation of the **Navigate** button in the attributes of the `PO` node of the Custom Report View.

Configuring Navigation to a Defined View

Navigation to a defined standard view, configured report or object profile from the configured chart report requires the following configurations:

- In the data source of the chart report, a column must be added for the definition of the navigation target.

The syntax required for definition of the view is the following:

```
View=ViewType:ViewName
```

The `ViewType` can be one of the following:

- ***Report*** for a configured report
- ***GraphicView*** for a standard Alfabet view
- ***ObjectView*** for an object profile

`ViewName` is the name of the view.

If the navigation target is a configured report, parameters values can be handed over to the view that opens.

The query of the report must contain each parameter definition as a variable defined as

```
@ParameterName
```

Please note that the old syntax of the Alfabet query language defining parameters as:`ParameterName` is not supported.

The parameter @BASE that refer to the base object for that the link is opened cannot be changed via parameter settings.

The parameters can be used in WHERE clauses of the query without defining a filter for the report, because the parameter values are already defined in the navigation link.

Optionally, filters can be defined for the report to allow the user opening the report to alter the parameter values. In this case the parameter values in the navigation link are used as default values for the filter when opening the report. The filter settings are also stored in the user context settings for the report. That means, when the user first opens the report via drilldown from a chart report and then afterwards opens the report in any other context, For example, via the **Reports** functionality, the report will open with the last filter settings derived from the drill-down.

For each parameter the following string must be added to the navigation target definition:

```
&ParameterName=ParameterValue
```

The setting of three parameters would result in the following navigation link:

```
View=ViewType:ViewName&ParameterName1=ParameterValue1&ParameterName2=Para
meterValue2&ParameterName1=ParameterValue2
```

.

The assignment of an object to each row in the report can also be performed by adding an additional row to the dataset that returns the `REFSTR` of an object related to the displayed results.

- The attribute **Navigation View Column** is set in the **Business Graphic Definition** tab of the **Report Assistant** and specifies the column containing the navigation target definition.

The **Navigation View Column** supersedes the navigation setting of the **Object Column** attribute that can be set both to trigger navigation to the object view of objects represented by the graphic elements in the chart and to trigger coloring according to the fixed colors defined for single objects in the report. You can define reports that use coloring based on object colors and at the same time navigate to defined views in the report.

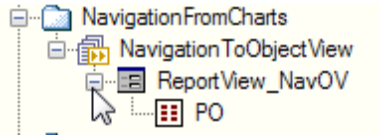
When **Object Column** is defined in parallel with **Navigation View Column**, the `REFSTR` in the **Object Column** is handed over to the navigation target defined in the **Navigation View Column** as base object that can be referred to by the parameter `BASE`.

- The **Navigate** button is activated for the report. By default, the **Navigate** button is deactivated for a new configured report. Activation of the button is required for technical reasons to activate the navigation functionality. In the report, the button remains inactive in the toolbar.

  If you hide the inactivated **Navigate** button in the toolbar via configuration of the view scheme used for the user's profile, the hyperlinks to navigate to objects are not displayed any longer.

Do the following to activate the navigate button:

1) In the explorer of the Reports tab of Alfabet Expand, expand the Custom Report View Node of the chart report for that you want to activate the **Navigate** button.



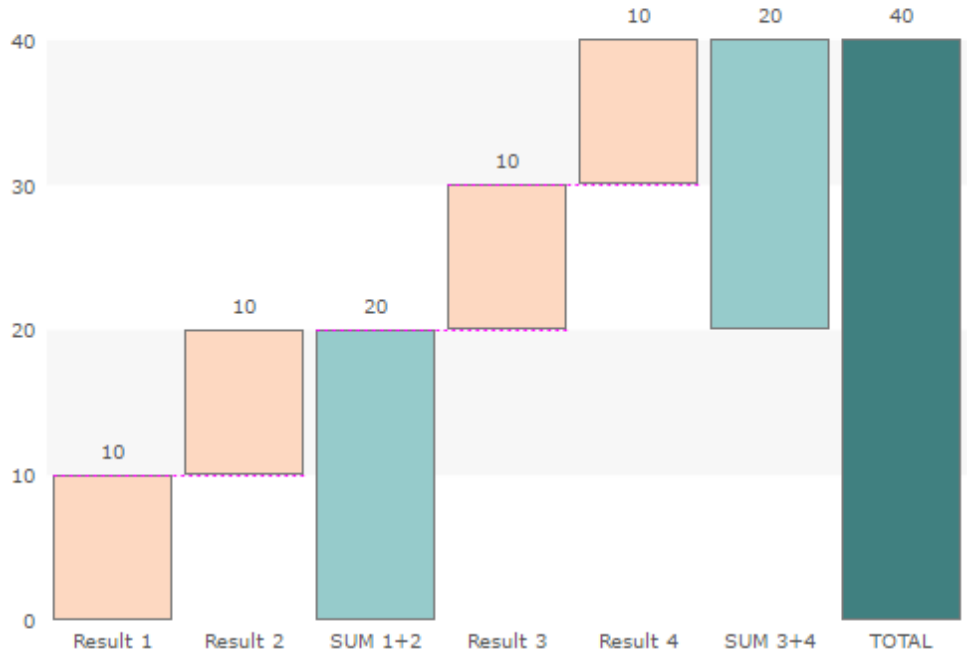
2) Click the PO element under the Custom Report View Node. The attributes of the PO-element are displayed in the attribute window.
3) In the **Buttons** section of the attribute window, set the attribute **Has Navigate Button** to `True`.

Configuring Display of the Sum of Multiple Columns in Waterfall Chart Reports

In waterfall charts, columns can be integrated into the report that sum up the values of all columns displayed left of them. There are two types of sum columns:

- `SUM` columns sum up all results starting at the last integrated `SUM` column.
- `CUMSUM` columns sum up all results independent of the number of `SUM` or `CUMSUM` columns that are integrated in the result dataset left to them.

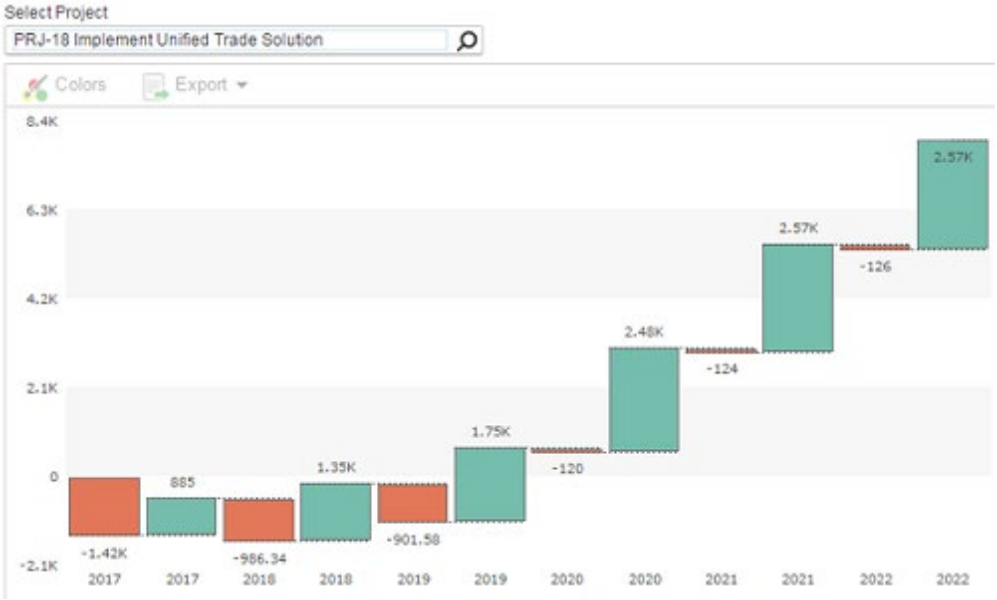
IN the following example this is demonstrated with easy result values of ten for each result column. The light green `SUM` columns added to the report sum up two result colums each, while the dark green `CUMSUM` column at the end of the report displays the sum of all values in the result columns of the report:



The inclusion of `SUM` or `CUMSUM` columns requires a change in the query definition of the configured waterfall chart report:

1) Add a new column to the result dataset of the configured report that is empty for result columns, returns `SUM` for `SUM` columns and `CUMSUM` for `CUMSUM` columns.
2) In the tab **Additional Attributes** of the business chart report assistant, enter the name of that column into the **Sum Info Column** attribute.
3) For each `SUM` or `CUMSUM` column, enter a new row to your result dataset at the position that the `SUM` or `CUMSUM` column is supposed to be displayed.

A configured waterfall chart report displays the costs and incomes resulting from a project over time. For each year two columns are displayed: one for costs and one for incomes.



The waterfall chart is based on the following query:

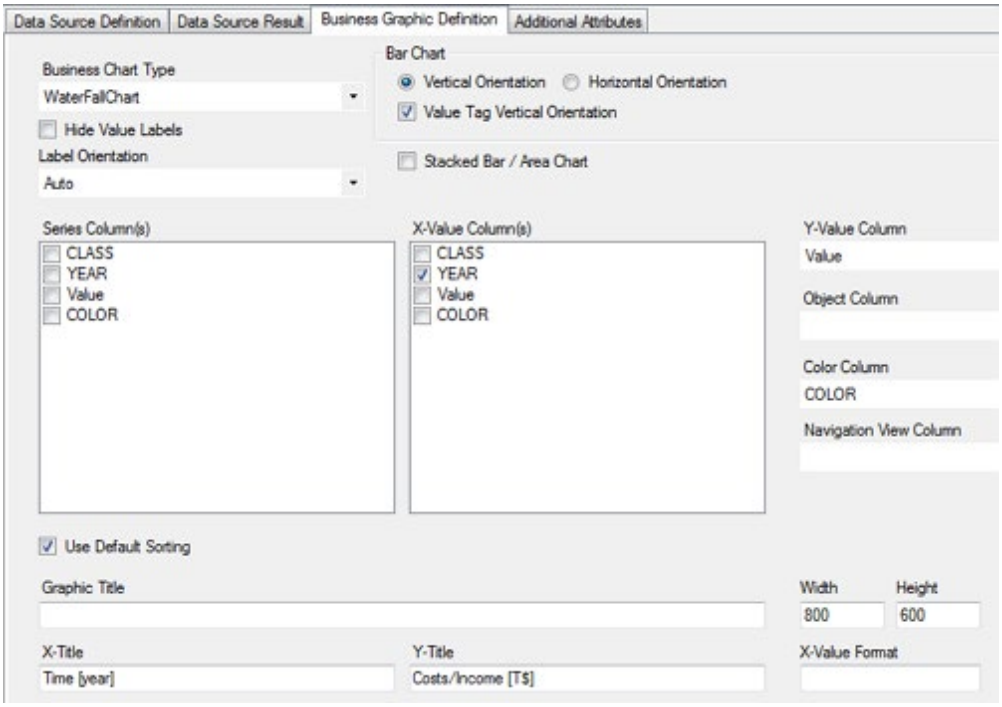
```
SELECT REFSTR,CLASS, YEAR, CASE WHEN CLASS = 'CostType' THEN -
(SUM(VALUE))ELSE SUM(VALUE)END AS Value, COLOR

FROM (

   SELECT NULL AS REFSTR, dbo.SCF_GetClassName(bv.MONETARYTYPE) AS
   CLASS, bv.YEAR, bv.VALUE, CASE WHEN
   dbo.SCF_GetClassName(bv.MONETARYTYPE) = 'CostType' THEN '#F99E68'
   ELSE '#33C0C0' END AS COLOR

   FROM BUSINESSCASE bc, BUDGETVALUE bv

   WHERE bc.OWNER = @BASE

   AND bv.OWNER = bc.REFSTR

) x

GROUP BY REFSTR, CLASS, YEAR,COLOR

ORDER BY YEAR, CLASS
```

In the Alfabet database, the values for cost types and income types are both stored as positive values. The display of costs in the negative range of the graphic is performed in the query with a CASE expression based on the monetary type of the budget value. Coloring is also depending on the monetary type. Grouping is performed to sum the values of all costs for the same object, monetary type and year. The results are ordered by year and monetary type to view the costs and incomes in the order that they will occure in the project.

The query results in the following dataset:

	CLASS	YEAR	Value	COLOR
1	CostType	2017	-1,420.31	#F99E68
2	IncomeType	2017	885.00	#33C0C0
3	CostType	2018	-986.34	#F99E68
4	IncomeType	2018	1,350.00	#33C0C0
5	CostType	2019	-901.58	#F99E68
6	IncomeType	2019	1,750.00	#33C0C0
7	CostType	2020	-120.00	#F99E68
8	IncomeType	2020	2,480.00	#33C0C0
9	CostType	2021	-124.00	#F99E68
10	IncomeType	2021	2,570.00	#33C0C0
11	CostType	2022	-126.00	#F99E68
12	IncomeType	2022	2,570.00	#33C0C0

In the **Business Graphic Definition** tab of the report assistant, the `Value` column is defined as `Y Value Column`, The `COLOR` column is defined as **Color Column** to provide coloring of the bars, and the `YEAR` column is defined as **X-Value Column(s)** . It would also be possible to use the `CLASS` column as **X-Value Column(s)**. This would change the labels on the x-axis, but the outlook of the bars would be the same.



To add `SUM` columns to the report that sum up the costs and incomes of each year and to add a cumulative sum at the end of the report, the following changes have to be performed on the query:

- An additional column must be added to the dataset resulting from the query. The column can return an empty string for the columns containing data. In the example the column is named `SUMCOL`. In the example, the string returned for the column `CLASS` is also changed to return meaningful wording because the `CLASS` definition will be used instead of the `YEAR` definition for labels on the x-axis of the waterfall chart.
- Additional rows must be added to the dataset using a `UNION ALL` command to add a `SUM` column for each year. The same query specification that is used to find the budget values

for each year is also used to add the sum columns. This method ensures that SUM columns are returned for all years for that the dataset contains budget values. In the column named SUMCOL the value SUM is returned to specify that this is a SUM column. The Value of the budget is ignored for these rows and set to '1' for all results. The CLASS column returns the string SUM together with the current year. It is important that in alphabetical sorting, the CLASS defined for the SUM column is having a higher position than the CLASS definitions for the data columns because the SUM column must be added to the end of each year.

- An additional row must be added to the dataset using a UNION ALL command to add a CUMSUM column at the end of the report. This row must return CUMSUM for the column named SUMCOL and returns the string 'TOTAL' for the column CLASS and for the column YEAR, a year that is higher than the last possible end date of a project is specified, to display the column at the end of the report. The values in all other columns are technically not taken into account.
- A sorting is performed on the top level, that means that all results gathered via UNION ALL are sorted by YEAR and then by CLASS.

This results in the following query:

```
SELECT REFSTR, CLASS, YEAR, Value, COLOR, SUMCOL

FROM (

    SELECT REFSTR,CASE WHEN CLASS = 'CostType' THEN 'Costs'ELSE
    'Profits' END AS CLASS, YEAR, CASE WHEN CLASS = 'CostType' THEN -
    (SUM(VALUE)) ELSE SUM(VALUE)END AS Value, COLOR,SUMCOL FROM (

        SELECT NULL AS REFSTR, dbo.SCF_GetClassName(bv.MONETARYTYPE)
        AS CLASS, bv.YEAR, bv.VALUE, CASE WHEN
        dbo.SCF_GetClassName(bv.MONETARYTYPE) = 'CostType' THEN
        '#F99E68' ELSE '#33C0C0' END AS COLOR,'1' AS SUMCOL

        FROM BUSINESSCASE bc, BUDGETVALUE bv

        WHERE bc.OWNER = @BASE

        AND bv.OWNER = bc.REFSTR

    ) x

    GROUP BY REFSTR, CLASS, YEAR,COLOR,SUMCOL

    UNION ALL

    SELECT REFSTR, CLASS, YEAR, '1' AS Value, COLOR,SUMCOL

    FROM (

        SELECT NULL AS REFSTR, 'SUM ' + CAST (bv.YEAR AS varchar) AS
        CLASS, bv.YEAR, '1' AS VALUE, '#C0C0C0' AS COLOR, 'SUM' AS
        SUMCOL FROM BUSINESSCASE bc, BUDGETVALUE bv

        WHERE bc.OWNER = @BASE

        AND bv.OWNER = bc.REFSTR

    ) x

    GROUP BY REFSTR, CLASS, YEAR, COLOR,SUMCOL

    UNION ALL

    SELECT NULL AS REFSTR, 'TOTAL ' AS CLASS, '2030' As YEAR, '1' As
    Value, '#408080' AS COLOR, 'CUMSUM' AS SUMCOL
```



```
) x
ORDER BY YEAR, CLASS
```

The query returns the following dataset:

	CLASS	YEAR	Value	COLOR	SUMCOL
1	Costs	2017	-1,420.31	#F99E68	1
2	Profits	2017	885.00	#33C0C0	1
3	SUM 2017	2017	1.00	#C0C0C0	SUM
4	Costs	2018	-986.34	#F99E68	1
5	Profits	2018	1,350.00	#33C0C0	1
6	SUM 2018	2018	1.00	#C0C0C0	SUM
7	Costs	2019	-901.58	#F99E68	1
8	Profits	2019	1,750.00	#33C0C0	1
9	SUM 2019	2019	1.00	#C0C0C0	SUM
10	Costs	2020	-120.00	#F99E68	1
11	Profits	2020	2,480.00	#33C0C0	1
12	SUM 2020	2020	1.00	#C0C0C0	SUM
13	Costs	2021	-124.00	#F99E68	1
14	Profits	2021	2,570.00	#33C0C0	1
15	SUM 2021	2021	1.00	#C0C0C0	SUM
16	Costs	2022	-126.00	#F99E68	1
17	Profits	2022	2,570.00	#33C0C0	1
18	SUM 2022	2022	1.00	#C0C0C0	SUM
19	TOTAL	2030	1.00	#408080	CUMSUM

In the settings of the **Business Graphic Definition** tab of the report assistant, the **X-Value Column(s)** definition is changed to `CLASS`.

The configured report now includes sums of values for all years and a total at the end of the report:

Combining Multiple Chart Types in One Chart

Line, area, and bar charts can be combined in a combination chart. The combination chart offers the following features for display of different kind of charts in the same graphic:

- Business charts of two different chart types can be combined, For example, some series can be displayed as a bar and some as a line.
- Two different y axes can be defined, For example, to combine charts that require a different scale because the series are having different ranges of values or units of measure.
- For a better graphic display of the combined chart types, 3D effects can be added to the configured report.

This section informs about the additional configuration required to implement a combination chart. The data displayed in each series is defined as described for individual chart types.

The following information is available:

- Combining Two Different Chart Types
- Implementing a Second Y-Axis
- Implementing Three-Dimensional Display

Combining Two Different Chart Types

Bar, area and line charts can be displayed next to each other in a single graphic with the following restriction:

- Stacked bars or stacked areas cannot be displayed in a combination chart.

To display different series in different chart types, add the following configuration to your basic chart definition returning all series in one chart type:

1) In the **Data Source Definition** tab, add an additional column to your dataset that returns the chart type for each series. The chart type can be one of the following:
   - bar
   - line
   - area

   Please note the following about the definition of the chart type:
   - All data belonging to the same series must return the same value.
   - If no data or any other than the allowed values are returned, the series will be displayed as bar chart.
2) Click the **Check Data Source** button in the **Data Source Result** tab. The result of the test is displayed in the **Data Source Errors** field. The next step can only be performed when the query passes the test without errors.
3) Define the following attributes in the **Business Graphic Definition** tab in addition to the basic configurations for display of the series:
   - **Business Chart Type**: Select `CombinationChart`.
   - **Chart Type Column**: Select the column in the dataset of the query defined in the **Data Source Definition** tab that returns the chart type.
   - **Stacked Bar / Area Chart**: Make sure that the checkbox is deselected. Stacked bars or stacked areas cannot be displayed in a combination chart.

Implementing a Second Y-Axis

The use of a secondary y-axis for some series is not limited to a combination report displaying more than one chart type:

- For bar charts, you can follow the steps described below without any additional configuration.
- For line and area charts and charts displaying multiple chart types, you must define the chart type for each series as described in the section Combining Two Different Chart Types. If all series return the same chart type, only one chart type is displayed.

Add the following configuration to your chart definition to use a secondary y-axis:

1) In the **Data Source Definition** tab, add an additional column to your dataset that returns `Secondary` for all data belonging to the series that shall use the secondary y-axis. If any other value is returned, the primary y-axis is used.

   Please note that for any chart type other than bar chart you must also define the chart type in a separate column as described in the section Combining Two Different Chart Types.

2) Click the **Check Data Source** button in the **Data Source Result** tab. The result of the test is displayed in the **Data Source Errors** field. The next step can only be performed when the query passes the test without errors.

3) Define the following attributes in the **Business Graphic Definition** tab in addition to the basic configurations for display of the series:

   - **Business Chart Type**: Select `CombinationChart`.
   - **Show Secondary Y-Axis**: Select the checkbox.
   - **Secondary Y-Title**: Optionally define a caption to be displayed on the secondary y-axis.
   - **Y-Axis Selection Column**:

4) Optionally define the following attributes in the **Additional Attributes** tab:

   - **Secondary Y Maximum Value**: Enter a maximum value for the secondary y-axis. If no maximum value is defined, the maximum value of the y-axis is derived from the highest y value in the dataset. Definition of a fixed maximum value can For example, be used to streamline the y-axis definitions of multiple charts for comparison reasons. If an y value in the result dataset of the report exceeds the defined maximum value, the defined maximum is ignored and the y-axis is extended to display all values in the dataset.
   - **Secondary Y Minimum Value**: Enter a minimum value for the secondary y-aaxis. If no minimum value is defined, the minimum value of the y-axis is 0 or, if negative values are displayed, derived from the lowest y value in the dataset. Definition of a fixed minimum value can For example, be used to highlight differences if values are all in a high range and differences would not be clearly visible if values are displayed on a scale starting from 0. If an y value in the result dataset of the report is lower than the defined minimum value, the defined minimum is ignored and the y-axis is extended to display all values in the dataset.
   - **Secondary Y Number Prefix**: Enter a string that will be displayed in front of all numbers on the secondary y-axis and in front of all label numbers in the chart that are referring to the secondary y-axis . This value can be set to display values For example, with a currency symbol in front without converting the numbers into a string on the SQL layer.
   - **Secondary Y-Axis Color**: The color used for display of the secondary y-axis if **Secondary Y-Axis Thickness** is set to a value higher than zero.
   - **Secondary Y-Axis Thickness**: The width of the line drawn on the secondary y-axis. By default, **Secondary Y-Axis Thickness** is set to zero and no line is drawn.

Implementing Three-Dimensional Display

Three dimensional display is only available for a subset of combination charts:

- **Bars** are fully supported. Bars can be either displayed with a three-dimensional effect on the bar only, or as fully three-dimensional cubes.
- **Lines** can be added to three-dimensional combination charts, but they are nevertheless displayed two-dimensional. Please note that the display of lines requires the configuration described in the section Combining Two Different Chart Types in addition to the configuration described below.
- **Areas** are not supported. If the combination chart includes area charts, these will be displayed as bars instead.

The three-dimensional display requires the following settings for the configured report:

1) Click the **Check Data Source** button in the **Data Source Result** tab. The result of the test is displayed in the **Data Source Errors** field. The next step can only be performed when the query passes the test without errors.
2) Define the following attributes in the **Business Graphic Definition** tab in addition to the basic configurations for display of the series:
   - **Business Chart Type**: Select `CombinationChart`.
3) Define the following attributes in the **Additional Attributes** tab:
   1) **3D Effect on Bar**: Select True if you want to display a 3D effect inside the two-dimensional bars.
   2) **Chart Appearance**: Select 3D from the drop-down list to display bars as cubes.

Defining a Circular Roadmap Report

Circular roadmap reports are designed in the **Report Assistant**. For information about how to open the **Report Assistant**, see Creating a New Graphic Report. For basic information about how to work with the **Report Assistant**, see Configuring the Report with the Report Assistant.

The content and layout of a circular roadmap report is defined by adding elements to the explorer of the **Report Assistant** and setting the attributes in the attribute pane of the **Report Assistant** window.

It is recommended to use native SQL for creating circular roadmap reports. Alfabet query language does not provide the ability to include anything else than data from the tables in the Alfabet database into the search results, which excludes some of the features of the report.

The following table lists the types of explorer node elements that can be added to the report configuration to specify the content of the report. Information is provided in the table about the purpose of each explorer node element:

Explorer Node Element	Required to Configure:
Root Node	General layout of the report.

Explorer Node Element	Required to Configure:
Item Query	Definition of the objects that are displayed as items in the shells and sectors of the circular roadmap report.
Sector Query	Definition of the sectors of the circular roadmap report.
Shell Query	Definition of the shells of the circular roadmap report.

Defining the Content of the Circular Roadmap Report

The circular roadmap report defines the relationship of objects to the following two dimensions in a circular graphic:

- The distance of the object elements from the center of the circle is derived from a date or a numeric value like a date or an indicator.
- The circle is devided into sectors, that typically represent objects. An object element is placed into the sector representing a related object.

To define a circular roadmap, three queries must be provided:

- A **Shell Query** that defines a numeric or time scale from the center to the border of the circle. The scale devides the circle into shells that can be highlighted with different colors.
- A **Sector Query** that defines the division of the circle into sectors.
- An **Item Query** that finds the objects to be displayed in the circle and their position within the shells and sectors.

Optionally, a **Connections Query** can be specified to display connecting arrows between items in the circular roadmap.

The content, design and navigation behaviour of the configured report is mainly directly defined within the three queries the circular roadmap report is based on. The data from the query is taken over if the name of the column in the query output that contain the data is added to the respective attribute in one of the sections **Common**, **Data**, **Special Item Data** and **Special Sector Data** of the report.

Some of the attributes in the section **Data** are valid for all mandatory queries. That means that the column in the result dataset that contains For example, the specification of the label for shells, sectors and items must be identical in all queries or the circular roadmap report.

Basic information about taking over data from queries by defining attributes of the root node of the report is given in the section Mapping of Query Data to Attributes.

The query parameter `BASE` can be used in all three queries to refer to the base object if the attribute **Apply to Class** of the configured report is set.

The report can only be displayed if all three queries are available, valid and combined in the correct way. This is a very complex configuration and Therefore, it is recommended to define all queries separately in configured reports of the **Type** `Query` or `NativeSQL` to check whether the result of each query results in the required dataset prior to combining the data in the report.

In the following the basic requirements for combining the queries is described in detail:

- Defining the Shells
- Defining the Sectors
- Defining the Items
  - Defining the Design of the Items in the Circular Roadmap Report
- Defining Connections Between Items

Defining the Shells

The shells of the circular roadmap are the graphic representation of a scale that either covers a range of dates or a range of numbers. The **Shell Query** must either define dates or numbers. The first row of the dataset defines the starting point of the scale. Each following row defines the end point of the respective shell on the scale. Object elements in the report are located on the axis of the scale according to their defined value. For example, if a scale has the shells 0-1, 1-2, 2-3, and 3-4, an object with the y value 3.5 is positions in the middle of the shell 3-4.

The query definition must at least contain one column that returns the numeric value or date. The name of that column must be specified in the attribute **Value Column** in the section **Data** of the root node of the circular roadmap report. Additional columns of the dataset can provide the following data:

Attribute	Column must return:	Mandatory	Exclusive use of attribute for this query
Value Column	A date or numerical value that marks the end of the shell on the scale. (The value defined in the first row of the dataset defines the start of the scale).	Yes	No, attribute is used by all queries of the circular roadmap report.
Label Column	A string that shall be displayed as label of the shell. Labels are displayed from the center of the cirlce to the right if **Label Column** is defined.	No, if not defined, no labels are displayed.	No, attribute is used by all queries of the circular roadmap report.

Attribute	Column must return:	Mandatory	Exclusive use of attribute for this query
Color Column	A HTML compliant color definition for coloring of the shell.	No, if not defined, no legend is displayed for shell coloring.	No, attribute is also used for **Item Query**.
Legend Group Column	A string that shall be displayed as heading for the legend entry defined with **Legend Entry Column** in the same row of the dataset. All legend entries that have the same **Legend Group Column** are grouped and displayed under the heading defined with **Legend Group Column**.	No, if the attribute is not defined, the legend does not display entries for shell coloring.	No, attribute is also used for **Item Query**. Please note that
Legend Entry Column	A string that shall be displayed next to the color of the shell in the legend of the report.	No, if not defined, and **Legend Group** Column is defined, the legend displays entries for shell coloring with the color followed by the value.	No, attribute is also used for **Item Query**.

The color and label defined defined for the first row in the dataset define the center point of the circle and are not visible in the report. But the color defined for the center point is included into the legend, if a legend group for shells is defined.

For example, a time scale is defined for the shell definition. The time scale covers a range of 5 years starting from the current date. That means that the y-axis is representing dates with each shell covering one year, that means 365 days. The following native SQL query is defined as **Shell Query**. A color and a label are also assigned to each shell.

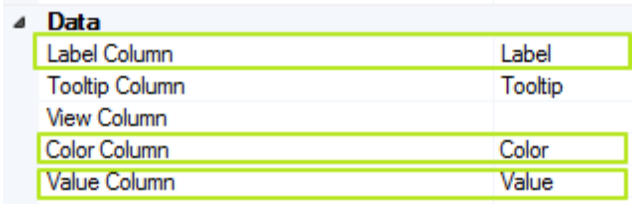
```
SELECT NULL AS REFSTR, '#ffffff' AS Color, 'today' AS Label,
CURRENT_TIMESTAMP AS Value UNION ALL SELECT NULL AS REFSTR,
'#d8d8ff' AS Color, '1. yr' AS Label,
DATEADD(YEAR,1,CURRENT_TIMESTAMP) AS Value UNION ALL SELECT NULL AS
REFSTR, '#e0e0ff' AS Color, '2. yr' AS Label,
DATEADD(YEAR,2,CURRENT_TIMESTAMP) AS Value UNION ALL SELECT NULL AS
REFSTR, '#e8e8ff' AS Color, '3. yr' AS Label,
DATEADD(YEAR,3,CURRENT_TIMESTAMP) AS Value UNION ALL SELECT NULL AS
REFSTR, '#f0f0ff' AS Color, '4. yr' AS Label,
DATEADD(YEAR,4,CURRENT_TIMESTAMP) AS Value UNION ALL SELECT NULL AS
REFSTR, '#f8f8ff' AS Color, '5. yr' AS Label,
DATEADD(YEAR,5,CURRENT_TIMESTAMP) AS Value
```

The query results in the following dataset:

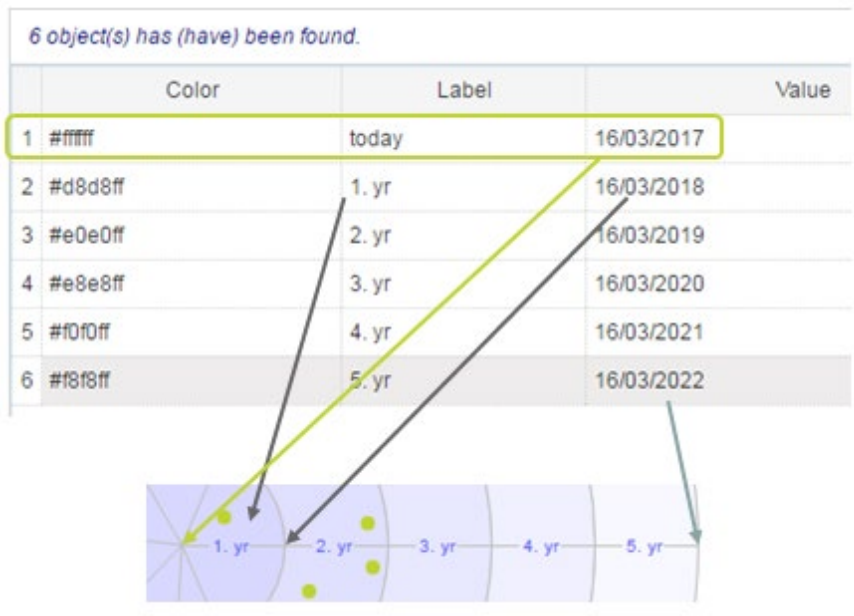
6 object(s) has (have) been found.

	Color	Label	Value
1	#ffffff	today	16/03/2017
2	#d8d8ff	1. yr	16/03/2018
3	#e0e0ff	2. yr	16/03/2019
4	#e8e8ff	3. yr	16/03/2020
5	#f0f0ff	4. yr	16/03/2021
6	#f8f8ff	5. yr	16/03/2022

In the root node of the report, the data is mapped to the attributes Value Column, Label Column and Color Column:



In the report, each `Value` returned in the dataset represents the border of a shell. The `Value` of the first row is the starting point of the scale. The `Label` and `Color` defined in that row are ignored. The `Value` in the following rows each define the endpoint of a shell. The `Label` and `Color` defined for a `Value` are used for the shell for that the `Value` marks the end point:



The object elements in the report are displayed along the shell according to the date defined for them as y value.

The shell query does not have to be related to the item query by refering to the same objects, but you must ensure that all possible values for placement on the shell axis that are returned for item objects fall into the defined range or the scale.

For example, the **Shell Query** definition for a circular roadmap report with the shells representing indicator values can be defined completely unrelated to the indicator values:

```
SELECT NULL As REFSTR, '0' As Value, 'very low' AS Label, '#E6EDD6'
As Color

UNION ALL

SELECT NULL As REFSTR, '1' As Value, 'very low' AS Label, '#E6EDD6'
As Color

UNION ALL

SELECT NULL As REFSTR, '2' As Value, 'low' AS Label, '#DBDBDB' As
Color

UNION ALL

SELECT NULL As REFSTR, '3' As Value, 'middle' AS Label, '#FCC8A9' As
Color

UNION ALL SELECT NULL As REFSTR, '4' As Value, 'high' AS Label,
'#FA9558' As Color

UNION ALL

SELECT NULL As REFSTR, '5' As Value, 'very high' AS Label, '#E15806'
As Color
```

or, if you are not sure which range is defined for the indicator as:

```
SELECT DISTINCT NULL As REFSTR,

CASE WHEN ind.VALUE <=1 THEN '0' ELSE CAST(ind.VALUE AS INT) END AS
Value,

CASE WHEN ind.VALUE > 1 AND CAST(ind.VALUE AS INT)= 1 THEN 'very
low' WHEN ind.VALUE > 1 AND CAST(ind.VALUE AS INT)= 2 THEN 'low'
WHEN ind.VALUE >1 AND CAST(ind.VALUE AS INT)=3 THEN 'middle'WHEN
ind.VALUE >1 AND CAST(ind.VALUE AS INT)= 4 THEN 'high' ELSE 'very
high' END AS Label,

CASE WHEN ind.VALUE > 1 AND CAST(ind.VALUE AS INT)= 1 THEN '#E6EDD6'
WHEN ind.VALUE > 1 AND CAST(ind.VALUE AS INT)= 2 THEN '#DBDBDB' WHEN
ind.VALUE >1 AND CAST(ind.VALUE AS INT)=3 THEN '#FCC8A9'WHEN
ind.VALUE >1 AND CAST(ind.VALUE AS INT)= 4 THEN '#FA9558' ELSE
'#E15806' END As Color

FROM INDICATOR ind

WHERE ind.NAME LIKE 'Criticality (aggr.)'
```

Both queries result in the same result dataset:

	Value	Label	Color
1	0	very high	#E6EDD6
2	1	very low	#C0C0C0
3	2	low	#FCC8A9
4	3	middle	#FA9558
5	4	high	#E15806
6	5	very high	#E15806

But the second query provides security that all values defined for the indicator are included into the shell definition. It also reduces the shell definition to the subset of shells that are required. If only indicator values between 0 and 3 are defined for the objects in the Alfabet database, the report will display only three instead of 5 shells and the sizing is optimized for display of differences between indicator settings.

The design of the shells can be further specified with attributes of the root node of the circular roadmap report assistant. For more information see Defining the Basic Design of the Circular Roadmap Report.

Defining the Sectors

The **Sector Query** typically returns objects from the Alfabet database, but this is not a requirement. The **Sector Query** must return a dataset with a key property that is different for each result in the return dataset and can be used to map the object elements unambiguously to a sector. The column of the dataset that returns the key property must be defined in the attribute **Sector Id Column** of the root node of the circular roadmap. If the sector query returns objects of the Alfabet database, and the result returns only one row per object, the `FIND` class of the Alfabet query or the `REFSTR` returned via the first definition in the `SELECT` clause of a native SQL query can be used to identify the sector. In this case, **Sector Id Column** does not need to be defined.

The columns of the result dataset can return information about the sectors that is used for display of the sectors in the circular roadmap report. The following attributes of the root node of the circular roadmap report can be mapped to columns in the result dataset:

Attribute	Column must return:	Mandatory	Exclusive use of attribute for this query
Section **Special Sector Data**			
Sector Id Column	A key property that is different in each row of the result dataset and that can be related	No, if the `FIND` class of the Alfabet query or the `REFSTR` returned via the first definition in the `SELECT` clause of a	Yes

Attribute	Column must return:	Mandatory	Exclusive use of attribute for this query
	with the object items in the circular roadmap report.	native SQL query is used to identify the sector, the setting is not required.	
Section **Data**			
Value Column	A numerical value that is used to calculate the size of the sectors. The circle size of a sector equals the share of its value on the overall sum of values. The numeric value is also used to calculate the horizontal deviation of the position of items to the middle axis of a sector, that means how far left or right of the middle axis an item is displayed. This calculation depend on the **Value Column** value of the sector definition and the **Axial Value Column** value of the item definition.	Yes. This attribute is required for calculating the position of items within a sector. To define equally sized sectors, the **Value Column** in the result dataset can be defined to return the same value in each row.	No, also used by **Shell Query** and **Item Query**.
Label Column	The text displayed as label of each sector outside the circle.	No, if not set, no labels are displayed.	No, also used by **Shell Query** and **Item Query**.
Tooltip Column	The text displayed as tooltip when the user moves the mouse over a sector.	No, if not set, no tooltips are displayed.	No, also used by **Item Query**.
Legend Group Column	A string that shall be displayed as heading for the legend entry defined with **Legend Entry Column** in the same row of the dataset. All legend entries that have the same **Legend Group Column** are grouped and displayed under the heading defined with **Legend Group Column**.	No, if not set, sectors are not included into the legend.	No, also used by **Shell Query** and **Item Query**.
Legend Entry Column	A string that shall be displayed for the sectors in the legend. The string is displayed as is, that means that if you would For example, explain the abbreviations used as labels for	No, if not defined, but **Legend Group Column** is defined, sector labels are listed in the legend group.	No, also used by **Shell Query** and

Attribute	Column must return:	Mandatory	Exclusive use of attribute for this query
	sectors, the column must return both the abbreviation and the full name with a separator.		**Item Query**.

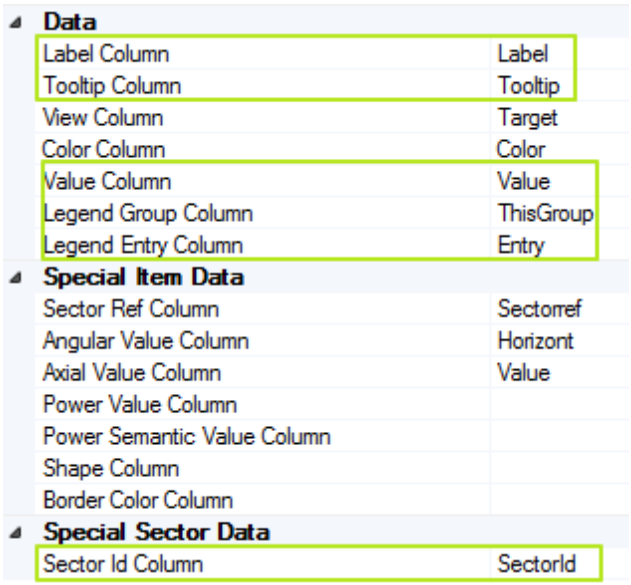
The following example is a **Sector Query** that returns organisations subordinate to a selected base organisation. This **Sector Query** will be used in the description of the **Item Query** in combination with the indicator scale example for a **Shell Query** to explain mapping of object elements to shells and queries.

```
SELECT REFSTR, SectorId, Tooltip, Label, ThisGroup, Entry,
COUNT(ICTO) As Value

FROM(

   SELECT org.REFSTR as REFSTR, org.NAME As SectorId, org.SHORTNAME
   As Label, org.NAME As Tooltip, 'Sectors' As ThisGroup,
   org.SHORTNAME + ' = ' + org.NAME As Entry, icto.REFSTR As ICTO

   FROM ORGAUNIT org

   INNER JOIN ICTOBJECT icto ON icto.OWNER=org.REFSTR

   WHERE org.BELONGSTO = @BASE

)x

GROUP BY REFSTR, SectorId, Label, Tooltip, ThisGroup, Entry
```

The column `Value` is defined in the attribute **Value Column** of the root node attributes. It returns the number of ICT Objects assigned to the organization. When rendering the graphic, this value will be used both for sizing of the sectors representing the organizations and for calculating how much left or right of the middle axis of the sector an items is displayed. The column SectorId returns the name of the organization, that is used to identify the sector for placing of the items. The column SectorId is define in the attribute Sector Id Column in the root node attributes.

The label of the sectors displays the short name of the organization. To provide the user with the full name of the organization, a tooltip column is added that displays the name of the organization when a user places the mouse in a sector and a legend group has been defined with a string concatenated from the shortname of the organization, an equal sign and the name of the organization.

The columns are mapped as follows:



The design of the shells can be further specified with attributes of the root node of the circular roadmap report assistant. For more information see Defining the Basic Design of the Circular Roadmap Report.

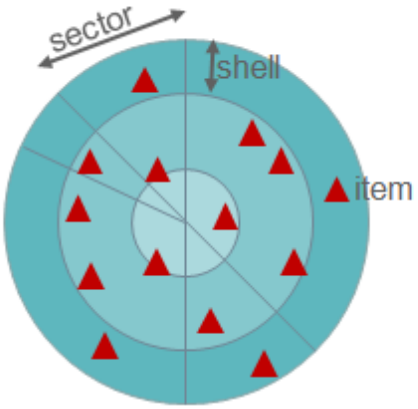
Defining the Items

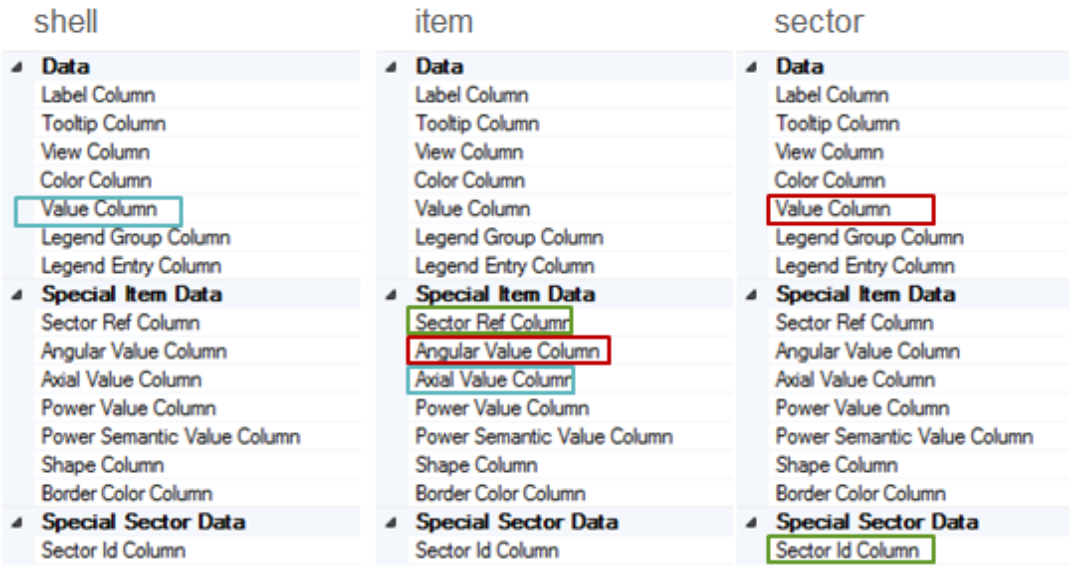
The items of the circular roadmap have two dimension of configuration. First, the positioning of the items in the report must be defined. You can then optionally define the items to be colored, sized and formed differently.

The positioning of items includes three mappings:

- The position relative to the shell axis must be defined.
- The mapping to a sector must be defined.
- The position of the item relative to the middle axis of the sector must be defined to avoid multiple items with the same shell axis value and sector mapping to be placed on top of each other.

To position the items, the columns in the result dataset of the Sector Query, Shell Query and Item Query must be set in relation to each other. This is performed by defining the columns in the result datasets to be related with each other and to map the related column to attributes in the root node of the circular roadmap report. The following columns of the result datasets are involved:





The **Value Column** defined for the **Shell Query** defines a scale either as date format or numerical value. The **Axial Value Column** of the **Item Query** must contain values that are in the range of this scale. For example, if the **Shell Query** defines the range of a year per shell for 2015 to 2025, and the **Axial Value Column** for an item returns July 1st, 2016, the item is placed in the middle of the shell for 2016.

The **Sector Id Column** of the **Sector Query** must return a unique value for each sector. The **Sector Ref Column** of the **Item Query** must return values that are identical to the values of the **Sector Id Column** to map each item to one of the defined sectors. For example, of the sectors display domains and the **Sector Id Column** contains the names of the domains, the **Sector Ref Column** of the **Item Query** must return a domain name for each item in the **Sector Ref Column**.

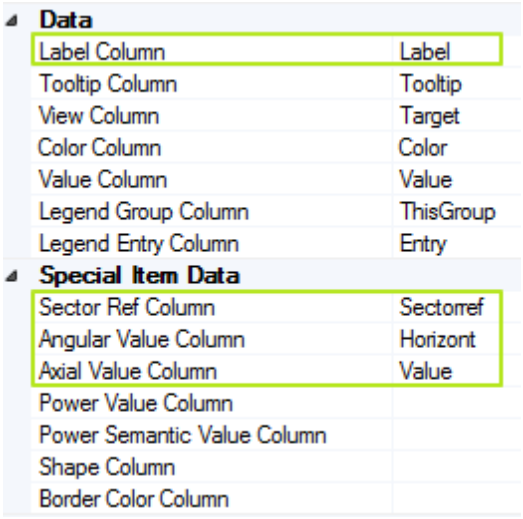
The **Value Column** of the **Sector Query** defines the size of the sectors in the graphic, but also the size of the sector for placement of the items left and right of the middle axis of the sector. The **Angular Value Column** defined for the item in the **Item Query** is used together with the **Value Column** of the **Sector Query** to define the exact placement of the item in the sector. For example, if the number of items in a sector is used to define the sector size, a row count for

items returned for the same sector in the result dataset can be used to specify a different number for each item within the range of the overall number of items in the sector.

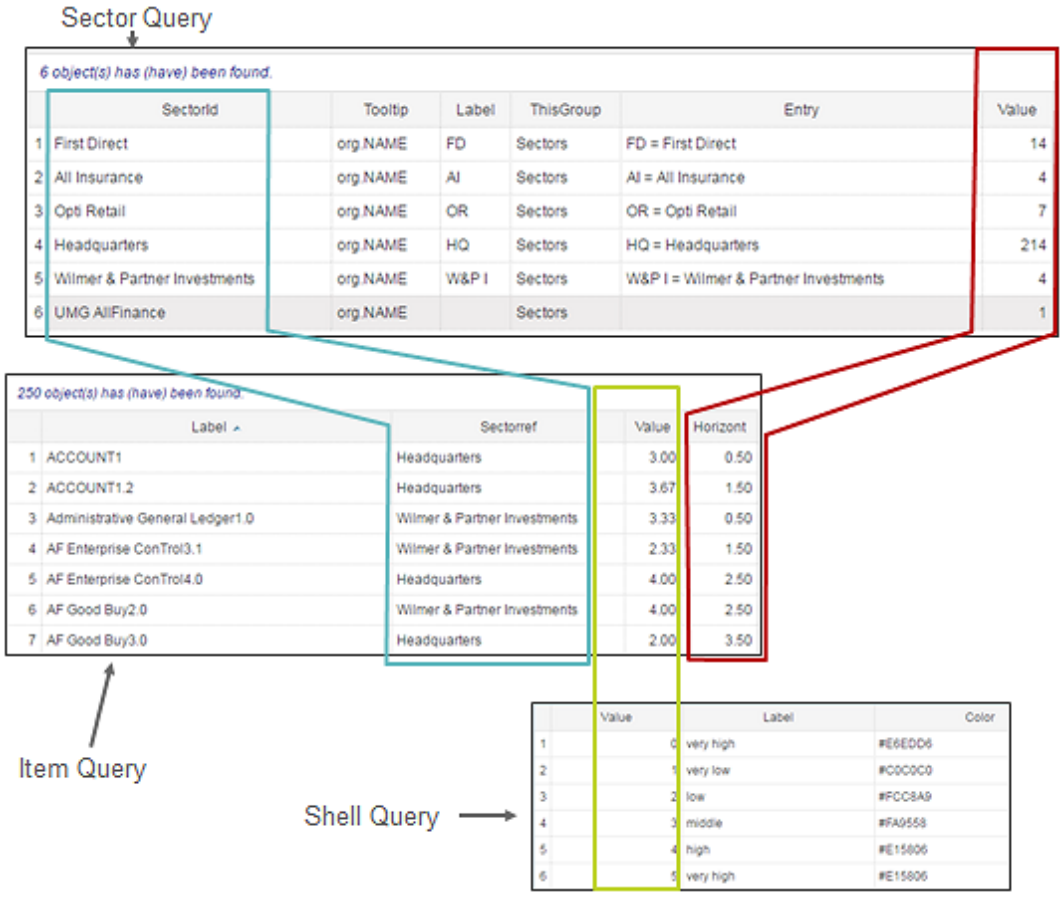
The **Item Query** to be used with the **Shell Query** and the **Sector Query** in the examples in the sectons above defines applications owned by the organizations displayed as sectors. The placment in the shells depends on an indicator value:

```
SELECT REFSTR, Label, Sectorref, Value, ROW_NUMBER() OVER (PARTITION
BY Sectorref ORDER BY Label) - 0.5 As Horizont

FROM(

SELECT app.REFSTR as REFSTR, app.NAME + app.VERSION As Label,
org.NAME AS Sectorref, ind.VALUE As Value, NULL As Horizont FROM
APPLICATION app INNER JOIN ICTOBJECT icto ON app.ICTOBJECT =
icto.REFSTR INNER JOIN ORGAUNIT org ON icto.OWNER=org.REFSTR INNER
JOIN INDICATOR ind ON ind.OBJECT = app.REFSTR WHERE org.BELONGSTO =
@BASE AND ind.NAME LIKE 'Criticality (aggr.)')x
```

The columns for the result dataset are mapped to the attributes in the root node of the circular roadmap as follows:



The result dataset of the **Item Query** is mapped to the result dataset of the **Shell Query** and the **Sector Query** to render the report:



The indicator that is used to define the position of the items in the shells is a calculated indicator that is returned in the `Value` column if the **Item Query**. The position of the items is Therefore, within the shells and not always exactly on the borders of the shells.

The names of the organisations returned in the `Sectorref` column of the **Item Query** each match a name of an organisation returned in the `SectorId` column of the **Sector Query**, mapping the items to a defined sector. Within the sector, the value defined for `Horizont` is used together with the number defined in the **Sector Query** in the column `Value` to define the relative position of the items with regard to the middle axis of the sector. In the example above, the sector `Wilmer & Partner Investments` has a `Value` of 4. Three of the results for items mapped to the sector are included into the example graphic. They have `Horizont` values 0.50, 1.50 and 2.50. The values are calculated as the row number within the result set of the sector minus 0.5 to make sure that the items number 4 is not placed directly on the border. These values are all different and are all in the range of 4.

Items that are defined for a circular roadmap can be presented as geometric shape that may differ in different size, shape and coloring for each item. Alternatively, items can be represented by icons. In the **Item Query**, you can define the design of the items per item. The column of the result is then defined in the attributes of the root node of the circular roadmap report. The following attributes of the root node of the circular roadmap report can be mapped to columns in the result dataset:

Attribute	Column must return:	Mandatory	Exclusive use of attribute for this query
Section **Special Item Data**			
Sector Ref Column	Each value returned in this column must match a value returned in the column **Sector Id Column** of the **Sector Query** to map the item to a sector. If no **Sector Id Column** is defined for the Sector Query, but sectors represent objects of the objet class that is the `FIND` class of the Alfabet query or for that `REFSTR` values are returned in the first argument of the `SELECT` clause of the **Sector Query**, the column must return the `REFSTR` value of one of the sectors.	Yes	Yes
Angular Value Column	The **Angular Value Column** must return a number that is used together with the **Value Column** of the **Sector Query** to define the exact placement of the item in the sector.	Yes	Yes
Axial Value Column	The **Value Column** defined for the **Shell Query** defines a scale either as date format or numerical value. The **Axial Value Column** of the **Item Query** must contain values that are in the range of this scale.	Yes	Yes
Power Value Column	A numeric value that is used to calculate the size of the item. The higher the value in the **Power Value Column**, the bigger the size of the item. The absolute size range for sizing of objects is defined with the attributes **Min Item Size** and **Max item Size** in the section **Common**.	No, if not set, all items are equally sized. If items are represented by icons, this setting is ignored.	Yes
Power Semantic Value Column	If **Power Value Column** is defined to size items differently and the attribute **Add Power To Tooltip** in the section **Common** is set to `True`, the value in the **Power Value Column** is added in parantheses to the tooltip text at the end of the string defined with **Tooltip Column**. If you would like to display a string explaining the power value instead of the numeric value, you can define a string in the dataset of the **Item Query** and set **Power Semantic Value Column** to the name of the respective	No, if not set, the numerical vaule will be added to the tooltip if the attribute **Add Power To Tooltip** in the section **Common** is set to `True`	Yes

Attribute	Column must return:	Mandatory	Exclusive use of attribute for this query
	column. The string defined in that column will be added to the tooltip instead of the numerical value.		
Border Color Column	A HTML compliant color definition for coloring of the item border if the item is displayed as geometric shape.	No, if items are displayed as geometric shapes and this attribute is not set, the item border is transparent. If items are represented by icons, this setting is ignored.	Yes
Section **Data**			
Label Column	The text displayed as label next to the item. Labels are only displayed if the attribute **Show Item Label** in the section **Common** is set to **True**.	No, if not set, no labels are displayed.	No, also used by **Shell Query** and **Sector Query**.
Tooltip Column	The text displayed as tooltip when the user moves the mouse over an item.	No, if not set, the label is displayed as tooltip.	No, also used by **Sector Query**.
View Column	Definition of a link target that shall open if a user double-clicks on an item. The format is `View=ViewType:ViewName` The `ViewType` can be one of the following: • ***Report*** for a configured report • ***GraphicView*** for a standard Alfabet view • ***ObjectView*** for an object profile `ViewName` is the name of the view.	No, if not set, navigation is deactivated.	Yes

Attribute	Column must return:	Mandatory	Exclusive use of attribute for this query
	For detailed information about the required configuration to activate navigation, see Defining Navigation from the Report to Alfabet Views.		
Color Column	A HTML compliant color definition for coloring of the item, if the item is displayed as geometric shape..	No, if items are displayed as geometric shape and this attribute is not set, a color is assigned randomly. All items are equally colored. If items are represented by icons, this setting is ignored.	No, als used by **Shell Query**.
Legend Group Column	A string that shall be displayed as heading for the legend entry defined with **Legend Entry Column** in the same row of the dataset. All legend entries that have the same **Legend Group Column** are grouped and displayed under the heading defined with **Legend Group Column**.	No, if not set, no legend for items is displayed.	No, als used by **Shell Query** and **Sector Query**.
Legend Entry Column	A string that shall be displayed for the items in the legend. If set, the string is displayed after a graphic combining all design options except the size defined for the item. If the same string is defined for multiple design options, the first design option found for the string is displayed for the entry.	No, if not set and **Legend Group Column** is set, a separate legend entry will be added for each item.	No, als used by **Shell Query** and **Sector Query**.
Common Section			
Shape Source > **Shape Name Column**	The column defined with this attribute must return one of the following values to define a shape for the item: `Rectangle`, `Triangle`, `Circle`, `Diamond` or `FlippedTriangle`. Please note that this attribute is only available in the context of specific settings for Shape Source that are described in detail in the section Defining the Design of the Items in the Circular Roadmap Report.	No, by default items are displayed as circles. This setting is only required if items shall be displayed in different geometric shapes. If items are all represented by the same geometric shape or	Yes

Attribute	Column must return:	Mandatory	Exclusive use of attribute for this query
		by an icon, this setting is not required.	
Shape Source > Shape Name Column	The column defined with this attribute must return the name of an icon from the 22x22 icon gallery. Please note that this attribute is only available in the context of specific settings for Shape Source that are described in detail in the section Defining the Design of the Items in the Circular Roadmap Report.	No, by default items are displayed as circles. This setting is only required if items shall be displayed in different geometric shapes. If items are all represented by the same geometric shape or by an icon, this setting is not required.	Yes

The complete settings for the various options to represent items either via geometric shapes or via icons and either static or via the query are described below.

The design of the shells can be further specified with attributes of the root node of the circular roadmap report assistant. For more information see Defining the Basic Design of the Circular Roadmap Report.

Defining the Design of the Items in the Circular Roadmap Report

The items in the circular roadmap can be represented either by icons or by geometric shapes. The following options are available:

- A geometric shape is assigned per item via the **Item Query**.
- The same geometric shape is used for all items. This is the default behaviour. The default shape is a circle.
- An icon from the 22x22 icon gallery is assigned per item via the **Item Query**.
- The icon of the object represented by the item is used to display the item. The following applies:
  - All objects that have an icon assigned are represented by the object's icon.
  - If no icon is assigned to the object, the object class icon defined in the relevant class settings is used.
  - If the item is not representing an object, items are displayed as rectangles.

The display of items is mainly configured with the attribute **Shape Source** in the section **Common** of the attributes of the root node of the report assistant. The attribute is expandable and different settings are required for each type of graphic item representation. In addition, other attributes or results from the **Item**

Query might be required. The required configurations are listed in the following separately for each type of graphic item representation.

To display all items with the same geometric shape:

1) In the root node attributes of the report assistant, expand the **Shape Source** attribute.
2) Set the following attributes in the order they are listed here:
   - **Shape Type**: Select `GeometricShape`.
   - **Shape Source**: Select `ReportDefinition`.
   - **Fixed Shape**: Select a shape from the drop-down list.
3) By default, all items are displayed in the same, randomly selected color. Optionally, you can define the background color and border color of the items per item in the query defined with the element **Item Query**. Colors must be returned as HTML compliant color code. The name of the columns returning the color definitions must be entered into the attributes **Color Column** and **Border Color Column** in the root node of the report assistant.
4) Optionally, change the size of your items with one of the following settings:
   - If all items shall be displayed with the same size, you can change the size of the item by changing the atribute **Min. Item Size** in the root node of the report assistant.
   - If items shall be differently sized, the query defined for the Item Query element must include a column that returns a numeric value that is used to calculate the size of the item. The higher the value in the column, the bigger the size of the item. The name of the column must be entered into the attribute **Power Value Column** in the root node of the report assistant. The absolute size range for sizing of objects is defined with the attributes **Min. Item Size** and **Max. Item Size** in the root node of the report assistant.

To display items with different geometric shapes:

1) In the **Item Query**, add a column that returns one of the following values to define a shape for each item: `Rectangle`, `Triangle`, `Circle`, `Diamond` or `FlippedTriangle`.
2) In the root node attributes of the report assistant, expand the **Shape Source** attribute.
3) Set the following attributes in the order they are listed here:
   - **Shape Type**: Select `GeometricShape`.
   - **Shape Source**: Select `Query`.
   - **Shape Name Column**: Enter the name of the column in the **Item Query** that returns the shape definition.
4) By default, all items are displayed in the same, randomly selected color. Optionally, you can define the background color and border color of the items per item in the **Item Query**. Colors must be returned as HTML compliant color code. The name of the columns returning the color definitions must be entered into the attributes **Color Column** and **Border Color Column** in the root node of the report assistant.
5) Optionally, change the size of your items with one of the following settings:
   - If all items shall be displayed with the same size, you can change the size of the item by changing the atribute **Min. Item Size** in the root node of the report assistant.

- If items shall be differently sized, the query defined for the Item Query element must include a column that returns a numeric value that is used to calculate the size of the item. The higher the value in the column, the bigger the size of the item. The name of the column must be entered into the attribute **Power Value Column** in the root node of the report assistant. The absolute size range for sizing of objects is defined with the attributes **Min. Item Size** and **Max. Item Size** in the root node of the report assistant.

To display the object icon as item:

1) In the root node attributes of the report assistant, expand the **Shape Source** attribute.
2) Set the following attributes in the order they are listed here:
   - **Shape Type**: Select `Icon`.
   - **Shape Source**: Select `Object`.

To display items with different icons selected from the 22x22 icon gallery:

1) In the **Item Query**, add a column that returns the name of an icon from the 22x22 icon gallery.
2) In the root node attributes of the report assistant, expand the **Shape Source** attribute.
3) Set the following attributes in the order they are listed here:
   - **Shape Type**: Select `Icon`.
   - **Icon Source**: Select `Query`.
   - **Icon Name Column**: Enter the name of the column in the query defined with the element **Item Query** that returns the icon name.

Defining Connections Between Items

To display connections between items in the report, all relevant data for rendering the connections must be provided via th optional **Connections Query**. The **Connection Query** must at least return the `REFSTR` of the object that is represented by the item the connection is coming from and the `REFSTR` of the object represented by the item the connection is ending at. That means that both values must be identical to a `REFSTR` returned with the first `SELECT` statement of the native SQL query for definition of the **ItemQuery**.

The columns in the result dataset of the **Connections Query** must then be mapped to attributes in the root node of the report assistant that are available when expanding the attribute **Connection Query Columns** in the section **Data**.

In addition to the mandatory definition of the start item and end item of the connection, the query can return styling parameters, legend texts and a link target to activate navigation from the connection to an Alfabet view. The columns must be mapped to the following sub-attributes of the attribute **Connection Query Columns**:

Attribute	Column must return:	Mandatory
Start Ref Column	The `REFSTR` of the object represented with by the item the connection is starting at. The `REFSTR` must be identical to a `REFSTR` returned in the first `SELECT` argument in the **Item Query** when written in native SQL or the `FIND` class in an Alfabet query.	Yes
End Ref Column	The `REFSTR` of the object represented with by the item the connection is ending at. The `REFSTR` must be identical to a `REFSTR` returned in the first `SELECT` argument in the **Item Query** when written in native SQL or the `FIND` class in an Alfabet query.	Yes
Line Style Column	The style of the connecting line. Allowed values are `Solid`, `Dash`, `DashDot`, `DashDotDot` and `Dot`.	No, by default connections are displayed as solid lines (`Solid`).
Line Weight Column	The width of the connecting line as integer.	No, the default value is 1.
Line color Column	The color of the connecting line as HTML compliant color code.	No, by default connections are displayed as black lines.
Start Arrow Style Column	The style of the start point of the connection. Allowed values are `None`, `Angle`, `Triangle`, `Square`, `Circle` and `Rhomb`. Navigation from a connection and display of a tooltip is only available if the user points to either the start or end point of the connection. If you would like to activate navigation or the display of a tooltip, definition of a start arrow style will make it easier for the user to find the connection start.	No, by default connections are displayed with no extra decoration at the start point (`None`).
End Arrow Style Column	The style of the end point of the connection. Allowed values are `None`, `Angle`, `Triangle`, `Square`, `Circle` and `Rhomb`. Navigation from a connection and display of a tooltip is only available if the user points to either the start or end point of the connection. If you would like to activate navigation or the display of a tooltip, definition of an end arrow style will make it easier for the user to find the connection end.	No, by default connections are displayed with no extra decoration at the end point (`None`).
Tooltip Column	A string that is displayed as tooltip when the user moves the cursor over the start or end point of the connection.	No, by default no tooltips are displayed.

Attribute	Column must return:	Mandatory
Link Ref Column	The `REFSTR` of an object. If this column is specified, navigation from the start and end point of the connection is activated. When the user clicks and holds the end point, a preview of the object with the `REFSTR` returned in the **Link Ref Column** opens. The user can navigate to the object profile or object cockpit of the object by clicking the **Show Details** button in the preview. If **View Column** is defined, the defined navigation target opens instead of the object profile or object cockpit of the object.	No, if not set, navigation is deactivated.
View Column	Definition of a link target that shall open if a user double-clicks on an item. The format is `View=`*`ViewType`*`:`*`ViewName`* The `ViewType` can be one of the following: • ***Report*** for a configured report • ***GraphicView*** for a standard Alfabet view • ***ObjectView*** for an object profile `ViewName` is the name of the view. For detailed information about the required configuration to activate navigation, see Defining Navigation from the Report to Alfabet Views.	No, if not set, but **Link Ref Column** is set, navigation is opening the object profile or object cockpit of the object defined with **Link Ref Column**.
Legend Group Column	A string that shall be displayed as heading for the legend entry defined with **Legend Text Column** in the same row of the dataset. All legend entries that have the same **Legend Group Column** are grouped and displayed under the heading defined with **Legend Group Column**. A legend is only displayed for connections if both **Legend Group Column** and **Legend Text Column** are defined.	No, if the attribute is not defined, the legend does not display entries for connection styles.
Legend Text Column	A string that shall be displayed next to the style of the connection in the legend of the report. A legend is only displayed for connections if both **Legend Group Column** and **Legend Text Column** are defined.	No, if the attribute is not defined, the legend does not display entries for connection styles.

Defining the Basic Design of the Circular Roadmap Report

The basic design of the report is designed directly in the attributes of the root node element of the **Report Assistant** explorer in the section **Common**:

Attribute	Description
Width	Defines the overall width of the resulting report in mm (1 mm = 4 pixel).
Height	Defines the overall height of the resulting report in mm (1 mm = 4 pixel).
Add Power To Tooltip	If set to `True`, the value defined in the column of the **Item Query** dataset defined as **Power Semantic Value Column** in the section **Special Item Data** is written in parentheses to the end of the tooltip. If **Power Semantic Value Column** is not defined, the numerical value from the column of the **Item Query** dataset defined as **Power Value Column** is written in parentheses to the end of the tooltip.
Left Item - Left Label	If set to `False`, all item labels are displayed on the right of the item. If set to `True`, the labels of the items on the left side of the graphic are displayed left of the corresponding item. Defines the color of the item labels. This setting is only relevant if **Show Item Label** is set to `True`.
Draw Shell Grid	If set to `True`, shells are separated with lines.
Draw Sector Grid	If set to `True`, sectors are separated with lines.
Show Item Label	If set to `True`, labels are displayed for items.
Plot Area Background Color	Defines the background color of the circular roadmap report, that means the color of the area surrounding the circle.
Plot Area Border Color	Defines the border color of the area surrounding the circular roadmap report.
Default Shell Color	Defines the coloring that shall be applied to shells if no color is defined in the **Shell Query**.
Sector Label Color	Defines the color of the sector labels.

Attribute	Description
Shell Label Color	Defines the color of the shell labels.
Item Label Color	Defines the color of the item labels. This setting is only relevant if **Show Item Label** is set to `True`.
Grid Color	Defines the color of grid lines in the report. Defines the color of the item labels. This setting is only relevant if **Draw Sector Grid** and/or **Draw Shell Grid** is set to `True`.
Plot Sector Line Extension	If set to `False`, grid lines for sectors end at the border of the circle. If set to `True`, the grid lines for sectors are extending the border of the circle. This setting is only relevant if **Draw Sector Grid** is set to `True`.
Min Item Size	The minimum size of items in the report. If the attribute **Power Value Column** in the section **Special Item Data** is not set, this is the size of all items in the report.
Max Item Size	The maximum size of items in the report if sizing is performed according ot the values in the column of the **Item Query** dataset that is defined with the attribute **Power Value Column** in the section **Special Item Data**.

Defining Reports to Analyse Data Cubes

Cubes are multi-dimensional blocks of facts that are structured by dimensions. A dimension is a facet of your enterprise relevant for analysis, whereas a fact is a measurement related to the elements within each dimensions.

For example, if you want to analyse the cost of applications in the organizations of your enterprise, typical dimensions are Application, Organization, Time, and Cost Types. Each organization of the enterprise is an element of the dimension "Organization". Elements of the dimension Time may be For example, month, or year.

The costs are the facts that are stored in the cube in relation to the dimensions.

A cube allows to aggregate data along the dimensions. The dimensions can be structured hierarchically, and the facts can be aggregated on the various levels of the hierarchy. Each element of a dimension can also have various attributes, and aggregation can be performed by aggregating elements wich are identical in a defined attribute.

Cubes are defined outside of Alfabet in the Microsoft® Business Analysis Services that are part of the distribution of Microsoft SQL Server. The definition of data cubes is independent from the Alfabet database. Data in a cube can be defined independent of any data source or can take over data from one or multiple databases. Therefore, cubes can be build on data from the Alfabet database in combination with data from any other data source or with data that is added manually to the cube.

The data in the cube represents a snapshot of the data in the source databases and must be updated in regular intervals to reflect changes made in the underlying source database. Regular update intervals can be configured for a cube in the Microsoft® Business Analysis Services.

Alfabet provides an interface for the analysis and graphical display of data in the cube. The analysis is done with the embedded third party components DevExpress®. Data in a cube can be displayed in a table as well as in a chart or a portfolio and the data can easily be structured by dimensions and aggregated along the axes of the report. The user can drill-down from the analysis view to the object profile of Alfabet objects.

To analyse the data that is created and maintained by the Alfabet users via the user interface in a cube, you must mainly work in third-party components either outside or embedded to the Alfabet components:

- The cube is created and updated in the Microsoft® Business Analysis Services that is a component of Microsoft SQL Server. For information about the creation and management of cubes, see the documentation of the Microsoft® Business Analysis Services version you are working with.

  The cube can take over data from any external source. The Alfabet database on Oracle database servers can also be used as a data source.

- The analysis of data is performed within a Pivot grid analysis view based on the embedded component DevExpress®embedded into the Alfabet components. The Alfabet Web Application must be configured to allow PivotTable handling. For more information about the required settings, see *Prerequisits for Using DevExpress®* in the *Technical Requirements*.

To analyse the cube designed with Microsoft® Business Analysis Services with one of the third party components based Pivot grid analysis views on the Alfabet interface, a configured report must be defined in Alfabet Expand that fetches the data from one or multiple cubes of a cube project for analysis.

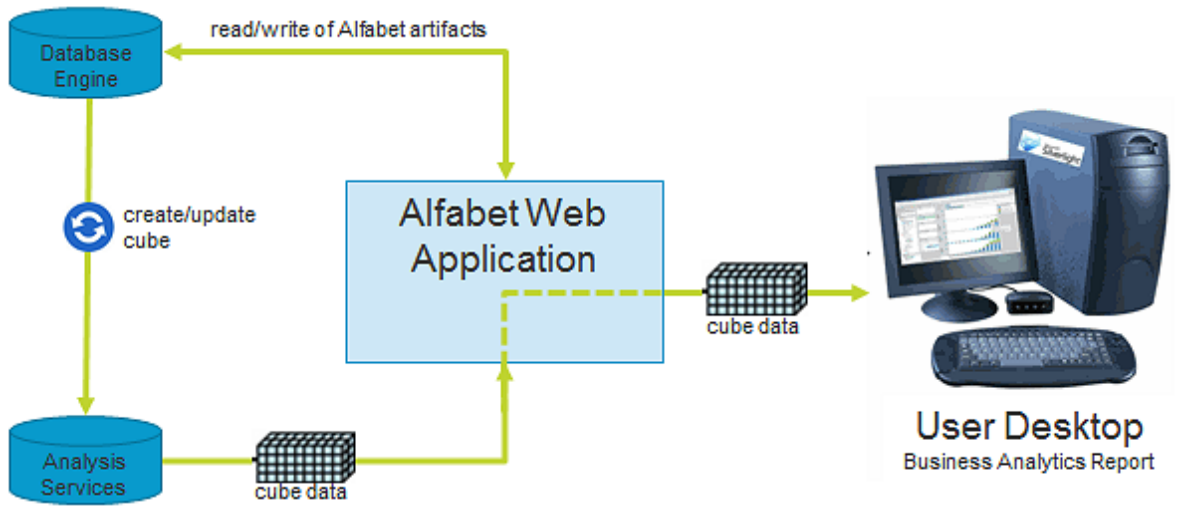


FIGURE: Overview of the components involved in cube reporting

Cube reports are designed in the **Report Assistant**. For information about how to open the **Report Assistant**, see Creating a New Graphic Report. For general information about how to work with the **Report Assistant**, see Configuring the Report with the Report Assistant.

Defining the Access to the Cube Data in the Report

In the **Report Assistant**, set the following attributes to define access of the Alfabet Web Application to the cube that shall be analysed.

Attribute	Description
Data Source	Enter the name of the Microsoft® SQL Server hosting the cube in the Microsoft® Business Analysis Services. All or part of the specification can be defined as server variable. Server variables are substituted with the value defined for the server variable in the server alias configuration of the Alfabet Web Application. This enables the same configuration of the report to be used in different environments with a different Microsoft® SQL Server hosting the cube in the Microsoft® Business Analysis Services. For information about how to define server variables, see Using Server Variables in Configurations in the chapter Getting Started with Alfabet Expand.
Extended Parameters	Optionally, you can define a string to add parameters required for connecting to the database to the default connection string used for database connection.
Roles	In the Microsoft® Business Analysis Services, you can restrict access to cube data based on roles. If you have used roles when defining the cube, you can specify the role that shall be used by the Alfabet Server to access the cube in this attribute.
Initial Catalog	Enter the name of the project containing the cube or cubes that shall be analysed.
Cube	Enter the name of the cube that shall be available via the report.
Alfabet Ref Fields	To activate navigation from the Pivot grid analysis to the object profile of Alfabet objects, the REFSTR of the object class must be available as field in the dimension of the cube. The field referencing the object in the cube must be specified fully qualified in the **Alfabet Ref Fields** attribute. The qualifiers must be written in squlare brackets. Dimension fields must start with $. For example,: `[$Application].[REFSTR]` Multiple fields can be specified as comma separated list.

Providing Context Sensitive Online Help for the Pivot Grid Analysis View of the Cube Based Report

If you would like the user to be informed about the functionality of the Pivot grid analysis view via the context sensitive help, you can do the following:

- An Alfabet specific standard help file is available. To display the standard help, enter `PivotTableAnalysis.html` in the **Help Index** attribute.
- A customer specific help can be opened. The method for defining customer specific help is described in the section Specifying Custom Help for a Configured Report.

Defining Dynamic Lane Reports

Dynamic lane reports are designed in the **Report Assistant**. For information about how to open the **Report Assistant**, see Creating a New Graphic Report. For more information about how to work with the **Report Assistant**, see Configuring the Report with the Report Assistant.

The content and layout of a dynamic lane report is defined by adding elements to the explorer of the **Report Assistant** and setting the attributes in the attribute pane of the **Report Assistant** window.

The content and layout of a lane report is defined in the root node element.

The following table lists the explorer node elements that can be configured to specify the content of the report as well as provides information about the purpose of each sub-element:

Explorer Node Element	Required to Configure:
Root Node of the report	Definition of the layout of the report.
Lane	Definition of a new lane added to the report. A lane consists of a row of objects and optionally a definition of a link to the next lane in the report. The next lane is the lane right of the current lane.
Node	Definition of the layout of the object row of the lane. The Node element is a child of the Lane element.
Link	Definition of the layout of the connections between this object row and the object row right of this object row. The Link element is a child of the Lane element.
Query	If defined as child of the Node element, the Query element defines the objects displayed in the object row of the lane. If defined as child of the Link element, the Query element defines the kind of connections displayed in the diagram between the object rows.

Explorer Node Element	Required to Configure:
Color Rule	Definition of the colors used in the current cell of the grid report and the rules that define which objects are displayed in which color. For more information about the definition of color rules, see Defining Color Rules.
Indicator Rule	Definition of one or multiple icons displayed in the boxes representing the objects in the current cell of the lane report. The icons can either represent an indicator or any aspect defined via a query. For more information, see the section Defining Indicator Rules.

UKA new dynamic lane report is available in which the number of lanes is not fixed but dynamically specified via the query. The configured report is of the type Custom and based on the new template DynamicLaneReport. Please note the following: • In the report assistant, a Query node must be added to the explorer to define the content of the configured dynamic lane report via either Alfabet query language or native SQL. The query must return a grouped dataset. The first grouping level defines the lanes, the second grouping level defines the nodes on the lane and the third grouping level defines connections between the lanes where one of the columns of the dataset must return the REFSTR of one of the objects in the next lane for that the connection is to be established. • The respective column must be mapped to the attribute Link To Column in the root node of the dynamic lane report. Connections can also be re-directed to the previous lane. In this case, a dataset column returning the value PREV for all connections that shall be redirected to the previous lane must be included into the query result table and the column must be mapped to the Link Direction Column attribute in the root node of the dynamic lane report. • The root node of the report assistant provides attributes to configure the look of the lane report, the coloring and line styles either by mapping data from the query to attributes or by direct definition in the attribute. • Drill-down to the object profile of the object represented in the lanes is provided. Drill-down from connections is provided if the connection represents an object is defined for the connection. • This functionality will be documented with a future patch release.

Defining a Gallery Report

Gallery reports are designed in the **Report Assistant**. For information about how to open the **Report Assistant**, see Creating a New Graphic Report. For more information about how to work with the **Report Assistant**, see Configuring the Report with the Report Assistant.

The report assistant of the gallery report does not have an explorer. All attributes are on the root level and are displayed as a list of attribute fields for defining the configured report.

The following information is available:

- Defining the Basic Design of the Gallery Report
- Defining the Content of the Gallery Report

Defining the Basic Design of the Gallery Report

The basic design of the gallery report, like For example, the background color and any design that applies to all content independent from the displayed data is defined in the section **Common**:

Attribute	Description
Report Background Color	Select a color for the background between object boxes for the configured report from the color selector. For information about how to select colors, see Defining Color Attributes.
Area Margin	Define the spacing between the border of the graphic and the object boxes in the report in pixel.
Rows	Define how many rows shall be displayed for the gallery report. The object boxes resulting from the result dataset of the query of the report are distributed equally among the rows. if the graphic size exceeds the screen size in length or in width, scrolling might be required to view all objects of the report.
Node Width	Define the width of the individual object boxes in pixel.
Node Height	Define the height of the individual object boxes in pixel.
Node Radius	Define how much the edges of the object boxes are rounded off. If you define 0, the object boxes are displayed as squares.
Space Between Nodes Vertical	Define the vertical spacing between the object boxes in pixel.
Space Between Nodes Horizontal	Define the horizontal spacing between the object boxes in pixel.
Node Background Color	Select a default color for coloring of the object boxes for which no object specific coloring is defined via the query of the configured report. For information about how to select colors, see Defining Color Attributes.
Node Text Color	Select a default text color for coloring of the object boxes for which no object specific text coloring is defined via the query of the configured report. For information about how to select colors, see Defining Color Attributes.

Attribute	Description
Node Border Color	Select a default border color for coloring of the object boxes for which no object specific border coloring is defined via the query of the configured report. For information about how to select colors, see Defining Color Attributes.
Show Class Icon	Set to `True` if you want to display a class specific icon in front of the text in the middle of the object boxes. Please note the following: • If no text is defined for the boxes via the query of the configured report and the attribute **Image Column**, icons are not displayed, even if **Show Class Icon** is set to `True`. • If the report is based on a native SQL query and the query results are not referencing an object class either by returing the `REFSTR` of objects as first argument of the `SELECT` statement or via a `SetRowReference` instruction, icons are not displayed, even if **Show Class Icon** is set to `True`. • If object specific icons are defined for single objects in the result dataset, these icons are displayed instead of the class icon. • If a stereotype is defined via the query and the attribute Stereotype Column is set, the icons defined for the object class stereotypes are displayed instead of the icon for the object class.
Shadow	Set to `True` if you want to display object boxes with a shadow.

Defining the Content of the Gallery Report

The content of the gallery report and all object specific design and information is defined in a native SQL or Alfabet query in the section **Query** of the gallery report attributes. The content is then mapped to attributes in the section **Data** of the gallery report attributes to map the information to specific parts of the gallery report design. For general information about how to map the query result dataset to attributes of a configured report, see Mapping of Query Data to Attributes.

For each row in the result dataset a box is added to the report. The underlying technology assigns the box to the object that is the `FIND` class of the Alfabet query or the class for the the first argument of the `SELECT` statement returns a `REFSTR`. That means that For example, the assignment of a class icon to the text in the boxes is derived from this object and the default navigation by double-clicking on an object box opens the object profile of this object.

The following special cases may apply:

- The Alfabet query language instruction `SetRowReference` can be used in both Alfabet queries and native SQL queries to specify any other object class for that data is included into the result dataset to the referenced object class represented in the object boxes of the report.
- If the first argument of the `SELECT` statement returns the `REFSTR` values for multiple object classes, class icons and navigation is set according to the correct object class for each object box.

Additional information can be displayed in each corner of the object box. The information has to be added to the query results and the column of the result dataset mapped to the respective attribute **Column 1** - **Column 4** of the gallery report.

Each box corner can be regarded as a cell in a tabular report, that means that color assignment and picture assignment via Alfabet query language instructions can be used to design the box corners. The column names defined for the box corners are displayed as tooltip when the user moves the cursor over the information displayed in the corner.

The following data can be defined via the query of the configured report and mapped to the following attributes of the configured report to be used for the design of the gallery report:

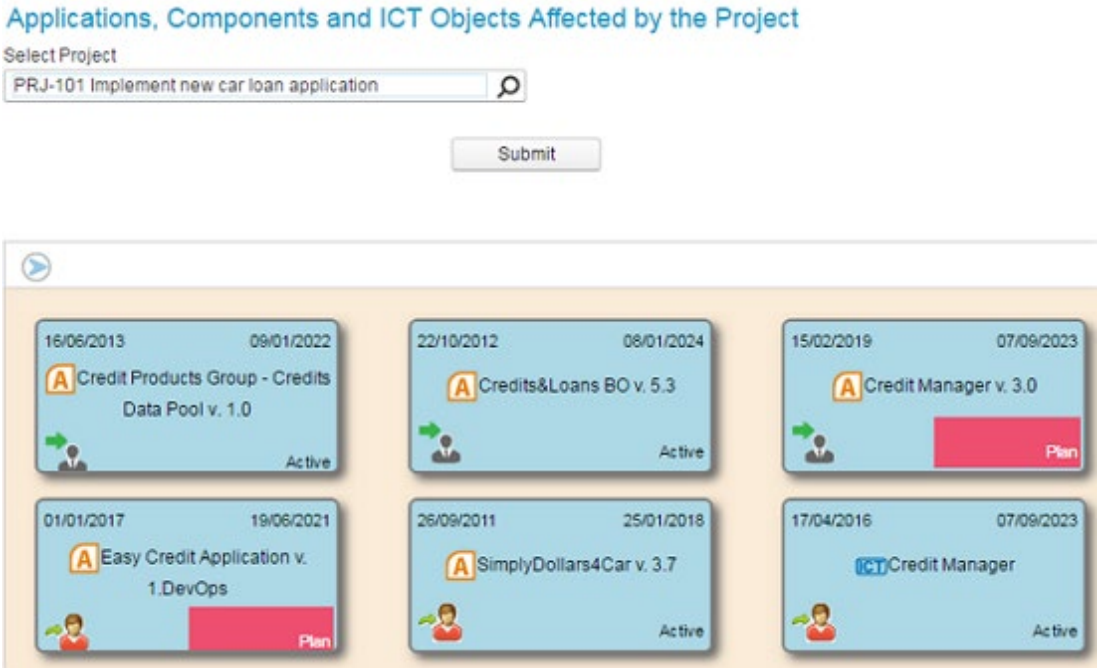
Attribute	Description	Expected Query Output
Image Column	Defines the name of the column containing the text to be displayed in the center of the object boxes.	A string.
Stereotype Column	Defines the name of the column containing the name of the stereotype of the current object. If this attribute is defined, and the attribute Show Class Icon is set to True, the stereotype icon is displayed instead of the object class icon left to the text in the center of the object boxes.	The value of the **Stereotype** attribute of the object.
Node Background Color Column	Defines the name of the column containing a HTML compliant color definition for coloring of the background of the object boxes. If this attribute is not set, all object boxes are displayed with the background color defined with the attribute **Node Background Color**.	An HTML compliant color code, like For example,: `#11A6D0`
Node Text Color Column	Defines the name of the column containing a HTML compliant color definition for coloring of the text in the object boxes. If this attribute is not set, the color defined with the attribute **Node Text Color** is used	An HTML compliant color code, like For example,: `#11A6D0`

Attribute	Description	Expected Query Output
	for the text in all object boxes.	
Node Border Color Column	Defines the name of the column containing a HTML compliant color definition for coloring of the border of the object boxes. If this attribute is not set, the color defined with the attribute **Node Border Color** is used for the text in all object boxes.	An HTML compliant color code, like For example,: `#11A6D0`
Column 1	Defines the information to be displayed in the upper left corner.	Any string to display.
Column 2	Defines the information to be displayed in the upper right corner.	Any string to display.
Column 3	Defines the information to be displayed in the lower left corner.	Any string to display.
Column 4	Defines the information to be displayed in the lower right corner.	Any string to display.
View Column	Defines the name of the column containing the definition of an alternative link target for the double click action on the object boxes. By default, the standard object profile of the object opens when a user double clicks on an object box.	`View=`*`ViewType`*`:`*`ViewName`* The `ViewType` can be one of the following: • ***Report*** for a configured report • ***GraphicView*** for a standard Alfabet view • ***ObjectView*** for an object profile `ViewName` is the name of the view. Please note that the link target only opens if each result in the query is returning a base class. In Alfabet queries this is the `FIND` class. In native SQL queries, the first argument of the `SELECT` statement must return the `REFSTR` of an object

Attribute	Description	Expected Query Output
		in the Alfabet database. This is also required if the link target does not refer to an object as base object. For more information about defining navigation to a target view, see Defining Navigation from the Report to Alfabet Views.
Legend Caption Column	Defines the text to be displayed in the legend for the given node type. The legend gives information about each different node. If the class icons are displayed, each class icon displayed is added to the legend. If bubbles are displayed, each differently colored bubble is explained.	Any string to display in the legend.

A simple example for a gallery report configuration is available in the section Mapping of Query Data to Attributes.

In the following example, a gallery report is configured to display objects of different object classes and information as icons or with a colored background. The report gives a user an overview about the applications and ICT Objects assigned as affected architecture to a project. The user can see from an image displayed in the lower left corner, whether he/she is responsible for the object. If he/she is responsible, the person icon is displayed in red. In the lower right corner, the report displays the information about the object state of the application or ICT object. If hte application is planned, the object state is displayed with a red background, because it is likely that the current user is currently involved in the implementation and action is required. In the upper corner of the boxes, the start date and end date of the application or ICT object is displayed to give a hint about the time frame for actions:



Defining Gantt Chart Reports

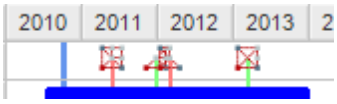
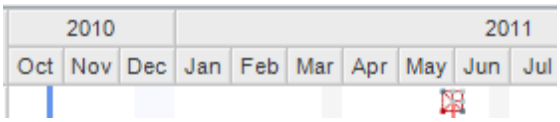
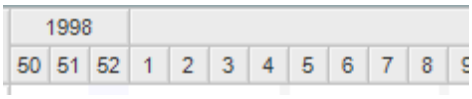
The general design of the report is defined via the attributes of the root node element of the explorer in the **Report Assistant**. The root node element can contain one or multiple elements **Item** to define the objects for which a row is added to the report and to configure the data displayed for the object in the gantt, For example, whether start and end date and/or a lifecycle is displayed. **Item** elements can be hierarchically structured in the report.

Defining the General Layout and Time Scale of the Gantt Chart

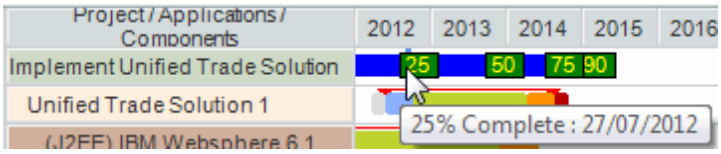
The general layout of the report is defined with the attributes of the root node element.

To edit the attributes of the root node, click the root node in the explorer and edit the attributes described below in the attribute window:

Attribute	Description
Caption	The x-axis field on top of the y-axis displays the caption text as a headline for the objects displayed in the report.

Attribute	Description
	
Caption Width	Defines the width of the y-axis fields in mm (1 mm = 4 pixel).
Calendar Step	Defines the default scaling of the time information on the x-axis. The time line can have one field for each year or one for each quarter or month or week of the year with the information about the year on top.  *FIGURE: Yearly scale*  *FIGURE: Quarterly scale*  *FIGURE: Monthly scale*  *FIGURE: Weekly scale* The user can change between the scaling options at runtime using the zoom in and zoom out options of the floating toolbar.
Today Based	Defines how the start and end year of the time scale is calculated. If set to `False`, the time scale will start with the year defined with the parameter **Start Year** and will end with the year defined with the parameter **End Year**. Or, if those parameters are not defined, all years for which objects in the report have a lifecycle defined will be displayed.

Attribute	Description
	If set to `True`, the time scale will start with the current year minus the number of years defined with the parameter **Start Year** and will end with the current year plus the number of years defined with the parameter **End Year.** **NOTE**: **Today Based** influences both the start and end year of the time scale. If **Today Based** is set to `"true"` and you do not set the parameters **Start Year** and **End Year**, the time scale will be limited to the current year.
Start Year	Defines the start year for the time scale of the Gantt report. The definition of the start year depends on the setting for the parameter **Today Based**. If **Today Based** is set to `False`, a year must be defined with ***StartYear*** to start the time scale of the report with the defined year. If ***TodayBased*** is set to `True`, an integer or -1 must be defined with ***StartYear***. The time scale starts with the year matching the current year minus the number of years defined with ***StartYear*** or, if ***StartYear*** is set to -1 or not defined at all, with the current year.
End Year	Defines the end year for the time scale of the Gantt report. The definition of the end year depends on the setting for the parameter **Today Based**. If **Today Based** is set to false, a year must be defined with **End Year** to end the time scale of the report with the defined year. If **Today Based** is set to true, an integer or -1 must be defined with **End Year**. The time scale ends with the year matching the current year plus the number of years defined with **End Year** or, if **End Year** is set to -1 or not defined at all, with the current year.
Show Legend	When set to `True`, a legend is shown for the lifecycle phases and the object classes in the report. 
Show Enterprise Milestones	When set to `True`, enterprise milestones are displayed in a separate row on top of the object lifecycles in the report. The target date of the enterprise milestone is displayed as vertical line. When you move the cursor over a milestone, a tooltip displays the name and dates of the milestone. double-clicking on a milestone symbol opens the enterprise milestones object view. 

Attribute	Description
Show Project Milestones	When set to `True`, project milestones are displayed in the row representing the project. When you move the cursor over a milestone, a tooltip displays the name and dates of the milestone. 
All Columns As Caption	If set to `True`, the y-axis of the report shows one single object box that contains all information about the current object in the dataset resulting from the query in the **Item** element. If set to `False`, the y-axis displays the dataset resulting from the query in the **Item** element as a table. **NOTE**: The information included into the dataset for technical reasons, like the start and end date of the object included in the dataset, is not diplayed in the object information.
dataset Width	This attribute is only relevant if the attribute **All Columns As Captions** is set to `False`. Define the width of the dataset in the report in pixel to restrict the space used for display of the additional information. The first row of the dataset is regarded as basic object information. The configured width is the width of the additional dataset rows only.

Defining the Hierarchy of the Objects Displayed in the Report

The root node element can contain one or multiple **Item** elements. These first level **Item** elements must have a query definition in their attribute specifications in order to find the objects displayed in the report. The query can either be a native SQL query or an Alfabet query.

If you are not familiar with Alfabet queries, see the chpater Defining Queries.

For special rules that apply to the definition of native SQL queries in the context of configurations for Alfabet, see the section Defining Native SQL Queries.

The information displayed on the y-axis of the Gantt report is identical to the column values that would result from a tabular output of the defined query.

Report columns that are defined for technical reasons are not displayed in the object information in the report. This includes:

- The first column of the output of a native SQL query. This column must specify the `REFSTR` of the object for technical reasons and is ignored for the visible output.
- The **Start Date** and **End Date** property of an object. The start date and end date are instead displayed on top or instead of the lifecycle information for the object. See below for details of the configuration.

The information about the object in each row provided in the dataset resulting from the query can be displayed in two ways dependent on the attribute **All Columns As Caption** of the report:

- If **All Columns As Caption** is set to `True`, all information is displayed in a single object box prior to the lifecycle information.

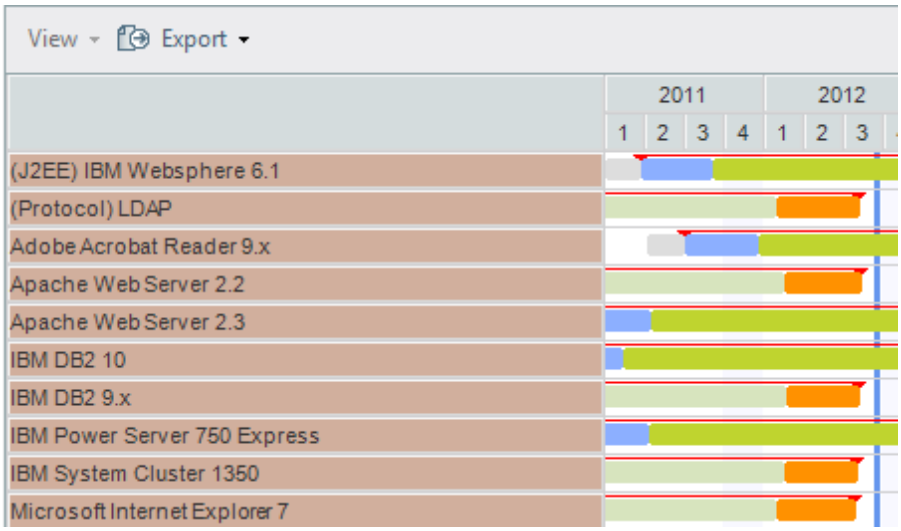


FIGURE: Name and Version of each Application are displayed in a single box

- If **All Columns As Caption** is set to `False`, a column is displayed for each column of the dataset. The first column is regarded as object box and has the title defined in the attribute **Caption** of the root node of the report. All other columns are regarded as additional information and have a header title that is identical to the header title of the dataset table:

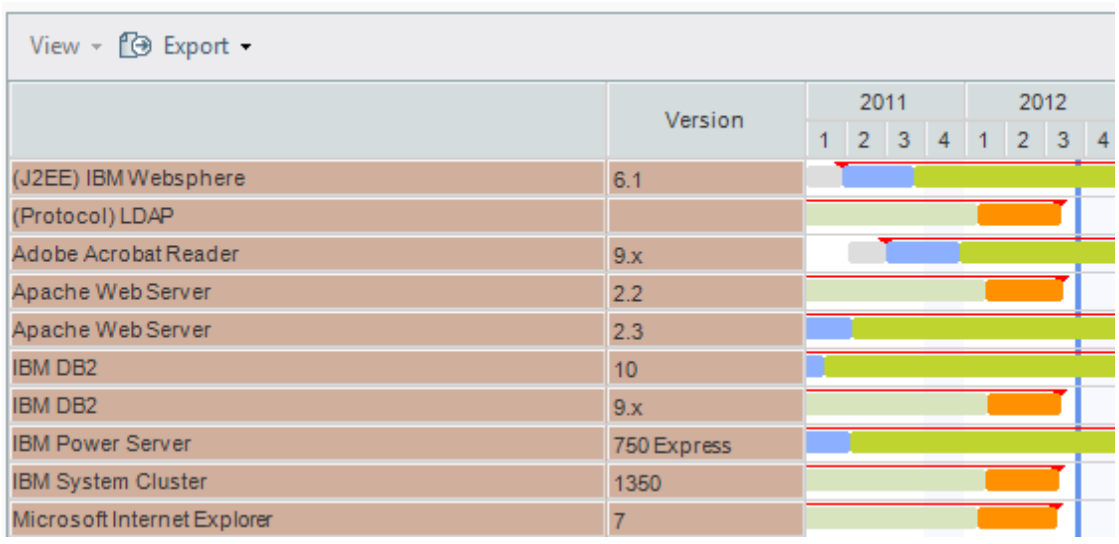


FIGURE: Name and Version of each Application are displayed in a separate column

The attribute **dataset Width** can be used to restrict the size of the dataset containing the additional information.

If the attribute **All Columns as Caption** is set to `False`, indicators can be displayed as icons in the report. If the information displayed about an object includes an indicator

value, and the indicator type is configured to display icons representing values, the icon is displayed in the gantt report instead of the value of the indicator.

The report can either display a flat list of objects and their lifecycle, or a grouped report can be configured. There are two methods to configure a grouped report:

- The complete content of the report is defined in one query resulting in a grouped dataset.

  If the attribute **All Columns as Caption** is set to `False` the information about each grouping level is displayed in separate columns of the dataset. The header titles are identical to the header titles resulting from the query.

- A separate query is added to the report configuration for each object class added to one of the levels of the report.

  When separate queries are defined, a high number of queries must be executed by the database to display the report. This can lead to performance issues. Therefore, it is recommended to base the report on a grouped dataset defined in a single query. Nevertheless for some use cases, like For example, defining different indicators for objects assigned to the same level, it is required to define multiple queries for the report.

  If the attribute **All Columns as Caption** is set to `False` the dataset is build from the first level. The header titles are identical to the header titles resulting from that query. The information of the subordinate datasets is written in the resulting columns in the order of their specification without regard of the correctness of the header titles. Therefore, it is recommended to base the report on a grouped dataset defined in a single query when displaying information about objects in a dataset.

Defining a Grouped Report Based on a Single Query

By default, the color of the boxes and text in the boxes on the y-axis is identical to the background color and foreground color defined for the object class or the object class stereotype in the relevant class settings for the object class. The coloring can be changed by defining color rules for the report. The object boxes are then colored according to the defined color rules. Boxes representing objects of the same object class may then have different colors assigned because of a color rule defined to color objects For example, dependent on their object state. For more information about the configuration of color rules, see Defining Color Rules.

The report can either display a flat list of objects and their lifecycles, or a grouped report can be configured with the query. For a grouped report, the y-axis of the Gantt report displays the information defined for the column headers of the respective group.

Grouped reports are defined via `GroupBy_Ex` instructions defined in Alfabet query language. Alfabet query language instructions can also be combined with native SQL queries.

A detailed description about how to define a grouped report using the `GroupBy_Ex` command is given in the section Grouping Query Results in Expandable Reports in the chapter Defining Queries.

Report columns that are defined for technical reasons are not displayed in the object boxes in the report. This includes:

- The first column of the output of a native SQL query. This column must specify the `REFSTR` of the object for technical reasons and is ignored in the visible output.
- `StartDate` and `EndDate` property of an object. The start date and end date are instead displayed on top or instead of the lifecycle information for the object. See below for details of the configuration.

If the query output results in a grouped report, an **Item** element without a query specification must be added to the explorer tree for each grouping level in a hierarchical structure, that means each subordinate level **Item** element as child of the superordinate **Item** element. The hierarchy of **Item** elements correspond to the grouping levels. To enhance the overview over the report structure, an attribute **Name** is available for the item elements that allow to rename the **Item** explorer nodes.

The attribute **Visible** of the **Item** element allows you to define rows that are not displayed in the report. This is useful to display the Gantt charts of objects that are related via an intermediate object without displaying the information about the intermediate object class.

For each element **Item** the display of the lifecycle and/or the start and end dates of the objects can be defined by setting different attributes of the **Item** element.

Item elements are not defined per object class but per grouping level. You can define grouped datasets with more than one object class per level. In this case, however, the settings in the **Item** element for the level are valid for all object classes displayed in the level.

Defining a Grouped Report on Basis of One Query Per Object Class

Item elements can be structured hierarchically. If an element **Item** contains a child element **Item**, for each row added to the report on basis of the definition in the parent **Item** element, subordinate rows are added to the report on basis of the definition in the child element **Item**. To enhance the overview over the report structure, an attribute **Name** is available for the item elements that allow the **Item** explorer nodes to be renamed.

For each element ***Item*** an Alfabet query or a native SQL query can be specified that defines for which objects rows are added to the report.

Note the following for the definition of the query for the **Items Query** attribute:

- If an element ***Item*** is child of a parent **Item** element, the query defined for the **Item** element must contain the parameter `BASE` in a `WHERE` clause to map the objects of this layer to the objects in the parent layer. `BASE` is identical to the `REFSTR` of the object in the parent layer.

  This specification is not mandatory for the top layer objects. The attribute `BASE` can only be used in the Alfabet query or native SQL query defining the top layer objects if the report is assigned to an object class with the attribute **Apply To Class**.

- For each **Item** element you can specify an attribute **Param Name** that allows a name to be defined that can be used as a parameter in queries of other **Item** elements in order to refer to the class found by the query in the selected **Item** element. In the queries of other **Item** elements, the

parameter returns the `REFSTR` of the `FIND` class of the current Alfabet query or the `REFSTR` defined as first `SELECT` argument in the native SQL query.

For information about the use of parameters in Alfabet queries, see Referring to the Current Alfabet Context in a WHERE Condition. For information about the use of parameters in native SQL queries, see Using Alfabet Parameters in the section Defining Native SQL Queries.

- The information displayed in the boxes on the y-axis of the Gantt chart report is identical to the column headers that would result from a tabular output of the defined query.

Report columns that are defined for technical reasons are not displayed in the object boxes in the report. This includes:

- `StartDate` and `EndDate` property of an object. The start date and end date are instead displayed on top or instead of the lifecycle information for the object. See below for details of the configuration.
- The first column of the output of a native SQL query. This column must specify the `REFSTR` of the object for technical reasons and is ignored in the visible output.

By default the color of the boxes on the y-axis and the color of the text in the boxes is identical to the background and foreground colors defined for the object class or the object class stereotype in the relevant class settings for the object class. For more information, see Configuring Class Settings for Object Classes and Object Class Stereotypes. The coloring can be changed by defining color rules for the report. The object boxes are then colored according to the defined color rules. Boxes representing objects of the same object class may then have different colors assigned because of a color rule defined to color objects For example, dependent on their object state. For more information about the configuration of color rules, see Defining Color Rules.

The attribute **Visible** of the **Item** element allows you to define rows that are not displayed in the report. This is useful to display the Gantt charts of objects that are related via an intermediate object without displaying the information about the intermediate object class.

For each element **Item** the display of the lifecycle and/or the start and end dates of the objects can be defined by setting different attributes of the **Item** element.

Defining the Attributes of the Item Element

The table below lists all attributes of the ***Item*** element. This also includes elements that define the display of lifecycles and start and end dates. A detailed description on how to specify those parameters to configure the chart is given in the sections Defining the Display of Lifecycles and Defining the Display of Start and End Dates..

Attribute	Description
Param Name	The **Param Name** attribute allows a parameter name to be defined that can be used in all queries defined for the report in order to refer to the `REFSTR` of the current object of the **Item** level that the attribute is defined for.

Attribute	Description
LC Query	Defines the object's lifecycle displayed in the report. If the parameter is set to `Default` and a lifecycle is defined for the object, all lifecycle states are displayed automatically in the chart. If you want to display any other lifecycle information, you must define the lifecycle that shall be displayed via an Alfabet query or a native SQL query. For more information about how to define the query see Defining the Display of Lifecycles.
Gantt Item Type	Defines the display mode of the start and end date of the object. For more information about the different modes to display the start and end date, see Defining the Display of Start and End Dates.
Color	Defines the color for the display of the start and end date information for the object class.
Visible	Defines whether information about the object class is visible in the report.
Name	Defines a name for the ***Item*** element. The name is used in the explorer tree as name for the ***Item*** node.
Alfabet Query/ Query as Text/ Native Sql	Defines the objects displayed in the report via an Alfabet query or a native SQL query. For each object found by the Alfabet query a line is added to the report. If you want to define an Alfabet query, you can either use the **Alfabet Query Builder** available via the **Alfabet Query** attribute or write the Alfabet query in a text editor available via the **Query as Text** attribute. If you want to define a native SQL query, use the text editor available via the **Native Sql** attribute. The text editor has separate tabs for the definition of the native SQL query and the definition of Alfabet query language instructions that can be used in combination with native SQL queries to built a grouped dataset. To open an editor, click in the respective attribute field to display the **Browse**  button and click the button.

Defining the Display of Lifecycles

The lifecycle of an object defined in an element **Item** of the report is displayed if:

- A lifecycle is defined that is related to the object and
- The attribute **LC Query** is defined in the **Item** element that contains the query finding the objects for that the lifecycledefinition is valid. If the report is based on a single query returning a grouped dataset, the **LCQuery** attribute must be defined for the **Item** element containing the object query only and must return lifecycle definitions for all objects in the report.

If a lifecycle is directly defined for the object, the attribute **LC Query** can be set to `Default`. All lifecycle states defined for the object are then displayed.

If you want to display only a subset of the lifecycle states or the lifecycle an object assigned to the object found by the report's query for the **Item**, you must define a query in the `LCQuery` attribute. The query must match the following conditions:

- The object found by the query must be of the object class `TimeStatus`.
- The query must be assigned to the object found for the **Item** element by using the parameter `BASE` in a `WHERE` clause of the query. The parameter `BASE` equals the `REFSTR` of the current object found by the query defined in the **Item** element.
- For Alfabet queries: the Show properties of the report must specify the following attributes of the object class `TimeStatus`: `Owner`, `Status`, `StartDate` and `EndDate`
- For native SQL queries: The `SELECT` clause of the query must define the `REFSTR` of the object class TimeStatus as first property followed by the definition of the properties `Owner`, `Status`, `StartDate` and `EndDate`. All other properties defined in the `SELECT` clause are ignored.
- the query must return a dataset with the column headers named `Owner`, `Status`, `StartDate` and `EndDate`.

The following example is a simple native SQL query finding the lifecycle elements of the object found for the **Item**:

```
SELECT ts.REFSTR, ts.OWNER AS 'Owner', ts.STATUS AS 'Status',
ts.STARTDATE AS 'StartDate', ts.ENDDATE AS 'EndDate'

FROM TIMESTATUS ts

WHERE ts.OWNER = @BASE
```

Defining the Display of Start and End Dates

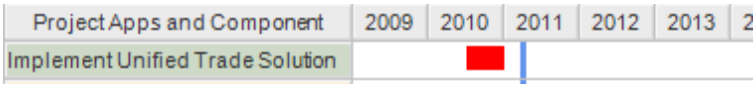
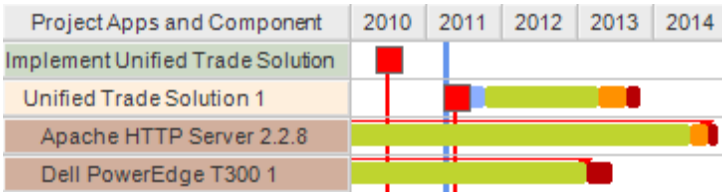
The start and end date of an object defined in an element **Item** of the report is displayed if:

- The object has a start and end date defined and
- the query defining the **Item** content includes both the `StartDate` and `EndDate` property of the object class in the **Show Properties** of the Alfabet query or in the `SELECT` statement of the native SQL query.

For grouped reports, the columns for the `StartDate` and `EndDate` properties of the basic find class of the report are relevant for the display of start and end dates of all levels of the report. To define a report with start and end dates in various levels:

- Add all `StartDate` and `EndDate` properties for all object classes in all levels to the **Show Properties** of the Alfabet query or the `SELECT` clause of the native SQL query.
- Use the `JOINCOLUMNS` command of the Alfabet query language to join all `EndDate` properties in the `EndDate` column the basic find class and all `StartDate` properties in the `StartDate` column for the basic find class.

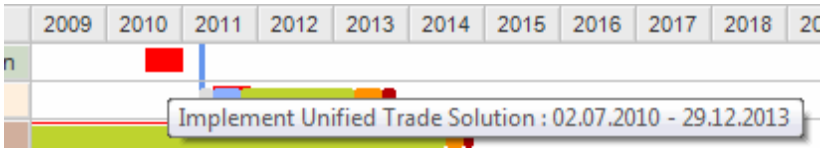
By default, the start and end dates are displayed as triangles connected by a line. You can change the display mode of the start and end dates by setting the **Gantt Item Type** attribute of the **Item** element. The following display modes are available:

Gantt Item Type	Display	Example
`Rectangle`	A rectangle with rounded edges is displayed between start and end date.	
`StartEndLine`	A line starting and ending with a triangle is displayed in the upper part of the cell for the gantt chart. This representation can be used to display lifecycle and start and end time information in parallel.	
`Milestone`	A rectangle with sharp edges is displayed at the start date of the object.	
`IconLine`	A rectangle with sharp edges is displayed at the start date of the object. A vertical line is drawn from the rectangle to the lower border of the diagram. The line is displayed behind the other objects in the Gantt chart. When the start and end date information is displayed in parallel to the lifecycle of the object, the rectangle is displayed on top of the lifecycle.	

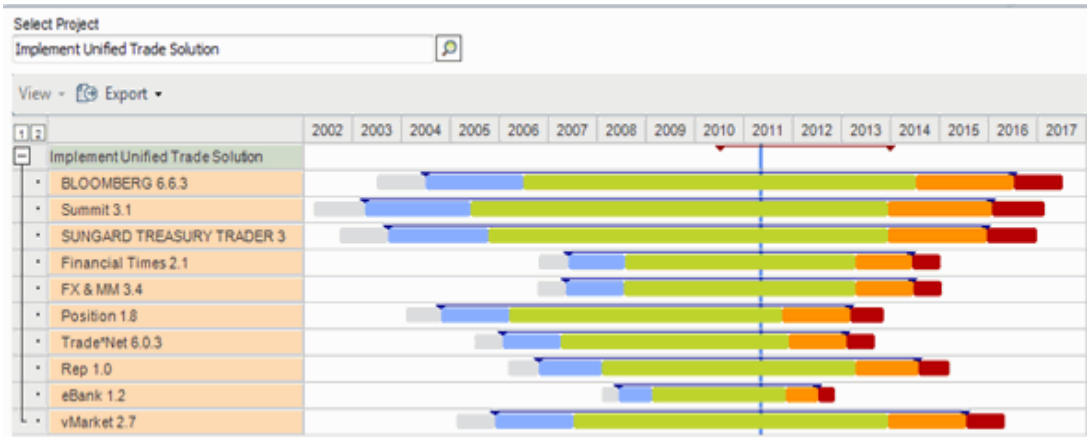
If both start and end date and the lifecycle of an object shall be displayed, you must select the default **Gantt Item Type** `StartEndLine` that is displayed on top of the lifecycle or IconLine, that is displayed in front of the lifecycle. For Milestone and IconLine, the start and end date information is displayed behind the lifecycle information and is Therefore, not visible.

The color of all start and end date representations can be changed by setting the attribute **Color** of the **Item** element. By default, all start and end date information is displayed in red.

When you move the cursor over the start and end date information, a tooltip is displayed that shows the object's name as well as the start and end date of the object:



The following example displays the lifecycles of applications assigned to the project architecture of a selected project. The start and end dates of both the project and the applications is also displayed in the report:



The report can For example, be created on basis of the following Alfabet query:

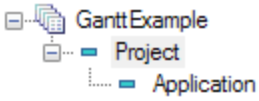
```
ALFABET_QUERY_500

FIND Project

    InnerJoin ProjectArch ON ProjectArch.Project = Project.REFSTR

    InnerJoin Application ON ProjectArch.Object = Application.REFSTR

WHEREProject.REFSTR =:BASE

Instructions

    GROUPBY_EX("Project.REFSTR","Application.REFSTR","Project",0);

    REMOVECOLUMNS("Project.REFSTR,Application.REFSTR");

    JOINCOLUMNS("Project.StartDate,Application.StartDate","Project.St
    artDate", "");

    JOINCOLUMNS("Project.EndDate,Application.EndDate","Project.EndDat
    e","");

EndOfInstructions

QUERY_XML

<QueryDef>

    <ShowProperty Type="Property" ClassName="Project" Name="REFSTR"
    />
```



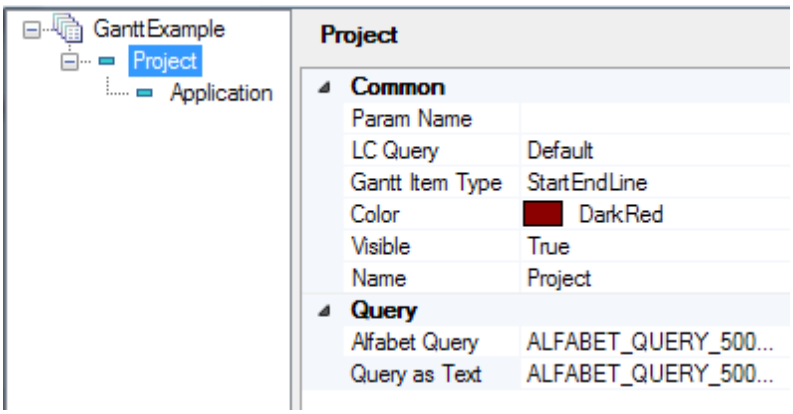
```
        <ShowProperty Type="Property" ClassName="Project" Name="Name" />
        <ShowProperty Type="Property" ClassName="Project"
        Name="StartDate" />
        <ShowProperty Type="Property" ClassName="Project" Name="EndDate"
        />
        <ShowProperty Type="Property" ClassName="Application"
        Name="REFSTR" />
        <ShowProperty Type="Property" ClassName="Application" Name="Name"
        />
        <ShowProperty Type="Property" ClassName="Application"
        Name="Version" />
        <ShowProperty Type="Property" ClassName="Application"
        Name="StartDate" />
        <ShowProperty Type="Property" ClassName="Application"
        Name="EndDate" />
        <SortProperty Type="Property" ClassName="Project" Name="REFSTR"
        />
        <SortProperty Type="Property" ClassName="Application"
        Name="REFSTR" />
    </QueryDef>
```

The query results in a grouped report. The `StartDate` and `EndDate` properties of both project and applications are added to the Show properties. The instructions join the `StartDate` values and the `EndDate` values respectively to allow display of start and end date information in all levels of the report.

In the **Report Assistant** of the report, one **Item** element must be added to the root node for the definition of the first level. The first level Item element contains a second level **Item** element:

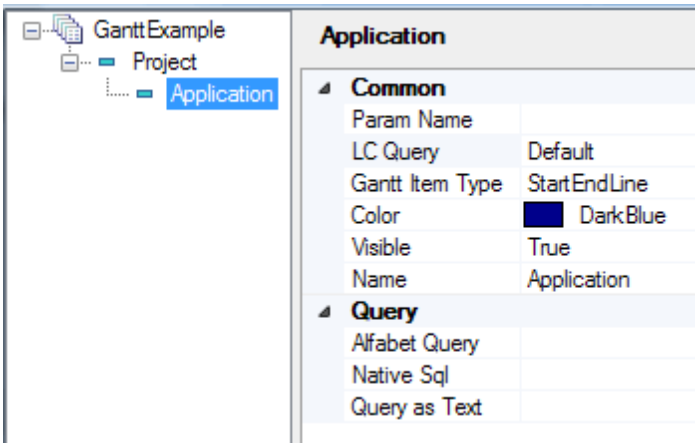


The Alfabet query defining the grouped dataset is defined in the attributes of the first level **Item** element. The first level element also contains the definition of the display of start and end date information for the projects displayed as first level. Projects have no lifecycle defined and Therefore, the `LCQuery` attribute is not considered for this **Item**.



The second grouping level of the report displays information about the objects of the object class Application. The attributes of the **Item** element define the display of the lifecycle and start and

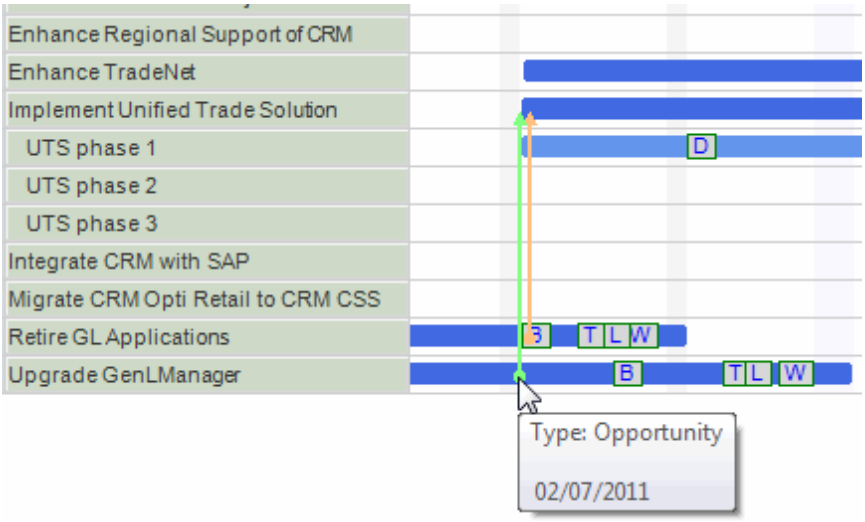
end date information for the applications assigned to the project. Applications are found via the query defined for the first level Item element, Therefore, no query definition to find objects is defined in the **Item** attributes:



Displaying Connections between Objects

Gantt charts can include information about relations between objects displayed in the Gantt chart, For example, migration links that indicate the time when an object is substituted by another one in the chart or project dependencies that indate the dependency of one project from the progress of another project.

Connections are displayed as an arrow pointing from one object to another at a given time in the timescale of the lifecycle diagram:



When an object that is connected to another object via a connection is included multiple times in the report, the connection will only be displayed for the first occurence of the object in the report.

Different types of connections can be implemented that can be destinguished by color and by name. When moving the cursor over the connection, a tooltip is displayed. The tooltip informs either about the name of the objects that are connected or displays additional information about the connection defined in the query the connection is based on. The date of the connection is always displayed in the tooltip.

Customer configured connections are not included into the legend. Instead, the tooltip can be used to give the required information to destinguish between different kind of connections.

Connections are based on a customer defined query that must result in a dataset with at least three columns returning the following values:

- The `REFSTR` property of the object that the connection is starting at.
- The `REFSTR` property of the object that the connection is ending at.
- The time in the time scale of the gantt diagram at that the connection shall be displayed.

Optinally, additional columns can be defined for the resulting dataset. If the resulting dataset includes additional columns, the content of the respective columns in a row is displayed as tooltip for the connection. If no additional columns are defined, the tooltip displays the information of the object boxes of the connected rows in the report. The date of the connection is added to the tooltip in both cases.

For each row in the resulting dataset, a connection is displayed in the diagram.

Connections are defined as sub-nodes of the report's root node. Right-click the report's root node in the **Report Assistant** and select **Add New Connection** to add a connection element to the report and define the connection with the following attributes:

Attribute	Description
From Column	Defines the name of the column returning the REFSTR of the object in the report that the connection starts with.
To Column	Defines the name of the column returning the REFSTR of the object in the report that the connection ends with.
Date Column	Defines the name of the column returning the date at which the connection is displayed in the diagram.
Color	Defines the color for the display of the connection in the gantt diagram.
Name	Defines a name for the connection. The name is used in the explorer tree of the **Report Assistant** as name for the **Connection** node.
Alfabet Query/ Query as Text/ Native Sql	Defines the objects that are to be connected and the date at which the connection is displayed in the report via an Alfabet query or a native SQL query. The query must result in a dataset with at least three columns returning the REFSTR of the start and end objects and the date of the connection. The output of any additional column is displayed in the tooltip for the connection in the diagram. If you want to define an Alfabet query, you can either use the **Alfabet Query Builder** available via the **Alfabet Query** attribute or write the Alfabet query in a text editor available via the **Query as Text** attribute.

Attribute	Description
	If you want to define a native SQL query, use the text editor available via the **Native Sql** attribute. The text editor has separate tabs for the definition of the native SQL query and the definition of Alfabet query language instructions that can be used in combination with native SQL queries to built a grouped dataset. To open an editor, click in the respective attribute field to display the **Browse** button and click the button.

A gantt report displays the lifecycle of all applications that are part of the affected architecture of a selected project. If a migration is defined for the project, the migration rules affecting the applications displayed in the gantt chart shall be displayed as connections.

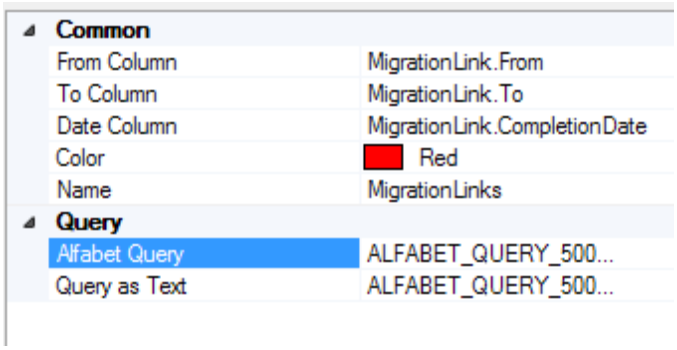
A connection is defined for the report with the following Alfabet query finding migration rules (=MigrationLink) defined for migrations of the current project and returning the From object, To object and completion date of the migration rule:

```
ALFABET_QUERY_500

FIND

MigrationLink

InnerJoin Migration ON MigrationLink.Migration = Migration.REFSTR

InnerJoin Project ON Project.Migrations = Migration.REFSTR

WHERE

    Project.REFSTR =:BASE

AUTODSINFO

QUERY_XML

<QueryDef>

    <ShowProperty Type="Property" ClassName="MigrationLink"
    Name="From" />

    <ShowProperty Type="Property" ClassName="MigrationLink" Name="To"
    />

    <ShowProperty Type="Property" ClassName="MigrationLink"
    Name="CompletionDate" />

    <SortProperty Type="Property" ClassName="MigrationLink"
    Name="From" />

    <SortProperty Type="Property" ClassName="MigrationLink" Name="To"
    />

    <SortProperty Type="Property" ClassName="MigrationLink"
    Name="CompletionDate" />

</QueryDef>
```

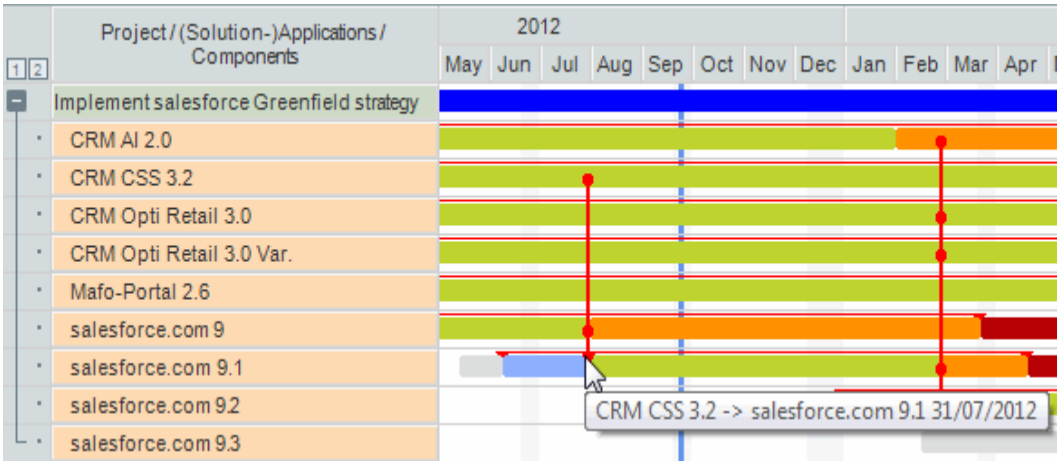
The dataset resulting from the query returns three columns:

Migration Rule Source	Migration Rule Target	Migration Rule Completion Date
26-200-0	26-620-0	30/03/2014
26-514-0	26-620-0	28/11/2014
26-516-0	26-620-0	28/11/2014
26-590-0	26-620-0	30/03/2014
26-591-0	26-620-0	30/09/2012
26-592-0	26-620-0	30/03/2014
26-593-0	26-620-0	30/09/2012
26-597-0	26-620-0	30/03/2014
26-598-0	26-620-0	30/09/2012

In the attributes of the connection element in the report, the technical name of the columns in the dataset, that is the ClassName and Name attribute of the respective Show property separated with a dot are defined as From Column, To Column and Date Column:



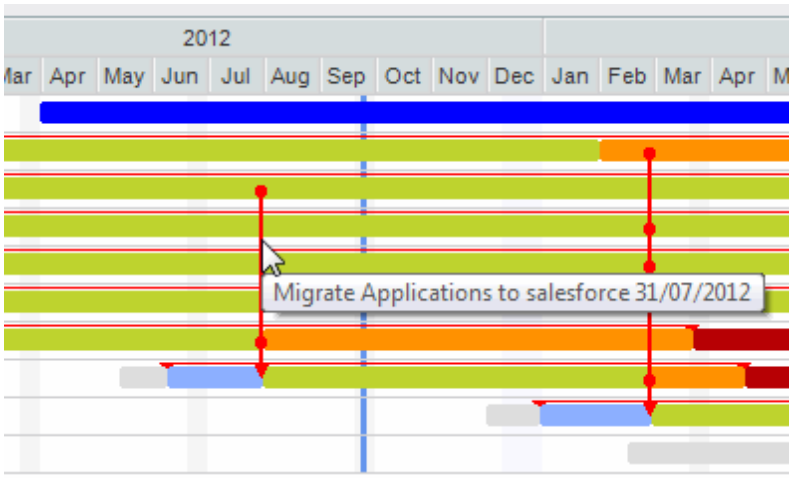
The resulting report shows the connections as red arrows between objects. In the example, multiple migrations are overlapping, because their completion date and target object is identical. You can see a red dot at each lifecycle where there is a start point for the migration rule.



If you want the report to display the name of the migration for which the migration rule is defined in the tooltip, the name of the Migration must be added as additional Show property to the query defined for the connection, resulting in a dataset with four columns:

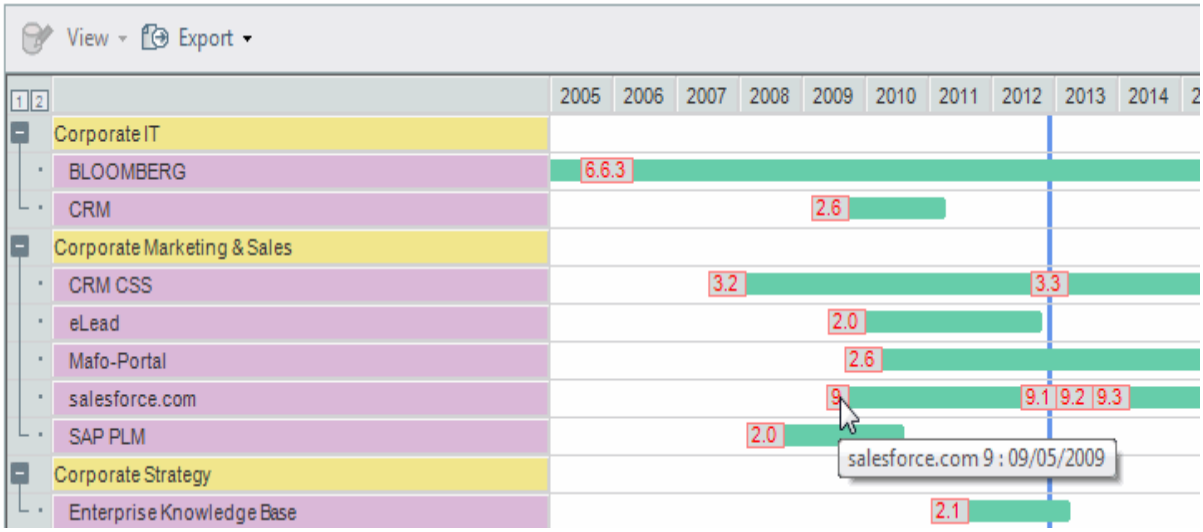
Migration Rule Source	Migration Rule Target	Migration Rule Completion Date	Migration Name
76-2694-0	76-3331-0	31/07/2012	Migrate Applications to salesforce
76-2694-0	76-3333-0	31/10/2013	Migrate Applications to salesforce
76-2760-0	76-3332-0	28/02/2013	Migrate Applications to salesforce

The tooltip is then changed to display the name of the migration:



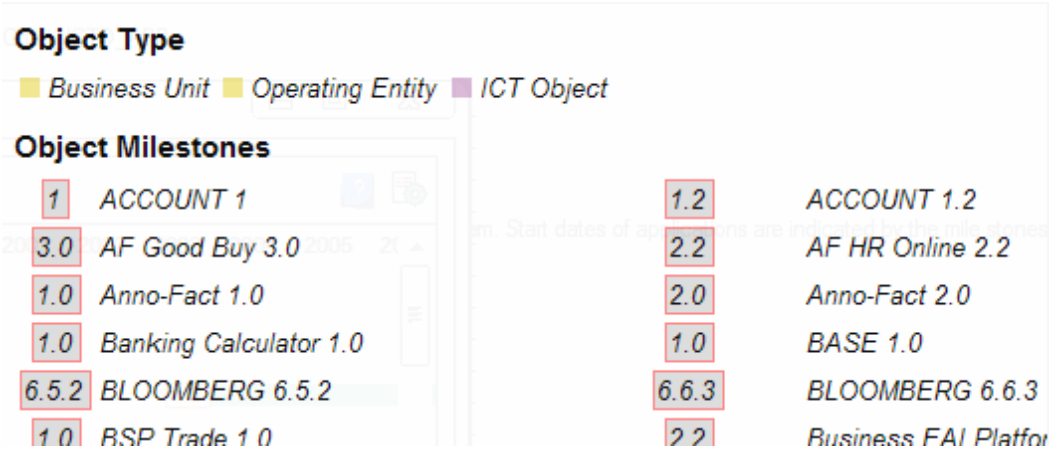
Displaying Customer Defined Milestones In the Lifecycle

Relevant events can be defined as milestones for the lifecycle of objects in the Gantt chart. For example, the assignment of new versions of an application to an ICT object can be displayed on the lifecycle of the ICT object to give information about software updates.



The customer configured milestones are displayed as a rectangle diplaying a customer defined label text in the lifecycle of the objects in the report. If the user moves the cursor over the milestone, a tooltip is displayed giving customer defined information and information about the exact date of the milestone.

In the legend of the report, milestones are listed with the information displayed in the tooltip as explanatory text.



The milestone definition is based on a customer defined native SQL or Alfabet query that returns the information about the location of the milestone in the time scale and for the lifecycle of which object the milestone shall be displayed. The query can also return the information displayed in the label and in the tooltip and legend for the milestone. Alternatively, the tooltip and label text can be a static text used for all milestones that is defined in the attributes of the milestone configuration.

The query defined for the milestone must return at least two columns:

- The `REFSTR` of the object for that the milestone shall be displayed in the lifecycle. If the query returns results for objects that are not displayed in the Gantt chart report, the respective row in the result dataset is ignored.
- A date that defines the time on the time scale of the Gantt Chart that the milestone is displayed at. If the query returns results for a time that is not displayed in the Gantt chart report, the respective row in the result dataset is ignored.

The query defined for the milestone can optionally return columns to display individual information for each milestone:

- A string that shall be displayed as a label in the milestone. The string should be short because the length of the milestone is matching the length of the string.
- A string that is displayed in a tooltip if the user moves the cursor over the milestone. The string is displayed together with the date information, that is automatically added to the string.

If you do not want an individual label and tooltip text to be displayed for a milestone, you can alternatively define a fixed label and/or tolltip text in the milestone definition in the **Report Assistant**.

Milestones are defined as sub-nodes of the first level **Item** element of the Gantt report.

Right-click the **Item** element in the **Report Assistant** and select **Add New Milestone** to add a milestone element to the report and define the milestone with the following attributes:

For technical reasons, the option **Add New Milestone** is available in the context menu of all **Item** elements in the hierarchy. Nevertheless, only milestones defined for first level **Item** elements are displayed in the report.

Attribute	Description
Background Color	Defines the background color of the milestones in the Gantt Chart.
Foreground Color	Defines the text color of the milestones in the Gantt Chart.
Border Color	Defines the color of the border of the milestones in the Gantt Chart.
Label	Defines the text to be displayed in all milestones displayed as a result of the query defined with the attribute **Alfabet Query/ Query as Text/ Native Sql**. Alternatively, an individual label text for each displayed milestone can be derived from the query result dataset. The text specified with the attribute **Label** is ignored if the **Label Column** attribute is defined.
Hint	Defines the text to be displayed in the tooltip of all milestones displayed as a result of the query defined with the attribute **Alfabet Query/ Query as Text/ Native Sql**. Alternatively, an individual tooltip text for each displayed milestone can be derived from the query result dataset. The text specified with the attribute **Hint** is ignored if the **Hint Column** attribute is defined.
Name	Defines a name for the milestone. The name is used in the explorer tree of the **Report Assistant** as name for the ***Milestone*** node.
Row Object Column	The name of the column in the result dataset of the query defined with the attribute **Alfabet Query/ Query as Text/ Native Sql** that returns the REFSTR of the object the milestone is valid for. For information about column name specifications, see Defining Column Names and Captions in the chapter Defining Queries.
Date Column	The name of the column in the result dataset of the query defined with the attribute **Alfabet Query/ Query as Text/ Native Sql** that returns the date in the time scale of the Gantt Chart the milestone shall be displayed at. For information about column name specifications, see Defining Column Names and Captions in the chapter Defining Queries.
Label Column	The name of the column in the result dataset that contains the string to be displayed as label in the milestones. Alternatively, you can use the attribute **Hint** to define a static hint displayed for all milestones in the Gantt chart.

Attribute	Description
	For information about column name specifications, see Defining Column Names and Captions in the chapter Defining Queries.
Hint Column	The name of the column in the result dataset that contains the string to be displayed in the tooltip of the milestones. Alternatively, you can use the attribute **Label** to define a static label displayed in all milestones in the Gantt chart. For information about column name specifications, see Defining Column Names and Captions in the chapter Defining Queries.
Alfabet Query/ Query as Text/ Native Sql	Defines the milestones to be displayed in the report. The query must return a dataset with at least 2 columns defining the location of the milestone in the Gantt Chart. One column must return a date to define the location of the milestone in the time scale, the other column must return the `REFSTR` of the object for that the milestone shall be added to the lifecycle information. Optionally, information about the label and tooltip of the milestone can also be derived from columns of the query result dataset. If you want to define an Alfabet query, you can either use the **Alfabet Query Builder** available via the **Alfabet Query** attribute or write the Alfabet query in a text editor available via the **Query as Text** attribute. If you want to define a native SQL query, use the text editor available via the **Native Sql** attribute. The text editor has separate tabs for the definition of the native SQL query and the definition of Alfabet query language instructions that can be used in combination with native SQL queries. To open an editor, click in the respective attribute field to display the **Browse** button and click the button.

For example, a report that displays the lifecycle of ICT Objects that are grouped by the organization they are owned by, shall display the start date of applications assigned to the ICT object as milestones.

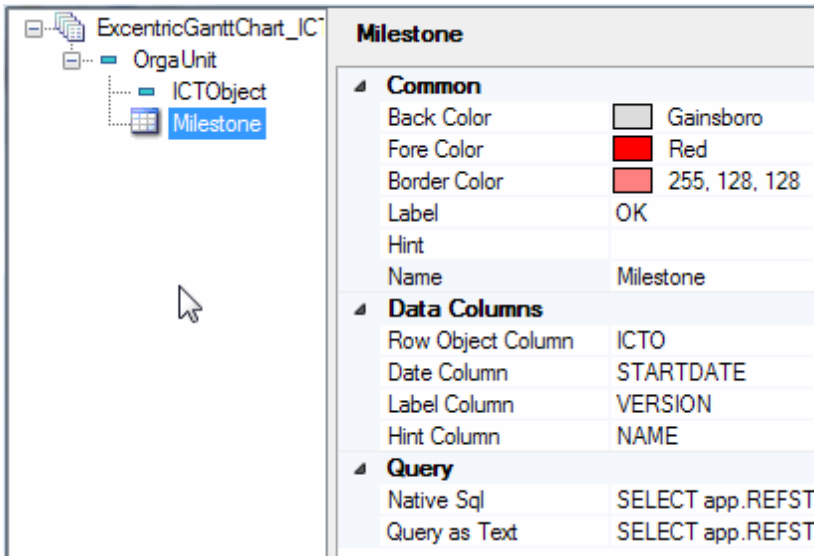
The native SQL query in the milestone definition finds all applications that are assigned to ICT objects owned by organizations.

```
SELECT app.REFSTR, icto.REFSTR AS ICTO, app.STARTDATE, app.VERSION,
app.NAME + ISNULL(' ' + app.VERSION, '') AS NAME

FROM ORGAUNIT ou, ICTOBJECT icto, APPLICATION app

WHERE icto.OWNER = ou.REFSTR

AND app.ICTOBJECT = icto.REFSTR
```

The query returns the REFSTR of the ICT Object, the start date of the application, the version number of the application and the name of the application. The dataset that would result from the query displays one row for each milestone that are added to the Gantt chart if both the start date is within the time scale displayed in the report and the REFSTR of the ICT object is identical with the REFSTR of an ICT object in the report.

	ICTO	STARTDATE	VERSION	NAME
1	26-126-0	14/11/2009 00:00:00 000	2.2	Business EAI Platform 2.2
2	26-129-0	12/02/2009 00:00:00 000	2.2	Groupware Services 2.2
3	26-68-0	25/08/2009 00:00:00 000	2.6	Mafo-Portal 2.6
4	26-94-0	24/03/2010 00:00:00 000	2.5	PS Global 2.5
5	26-105-0	24/01/2009 00:00:00 000	2.9	ALLFiance PISA 2.9
6	26-21-0	10/07/2009 00:00:00 000	2.0	SAP@OptiRetail 2.0
7	26-23-0	17/04/2003 00:00:00 000	4.0	SAP@AI 4.0
8	26-27-0	03/02/2009 00:00:00 000	2.9	BookIT 2.9
9	26-41-0	27/03/2009 00:00:00 000	2.2	Corporate FI-CO 2.2
10	26-43-0	19/01/2011 00:00:00 000	2.8	SAP International 2.8

In the definition of the milestone, the names of the columns from the result dataset are defined in the attributes Row Object Column, Date Column, Label Column and Hint Column to enable the creation of milestone objects from the query results.



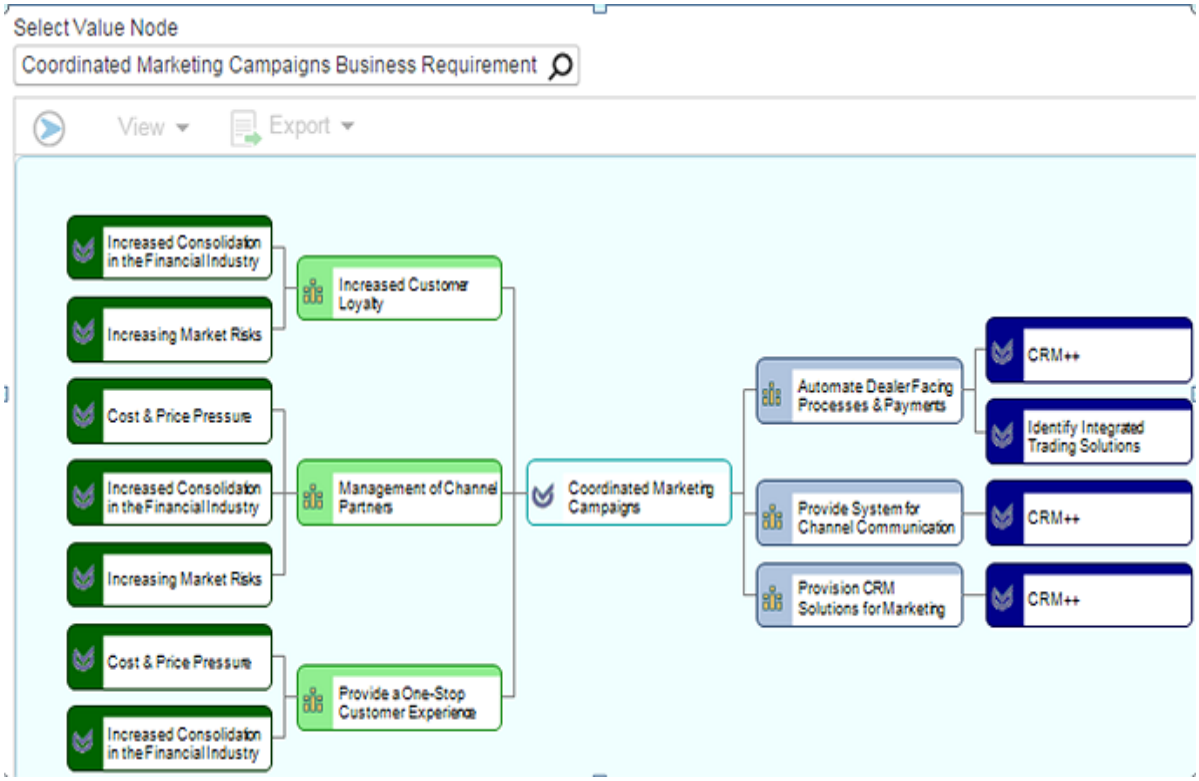
Defining a Grid Report

Grid reports are designed in the **Report Assistant**. For information about how to open the **Report Assistant**, see Creating a New Graphic Report. For more information about how to work with the **Report Assistant**, see Configuring the Report with the Report Assistant.

The content and layout of a grid report is defined by adding elements to the explorer of the **Report Assistant** and setting the attributes in the attribute pane of the **Report Assistant** window. The general design of the grid and the number of cells that contain graphics is defined via the attributes of the root node element.

A grid report is like a table that contains graphic reports in every table cell. For each cell, an element **Cell** defines the location of the cell in the table and the content of the cell. In the cells, objects are displayed as boxes and the relationship between the boxes is displayed as one of the following:

- A tree where related objects are connected with the branches of the tree. The tree can span in one or two directions. Whether the tree spans horizontally or vertically depends on the configuration of the report.



- A cluster map where boxes representing objects are stacked in each other hierarchically. If multiple objects are displayed in the box of a subordinate object, the boxes can be displayed on top of each other or next to each other.

For each type of cell content, the size, coloring and text attributes of the objects in a cell can be individually configured. A default for all cells can be defined in **Default Attribute Item** elements that can be defined as a sub-element of the ***Default Attribute Items*** node below the root node.

The design of the boxes representing objects in the configured report can be changed from the standard design with a white rectangle as background for text within the colored object box to fully colored object boxes. For more information see Defining the General Display of Object Boxes in Treemap, Diagram, Lane and Layered Diagram Reports.

The following table lists all elements that can be configured to specify the content of the report and provides information about the purpose of each sub-element:

XML Element	Purpose
Root Node	General layout of the report and number of rows and columns containing grid cells.
Cell	Definition of a cell in the report. The attributes of this element specify the general layout of the cell and its location in the grid table. The content is specified by the child elements **Item**. One element **Cell** must be defined as a child of the **Grid Report Def** element for each cell of the grid table. For example, if the **Grid Report Def** element defines that the grid table has two columns and three rows, the number of `Cell` elements in the report is 2 x 3 = 6.
Item	Definition of the objects to be displayed in the report. *For reports displaying a tree:* Definition of the content of one layer of the graphic tree. The tree is built starting with the `Item` element that is a child of the `Cell` element. This element can then contain one or two other `Item` elements as child elements and define the second layers. The third layer is defined by the child element of the second layer `Item` elements, and so on. *For reports displaying a cluster map:* Definition of one level of boxes in the hierarchy. The upper level boxes are defined as child element of the `Cell` element. This element can then contain one or multiple other `Item` elements as a child element to define the second layer. The third layer is defined by the child element(s) of the second layer, and so on.
Query	Definition of the content of either the current layer of the report or of all layers of the report in a grouped dataset.
Default Attribute Items	Container for the definition of **Default Attribute Item** elements.

XML Element	Purpose
Default Attribute Item	Definition of the default box layout for a single object class for all cells in the report.
Color Rule	Definition of the colors used in the current cell of the grid report and the rules that define which objects are displayed in which color. For more information about the definition of color rules, see Defining Color Rules.
Indicator Rule	Definition of one or multiple indicator or object related icons displayed in the boxes representing the objects in the current cell of the grid report. For more information about the definition of indicator rules, see .

Defining the Cells of the Grid Report

The basic design of the report and the number of cells in the report is configured directly in the attributes of the root node element of the report. For each cell of the report, the element **Cell** must be added as a child element to the root node.

A grid report can be regarded as an invisible table with each element **Cell** of the report configuring a cell in the table. The number of rows and columns in the report table is defined with the attributes **Rows** and **Columns** of the root node element. Each column and row is assigned a number. Columns are numbered from left to right, starting with zero and rows are numbered from top to bottom starting with zero. Therefore, a cell can be specified by the column and row number of the cell.

	Column 0	Column 1	Column 2	Column 3
Row 0	0,0	1,0	2,0	3,0
Row 1	0,1	1,1	2,1	3,1
Row 2	0,2	1,2	2,2	3,2
Row 3	0,3	1,3	2,3	3,3
Row 4	0,4	1,4	2,4	3,4
Row 5	0,5	1,5	2,5	3,5

FIGURE: Grid report table with naming of the table cells as "column number, row number"

The position of the cell within the report is defined by means of the attribute **Rectangle** of the **Cell** element. The attribute **Rectangle** specifies four numbers:

- The column number of the cell

- The row number of the cell
- The number of columns a cell spans
- The number of rows a cell spans

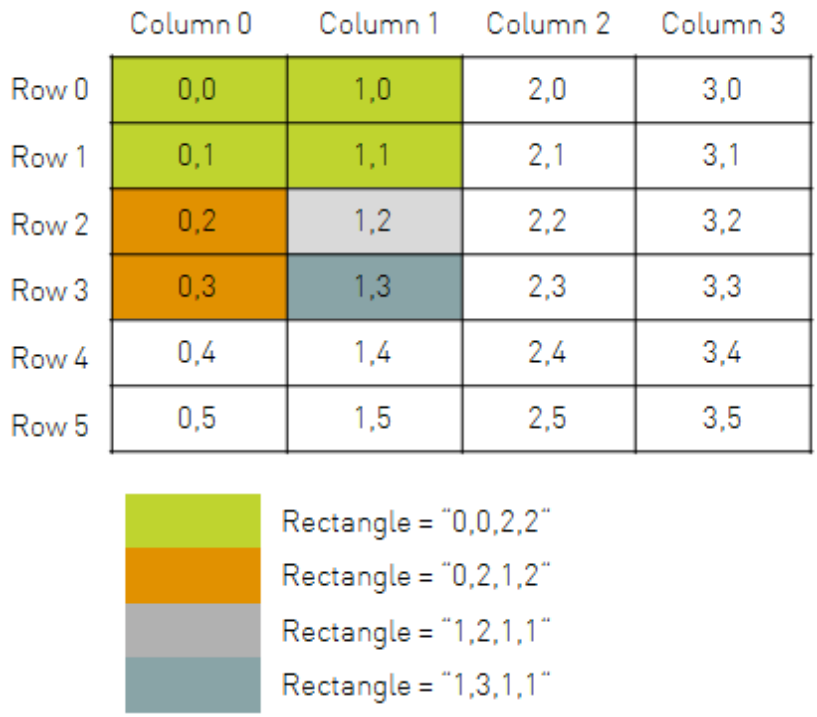


FIGURE: Grid report table with specification of the location of four different grid report cells

The following table lists all attributes of the root node element and the element **Cell** that are used to define the table layout.

The layout of object boxes is in general defined in the elements specifying the content of the objects in the report. Only a few attributes of the **Cell** element define the layout of items in the report. These attributes depend on the type of graphic display in the report and are described in separate sections in the table.

Attribute	Description
Root Node	
Rows	Defines the number of rows that the report will have.
Columns	Defines the number of columns that the report will have.
Row Height	Defines the height of the table rows in the grid report in mm (1 mm = 4 pixel). In the menu of the grid report, the option **View** > **Definition Size** allows the view mode to be defined:

Attribute	Description
	• If **Definition Size** is selected, the height and width of the table cells will be displayed as specified with the attributes **Row Height** and **Column Width**. If the content of the cell does not fit into the specified size, the handling of cell content will depend on the setting of the attribute **Auto Fit** of the **Cell** node. If **Auto Fit** is set to `true`, the content of the cells is drawn to a smaller scale to fit to the defined size. If **Auto Fit** is set to `False`, the content is not scaled and only part of the content is visible. • If **Definition Size** is deselected, the grid will be displayed in a full size view, whereby the cell sizes of the grid table will be computed based on the space required by the content of the cell.
Column Width	Defines the width of the table columns in the grid report in mm (1 mm = 4 pixel). In the menu of the grid report, the option **View** > **Definition Size** allows the view mode to be defined: • If **Definition Size** is selected, the height and width of the table cells will be displayed as specified with the attributes **Row Height** and **Column Width** . If the content of the cell does not fit into the specified size, the handling of cell content will depend on the setting of the attribute `Auto Fit` of the `Cell` node. If **Auto Fit** is set to `true`, the content of the cells is drawn to a smaller scale to fit to the defined size. If **Auto Fit** is set to `False`, the content is not scaled and only part of the content is visible. • If **Definition Size** is deselected, the grid will be displayed in a full size view, whereby the cell sizes of the grid table will be computed based on the space required by the content of the cell.
Cell Background Color	Defines the default background color of the grid cells in the report. This value can be overwritten by the **Background Color** attribute in the **Cell** element that defines a single cell.
Cell Border Color	Defines the default border color of the grid cells in the report. This value can be overwritten by the **Border Color** attribute in the **Cell** element that defines a single cell.
Cell Foreground Color	Defines the default cell text color of the grid cells in the report. This value can be overwritten by the **Foreground Color** attribute in the **Cell** element that defines a single cell.
Cell Caption Height	Defines the default height of the cell captions of the report in mm (= 4 pixel). This value can be overwritten by the **Caption Height** attribute in the **Cell** element that defines a single cell.
Show Legend	Defines whether a legend is shown for the color coding of the objects in the report.

Attribute	Description
Item Border Width	Defines the width of the border of the boxes in the report. the default value is 1 point. when using the border color as object property indicator via a color rule, it is recommended to set the border width to a higher value.
Cell	
Name	Defines the name of the **Cell** node in the explorer.
Layout Type	Defines the graphic design of the cell content. Currently, only three options can be selected: • `TreeHorizontal` displays a tree in a horizontal direction. • `TreeVertical` displays a tree in a vertical direction. • Cluster displays a cluster map. Whether boxes in the cluster are displayed a in horizontal or vertical direction is defined in the **Item Layout Type** attribute of the **Cell** element.
Rectangle	Defines the position and size of the cell. The first two numbers define the position of the cell in the table by specifying the number of the column from the left and the number of the row from top. Please note that the first row/column has the number "0". The next two numbers specify the width and height of the cell. **NOTE**: To define the attribute, click the rectangle in front of the attribute to expand it and define the values for the cell position and size in the respective sub-attributes.
Caption	Defines the caption of the cell in the grid report. **Note**: The caption is displayed in the color defined with the attribute **Foreground Color**. If **Foreground Color** is set to transparent, the caption will not be displayed.
Caption Height	Defines the height of the cell caption in mm (= 4 pixel)
Background Color	Defines the background color of the grid cell.
Foreground Color	Defines the text color of the grid cell.

Attribute	Description
Border Color	Defines the border color of the grid cell.
Auto Fit	Defines the handling of cell content if **View** > **Definition Size** is selected by the user in the toolbar of the report.: If **Definition Size** is selected, the height and width of the table cells will be displayed as specified in the attributes **Row Height** and **Column Width**. If the content of the cell does not fit into the specified size, the handling of cell content will depend on the setting of the attribute **Auto Fit** of the **Cell** node. If **Auto Fit** is set to true, the content of the cells will be drawn to a smaller scale to fit to the defined size. If **Auto Fit** is set to **False**, the content will not be not scaled and only part of the content is visible.
Item Spacing	Defines the spacing between items (object boxes) in the grid in mm (1 mm = 4 pixel). Click the rectangle in front of the attribute to expand it and define the horizontal spacing with the sub-attribute **Horizontal** and the vertical spacing with the sub-attribute **Vertical**. If you want to define identical values for horizontal and vertical spacing, you can enter the value in the sub-attribute **All** to set it simultaneously for all sub-attributes. **All** is defined as -1 if Horizontal and Vertical are different.
Item Padding	Defines the spacing between items (object boxes) and the border of the grid cell in mm (1 mm = 4 pixel). Click the rectangle in front of the attribute to expand it and define the spacing for the different sides of the cell with the sub-attributes **Left**, **Top**, **Right**, and **Bottom**. If you want to define equal values for all cell border spacings, you can enter the value in the sub-attribute **All** to set it simultaneously for all sub-attributes. **All** is defined as -1 if different values are defined for sub-attributes.
Hor. Align	Defines the horizontal alignment of the cell content.
Ver. Align	Defines the vertical alignment of the cell content.

Attributes for trees only:

Direction	Defines the direction of display of objects. Select Bidirectional to display object in two directions. Whether objects are displayed in horizontal (left and right) or vertical (top and bottom) directions is specified with the attribute **Layout Type**. Select Regular to display objects in one direction only, from left to right or from top to bottom, depending on the setting for **Layout Type**. Select Invers to display objects in one direction only, from right to left or from bottom to top, depending on the setting for **Layout Type**.

Attribute	Description
Tree Color	Defines the color of the tree displayed in the grid cell.
Attributes for clusters only:	
Item Layout Type	Defines the direction of object boxes in the first level of the report. Select Vertical to display objects next to each other from top to bottom or Horizontal to display object boxes next to each other from the left to the right.
Line Count	If many objects are found by a query that defines the objects in one layer of the reports, the defined space of the box can be too small to display all items next to each other. The **Line Count** element allows you to define how many objects are displayed next to each other (in horizontal or vertical direction, according to the **Layout Type** and **Item Layout Type** attributes of the **Cell** element). If the number of objects specified with the attribute **Line Count** is reached, the next boxes are displayed in a new row in reports with horizontal direction or in a new column in reports with vertical directions. **Note**: If the ***Line Count*** attribute is set to 1, the orientation specified in the **Item Layout Type** attribute is reversed.
Equal Width	Select `"true"` to display all boxes of the first layer of the report in equal width. For more information about the sizing of object boxes in cluster maps, see the section Defining the Size of Object Boxes in the Report.
Equal Height	Select `"true"` to display all boxes of the first layer of the report with equal height. For more information about the sizing of object boxes in cluster maps, see the section Defining the Size of Object Boxes in the Report.

Defining the Items Displayed in a Cell of a Grid Report

A cell in a grid report can either display object relationships in a tree structure or as nested boxes. For both types of reports, each level of the object boxes is defined with an Item element . An Item element triggers the display of objects of a single object class. The object class is defined with the attribute **Class Name** of the **Item** element.

The first level of the cell content is defined by an Item element that is a sub-node of the **Cell** node in the explorer. A **Cell** element can contain only one Item element to define the first level of the report. The first level **Item** element contains one or multiple child Item elements that define the next level of the tree or cluster map. The number of child elements to be defined for the root Item element depends on the type of report:

- For unidirectional trees:

Each **Item** element can have one child element defining the next level of the tree.

- For bidirectional trees:

The root **Item** element of a tree spanning in two directions can contain two Item elements, each defining a direction of the tree. All other **Item** elements of the report can have one child element defining the next level of the tree.

- For cluster maps:

Each **Item** element can have multiple child elements defining the next level of nested object boxes. This allows relationships to multiple objects to be displayed in one level of a report.

For cluster maps the attribute **Class Name** of the **Item** element can be left blank. In that case, all objects in the level are displayed in object boxes with the design defined in the **Item** element regardless of the object class the object belongs to.

The last level in the report is defined by an **Item** element without a child Item element.

The objects displayed in the levels of the report are found on via queries defined for the **Item** elements of the report. The queries can either be native SQL queries or Alfabet queries.

If you are not familiar with Alfabet queries, see the chapter Defining Queries.

For special rules that apply to the definition of native SQL queries in the context of configurations for Alfabet, see the section Defining Native SQL Queries.

The information displayed in the object boxes of the report is identical to the column headers that would result from a tabular output of the defined query.

Report columns that are defined for technical reasons are not displayed in the object boxes in the report. This includes the first column of the output of a native SQL query. This column must specify the `REFSTR` of the object for technical reasons and is ignored in the visible output.

There are two methods to define the content of the grid report:

- The complete content of the report is defined in one query resulting in a grouped dataset.
- A separate query is added to the report configuration to find the objects in a level of the report.

When separate queries are defined, a high number of queries must be executed by the database to display the report. This can lead to performance issues. Therefore, it is recommended to base the report on a grouped dataset defined in a single query.

Defining a Grid Report With a Single Query

The content of all level of the report is defined via an Alfabet query or a native SQL query in a Query element node. The **Query** node is a sub-node of the root **Item** node in a **Cell** element. The query finds the objects that are displayed in the report and determines the information that is displayed about the objects in the object boxes of the report. The query must return a grouped dataset. Each grouping level corresponds to a level in the report.

All other **Item** elements in the cell may not contain a sub-element **Query**.

The query must find objects of the object classes defined with the attribute **Class Name** of the **Item** elements of the corresponding level. Otherwise objects found by the query are not displayed in the report. For **Item** elements in cluster maps that does not have a **Class Name** defined, the report displays all objects found by the query without restrictions and a query can return objects from multiple classes.

Defining a Grid Report With One Query Per Item

The content displayed for each **Item** element in the report is defined via an Alfabet query or a native SQL query in a **Query** element node. The **Query** node is a sub-node of the respective **Item** element. The query must find objects of the object class defined with the attribute **Class Name** of the **Item** element. Otherwise objects found by the query are not displayed in the report. For **Item** elements in cluster maps that does not have a **Class Name** defined, the report displays all objects found by the query without restrictions and a query can return objects from multiple classes.

The object class defined for a level of the report is the same as the base class for the objects displayed in the next level. The Alfabet query or native SQL query defining the object class displayed in the next level must contain the parameter `BASE` in a `WHERE` clause to map the objects to the objects of the current layer. The parameter `BASE` is identical to the `REFSTR` of the objects in the current layer.

This specification is not mandatory for the top layer objects. The attribute `BASE` can only be used in the Alfabet query or native SQL query defining the top layer objects if the report is assigned to an object class with the attribute **Apply To Class**.

The use of the parameter `BASE` depends on the setting of the Cascading attribute in the Query element:

- If **Cascading** is set to `True`, a query in a layer cannot only refer to an object of the directly superordinate layer, but can also refer to objects of all other superordinate layers. The parameters to refer to objects of the superordinate layers are then specified as `BASE0` to refer to the directly superordinate layer, `BASE1` to refer to the layer above the directly superordinate layer and so on.
- If `Cascading` is set to False, a query in a layer can only refer to an object of the directly superordinate layer using the parameter `BASE`.

The layout of the object boxes in the level defined by an **Item** element are configured via attributes of the following elements:

- The attributes of the **Item** element for the current level define the layout of each single object box.
- The attributes of the **Item** element of the superordinate level define how boxes are displayed in the level. This includes, For example, the spacing between boxes. The `Cell` element contains the specification for the level layout display of the first level of the report.
- The attributes of the **Default Attribute Item** elements in the **Default Attribute Items** sub-node of the explorer allow a default layout of object boxes to be defined for a specified class. This default layout will be implemented if no layout specifications are specified in an **Item** element in any of the cells of the report that trigger the display of objects of the specified object class. The **Default Attribute Item** element allows you to define a consistent layout for multiple trees or cluster maps displayed in a grid report.

  The specifications in a **Default Attribute Item** element for an object class are overwritten with the specifications in an **Item** element for the object class with the following exceptions:

- When a tree is displayed in the cell, the **Item Size** setting in the **Default Attribute Item** element supersedes the **Item Size** specification in the root **Item** element of the cell.
- Values for **Text Attributes** in the **Default Attributes Item** element are ignored and must always be defined the **Item** element.

The table below lists all available attributes in the **Item** element and the allowed child Query element as well as the attributes of the **Default Attributes Item** element.

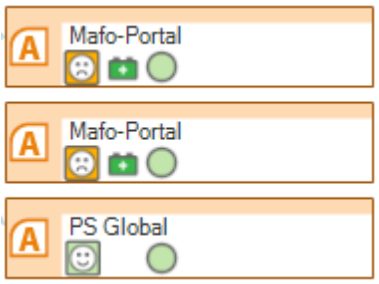
The attributes of the `Item` element defining the layout of the current level can be configured in the **Common** section of the property grid for the **Item** element. The attributes defining the subordinate level are listed in the **Layout section**. For the definition of the first level layout, see the description of attributes of the **Cell** element in Defining the Cells of the Grid Report.

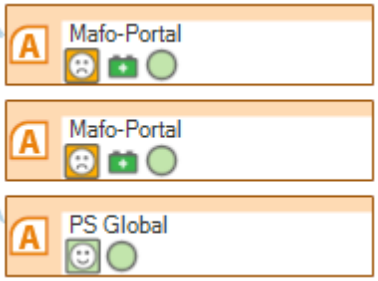
Elements (Bold) and Attributes	Description
Item	
Section Common (Attributes configuring the current level)	
Name	Defines the name of the **Item** node in the explorer of the **Report Assistant**.

Class Name	Defines the object class specified in this **Item** element.
	
	NOTE: For object classes allowing stereotype specification (For example, Project and Domain), the **Class Name** can specify an object of a specific stereotype. To define an **Item** element for a stereotype, the **Class Name** must be written with the following syntax:
	
	`ClassName:StereotypeName`
	
	For example,:
	
	`Project:StatementOfWork`

Background Color	Defines the color of the boxes representing the objects in the report. If no value is specified, the value specified in the **Default Attributes Item** element for the object class defined with the **Class Name** attribute will be used. For more information about color selection, see Defining Color Rules.
Foreground Color	Defines the color of the text in the boxes representing the objects in the report. If no value is specified, the value specified in the ***Default Attributes Item*** element for the object class defined with the **Class Name** attribute will be used.

Elements (Bold) and Attributes	Description
	For more information about color selection, see Defining Color Rules.
Border Color	Defines the border color of the boxes representing the objects in the report. If no value is specified, the value specified in the **Default Attributes Item** element for the object class defined with the **Class Name** attribute will be used. For more information about color selection, see Defining Color Rules.
Icon	Defines the icon displayed for the object class in the report. The icon must be an icon that has already been added to Alfabet Expand. For more information, see the section Adding and Maintaining Icons for the Alfabet Interface. If no value is specified, the value specified in the **Default Attributes Item** element for the object class defined with the **Class Name** attribute is used. If this value is also undefined, the icon defined in the class settings for the object class will be used. **NOTE**: If object class stereotypes are defined for an object class and an icon is not defined in the **Item** element or a **Default Attributes Item** element for the object class, the icon defined for the object class stereotype of the object displayed in a box is used per default. Therefore, different icons can be used for objects in the same level. **Default Attributes Item** elements can also be defined per object class stereotype to define a stereotype-based default icon for the report.
Rounded	Defines whether the edges of the object boxes are rounded.
Show Class Icon	Defines whether an icon is displayed for the object class in the report. If not otherwise specified via the Picture attribute, the icon implemented is that defined in the **Icon** attribute of the relevant class setting.
Show Indicator	Defines whether icons can be displayed in the boxes representing the objects in the current level of the report. The icons can either represent an indicator or any other aspect of the object data, like For example, objects associated with the current object. Icons are displayed for objects defined in the **Indicator Rule** elements in the report, independent of the level of the tree they are assigned to. The attribute **Show Indicators** allows the display of indicators to be limited to only the specified levels. For information about defining indicator rules, see the section Defining Indicator Rules.
Show Multiple Indicators	Defines whether display of icons defined via indicator rules is limited to one icon per object box or whether multiple icons can be assigned to an object box. If **Show Multiple Indicators** is set to `False`, a single icon can be displayed on the right of the object box. If multiple icons are found by indicator rules, the first icon found is assigned and all other icons are ignored.

Elements (Bold) and Attributes	Description
	If **Show Multiple Indicators** is set to `True`, one or multiple icons are displayed from left to right at the bottom of the object box. If the number of icons that are assigned to the object box via indicator rules exceeds the space available for display of icons in a single row, the first icons found are displayed and, if the object box space is exceeded, any other icons are ignored. The way icons are placed in the object box also depends on the attribute **Indicator Fill Policy** of the **Item** node. For more information about the placement of icons assigned via indicator rules, see the section Defining Indicator Rules.
Indicator Fill Policy	If **Show Multiple Indicators** is set to `True`, this attribute defines how icons assigned to an object box are placed in the row on the bottom of the object box: • `ByIndex`: This setting shall be used if multiple indicators are defined via multiple indicator rules, each returning a single icon. The position of the icon resulting from each indicator rule is defined by means of the attribute **Indicator Index** of the respective indicator rule that must be set to an unsigned integer starting with 0 for the left most position. If an **Indicator Index** attribute has been defined for each indicator rule and an indicator is not set for a specific object, the position of this indicator icon in the box will be left empty and the position of all other indicator icons will not be changed:  If the order of the **Indicator Index** attribute is not defined, the placement of indicators will be arbitrary. This method is designed for positioning only one icon per indicator rule. If an indicator rule returns multiple icons, only one of the icons is displayed. • `LeftAlign`: This setting shall be used if a single indicator rule is defined that returns multiple icons. It can also be applied when multiple indicator rules each return a single icon, but the position and order of icons is not important to understand the report. The Icons are displayed in the order of occurence within the indicator rule(s) from left to right. If a different number of indicator icons is returned, the remaining icons are moved to the left:

Elements (Bold) and Attributes	Description
	 If the indicator icons are all found by a single indicator rule, the order of icons depends on the order of results in the result dataset of the query of the indicator rule and may be different for different objects if no sort order is defined in the query.
Item Size	Click the rectangle in front of the **Item Size** attribute to expand it and define the width and height of the items (object boxes) of the current level in mm (1 mm = 4 pixel) by setting the sub-attributes **Width** and **Height**. If you define -1, the sub-attribute is undefined. For more information about the specification of object box sizes, see Defining the Size of Object Boxes in the Report.
Text Attributes	Click the rectangle in front if the **Text Attributes** attribute to define how text is displayed in the object boxes of the current level. If no specification is made, the **Text Attributes** defined in the **Default Attributes Item** element for the object class defined with **Class Name** will be used. **NOTE**: To allow line breaks in the text of the object box, set the sub-attribute **Word Wrap** to `true`.
Format	Allows static text to be added to the information displayed about an object in the object box. A string can be defined that contains place holders for the information added from the query results for the current object. The place holder is the column name for the property that would result from a tabular output of the report written in curly brackets. For example, the name and the version number of applications is to be displayed with the string " Version " between the name and version number. The report is based on an Alfabet query with `Application.Name` and `Application.Version` as Show properties. The string defined for the attribute Format resulting in the required output is: `{Application.Name} Version {Application.Version}` **NOTE**: The name and not the caption of the column of the tabular output of the query must be defined in the curly brackets. When the query is an Alfabet query, the correct syntax is ClassName.PropertyName or, if an alias is defined for the class, ClassAlias.PropertyName. A defined show name for the column in the report is ignored. When you define a native SQL query, setting an alias in the `SELECT` clause does not only change the caption but also the name of the database column.

Elements (Bold) and Attributes	Description
Attributes configuring the next level for trees:	
Direction	Defines the direction in that the tree starting with the item spans. This attribute is only allowed in the level under the root level if the attribute Direction of the **Cell** element is set to Bidirectional. Select Regular to display objects from left to right or from top to bottom, depending on the setting of the attribute **LayoutType** of the **Cell** element. Select Invers to display objects from right to left or from bottom to top, depending on the setting of the attribute **LayoutType** of the **Cell** element.
Attributes configuring the next level for cluster maps	
Hor. Align	Defines the horizontal alignment of the object boxes of the next level in object boxes of this level.
Ver. Align	Defines the vertical alignment of the object boxes of the next level in object boxes of this level.
Item Layout Type	Defines the direction of object boxes in the next level of the report. Select Vertical to display objects next to each other from top to bottom or Horizontal to display object boxes next to each other from the left to the right.
Line Count	If many objects are found by a query that defines the objects in one level of the report, the defined space of the box can be too small to display all items next to each other. The **Line Count** element allows you to define how many objects are displayed next to each other (in horizontal or vertical direction, according to the **Layout Type** and **Item Layout Type** attributes of the **Cell** element). If the number of objects specified with the attribute LineCount is reached, the next boxes will be displayed in a new row in reports with horizontal direction or in a new column in reports with vertical directions. **Note**: If the **Line Count** element is set to 1, the orientation specified in the **Item Layout Type** attribute will be reversed.
Equal Width	Select `true` to display all boxes of the next level of the report in equal width. **Note:** This parameter is ignored for the last level of boxes in the report. The last level contains no boxes inside and Therefore, all boxes are displayed with equal width anyway.
Equal Height	Select `true` to display all boxes of the next level of the report with equal height.

Elements (Bold) and Attributes	Description
	Note: This parameter is ignored for the last level of boxes in the report. The last level contains no boxes inside and Therefore, all boxes are displayed with equal height anyway.
Item Spacing	Defines the spacing between object boxes in the next level in mm (1 mm = 4 pixel).
Item Padding	Defines the spacing between object boxes of the next level and the border of the superordinate object box of this level in mm (1 mm = 4 pixel).
Query	
Name	Defines the name of the **Query** node in the explorer of the **Report Assistant**.
Alfabet Query/ Query as Text/ Native Sql	Defines a query to find the objects that are displayed in the report. How the query must be defined depends on the location of the **Query** element in the report.
Cascading	**For reports based on one Query element per Item element only**: If the **Cascading** attribute is set to `True`, a query in a layer cannot only refer to an object of the directly superordinate layer, but can also refer to objects of all other superordinate layers. The parameters to refer to objects of the superordinate layers are then specified as `BASE0` to refer to the directly superordinate layer, `BASE1` to refer to the layer above the directly superordinate layer and so on. **Note**: It is not possible use the the parameter `BASE`<number> to refer to the object class that the report is assigned to in the property **Apply To Class** of the report. Only the query of the first layer in the report can refer to the base object that the report is applied to via the parameter `BASE`. In this case, the **Cascading** attribute must be set to "false" for that layer. If the **Cascading** attribute is set to False, a query in a layer can only refer to an object of the directly superordinate layer using the parameter `BASE`.
Default Attributes Item	
Class Name	Defines the object class specified in this **Default Attributes Item** element.: **NOTE**: For object classes allowing stereotype specification (For example, Project and Domain), the **Class Name** can specify an object of a specific stereotype. To define an **Item** element for a stereotype, the **Class Name** must be written with the following syntax: `ClassName:StereotypeName`

Elements (Bold) and Attributes	Description
	For example,: `Project:StatementOfWork`
Background Color	Defines the color of the boxes representing the objects in the report. If no value is specified, the value specified in the class settings of the object class defined in the **Class Name** attribute will be used. For more information about the default coloring for object classes, see Configuring Class Settings for Object Classes and Object Class Stereotypes. For more information about color selection, see Defining Color Rules.
Foreground Color	Defines the color of the text in the boxes representing the objects in the report. If no value is specified, the value specified in the class settings of the object class defined in the `Class Name` attribute will be used. For more information about the default coloring for object classes, see Configuring Class Settings for Object Classes and Object Class Stereotypes. For more information about color selection, see Defining Color Rules.
Border Color	Defines the border color of the boxes representing the objects in the report. If no value is specified, the value specified in the class settings of the object class defined defined in the **Class Name** attribute will be used. For more information about the default coloring for object classes, see Configuring Class Settings for Object Classes and Object Class Stereotypes. For more information about color selection, see Defining Color Rules.
Icon	Defines the icon displayed for the object class in the report. The icon must be an icon that has already been added to Alfabet Expand. For more information, see the section section Adding and Maintaining Icons for the Alfabet Interface. If no value is specified, the value specified in the class settings of the object class defined in the **Class Name** attribute will be used. For more information, see Configuring Class Settings for Object Classes and Object Class Stereotypes.
Item Size	Click the rectangle in front of the **Item Size** attribute to expand it and define the default width and height of the items (object boxes) of the object class in mm (1 mm = 4 pixel) by setting the sub-attributes **Width** and **Height**. If you define -1, the sub-attribute is undefined. For more information about the specification of object box sizes, see Defining the Size of Object Boxes in the Report.
Text Attributes	Click the rectangle in front if the **Text Attributes** attribute to define how text is displayed per default in the object boxes for the object class.

Elements (Bold) and Attributes	Description
	NOTE: To allow line breaks in the text of the object box, set the sub-attribute **Word Wrap** to `true`.

Defining the Size of Object Boxes in the Report

The size of the object boxes in a grid report is calculated differently for reports displaying trees or cluster maps.

For Trees:

All object boxes of all levels of the tree have the same size. The size can be specified with the sub-attributes **Width** and **Height** of the **ItemSize** attribute in the following elements:

- the **Default Attributes Item** element
- the root **Item** element that is child of the **Cell** element.

If the size is defined both in the **Default Attributes Item** element and in the root **Item** element the greater value is selected. Selection is evaluated per sub-attribute, which means that the resulting size can have the height specified in one element and the with specified in the other element.

If none of the elements includes a size specification, object boxes are displayed with the default value of 40 x 15 mm (1 mm = 4 pixel).

The object box size can be defined via the sub-attributes **Width** and **Height** of the **Item Size** attribute. If -1 is specified for a sub-attribute, the sub-attribute is undefined. If the size specification in the **Default Attributes Item** element contains either only a specification of **Width** or only a specification of **Height**, the missing width or height specification is evaluated from the size definition of the root **Item** element of each cell..

For Clusters:

The size of the object boxes is defined individually for each level in the report. The size of object boxes can even differ within a level of the report. The size depends on both the content of the object box and the size specification in the report configuration.

The size can be specified with the attribute **Item Size** or the attributes **Width** and **Height** in the following elements:

- the **Default Attributes Item** element
- the **Item** elements.

The size specification in the **Item** element overwrites the size specifications in the **Default Attribute Item** element.

The size specification in the elements have different effects on the display of the object boxes in the levels:

- The size specification in the **Item** element is an absolute size. If the **Item** element of a level has an **Item Size** attribute with both sub-attributes **Width** and **Height** defined, the object boxes have the defined size regardless of the size of the content. If the object boxes of the subordinate level that are displayed within a box of this level do not fit into the defined size, the content is overlaying the heading of the upper level box an partially displayed outside the boxes:

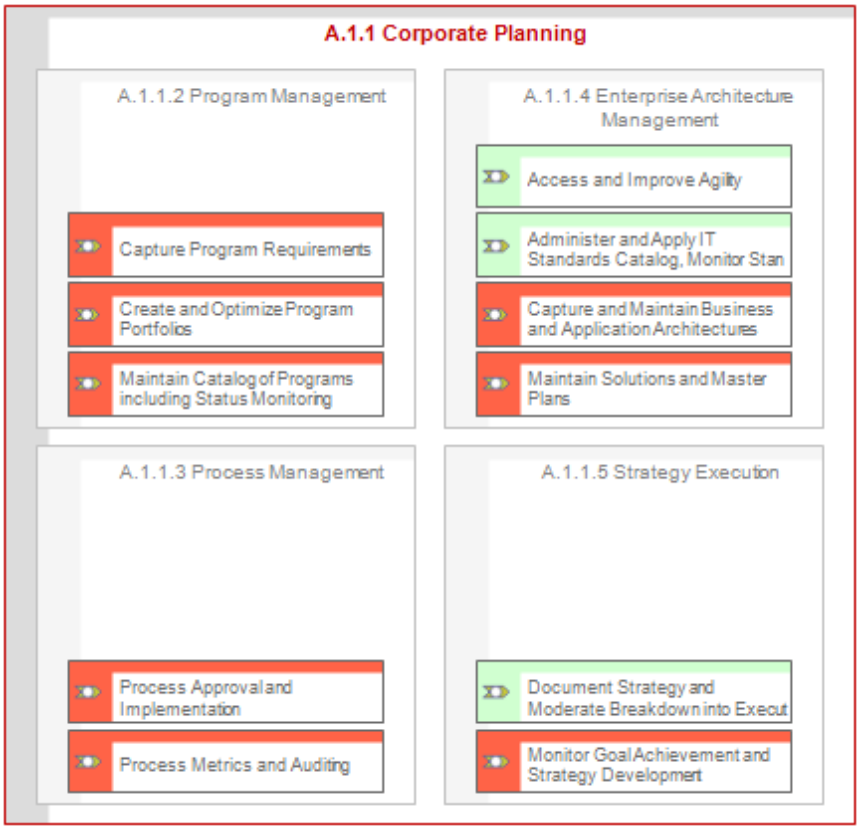
- Size specifications in the **Default Attributes Item** element are relative values that define a minimum size for elements without subordinate boxes for all levels in the report except the last one. Boxes of the last level of the report are displayed in the default size of 40,15 regardless of a size specification in the **Default Attributes Item** element if no size is specified in the **Item** element for the last level. Boxes of all other levels are displayed with the size defined in the **Default Attributes Item** element if the box content is smaller than the defined size. If the box content is bigger than the defined size, the box size is adapted to the content's size.

If none of the elements includes a size specification, object boxes that do not contain object boxes of a subordinate level are displayed with the default value of 40 x 15 mm (1 mm = 4 pixel) and all other box sizes are adapted dynamically to the content of the box.

If an element **Item** contains only a **Width** or only a **Height** specification, the size of the object boxes will adapt to the size of the content in the boxes, and the defined **Width** or **Height** will be used to overwrite the **Width** or **Height** specifications for the default size of boxes for the level.

If you want the boxes of a level to have the same size, but nevertheless the box size shall adapt to the size of the content of the box, you can set the attributes **Equal Width** and **Equal Height** in the **Item** element specification of the superordinate level or in the **Cell** element as superordinate level of the root `Item` element. These attributes are only valid if the **Item** element of the current level includes no size specification.

Equal Height is set in the root **Item** element. The **Item** element of the second level contains no size specification. Therefore, the second level elements are all displayed with the height of the box containing third-level sub-elements.



Defining Lane Reports

Lane reports are designed in the **Report Assistant**. For information about how to open the **Report Assistant**, see Creating a New Graphic Report. For more information about how to work with the **Report Assistant**, see Configuring the Report with the Report Assistant.

The content and layout of a lane report is defined by adding elements to the explorer of the **Report Assistant** and setting the attributes in the attribute pane of the **Report Assistant** window.

The content and layout of a lane report is defined in the root node element.

The following table lists the explorer node elements that can be configured to specify the content of the report as well as provides information about the purpose of each sub-element:

Explorer Node Element	Required to Configure:
Root Node of the report	Definition of the layout of the report.
Lane	Definition of a new lane added to the report. A lane consists of a row of objects and optionally a definition of a link to the next lane in the report. The next lane is the lane right of the current lane.
Node	Definition of the layout of the object row of the lane. The Node element is a child of the Lane element.
Link	Definition of the layout of the connections between this object row and the object row right of this object row. The Link element is a child of the Lane element.
Query	If defined as child of the Node element, the Query element defines the objects displayed in the object row of the lane. If defined as child of the Link element, the Query element defines the kind of connections displayed in the diagram between the object rows.
Color Rule	Definition of the colors used in the current cell of the grid report and the rules that define which objects are displayed in which color. For more information about the definition of color rules, see Defining Color Rules.
Indicator Rule	Definition of one or multiple icons displayed in the boxes representing the objects in the current cell of the lane report. The icons can either represent an indicator or any aspect defined via a query. For more information, see the section Defining Indicator Rules.

Defining the Content of a Lane Report

To define a lane report in the **Report Assistant**:

1) In the explorer, right-click the root node of the new report and select **Add New Lane**. A new **Lane** element is added to the report that has a **Node** and a **Link** child element.
2) Click the new **Lane** element and define the following attributes:

- **Name**: Define the name of the **Lane** element displayed in the explorer of the **Report Assistant**.
- **Caption**: Define the caption of the lane displayed above the object row of the lane in the report.
- **Has Link Lane**: Select `True` if the current lane is followed by another lane definition. Select `False` if the current lane is the last lane in the definition and Therefore, no links to objects in a following lane are required.

3) Expand the new **Lane** element, right-click the **Node** element and select **Add New Query**. A **Query** element is added to the **Node** element.
4) Click the **Node** element and define the layout of the object boxes in the lane. The attributes are described in the section Defining the Content of a Lane Report.
5) Click the **Query** element and define the following attributes:
   - **Name**: Define the name of the **Query** element displayed in the explorer of the **Report Assistant**.
   - **Alfabet Query/Native Sql/Query as Text**: Define a query to find the objects that are displayed in the object box column of the lane either in Alfabet query language or native Sql. All information that is given about the object in the dataset resulting from the query is displayed in the object box for the object in the lane.
6) Click the **Link** element and define the layout of the links in the lane. The attributes are described in the section Defining the Basic Layout of a Lane Report.
7) Right-click the **Link** element and select **Add New Query**. A **Query** element is added to the **Link** element.
8) Click the **Query** element and define the following attributes:
   - **Name**: Define the name of the **Query** element displayed in the explorer of the **Report Assistant**.
   - **Alfabet Query/Native Sql/Query as Text**: Define a query to find the objects that are displayed in the report either in Alfabet query language or native SQL. The query must result in a dataset with the following columns:
     - The REFSTR of the source object for the link. The object must be available in the object box column of the current lane to display the link.
     - The REFSTR of the target object for the link. The object must be available in the object box column of the next lane to display the link.

       By default a link is represented by an arrow from the current object box column to the object box column in the next lane. Therefore, the source object is the object in the current lane and the target object is the object in the next lane. you can change the direction of the link by setting the attribute **Reverse** of the **Link** element. When the direction is reversed, the source object is the object from the next lane and the target object is the object in the current lane.

     - If you want a label to be displayed in the link, at least one additional dataset column must be defined that returns the information to be displayed about the link in the label. The information of all datset columns that are not defined for technical reasons is displayed in the link label.

9) Click the **Link** element and define the following attributes:

- **From Column**: Enter the name of the database column of the dataset resulting from the query defined in the **Query** element that returns the REFSTR of the source object for the link. If the attribute **Reverse** is set to `False`, the source objects must be available in the object box column of the current lane. If the attribute **Reverse** is set to `True`, the source objects must be available in the object box column of the next lane.
- **To Column**: Enter the name of the database column of the dataset resulting from the query defined in the **Query** element that returns the REFSTR of the target object for the link. If the attribute **Reverse** is set to `False`, the target objects must be available in the object box column of the next lane. If the attribute **Reverse** is set to `True`, the target objects must be available in the object box column of the current lane.
- **Reverse**: Select `True` if you want the links to point from objects in the next lane to the objects in this lane. Select `False` if you want the links to point from obejcts in this lane to the objects in the next lane.

10) Repeat step 1 - 9 for each subsequent lane. For the last lane, repeat step 1 - 5 only.

Lanes are processed in the order given in the explorer. After having defined the lanes, you can reorder the lane definitions in the explorer, if applicable:

1) Click the root node of the report.
2) Click the **Browse** button in the attribute **Lanes** of the root node. A new window opens that lists all lane definitions of the report.
3) Click a lane definition in the list and use the **Up** and **Down** buttons on the upper right of the window to move the lane definition in the list.
4) Click **OK** to save your changes. The order of lane definitions in the explorer changes.
5) Click the **Lane** nodes in the explorer and check whether the setting of the **Has Link Lane** attributes is correct.
6) Adapt the link query definitions to the changes in the order of lanes. Links can only connect objects from one lane to the next lane in the definition.

Defining the Basic Layout of a Lane Report

The basic design of the lane report is configured via the attributes of the root node of the report.

For each lane, the design of the object boxes and links can be defined with the attributes of the Node and Link element of the lane definition.

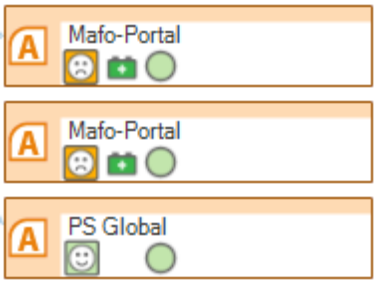
The table lists all attributes that are available for configuring the layout of the report in the respective elements:

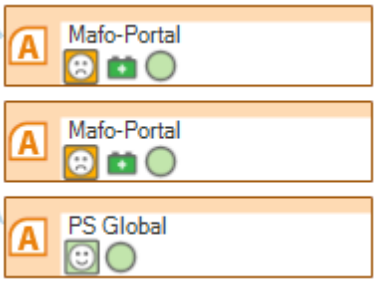
Attribute	Description
Root Node Element of	

Attribute	Description
the Report	
Show Legend	If set to `True`, a legend is shown for the color coding of the objects in the report and the indicator rules defined for the report.
Show Caption	If set to `True`, captions defined with the attribute **Caption** of the **Lane** elements of the report are displayed in the report.
Show Sub-Caption	If set to `True`, captions defined with the attribute **Caption** of the **Node** and **Link** elements are displayed in the report.
Lane Border Color	Defines the color of the borders displayed around object box rows and link rows in the report. By default, the border color is transparent and no border will be displayed.
Item Border Width	Defines the width of the border of the boxes in the report. The default value is 1 point. When using the border color as object property indicator via a color rule, it is recommended to set the border width to a higher value.
Caption Text Attributes	Click the rectangle in front if the **Text Attributes** attribute to define how the captions defined with the attribute **Caption** of the **Lane** elements of the report are displayed. **NOTE**: To allow line breaks in the text of the object box, set the sub-attribute **Word Wrap** to `true`.
Sub-Caption Text Attributes	Click the rectangle in front if the **Text Attributes** attribute to define how the captions defined with the attribute **Caption** of the **Node** and **Link** elements are displayed in the report. **NOTE**: To allow line breaks in the text of the object box, set the sub-attribute **Word Wrap** to `true`.
Node Element	
Caption	Defines a caption that is displayed as sub-caption beneath the Caption defined with the attribute **Caption** of the **Lane** element in top of the object box column defined by the **Node** element.
Width	Defines the width of the object box column.

Attribute	Description
Background Color	Defines the background color of the object box column.
Item Background Color	Defines the background color of the boxes representing the objects in the object box column. If the attribute is not defined, the value specified in the class settings of the object class represented by the object box will be used. For more information about the default coloring for object classes, see Configuring Class Settings for Object Classes and Object Class Stereotypes. For more information about color selection, see Defining Color Rules.
Item Foreground Color	Defines the text color of the boxes representing the objects in the object box column. If no value is specified, the value specified in the class settings of the object class represented by the object box will be used. For more information about the default coloring for object classes, see Configuring Class Settings for Object Classes and Object Class Stereotypes. For more information about color selection, see Defining Color Rules.
Item Border Color	Defines the border color of the boxes representing the objects in the object box column. If no value is specified, the value specified in the class settings of the object class represented by the object box will be used. For more information about the default coloring for object classes, see Configuring Class Settings for Object Classes and Object Class Stereotypes. For more information about color selection, see Defining Color Rules.
Icon	An icon from the icon gallery can be displayed in the object boxes of the lane. Select an icon from the drop-down list. If no icon is selected, the boxes are displayed without an icon. **NOTE**: Alternatively, the attribute **Show Class Icon** can be set to `True` to display the icon defined in the class settings of the object class in the object boxes. If an icon is selected with the attribute **Icon**, the setting of the attribute **Show Class Icon** is ignored.
Format	Defines a string to be displayed in the object boxes. By default, the object boxes display all information defined in the `Show` properties of the Alfabet query or `SELECT` statement of the native Sql query that is defined in the **Query** child element of the **Node** element. Alternatively, a static text can be configured that includes variables for data derived from the query. To display current query results in the string, add a variable in the format {dataset column name}. **NOTE**: The name and caption of a dataset column are not necessarily identical. Which value must be taken over depends on the way an Alfabet query or a native SQL query are

Attribute	Description
	defined. For more information about how to identify the correct technical name of a dataset column, see Defining Column Names and Captions in the chapter Defining Queries.
Text Attributes	Click the rectangle in front if the **Text Attributes** attribute to define how text is displayed per default in the object boxes for the object class. **NOTE**: To allow line breaks in the text of the object box, set the sub-attribute **Word Wrap** to `true`.
Alignment	This attribute alignes the object boxes in a column if other columns in the report are larger and this column is not completely filled with object boxes. Select `Top` to position the object boxes on top of the column. Select `Center` to position the object boxes at the center of the column. Select `Bottom` to position the object boxes on the bottom of the column.
Item Spacing	Defines the spacing between the object boxes in the object box column.
Item Padding	Defines the spacing between the column borders and the boxes in the column in number of pixels. Click the rectangle in front if the **Item Padding** attribute and define either one value for all spacings with the attribute **All** or single spacing values for the left, right, top and bottom border.
Item Height	Defines the height of the object boxes in the object box column in pixels.
Show Class Icon	Select `True` to display the class icon in the object boxes. the class icon is defined in the class settings of the object class. The **Show Class Icon** setting is ignored if an icon is defined with the attribute **Icon**.
Show Indicator	Select `True` to display an indicator icon in the upper right corner of the object box or multiple indicator icons at the bottom of the object box in a single row. Indicator icons are one displayed in the object box if an indicator rule is defined that specifies the dipslay of an indicator for the object. Indicator icons can represent an indicator value from the object evaluation or any other aspect that can be defined for the object via a query. Select `False` to exclude the display of the indicator icons for the object box column.
Show Multiple Indicators	Defines whether display of icons defined via indicator rules is limited to one icon per object box or whether multiple icons can be assigned to an object box. If **Show Multiple Indicators** is set to `False`, a single icon can be displayed on the right of the object box. If multiple icons are found by indicator rules, the first icon found is assigned and all other icons are ignored. If **Show Multiple Indicators** is set to `True`, one or multiple icons are displayed from left to right at the bottom of the object box. If the number of icons that are assigned to the object box via indicator rules exceeds the space available for display of icons in a single row,

Attribute	Description
	the first icons found are displayed and, if the object box space is exceeded, any other icons are ignored. The way icons are placed in the object box also depends on the attribute **Indicator Fill Policy** of the **Item** node. For more information about the placement of icons assigned via indicator rules, see the section Defining Indicator Rules.
Indicator Fill Policy	If **Show Multiple Indicators** is set to `True`, this attribute defines how icons assigned to an object box are placed in the row on the bottom of the object box: • `ByIndex`: This setting shall be used if multiple indicators are defined via multiple indicator rules, each returning a single icon. The position of the icon resulting from each indicator rule is defined by means of the attribute **Indicator Index** of the respective indicator rule that must be set to an unsigned integer starting with 0 for the left most position. If an **Indicator Index** attribute has been defined for each indicator rule and an indicator is not set for a specific object, the position of this indicator icon in the box will be left empty and the position of all other indicator icons will not be changed:  If the order of the **Indicator Index** attribute is not defined, the placement of indicators will be arbitrary. This method is designed for positioning only one icon per indicator rule. If an indicator rule returns multiple icons, only one of the icons is displayed. • `LeftAlign`: This setting shall be used if a single indicator rule is defined that returns multiple icons. It can also be applied when multiple indicator rules each return a single icon, but the position and order of icons is not important to understand the report. The Icons are displayed in the order of occurence within the indicator rule(s) from left to right. If a different number of indicator icons is returned, the remaining icons are moved to the left:

Attribute	Description
	 If the indicator icons are all found by a single indicator rule, the order of icons depends on the order of results in the result dataset of the query of the indicator rule and may be different for different objects if no sort order is defined in the query.
Link Element	
Caption	Defines a caption that is displayed as sub-caption beneath the Caption defined with the attribute **Caption** of the **Lane** element on top of the link column defined by the **Link** element.
Width	Defines the width of the link column.
Background Color	Defines the background color of the link column.
Item Background Color	Defines the color of the links in the report. If no value is specified, and the link is represented by an object that has class settings defined, the value specified in the class settings will be used. For example, a link can represent an information flow between two applications. For more information about the default coloring for object classes, see Configuring Class Settings for Object Classes and Object Class Stereotypes. If no value is specified, and the link is represented by an object that has no class settings defined, the link is invisible. If the link represents an object class, color rules can be defined to change coloring according to a given property value of the current object. For more information about color selection, see Defining Color Rules.
Item Foreground Color	Defines the text color of the boxes representing the objects in the object box column. If no value is specified, the value specified in the class settings of the object class represented by the link will be used. For more information about the default coloring for object classes, see Configuring Class Settings for Object Classes and Object Class Stereotypes.

Attribute	Description
	If the link represents an object class, color rules can be defined to change coloring according to a given property voalue of the current object. For more information about color selection, see Defining Color Rules. **NOTE**: This attribute is only relevant if the attribute **Show Label** is set to `True`.
Format	Defines a string to be displayed in the link label. By default, the link labels display all information defined in the `Show` properties of the Alfabet query or `SELECT` statement of the native Sql query that is defined in the **Query** child element of the **Link** element. Alternatively, a static text can be configured that includes variables for data derived from the query. To display current query results in the string, add a variable in the format {dataset column name}. **NOTE**: The name and caption of a dataset column are not necessarily identical. Which value must be taken over depends on the way an Alfabet query or a native SQL query are defined. For more information about how to identify the correct technical name of a dataset column, see Defining Column Names and Captions in the chapter Defining Queries. **NOTE**: This attribute is only relevant if the attribute **Show Label** is set to `True`.
Text Attributes	Click the rectangle in front if the **Text Attributes** attribute to define how text is displayed per default in the labels of the links. **NOTE**: To allow line breaks in the text of the link label, set the sub-attribute **Word Wrap** to `true`. **NOTE**: This attribute is only relevant if the attribute **Show Label** is set to True.
Show Label	If set to `True`, a label is displayed for each link in the report that provides information about the link. By default, the link labels display all information defined in the `Show` properties of the Alfabet query or `SELECT` statement of the native Sql query that is defined in the **Query** child element of the **Link** element. Alternatively, a static text can be configured that includes variables for data derived from the query. The attribute **Format** must be set to show static text in the labels.

Defining a Matrix Report

Matrix reports are designed in the **Report Assistant**. For information about how to open the **Report Assistant**, see Creating a New Graphic Report. For more information about how to work with the **Report Assistant**, see Configuring the Report with the Report Assistant.

The content and layout of a grid report is defined by adding elements to the explorer of the **Report Assistant** and setting the attributes in the attribute pane of the **Report Assistant** window.

The content and layout of a grid report is defined in the root node element.

The following table lists the explorer node elements that can be configured to specify the content of the report as well as provides information about the purpose of each sub-element:

Explorer Node Element	Required to Configure:
Root Node	General layout of the report.
Columns	Definition of the column header. The element is a container for one or multiple elements **Query** that define the objects displayed in the column header.
Rows	Definition of the row header. The element is a container for one or multiple elements **Query** that define the objects displayed in the row header.
Cells	Definition of the content of the cells in the report. The element is a container for one or multiple elements **Query** that define the objects displayed in the cells of the report.
Query	Definition of the content of the headers or cells of the report.
Color Rule	Definition of the colors used and the rules that define which objects are displayed in which color.
Indicator Rule	Definition of one or multiple icons displayed in the boxes representing the objects in the current cell of the lane report. The icons can either represent an indicator or any aspect defined via a query. For more information, see the section Defining Indicator Rules.

Defining the Basic Design of the Matrix Report

The basic design of the report depend on the sub-type of the matrix report. Two sub-types, `AffinityMatrix` and `Diagram` are available. For an overview of the differences between the two report types, see Matrix Reports in the section Types and Features of Configured Reports.

Customization of the basic report design is done via the attributes of the explorer root node. The report of the sub-type Diagram offers a greater flexibility for design changes than the report of the sub-type `AffinityMatrix`.

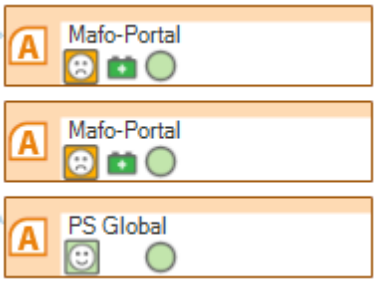
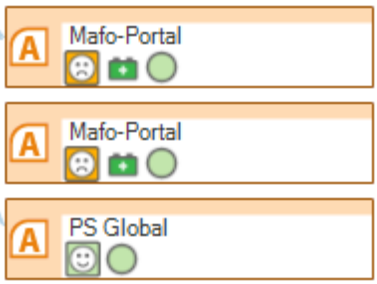
The design of the boxes representing objects in the report can be changed from the standard design with a white rectangle as background for text within the colored object box to fully colored object boxes. For more information see Defining the General Display of Object Boxes in Treemap, Diagram, Lane and Layered Diagram Reports.

Please note that the header cells for affinity matrices are displayed with text in white object boxes regardless of the setting for the general design.

The table below lists all attributes that specify the general layout of the report in the root node element and effect thereof.

The root node element also contains attributes for the definition of the cell content. These attributes are not listed in this section, but in the section Defining the Content of the Matrix Report.

Attribute	Description
Show Legend	Defines whether a legend is shown for the color coding of the objects in the report.
Render Type	Defines whether the matrix is displayed as an affinity matrix or a diagram. For an overview of the differences between the two report types, see Matrix Reports in the section Types and Features of Configured Reports.
Background Color	Defines the background color of the matrix. For more information about color selection, see Defining Color Attributes.
Grid Color	Defines the color of the borders between the cells of the matrix. For more information about color selection, see Defining Color Attributes.
Item Size	Defines the size of boxes that are displayed for all objects in the report in mm. (1 mm = 4 pixel). The size must be defined as length,height.
Item Border Width	Defines the width of the border of the boxes in the report. the default value is 1 point. When using the border color as object property indicator via a color rule, it is recommended to set the border width to a higher value.
Show Multiple Indicators	Defines whether display of icons defined via indicator rules is limited to one icon per object box or whether multiple icons can be assigned to an object box. If **Show Multiple Indicators** is set to `False`, a single icon can be displayed on the right of the object box. If multiple icons are found by indicator rules, the first icon found is assigned and all other icons are ignored. If **Show Multiple Indicators** is set to `True`, one or multiple icons are displayed from left to right at the bottom of the object box. If the number of icons that are assigned to the object box via indicator rules exceeds the space available for display of icons in a single row, the first icons found are displayed and, if the object box space is exceeded, any other icons are ignored. The way icons are placed in the object box also depends on the attribute **Indicator Fill Policy** of the **Item** node. For more information about the placement of icons assigned via indicator rules, see the section Defining Indicator Rules.
Indicator Fill Policy	If **Show Multiple Indicators** is set to `True`, this attribute defines how icons assigned to an object box are placed in the row on the bottom of the object box:

Attribute	Description
	• `ByIndex`: This setting shall be used if multiple indicators are defined via multiple indicator rules, each returning a single icon. The position of the icon resulting from each indicator rule is defined by means of the attribute **Indicator Index** of the respective indicator rule that must be set to an unsigned integer starting with 0 for the left most position. If an **Indicator Index** attribute has been defined for each indicator rule and an indicator is not set for a specific object, the position of this indicator icon in the box will be left empty and the position of all other indicator icons will not be changed:  If the order of the **Indicator Index** attribute is not defined, the placement of indicators will be arbitrary. This method is designed for positioning only one icon per indicator rule. If an indicator rule returns multiple icons, only one of the icons is displayed. • `LeftAlign`: This setting shall be used if a single indicator rule is defined that returns multiple icons. It can also be applied when multiple indicator rules each return a single icon, but the position and order of icons is not important to understand the report. The Icons are displayed in the order of occurence within the indicator rule(s) from left to right. If a different number of indicator icons is returned, the remaining icons are moved to the left:  If the indicator icons are all found by a single indicator rule, the order of icons depends on the order of results in the result dataset of the query of the indicator rule and may be different for different objects if no sort order is defined in the query.
Attributes for reports of the sub-type Diagram only	
Margin	Defines the space between the matrix and the border of the presentation object in number of pixels.

Attribute	Description
Border Color	Defines the color of the border between the matrix and the margin. For more information about color selection, see Defining Color Attributes.
Item Spacing	Defines the spacing between the boxes in a matrix cell in pixel.
Item Padding	Defines the spacing between the columns and between the boxes in the columns in number of pixels. The first number defines the vertical spacing between row borders and table content and the second number defines the horizontal spaces between column borders and cell content. The numbers must be specified comma-separated, that means vertical,horizontal.
Rounded Edges	Select `True` to display the diagram with rounded box edges, or `False` to display boxes with sharp edges.

Defining the Content of the Matrix Report

The explorer root node automatically contains the following child elements when opening the **Report Assistant**:

- one **Rows** element for the definition of the objects displayed in the row header cells on the left of the matrix.
- one **Columns** element for the definition of the objects displayed in the column header cells on top of the matrix.
- one **Cells** element for the definition of the objects displayed in the cells of the report.

The elements **Rows**, **Columns** and **Cells** must not have attributes. They are containers for one or multiple elements **Query**. The element **Query** defines the content of the header cells on basis of an Alfabet query or a native SQL query.

If you want to add objects from multiple classes to the row headers, column headers or cells, you must specify multiple elements **Query** within the respective container element, one for each object class.

Objects in a matrix cell are related to the objects in both the row and the column header. This relation must be defined in the query specifying the cell content. The query must be defined to deliver a tabular output that displays the `REFSTR` of both the object of the column header and the object in the row header in one of the table columns. That means that `REFSTR` of the object in the column header and the `REFSTR` of the object in the row header must be included in the Show properties of the Alfabet query or the `SELECT` clause of the native SQL query.

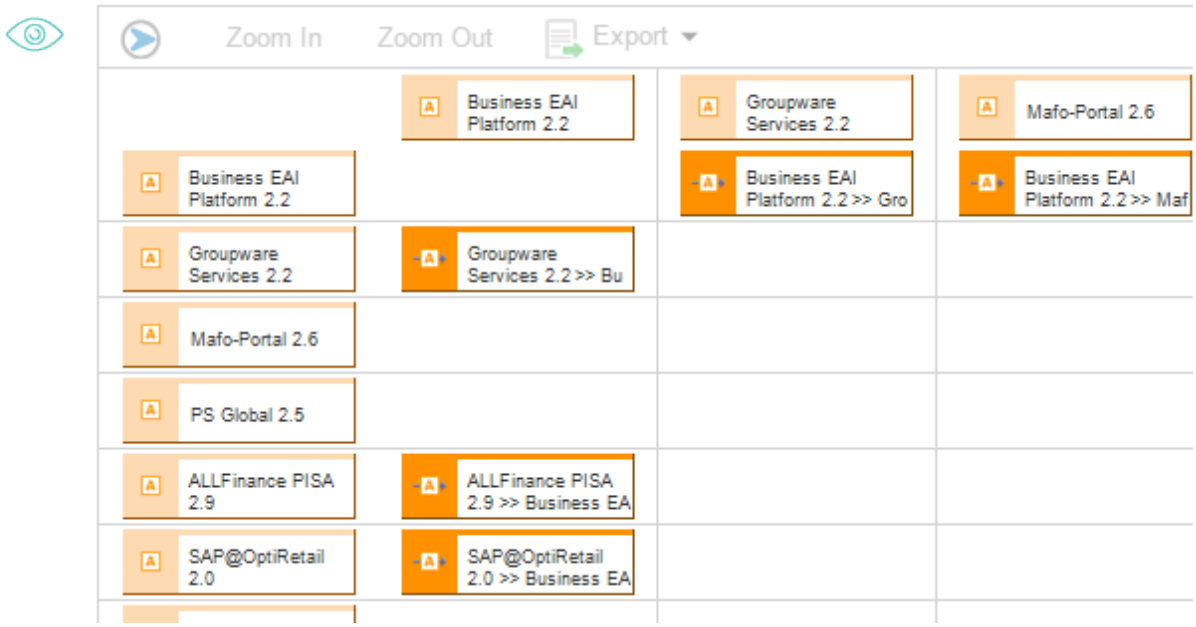
The position of this information within the Show property / the `SELECT` clause must be defined in the root node element with the attributes **Column Ref Index** and **Row Ref Index** . For example, if the `REFSTR` of the object in the column header is specified in the first column of the tabular output of the query, the **Column Ref Index** attribute must be set to 1.

In native SQL queries, the first column definition of the `SELECT` clause is not displayed in the resulting tabular output but exclusively used for technical processing of query results. Therefore, a **Column Ref Index** of 1 requires that the `REFSTR` of the object in the column header is the second column definition of the `SELECT` clause.

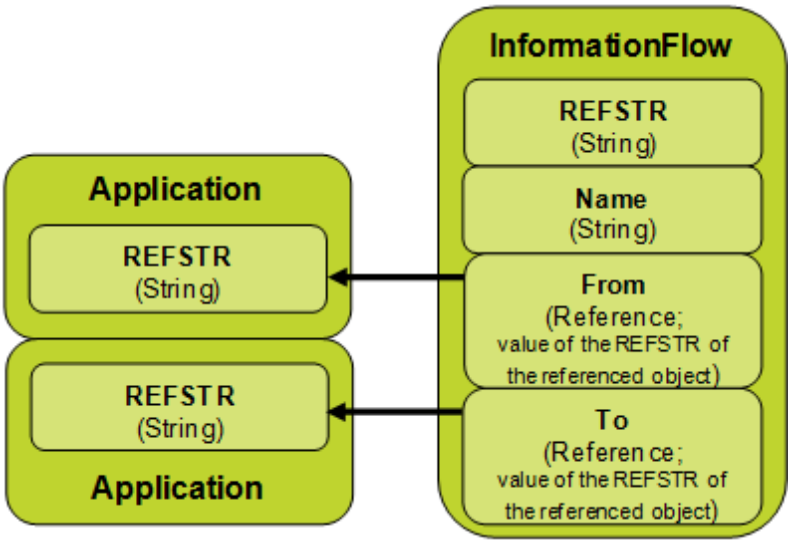
The columns of the tabular report output that are specified with **Column Ref Index** and **Row Ref Index** are not used for display of the object name in the matrix.

If multiple queries define cell content, the position of the `REFSTR`s of the column header and row header objects in the tabular output of the report must be identical for all queries.

A matrix of the sub-type AffinityMatrix shall display the information flows between applications. Applications are listed in both row and column headers. Each cell contains the information flows from the applications in the row headers to the applications in the column headers. The name of the business data that the information flow transfers is displayed in the boxes for the information flows:



The relation between the applications and the information flows are defined in the properties From and To of the information flow. The properties both reference a single application. The value of the properties is identical to the `REFSTR` property of the referenced application:



The report has a **Column Ref Index** of 2 and a **Row Ref Index** of 1.

The row headers list the applications the information flow is coming from and the **Row Ref Index** is 1. Therefore, when displaying the results from the query in a tabular report, the value of the property From of the information flow must be listed in the first column of the tabular query results.

The **Column Ref Index** is 2, which means that the property To of the the information flow must be listed in the second column of the tabular query output. All other columns define the information that shall be displayed in the object boxes in the cells of the report.

The queries resulting in the output in the image above are displayed below two times: with query definitions in Alfabet query language and with query definitions using native SQL. The query parts required for rendering of matrix cell content are marked in dark blue in the code.

When using Alfabet query language to define the report, the `FIND` class of each query must be the object class displayed in the part of the matrix defined by the Alfabet query. In the example, the find class for the cell content is `InformationFlow` although the information displayed in the boxes is exclusively about Business Data. The first and second Show properties of the Alfabet query defining the cell content define the position of the object in the matrix.

Column definition query:

```
ALFABET_QUERY_500
FIND Application
QUERY_XML
<QueryDef>
<ShowProperty Type="Property" ClassName="Application" Name="Name" />
<ShowProperty Type="Property" ClassName="Application" Name="Version"
/>
<SortProperty Type="Property" ClassName="Application" Name="Name" />
</QueryDef>
```

Row definition query:

```
ALFABET_QUERY_500
```



```
FIND Application
QUERY_XML
<QueryDef>
<ShowProperty Type="Property" ClassName="Application" Name="Name" />
<ShowProperty Type="Property" ClassName="Application" Name="Version"
/>
<SortProperty Type="Property" ClassName="Application" Name="Name" />
</QueryDef>
```

Cell definition query:

```
ALFABET_QUERY_500
FIND InformationFlow
QUERY_XML
<QueryDef>
<ShowProperty Type="Property" ClassName="InformationFlow"
Name="From" />
<ShowProperty Type="Property" ClassName="Application" Name="To" />
<ShowProperty Type="Property" ClassName="InformationFlow"
Name="Name" />
</QueryDef>
```

When using native SQL to define the report, the first column definition of the `SELECT` clause of all queries must define the `REFSTR` of the object to be displayed in the matrix. The second and third column of the `SELECT` clause of the native SQL query defining the cell content define the position of the object in the matrix.

Column definition query:

```
SELECTapp.REFSTR, app.NAME, app.VERSION
FROM APPLICATION app
```

Row definition query:

```
SELECTapp.REFSTR, app.NAME, app.VERSION
FROM APPLICATION app
```

Cell definition query:

```
SELECT inf.REFSTR,inf.A_FROM,inf.A_TO, inf.NAME
FROM INFORMATIONFLOW inf
```

The following table lists all attributes of the elements **Query** and **Root Node** that are required to define the content of matrix cells and headers.

Attribute	Description
Root Node	

Attribute	Description
ColumnRef Index	Specifies the column number in the tabular output of the query defined for cell content specification that contains the `REFSTR` of the object in the column header the cell content object is related to. The default is 1.
RowRefIndex	Specifies the column number in the tabular output of the query defined for cell content specification that contains the `REFSTR` of the object in the row header the cell content object is related to. The default is 2.
Cell Reference	Select `Implicit`. The value `Explicit` is displayed for reasons of backward compatibility and should not be used.
Query	
Name	Defines the name of the **Query** element in the explorer.
Alfabet Query / Native Sql / Query as Text	Defines the content, look and number of the column headers, row headers or cell content, depending on the container element for the **Query** element. A valid Alfabet query must be defined in either the **Alfabet Query** or **Query as Text** attribute or a native SQL query must be written in the **Native Sql** attribute. The query must find the relevant objects and defines which properties of the objects are displayed in the object boxes.
Cascading	Not relevant for matrix reports.

Defining Node Arc Reports or Node Arc Reports With Edge Bundling

Node arc reports are based on the design elements and layouts of standard Alfabet diagrams. The boxes displayed in the node arc report to represent objects are all based on customer defined diagram item templates. Instead of building the box design from scratch, the configured report references customer defined diagram item templates that are either used as defined or modified with the configurations done in the configured report.

Prior to creating a node arc report, you must Therefore, define diagram item templates for use in the configured report. For information about how to configured diagram item templates, see Configuring Custom Diagram Item Templates.

Please note the following about the configuration of diagram item templates for use in node arc reports:

- Standard diagram items cannot be used in configured reports. If you would like to use the standard design of a diagram item for a class in a node arc report, you must create a

custom diagram item template as copy of the standard diagram item template without changes in design.

- The information about the object displayed in the elements of the diagram item template can be modified via the configured report configuration. In that case, the information is taken over from a defined column in the result dataset of the query defined in the configured report. Matching of column data with the elements of the diagram item template is performed by name of the diagram item template element. If you define the configured report to display objects based on multiple different diagram item templates, make sure that the respective elements are all named identically for all configured diagram item templates.
- If you would like to display nodes within nodes, you must define an area for the display of the child nodes in the diagram item template to be used for the parent nodes. In addition, the **Layout Type** attribute must be set to `Scalable` for both the custom diagram item template used for rendering the outer and the inner nodes. Custom diagram item templates used for outer nodes must have an area for placement of the inner nodes defined. Please note that only one level of nesting is supported for nodes in node arc reports.

Node arc reports can be based on two different templates. For reports based on the template `NodeArcReport`, the user can select the diagram layout in runtime. The predefined filters will be the same as for standard Alfabet diagrams. Reports based on the template `NodeArcReportWithBundling` are always displayed in `Spring Layout`. The user can select at runtime, whether and how much connections are bundled to enhance visibility. The configuration options available are the same for both types of node arc reports.

Node arc reports are designed in the **Report Assistant**. For information about how to open the **Report Assistant**, see Creating a New Graphic Report. For more information about how to work with the **Report Assistant**, see Configuring the Report with the Report Assistant.

The content and layout of a node arc report is defined by adding elements to the explorer of the **Report Assistant** and setting the attributes in the attribute pane of the **Report Assistant** window.

The following table lists the explorer node elements that can be configured to specify the content of the report as well as provides information about the purpose of each sub-element:

Explorer Node Element	Required to Configure:
Root Node	Definition of the layout of the report.
Node Query	Definition of the dataset that defined which nodes are displayed with which information and design in the node arc report.
Arc Query	Definition of the dataset that defined which nodes are connected with each other and how the connections between the nodes are designed.

Explorer Node Element	Required to Configure:
Indicator Rule	Indicator rules can be defined to display icons within the nodes of the node arc report. Currently only indicator rules of the type `IconQuery` are supported. For more information about indicator rule definition, see .Defining Indicator Rules

The **Node Query** and **Arc Query** define the content and layout of the nodes and arcs, that means connections, in the node arc report. Which column in the result datasets of the **Node Query** and **Arc Query** provide which information required to build the node arc report with the correct content and layout is defined in the attributes of the root node of the node arc report. In the root node, it is also possible to define the general layout and a default layout for nodes and arcs that is used if nodes or arcs do not have a layout defined in the respective query result.

Defining the Nodes Displayed in the Node Arc Report

The nodes displayed in the node arc report are found by a query that has to be defined in the attribute **Alfabet Query** (or **Query as Text**) or **Native SQL** of the explorer node **Node Query**.

The query must return all information that is required to display the nodes of the configured report. The name of the column that contains the respective information must then be mapped with one of the attributes of the root node of the node arc report that are listed in the following table together with information about the required format of the information in the dataset and a description of the triggered functionality:

Attribute	Required value type in dataset	Description
Node Ref Column	REFSTR property value of an object	Returns the REFSTR of the object that is represented by the node. This definition is mandatory.
Node Shape Column	The name of an existing diagram item template.	Returns the diagram item template that the node should be based on. The diagram item template is defined by the name of the default diagram item template with the prefix `CustomItems:`, For example, `CustomItems:AppApplicationServer`.
Node Background Color Column	HTML color code	Returns the background color to be used for the diagram item that displays the current object. This value overwrites the background color defined for the diagram item template used to display the node.

Attribute	Required value type in dataset	Description
Node Navigation View Column	`View=ViewType:ViewName` The `ViewType` can be one of the following: • ***Report*** for a configured report • ***GraphicView*** for a standard Alfabet view • ***ObjectView*** for an object profile `ViewName` is the name of the view. If the link target is a configured report, optionally parameters can be specified for filling the filter fields of the report. For more information about defining navigation to a target view including definition of parameters, see Defining Navigation from the Report to Alfabet Views.	Defines the name of the column containing the definition of an alternative link target for the double click action on the object nodes. By default, the standard object profile of the object opens when a user double clicks on a node or clicks a node and then clicks the **Navigate** button. To enable navigation, both **Node Navigation View Column** and **Node Ref Column** must be defined. The `REFSTR` returned in the column defined with **Node Ref Column** is submitted as base object to the view that opens.
Node Icon Name Column	The name of an icon that is available in the icon gallery.	Returns the icon that shall be displayed on the node to represent the object. Icons must have been uploaded to the icon gallery to be used in the configured report. For more information about upload of icons see Adding and Maintaining Icons for the Alfabet Interface.
Node Tool Tip Column	String	Returns a text to be displayed as tooltip when the user points with the cursor to the node.
Node Legend Group Column	String	Returns a string that shall be displayed as heading for the legend entry defined with **Node Legend Text Column** in the same row of the dataset. All legend entries that have the same **Node Legend Group Column** value are grouped and displayed under the heading defined with **Node Legend Group Column**.
Node Legend Text Column	String	The text that shall be displayed for the kind of node in the legend. The legend shows an entry for each combination of coloring and icon in the node. The node icon is displayed in front, and the node legend text is displayed in a field with the background color identical to the one defined for the node.

Attribute	Required value type in dataset	Description
		Legend text is only displayed if a legend group for the text is also defined with **Node Legend Group Column**.
Node Labels and Areas	Any information about objects that can meaningfully be displayed on the nodes	With this attribute the information displayed about the object in the node is defined. Click into the attribute field and then click on the **Browse** button on the right of the attribute field to open an editor for the mapping of information from the result dataset of the query to elements defined to display information in the diagram item template that the node is based on. For each information to display: 1) Write the name of the dataset column containing the information into the field **Node Query Column**. 2) Write the name of the element in the diagram item template that shall be the container for the information into the field **Diagram Item Template Shape**. 3) Click the button **Add**. After having defined the mapping, click **OK** to close the editor and save your configuration. Please note that each dataset column can only be mapped once to an diagram item template element name. Therefore, if you want multiple diagram item templates to be used in your configured report, these shall all use the same names for elements displaying information.

Please note the following about the definition of the node query:

- **To display objects of multiple object classes as nodes in the node arc report**: As only one column name per functionality can be defined for the nodes in the attributes of the root node of the report, the query must return the values for all object classes that return the same type of information in columns of the same name. You can use For example, `UNION ALL` for a definition of multiple object classes in native SQL queries.
- **To display objects in a nested node infrastructure**: Nodes can be displayed within a two-level hierarchy nested into each other. To display child nodes within parent nodes, use the `GroupBy_Ex`

instruction to create a dataset with two levels with the parent node definition on the upper level. As only one column name per functionality can be defined for the nodes in the attributes of the root node of the report, the query must return the values for all object classes that return the same type of information in columns of the same name. Use `JoinColumn` instructions in both Alfabet queries or native SQL queries to join identical information from both levels into one column each.

For information about creating a grouped dataset with Alfabet instructions, see Grouping Query Results in Expandable Reports.

If nested nodes shall be defined, additional attributes must be defined in the root node of the report assistant. Most display options for nested nodes can either be defined statically by value definition in the attributes of the root node or dynamically in the query. If values are defined in the query, the values selectable for the static attributes must be returned in a column of the query and the attribute for dynamic value definition must be set to the column name of the column in the node query returning the value. Static values are then used as default values if values are not returned by the query:

Attribute for Static Definition	Attribute for Dynamic Definition	Description
Enable Box-in-Box		Set to `True` if you want to show two levels of nodes nested into each other. A section **In-Box Rendering** is added to the attributes of the root node of the report assistant if the attribute is set to `True`. Exand the **In-Box Rendering** in that section and define the attributes defined in this table in the following.
Node Resizing Model	Node Resizing Model Column Name	Defines how the availability of child nodes shall influence the size of nodes in the node arc report: • `KeepChildNodeSize`: Select if the size of inner nodes shall be maintained and the outer nodes shall be enlarged if this is required to fit in the inner nodes. • `KeepParentNodeSize`: Select if the size of the outer nodes shall be maintained and the size of the inner nodes shall be reduced to fit all inner nioes into the outer node. If text for the nodes • `KeepAllNodeSizes`: Select if all nodes shall have a fixed size and the display of inner nodes shall be limited to the number of inner nodes fitting into the available space in the placement area of the outer node.
Initial Child Item Size		Select the size of the inner nodes that shall apply if no node resizing is done. Please note that the size of the inner nodes must be smaller than the size of the outer nodes defined with the attribute **Item Size**. The values available in the drop-down list of the attribute **Initial Child Item Size** are configured in the XML object ***DiagramInformationFlowDef***. For more information, see the section Configuring the Sizes of Diagram Items in Automatically Generated Diagrams. Depending on the configuration, some sizes may be configured to hide the name, icon, and attributes. In this case, the user can point to the

Attribute for Static Definition	Attribute for Dynamic Definition	Description
		node to display a tooltip with the object's name or click-and-hold to open the preview of the object.
Minimal Child Node Size	Minimal Child Node Size Column Name	If the **Node Resizing Model** attribute was set to `KeepParentNodeSize`, you can optionally define a maximum required size of the inner nodes as an integer. If the inner nodes will not fit in to the outer node with the defined minimal node size, the number of displayed inner nodes will be restricted to the number of nodes fitting the available space and additional inner nodes returned by the node query for display in the outer node excluded from displayed.
Maximal Number Of Child Nodes	Maximal Number Of Child Nodes Column Name	Optionally define the maximum number of inner nodes that may be displayed per outer node as integer. This may be useful to restrict the size of the outer node if the **Node Resizing Model** attribute was set to `KeepChildNodeSize`.
Child Node Padding	Child Node Padding Column Name	Define the padding between inner nodes as integer.
Maximal Container Node Width	Maximal Container Node Width Column Name	If the **Node Resizing Model** attribute was set to `KeepChildNodeSize`, you can optionally define a maximum allowed width of the outer nodes as integer. If both the **Maximal Container Node Width** and **Maximal Container Node Height** attributes are defined, the number of displayed inner nodes will be restricted to the number of nodes fitting the available space and additional inner nodes returned by the node query for display in the outer node excluded from displayed.
Maximal Container Node Height	Maximal Container Node Height Column Name	If the **Node Resizing Model** attribute was set to `KeepChildNodeSize`, you can optionally define a maximum allowed width of the outer nodes as integer. If both the **Maximal Container Node Width** and **Maximal Container Node Height** attributes are defined, the number of displayed inner nodes will be restricted to the number of nodes fitting the available space and additional inner nodes returned by the node query for display in the outer node excluded from displayed.
Default Items Area Name	Container Area Name Column Name	Define a default name that will be used to identify the area in the custom diagram item template for the outer node if the definition in the query does not provide a valid area name.

Attribute for Static Definition	Attribute for Dynamic Definition	Description
Diagram Item To Container Area Map		Define the name of the area in the custom diagram item template for the outer node that shall contain the inner nodes. In the static definition, click the **Browse** button in the attribute. A new window opens. Click the **Add Diagram Items** button in the upper left corner, select a custom diagram item template that shall be used for display of outer nodes. the diagram item template is displayed in the window with custom diagram item template name on the left and a drop-down list of available areas in the custom diagram item template on the right. Select the relevant area from the drop-down list.

A default layout can be defined for the nodes in the configured report with the attributes listed in the following table. The default layout is used if a valid definition for the respective design is not provided in the query results. If the same design shall anyway be used for all nodes in the configured report, the default layout can be defined and the query does not have to return the design elements defined as default.

Attribute	Description
Default Node Shape Name	Specifies the name of the diagram item template that shall be used as default for all nodes that do not have a diagram item template defined in the node query. when you click into the attribute field and then click the **Browse** button on the right of the attribute field, a selector opens that lists all available diagram item templates sorted by object class in an expandable table. Click a row in the table and click **OK** to select the custom diagram item template. If you click a first level row with the object class name, the standard diagram item template for the object class is selected instead of a custom diagram item template. If neither the default nor a definition in the query is available, the standard diagram item template for applications will be used.
Default Node Label Area Name	Specifies a default for the area that information is displayed in for the case that in the Nodes And Areas Attribute a specification does only contain a column name but not a diagram item template area name.
Default Node Background Color	Specifies the default background color that will be used for all nodes for that no background color is defined via the node query. If neither the default nor a definition in the query is available, the standard diagram item template for applications will be used.

Defining the Arcs Connecting the Nodes in the Node Arc Report

The arcs displayed in the node arc report are found by a query that has to be defined in the attribute **Alfabet Query** (or **Query as Text**) or **Native SQL** of the explorer node **Arc Query**.

The query must at least return the RESTR of the start object and the end object for the arc. If the arc represents an object, the definition of the REFSTR of the object in the query has to be defined to enable navigation to hte object's object profile or any other view specified via an instruction. Styling of the arcs can also be defined individually via the query.

Attribute	Required value type in dataset	Description
Arc Start Node Ref Column	REFSTR property value of an object represented by a node	Returns the REFSTR value of the object the arc is starting at. This information is mandatory.
Arc End Node Ref Column	REFSTR property value of an object represented by a node	Returns the REFSTR value of the object the arc is pointing to. This information is mandatory.
Arc Label Column	String	Returns the text that shall be displayed as label on the arc. Labels are only displayed if this attribute is set and the attribute **Show Arc Labels** is set to `True`.
Arc Tooltip Column	String	Returns the text that shall be displayed as tooltip if the user moves the cursor over the arc line.

Arc Label Position Column	`Center`
	
	`Start`
	
	`End`
	
	`PureStart`
	
	`PureEnd`
	
	`PureCenter`
	
	`UserDefined`
		Returns the position of the label on the arc. While the label is positioned vertically on top of the line representing the arc when selecting `Start`, `Center` or `End`, the label is positioned vertically directly on the line when `PureStart`, `PureEnd` or `PureCenter` is used. `UserDefined` specifies that the value defined with **Default Label Position** is used.
		
		Labels are only displayed if the attribute **Arc Label Column** is set and the attribute **Show Arc Labels** is set to `True`.

Arc Reference Column	REFSTR property value of an object	Returns the REFSTR of the object represented by the arc. This attribute must be set to enable navigation from the arc in the node arc report.

Attribute	Required value type in dataset	Description
Arc Navigation View Column	`View=ViewType:ViewName` The `ViewType` can be one of the following: • ***Report*** for a configured report • ***GraphicView*** for a standard Alfabet view • ***ObjectView*** for an object profile `ViewName` is the name of the view. If the link target is a configured report, optionally parameters can be specified for filling the filter fields of the report. For more information about defining navigation to a target view including definition of parameters, see Defining Navigation from the Report to Alfabet Views.	Defines the name of the column containing the definition of a link target for the double click action on the arc arrows. By default, navigation is not provided for arcs. To enable navigation, both **Arc Navigation View Column** and **Arc Reference Column** must be defined. The `REFSTR` returned in the column defined with **Arc Reference Column** is submitted as base object to the view that opens.
Arc Color Column	HTML color code	Returns the color of the arc line in HTML compliant color coding.
Arc Label Back Color Column	HTML color code	Returns the color of the label background in HTML compliant color coding. Labels are only displayed if the attribute **Arc Label Column** is set and the attribute **Show Arc Labels** is set to `True`.
Arc Label Text Color Column	HTML color code	Returns the color of the text in the label in HTML compliant color coding. Labels are only displayed if the attribute **Arc Label Column** is set and the attribute **Show Arc Labels** is set to `True`.
Arc Label Border Color Column	HTML color code	Returns the color of the label border in HTML compliant color coding. Labels are only displayed if the attribute **Arc Label Column** is set and the attribute **Show Arc Labels** is set to `True`.

Attribute	Required value type in dataset	Description

Arc Line Style Column	`Solid`
	`Dash`
	`DashDot`
	`DashDotDot`
	`Dot`
		Returns the line style of the arc line. The line styles are limited to one of the preconfigured styles.

Arc Line Width Column	Integer	Returns the width of the arc line.
Arc Legend Text Column	String	Returns the text that shall be displayed next to the line color in the legend. The legend only destinguisches coloring. That means that independent from the styling selected for the line with and arrow style, lines are all displayed by a symbol with the same linestyle, but in the color returned in the column defined in the attribute **Arc Color Column** or, if no color is defined via the query, the color selected with the attribute **Default Arc Color**. the text returned in the column specified with the attribute Arc Legend Text Column is written next to the line symbol in the color and with the background color defined for the node labels. Legend text is only displayed if a legend group for the text is also defined with **Node Legend Group Column**.
Arc Legend Group Column	String	Returns a string that shall be displayed as heading for the legend entry defined with **Arc Legend Text Column** in the same row of the dataset. All legend entries that have the same **Arc Legend Group Column** value are grouped and displayed under the heading defined with **Arc Legend Group Column**.

Start Arc Arrow > Arrow Style Column	`None`
	`Angle`
	`ArrowedTriangle`
	`Circle`
	`LeftAngle`
		Returns the design of the arrow at the start of the line. The arrow styles are limited to one of the preconfigured styles.

Attribute	Required value type in dataset	Description

	`Rhomb`
	`RightAngle`
	`Triangle`	

Start Arc Arrow > Arrow Size Column	`Small`
	`Middle`
	`Large`
	`ExtraLarge`
		Returns the size of the arrow at the start of the arc as one of the allowed values between small and extra large.

End Arc Arrow > Arrow Style Column	`None`
	`Angle`
	`ArrowedTriangle`
	`Circle`
	`LeftAngle`
	`Rhomb`
	`RightAngle`
	`Triangle`
		Returns the design of the arrow at the end of the line. The arrow styles are limited to one of the preconfigured styles.

End Arc Arrow > Arrow Size Column	`Small`
	`Middle`
	`Large`
	`ExtraLarge`
		Returns the size of the arrow at the end of the arc as one of the allowed values between small and extra large.

A default layout can be defined for the arcs in the configured report with the attributes listed in the following table. The default layout is used if a valid definition for the respective design is not provided in the query results. If the same design shalll anyway be used for all arcs in the configured report, the default layout can be defined and the query does not have to return the design elements defined as default.

Attribute	Description
Default Arc Label Position	Select the default position position of the label of the arc if not returned via the query. While the label is positioned vertically on top of the line representing the arc when selecting `Start`, `Center` or `End`, the label is positioned vertically directly on the line when `PureStart`, `PureEnd` or `PureCenter` is used. `UserDefined` specifies that the value defined with **Default Label Position** is used.

Attribute	Description
	Labels are only displayed if the attribute **Arc Label Column** is set and the attribute **Show Arc Labels** is set to `True`.
Default Arc Color	Select a color that is used as default for the coloring of arc lines if a color is not defined via the column in the **Arc Node Query** defined with the attribute **Arc Color Column**.
Default Arc Label Back Color	Select a color that is used as default for the coloring of arc label backgrounds if a color is not defined via the column in the **Arc Node Query** defined with the attribute **Arc Label Back Color Column**. Labels are only displayed if the attribute **Arc Label Column** is set and the attribute **Show Arc Labels** is set to `True`.
Default Arc Label Text Color	Select a color that is used as default for the coloring of arc label text if a color is not defined via the column in the **Arc Node Query** defined with the attribute **Arc Label Text Color Column**. Labels are only displayed if the attribute **Arc Label Column** is set and the attribute **Show Arc Labels** is set to `True`.
Default Arc Label Border Color	Select a color that is used as default for the coloring of arc label borders if a color is not defined via the column in the **Arc Node Query** defined with the attribute **Arc Label Border Color Column**. Labels are only displayed if the attribute **Arc Label Column** is set and the attribute **Show Arc Labels** is set to `True`.
Default Arc Line Style	Select a line style that is used as default for the style of arc lines if a style is not defined via the column in the **Arc Node Query** defined with the attribute **Arc Line Style Column**.
Default Arc Line Width	Enter a line width in pixel that is used as default for the width of the arc lines if a width is not defined via the column in the **Arc Node Query** defined with the attribute **Arc Line Width Column**.
Start Arc Arrow > Default Arrow Style	Select an arrow style that is used as default for the arrows at the start of the arc if a style is not defined via the column in the **Arc Node Query** defined with the attribute **Start Arc Arrow** > **Arrow Style Column**.
Start Arc Arrow > Default Arrow Size	Select an arrow size that is used as default for the arrows at the start of the arc if a size is not defined via the column in the **Arc Node Query** defined with the attribute **Start Arc Arrow** > **Arrow Size Column**.

Attribute	Description
End Arc Arrow > Default Arrow Style	Select an arrow style that is used as default for the arrows at the end of the arc if a style is not defined via the column in the **Arc Node Query** defined with the attribute **End Arc Arrow** > **Arrow Style Column**.
End Arc Arrow > Default Arrow Size	Select an arrow size that is used as default for the arrows at the end of the arc if a size is not defined via the column in the **Arc Node Query** defined with the attribute **End Arc Arrow** > **Arrow Size Column**.

Defining the General Layout of the Node Arc Report

The general layout of the node arc report defines the default for the setting of diagram specific filters that are automatically added to the configured report view when the configured report view is created. When a user opens the node arc report, the layout defined by default is used to display the node arc report and the user can then change to a different layout using one of the following methods:

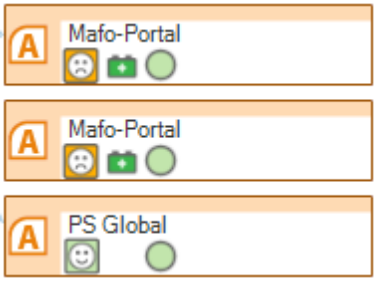
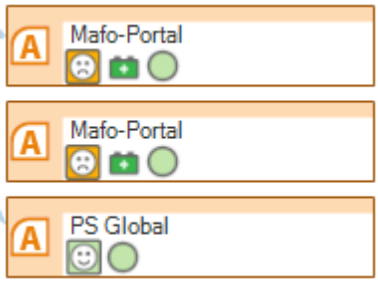
- Users can change the layout via the standard filters added to the report by default. If you do not want to the user to change the layout options defined as default, you can remove the corresponding filter fields from the filter panel. On demand, you can add custom filters to the filter panel. Definition of filters is described in the section Defining Filters for Configured Reports and Selectors
- Users with edit rights to the base object of the report can change the layout in the Alfabet Diagram Designer. The Alfabet Diagram designer opens when the user clicks the **Change Layout** button of the report. The layout of the report can be modified in the Alfabet Diagram Designer including changes to the size, alignment, coloring, and styling of the objects in the report. Additionally, the caption of the node arc report will be displayed in the Alfabet Diagram Designer. If objects are not found any longer via the query of the node arc reports, the objects and the connections of the objects to other objects will be removed from the rendering of the report. The changed layout will be saved for the current user and the node arc report will use the layout if the current user opens the report for the same base object. The user can re-open the diagram designer to change the layout or remove the layout at a later time.

Optionally, node arc reports can be configured to save a layout defined by one user for all users. The buttons **Change Layout** and **Delete Saved Layout** will be changed to **Change Saved Layout** and **Delete Shared Layout** if the layout is changed to inform the user that any change to the layout will also change the view for other users. Any changes performed by any authorized users will change the layout for all users.

In addition to the default settings for the standard filters, display of labels and legends are part of the standard layout definition.

The following attributes of the node arc report are part of the standard layout definition:

Attribute	Description
Layout	Dependent on the layout type, boxes and connections are differently arranged within the diagram. For example, Condensed displays all nodes in a circle with the connecting arcs between them. Which layout type is best for display of the report depends on the current content. For the same report, display of a high number of objects might require a different layout then the same report displaying only a few objects to give the best overview. Select a layout from the available layout types in the drop-down list. The layout will be used when the user first opens the diagram.
Item Size	Select the size of the nodes displayed in th the node arc report. The values available in the drop-down list of the attribute **Item Size** are configured in the XML object ***DiagramInformationFlowDef***. For more information, see the section Configuring the Sizes of Diagram Items in Automatically Generated Diagrams. Depending on the configuration, some sizes may be configured to hide the name, icon, and attributes. In this case, the user can point to the node to display a tooltip with the object's name or click-and-hold to open the preview of the object.
Format	Define the page format to use to display the diagram. The page format setting is not relevant if `Condensed` is selected as layout.
Landscape	Defines the orientation of the node arc report. If set to `True`, the node arc report is displayed in landscape orientation. If set to `False`, it is displayed in portrait orientation. The default value is `False`.
Show Arc Labels	Defines whether labels are displayed on the arcs of the node arc report. Labels are only displayed, if content and design is defined in the definition of the arcs and this attribute is set to `True`.
Show Legend	Defines whether a legend is shown for the node arc report. If set to `True`, legend texts and grouping should be defined as part of the node and arc definitions.
Indicator Fill Policy	If multiple indicator rules are defined for the node arc report, this attribute defines how icons assigned to a node are placed in a row on the bottom of the node: • `ByIndex`: This setting shall be used if multiple indicators are defined via multiple indicator rules, each returning a single icon. The position of the icon resulting from each indicator rule is defined by means of the attribute **Indicator Index** of the respective indicator rule that must be set to an unsigned integer starting with 0 for the left most position. If an **Indicator Index** attribute has been defined for each indicator rule and an indicator is not set for a specific object, the position of this indicator icon in the box will be left empty and the position of all other indicator icons will not be changed:

Attribute	Description
	 If the order of the **Indicator Index** attribute is not defined, the placement of indicators will be arbitrary. This method is designed for positioning only one icon per indicator rule. If an indicator rule returns multiple icons, only one of the icons is displayed. • `LeftAlign`: This setting shall be used if a single indicator rule is defined that returns multiple icons. It can also be applied when multiple indicator rules each return a single icon, but the position and order of icons is not important to understand the report. The icons are displayed in the order of occurence within the indicator rule(s) from left to right. If a different number of indicator icons is returned, the remaining icons are moved to the left:  If the indicator icons are all found by a single indicator rule, the order of icons depends on the order of results in the result dataset of the query of the indicator rule and may be different for different objects if no sort order is defined in the query.
Allow Others to View Saved Diagram	If set to `True`, users with edit permissions for the base object of the node arc report can define a diagram layout for the node arc report that will then be shared with all users. If another user opens the node arc report for the same base object, the shared layout will be displayed. Users with edit permissions to the base object of the node arc report can change or delete the shared layout.

Defining Portfolio Reports

Portfolio reports are designed in the **Report Assistant**. For information about how to open the **Report Assistant**, see Creating a New Graphic Report. For more information about how to work with the **Report Assistant**, see Configuring the Report with the Report Assistant.

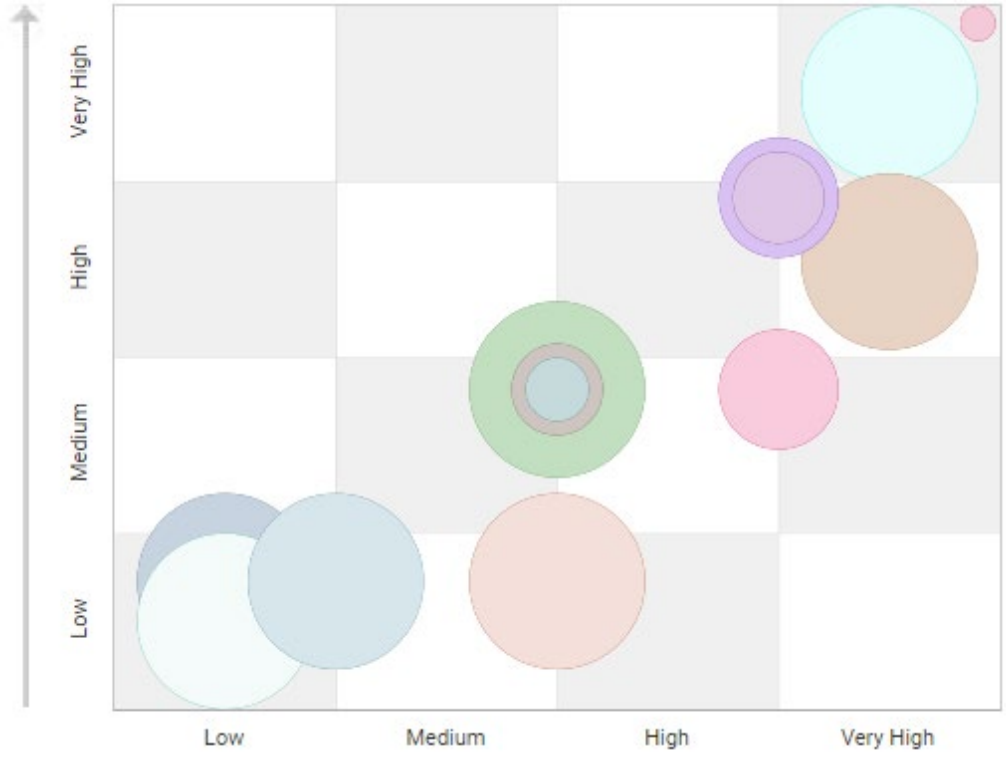
The content and layout of a portfolio report is defined by adding elements to the explorer of the **Report Assistant** and setting the attributes in the attribute pane of the **Report Assistant** window.

The following table lists the explorer node elements that can be configured to specify the content of the report as well as provides information about the purpose of each sub-element:

Explorer Node Element	Required to Configure:
Root Node	Definition of the layout of the report.
X-axis	Static definition of the X-axis of the portfolio.
Y-axis	Static definition of the Y-axis of the portfolio.
Power Axis	Static definition of the power axis of the portfolio. The power axis is the third dimension of evaluation displayed in a portfolio by variation of the size of the shapes representing the objects in the portfolio.
Query	Definition of the objects displayed in the portfolio and dynamic definition of portfolio layout and portfolio axes.
Connection	Definition of connections between objects in the portfolio.

The following sections guide you through the definition of various features of the portfolio.

The default layout displays objects represented by colored bubbles. If a power axis is defined, the bubbles will be sized according to the power axis value. The default portfolio displays an X-axis and Y-axis separated in four segments with the labels **Low**, **Medium**, **High** and **Very High**:



For the definition of a portfolio with default layout it is sufficient to define the base content as described in the sections Defining the Query the Portfolio is Based On and Defining the Objects and Evaluation Dimensions for the Portfolio Report. All other definitions are optional:

Defining the Query the Portfolio is Based On

The objects displayed in the portfolio, the indicators used for evaluation, and any variable design elements are defined in a query that can either be a native SQL query or Alfabet query.

It is recommended to use native SQL for creating portfolio reports. Alfabet query language does not provide the ability to include anything else than data from the tables in the Alfabet database into the search results, which excludes some of the features of the report.

The query must be defined in a ***Query*** node added to the **Report Assistant** explorer. To add the ***Query*** node:

Only one ***Query*** node can be defined for the portfolio definition.

1) Right-click the root node of the explorer in the **Report Assistant** and select **Add New Query** . A new ***Query*** element is added to the explorer node.
2) Click the ***Query*** node and define the attributes of the node. The following settings can be applied:

Attribute	Description
Name	Defines the name of the **Query** element displayed in the explorer of the **Report Assistant**.
Alfabet Query / **Query as Text** / **Native Sql**	Defines a query to find the objects that are displayed in the report, the indicator values that are used to calculate the location of the object shapes in the portfolio, and all optional dynamic design elements of the portfolio.

The query defined for the report must at least find the objects that shall be displayed in the report and the relevant values used for evaluation. In addition, configurations that are described in the following sections are based on optional data returned by the query.

Please note the following basic rules for the definition of the query:

- In native SQL queries, the first argument of the `SELECT` statement must return the `REFSTR` of the object to be displayed in the report in addition to defining the following information in the `SELECT` statement. The first argument is not part of the data set resulting from the query and cannot be mapped to attributes in the **Report Assistant**.
- In Alfabet queries, indicators cannot be added to the show properties using the show properties of the type `Indicator`.

The following information determines the objects defined in the query and the display of evaluation data for the X-axis, Y-axis and power axis:

Feature to be implemented	Mandatory/Optional	Column Content
Identification of the objects to be displayed in the portfolio report	Mandatory	The values of a unique property of the objects to be displayed in the report. This can either be the `REFSTR` of the object or any other unique property or combination of properties that ensures that the column in the report table can be mapped with objects in the database. If no tooltip is defined as described in the section Defining a Tooltip for Objects in the Portfolio Area, the information used to identify the object will be displayed as tooltip if the user moves the cursor around an object in the portfolio. A legend is available for the colors used for the objects in the portfolio. The information in the legend is identical to the value of the property that is used to identify the object. It is Therefore, recommended to Please note that the first argument of the `SELECT` statement in a native SQL query cannot be used for the report definitions. If the `REFSTR` returned in the first argument shall be used to identify the objects, it has to be added again as following argument.

Feature to be implemented	Mandatory/Optional	Column Content
The value determining the placement of the object bubble on the X-axis.	Mandatory	The values in the column must be numerical. Evaluation does not have to be based on indicator values, but any other aspect that can be expressed in numbers can be defined.
The value determining the placement of the object bubble on the Y-axis.	Mandatory	The values in the column must be numerical. Evaluation does not have to be based on indicator values, but any other aspect that can be expressed in numbers can be defined.
The value determining the size of the object bubble or geometric shape.	Optional	The values in the column must be numerical. Evaluation does not have to be based on indicator values, but any other aspect that can be expressed in numbers can be defined. The size is not variable for object representation via icons.

The following information can be returned by the query to define optional features for the portfolio report:

Feature to be implemented	For more information see:
Displaying a tooltip when the customer moves the cursor over the object bubble, shape or icon.	Defining a Tooltip for Objects in the Portfolio Area
Defining X-Axis and Y-Axis segment labels	Defining Axes for a Standard Layout Portfolio
Defining X-Axis and Y-Axis captions	Defining Axes for a Standard Layout Portfolio
Representing objects in the portfolio by different geometric shapes	Representing Objects as Geometric Shapes
Representing of objects in the portfolio by different icons	Representing Objects As Icons
Coloring of bubbles or geometric shapes in the portfolio	Defining the Coloring of Shapes in the Portfolio Report
Displaying a legend for the power axis of the portfolio report	Defining a Legend for the Power Axis of the Portfolio

Feature to be implemented	For more information see:
Defining connections between objects in the portfolio report	Defining Connections between Objects Displayed in the Portfolio
Defining a navigation target for navigation from objects in the portfolio	Defining Navigation from the Objects in the Portfolio to Alfabet Views
Defining a caption for the portfolio	Defining the Caption and Size of the Portfolio Report Area

Defining the Objects and Evaluation Dimensions for the Portfolio Report

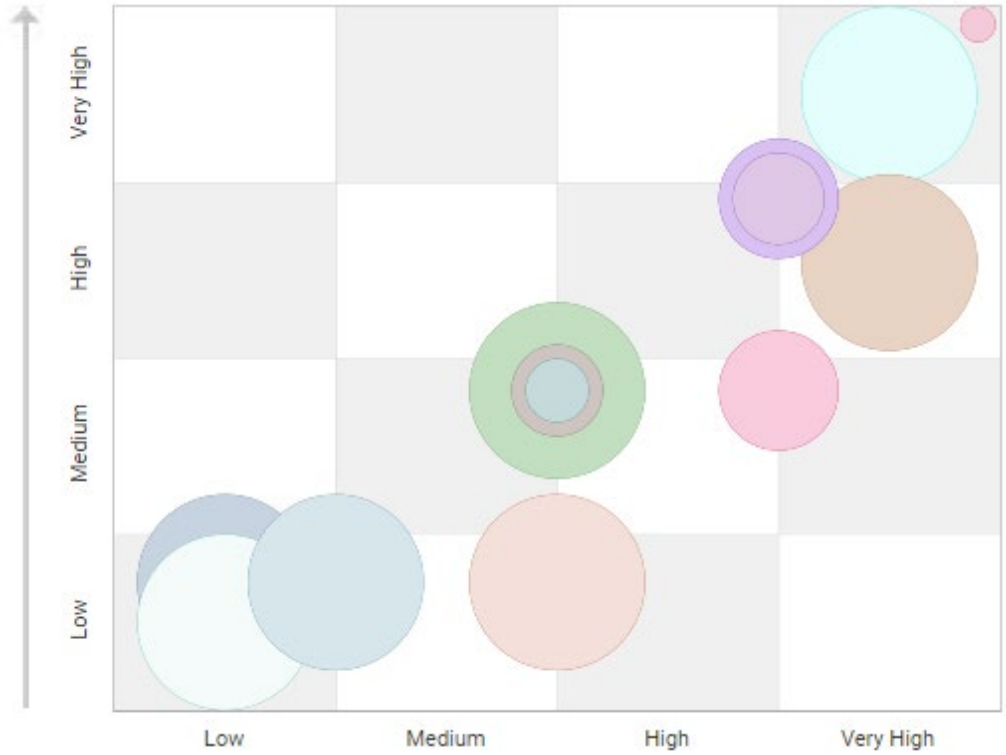
To define the basic content of the portfolio, the following attributes of the **Report Assistant** explorer's root node must be mapped to the column names in the dataset returned by the query defined in the ***Query*** element of the portfolio report described in the section Defining the Query the Portfolio is Based On:

Attribute	Description
X Value Column	Defines the name of the dataset column that contains the information about the placement of the object bubble on the X-axis.
Y Value Column	Defines the name of the dataset column that contains the information about the placement of the object bubble on the Y-axis.
Power Value Column	Defines the name of the dataset column that contains the values relevant for sizing of the object bubbles.
Image Column	Defines the name of the dataset column that contains the unique object c lass property used to identify the object.
Tooltip Column	Defines the name of the dataset column that contains the information to be displayed in a tooltip when the customer moves the cursor over the object bubble.

Defining the X-Axis And Y-Axis of the Portfolio Report

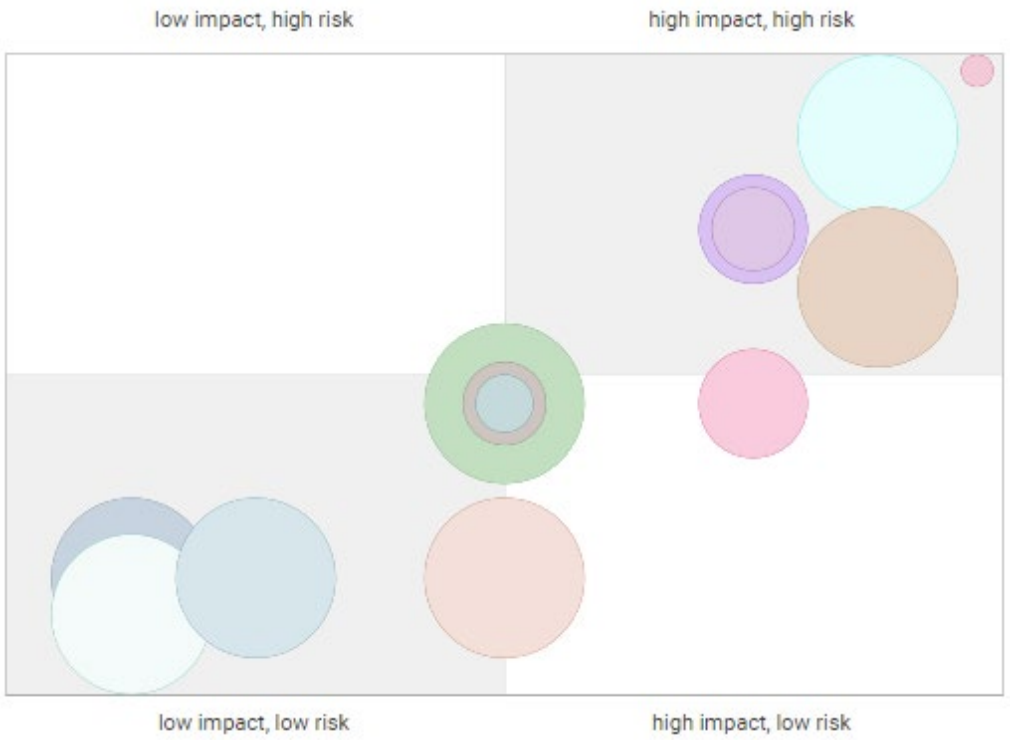
Portfolios can either have a quadrant design with a title for each quadrant or display an X-axis and Y-axis scale.

By default, the portfolio is displayed with an X-axis and Y-axis scale with four identically sized segments in each axis. The axis segments are by default labeled **Low**, **Medium**, **High** and **Very High**:



The following customization options are available for the axes of the portfolio:

- The number of segments on an axis and the titles displayed for the segments are configurable
- A caption can optionally be displayed for the axes.
- The minimum and maximum value can be defined for the axes. By default, the axes scope is defined by the minimum value and maximum value required to display the objects in the portfolio.
- The portfolio can alternatively be displayed in a quadrant layout with a title for each quadrant:



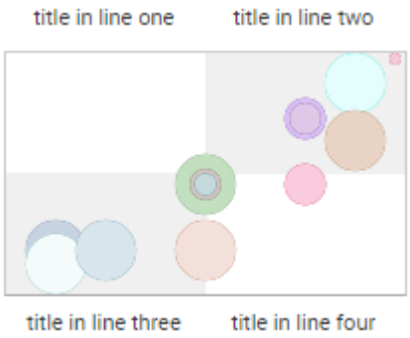
The configurations are described in the following sections:

- Defining a Quadrant Layout
- Defining Axes for a Standard Layout Portfolio
- Defining the Range of the Portfolio X-Axis and Y-Axis

Defining a Quadrant Layout

For the configuration of a quadrant layout, set the following attributes in the root node of the report assistant:

Attribute	Description
Quadrant Layout	Select `True`.
Quadrants	Click the **Browse** button in the attribute field and define each quadrant title in a separate line in the editor. The following graphic shows which quadrant title is defined by which line from top to bottom:

Attribute	Description
	

Defining Axes for a Standard Layout Portfolio

For the x-axis and y-axis layout of the portfolio, both the number of segments for an axis as well as the axis segment titles are configurable. In addition, a caption can be added for the axes.

Both axes captions and segmentations can be either defined statically for the report or dynamically via the query the report is based on. If both configurations are available, the dynamic design of the axes supersedes the static design.

Dynamic change of the X-axis and Y-axis design is For example, useful to define a flexible portfolio report with filters that allow the user to decide which evaluations to display. You can configured the caption and segment title of the axes in the portfolio to be adapted to the selected indicators.

To change the layout of the axes statically, set the following attributes of the Y-Axis and X-Axis elements in the **Report Assistant**:

Attribute	Description
X-Axis element	
Caption	Enter the caption to be displayed for the Y-axis.
Segment Names	Click the Browse button in the attribute field and define each segment title in a separate line in the editor. The number of segment titles defined equals the number of segments on the X-axis. Please note that there must not be a line break after the last specification. An empty row at the end of the definition will create an empty segment on the X-axis.

Attribute	Description
Y-Axis element	
Caption	Enter the caption to be displayed for the Y-axis.
Segment Names	Click the Browse button in the attribute field and define each segment title in a separate line in the editor. The number of segment titles defined equals the number of segments on the X-axis. Please note that there must not be a line break after the last specification. An empty row at the end of the definition will create an empty segment on the X-axis.

To change the layout of the axes dynamically, the data required to define the axes must be provided via the dataset returned by the query defined in the ***Query*** element of the portfolio report described in the section Defining the Query the Portfolio is Based On. The dataset column names must then be defined in the attributes of the root node of the Report Assistant. The following table lists the attributes that must be set and the required data to be returned in the query:

Attribute	Description
X-Caption Column	Enter the name of the dataset column returning the string to be displayed as X-axis caption. The first value in the defined dataset column that is not NULL will be displayed as X-axis caption.
Y-Caption Column	Enter the name of the dataset column returning the string to be displayed as X-axis caption. The first value in the defined dataset column that is not NULL will be displayed as X-axis caption.
X-Segment Names Column	Enter the name of the dataset column returning the segment title strings. The number of segment titles defined equals the number of segments on the X-axis. The segment titles must be separated with CHAR (10). For example, to define an axis with three segments having the titles **Small**, **Medium** and **Large**, the column can be defined as follows in a native SQL query: `'Small'+CHAR(10)+'Medium'+CHAR(10)+'Large' AS 'XSegments'`
Y-Segment Names Column	Enter the name of the dataset column returning the segment title strings. The number of segment titles defined equals the number of segments on the Y-axis. The segment titles must be separated with CHAR (10). For example, to define an axis with three segments having the titles **Small**, **Medium** and **Large**, the column can be defined as follows in a native SQL query: `'Small'+CHAR(10)+'Medium'+CHAR(10)+'Large' AS 'YSegments'`

Defining the Range of the Portfolio X-Axis and Y-Axis

For both quadrant and standard portfolio axis layout, the numerical range of the X-axis and the Y-axis can be defined.

By default, the range covered by the portfolio is defined by the highest and lowest value for an object in the portfolio. When the set of objects displayed is changed, the range covered by the axes also changes. The covered range is then segmented into equally sized segments. That means that very high can mark a range between For example, 4 and 5 but in another portfolio a range between 4.5 and 5.3. Defining a minimum and maximum value can be used For example, for the following reasons:

- With the flexible scaling of the portfolio axes, the objects with the minimum and maximum X-value and Y-value respectively will be displayed directly on the border of the portfolio diagram. The axes scope can be defined to allow spacing between all objects displayed in the portfolio and the portfolio border.
- With the flexible scaling of the portfolio axes, the positioning of object in the portfolio depends on the relative position with respect to other objects. For example, for an X-axis indicator that may have values from 1 to 5, an object that has an indicator setting of 3 will be displayed in the middle of the portfolio if the indicator settings for the other objects cover the complete range of 1 to 5, but will be displayed on the right border of the portfolio if compared to objects with indicator settings from 1 to 3. The axes scope can be defined to allow a better comparison of the display of objects in the portfolio for different filter settings.
- If the portfolio is displayed in standard layout with axes segments defined, the definition of the minimum and maximum value can be used to map the segment titles to defined numerical values.

If objects to be displayed in the portfolio have a value for an axis that lies outside of a defined range, the object is nevertheless displayed on the border of the diagram.

To define a range for the portfolio axis, set the following attributes in the X-Axis and Y-Axis elements in the **Report Assistant**:

Attribute	Description
X-Axis element	
Min Value	Enter a numerical value to start the X-axis with the defined value.
Max Value	Enter a numerical value to end the X-axis with the defined value.
Y-Axis element	
Min Value	Enter a numerical value to start the X-axis with the defined value.
Max Value	Enter a numerical value to end the X-axis with the defined value.

Defining the Graphic Representation of Objects in the Portfolio Area

While standard Alfabet portfolio report page views display the objects in the portfolio as differently colored bubbles only, configured portfolio reports offer a broader range of object representations.

Objects can be displayed in the configured portfolio report as a variety of geometric shapes or as icon. The shape or icon representing the objects can be identical for all objects or defined per object via the query defined in the ***Query*** element of the configured portfolio report.

If objects are represented by geometric shapes, the color of objects can also be defined via the query defined in the ***Query*** element of the configured portfolio report. The color can Therefore, represent an additional aspect of evaluation and a legend is available for the object shape and coloring.

By default, portfolio reports will display the objects in the report as bubbles in the same way as standard Alfabet portfolio report page views do.

The following configuration options are available:

- Representing Objects as Geometric Shapes
- Representing Objects As Icons
- Defining the Coloring of Shapes in the Portfolio Report

Representing Objects as Geometric Shapes

Configured portfolio reports can represent all objects with the same geometric shape or with different geometric shapes returned via the query of the portfolio report.

To configure use of the same geometric shape for all objects, expand the **Shape Source** attribute in the root node of the **Report Assistant** and set the following attributes in the order defined in the list:

Attribute	Description
Shape Type	Select `GeometricShape`.
Shape Source	Select `ReportDefinition`. This is the default value.
Fixed Shape	Select the shape that shall be used to display the objects in the portfolio. The following shapes are available: • `Rectangle` () • `Triangle` () • `Circle` ()

Attribute	Description
	• `Diamond` () • `FlippedTriangle` ()

To assign a shape to each object individually, the name of the geometric shapes must be provided via the dataset returned by the query defined in the ***Query*** element of the portfolio report described in the section Defining the Query the Portfolio is Based On. The use of the query result dataset for evaluation of object shapes must then be defined in the root node of the **Report Assistant**. Expand the **Shape Source** attribute and set the following attributes in the order defined in the table:

Attribute	Description
Shape Type	Select `GeometricShape`.
Shape Source	Select `Query`.
Shape Name Column	Enter the name of the column in the query dataset that returns the name of the shape for each object. The query can return any of the following shape names: • `Rectangle` () • `Triangle` () • `Circle` () • `Diamond` () • `FlippedTriangle` () If no or any other value is returned in the column, `Circle` is used as default.

Representing Objects As Icons

Configured portfolio reports can represent all objects as icons.

Representing objects as icons limits the availability of other features in the portfolio:

- Sizing of icons based on power axis values is not possible.
- Coloring cannot be defined for the object representations.

Icons can be represented by the icons defined for the object, or by any icon that has been uploaded to the 22x22 icon gallery. If objects defined for the object shall be used, the icon defined for the object in the standard object class property Icon is used to represent the object in the portfolio, or, if no icon is defined for the object, the icon defined for the object class or object class stereotype in the class settings is used.

For more information about uploading icons to the icon gallery, see Adding and Maintaining Icons for the Alfabet Interface.

For more information about assigning icons to an object class or object class stereotype via the class settings, see Specifying the Icons Implemented for Object Classes and Object Class Stereotypes.

To configure use of the object icon, expand the **Shape Source** attribute in the root node of the **Report Assistant** and set the following attributes in the order defined in the list:

Attribute	Description
Shape Type	Select `Icon`.
Shape Source	Select `Object`. This is the default value.

Please note that in the preview of the report that opens via the Review Report option in the context menu of the report the icons for the object class stereotype class settings are ignored and all objects are displayed either with the object icon or, if not defined, with the icon for the class settings for the object class without stereotype definition.

To use icons from the 22x22 icon gallery for display of objects in the portfolio, the name of the icon must be provided via the dataset returned by the query defined in the ***Query*** element of the portfolio report described in the section Defining the Query the Portfolio is Based On. The use of the query result dataset for evaluation of object shapes must then be defined in the root node of the **Report Assistant**. Expand the **Shape Source** attribute and set the following attributes in the order defined in the table:

Attribute	Description
Shape Type	Select `Icon`.
Icon Source	Select `Query`.
Icon Name Column	Enter the name of the column in the query dataset that returns the name of the icon for each object. If the icon name cannot be found in the icon gallery, there is no fallback and a missing icon symbol ( ) is displayed instead for the object in the portfolio.

Defining the Coloring of Shapes in the Portfolio Report

If objects in a portfolio are displayed as geometric shapes, object shapes are by default displayed in different colors. Icons cannot be colored.

The color used for an object is identical in all portfolios. The coloring of single objects can be changed via the following mechanisms:

- If a color is defined for an object via the private object class property **Color** of the object class, this color will be used to color the object in portfolio reports. The Color property is not part of standard custom editors. To set the coloring for objects of a defined class, a color selector must be defined in a custom editor for the respective object class. For information about defining custom editors, see .
- The color used for objects in the chart can be changed by the user viewing the portfolio if the **Colors**  button of the report is enabled or if a color selector is defined in a custom editor for the respective object class. The **Colors**  button is activated or deactivated in the server alias of the Alfabet Web Application. For information about the activation of the **Colors**  button, see *Allowing Users to Change the Color Assigned to Objects in Business Graphics* in the reference manual *System Administration*. Please note that the color defined for an object via the **Colors**  button changes the color assigned to the object via the private object class property **Color** and will Therefore, be applied to the object in all other reports that use object colors for all Alfabet users.

Instead of using the object colors, individual coloring of shapes can be defined for the portfolio report.

Please note that the legend for the portfolio displays one entry for each object returning the shape and color followed by the value of the object class property or object class properties used to identify the object. If you define an identical shape and color combination for different objects in the configured report, the legend will list these objects all with an identical shape and color in front.

To define the basic content of the portfolio, the following attributes of the **Report Assistant** explorer's root node must be mapped to the column names in the dataset returned by the query defined in the ***Query*** element of the portfolio report described in the section Defining the Query the Portfolio is Based On:

Attribute	Description
Color Column	Enter the name of the column that returns the HTML compliant color definition (For example, `#000000` for black).

Defining a Tooltip for Objects in the Portfolio Area

A tooltip is displayed if the user moves the mouse over an object in the portfolio report.

If no tooltip is defined via the settings in the **Report Assistant**, the information used to identify the object will be displayed as tooltip. If a power axis is defined, the numeric value of the power axis in parentheses will be added to the default tooltip after the object information.

Tooltip texts must be returned by the query defined in the ***Query*** element of the portfolio report described in the section Defining the Query the Portfolio is Based On. The texts are then mapped to attributes of the **Report Assistant** explorer's root node.

Tooltips can have the following elements:

- Any string defined in the query of the portfolio report
- Information about the value of the power axis. This can be either the numeric or semantic value of the indicator defined for the power axis.
- A caption for the power axis to inform the user about the meaning of the power axis value.

If all elements of the tooltip are defined, the syntax of the tooltip is:

tooltip text (power axis caption = power axis value)

To define the tooltips for the portfolio, the following attributes of the **Report Assistant** explorer's root node can optionally be set:

Attribute	Description
Tooltip Column	Defines the name of the dataset column that contains the information to be displayed in the tooltip. If not defined, the information returned in the dataset column mapped to the **Image Column** attribute is displayed as tooltip.
Add Power to Tooltip	If the **Power Value Column** attribute is defined and the **Add Power to Tooltip** attribute is set to `True`, the information about the power axis value for the current object will be added to the tooltip in parentheses. By default, this attribute is set to `True`.
Power Caption Column	Defines the name of the dataset column that returns the caption of the power axis. If the dataset column returns multiple different values, the first value that is not `NULL` returned in the dataset will be displayed. The caption for the power axis can alternatively be defined statically with the attribute **Caption** of the **Power Axis** element of the **Report Assistant**. If neither the **Power Caption Column** attribute nor the **Caption** attribute of the **Power Axis** element are defined, no caption will be displayed for the power axis value. If both attributes are defined, the setting of the **Power Caption Column** attribute supersedes the setting of the static **Caption** attribute of the **Power Axis** element. The power axis caption will only be added to the tooltip if the **Add Power to Tooltip** attribute is set to `True`. If a legend is defined for the power axis, the caption is also displayed as legend title to the left of the power axis value definitions. For more information about the definition of the power axis legend, see Defining a Legend for the Power Axis of the Portfolio.
Power Semantic	Defines the name of the dataset column that returns the semantic value of the indicator defined for the power axis.

Attribute	Description
Value Column	If the **Power Semantic Value Column** attribute is not defined, and the **Add Power to Tooltip** attribute is set to `True`, the numeric value defined in the **Power Value Column** attribute will be displayed in the tooltip.
Power Axis element:	
Caption	Defines the caption of the power axis. The caption for the power axis can alternatively be defined dynamically with the attribute **Power Caption** of the root node of the **Report Assistant**. If neither the **Power Caption Column** attribute of the root node nor the **Caption** attribute of the **Power Axis** element are defined, no caption will be displayed for the power axis value. If both attributes are defined, the setting of the **Power Caption Column** attribute of the root node supersedes the setting of the static **Caption** attribute of the **Power Axis** element. The power axis caption will only be added to the tooltip if the **Add Power to Tooltip** attribute of the root node of the **Report Assistant** is set to `True`. If a legend is defined for the power axis, the caption is also displayed as legend title to the left of the power axis value definitions. For more information about the definition of the power axis legend, see Defining a Legend for the Power Axis of the Portfolio.

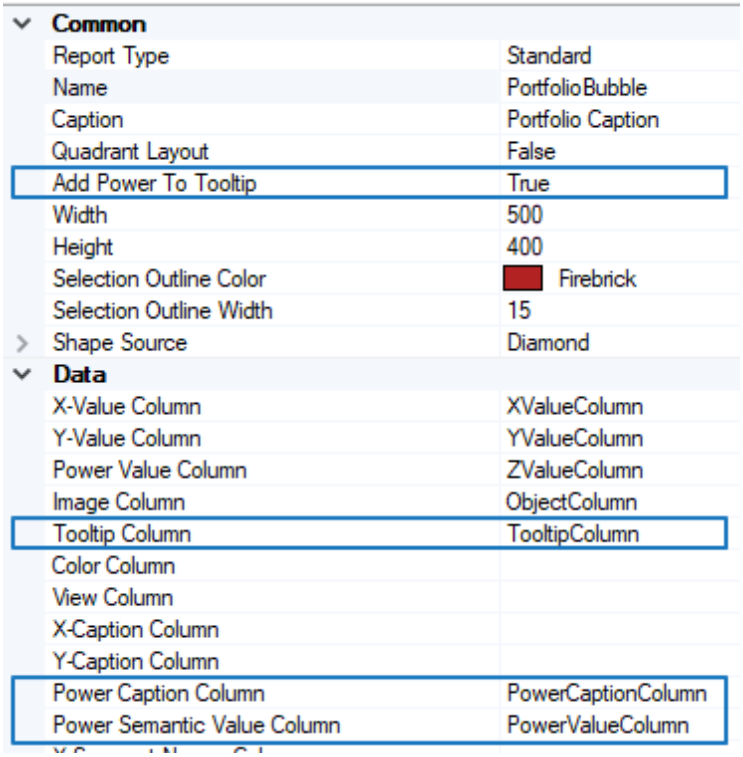
For example, the following query defines a portfolio report with a power axis definition for an application portfolio displaying applications in a selected application group. The tooltip shall display the name and version number of the object followed by the power axis value information that includes a string as title and the semantic value defined for the indicator. The tooltip information definition is highlighted with bold text:

```
SELECT DISTINCT app.REFSTR, app.REFSTR AS 'ObjectColumn',
ind.VALUE AS 'ZValueColumn',
ind2.VALUE AS 'XValueColumn',
ind3.VALUE AS 'YValueColumn',
app.NAME + ' ' + app.VERSION AS 'TooltipColumn',
ind.SEMANTICVALUE AS 'PowerValueColumn',
'Usability' AS 'PowerCaptionColumn',
FROM APPLICATION app
INNER JOIN INDICATOR ind ON ind.OBJECT = app.REFSTR
INNER JOIN INDICATOR ind2 ON ind2.OBJECT = app.REFSTR
LEFT JOIN INDICATOR ind3 ON ind3.OBJECT = app.REFSTR
INNER JOIN RELATIONS rel ON rel.TOREF = app.REFSTR
INNER JOIN APPLICATIONGROUP appg ON appg.REFSTR = rel.FROMREF
WHERE rel.PROPERTY = 'Applications'
```

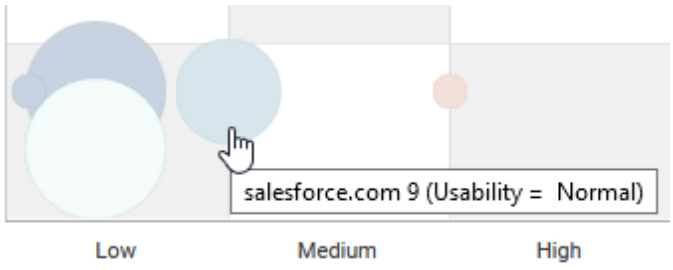


```
AND appg.REFSTR = @BASE
AND ind.NAME LIKE '%Usability'
AND ind2.NAME LIKE '%Customer Impact'
AND ind3.NAME LIKE '%Criticality (aggr.)'
```

In the report assistant, the values for the tooltip relevant parameters are defined to display all relevant information:

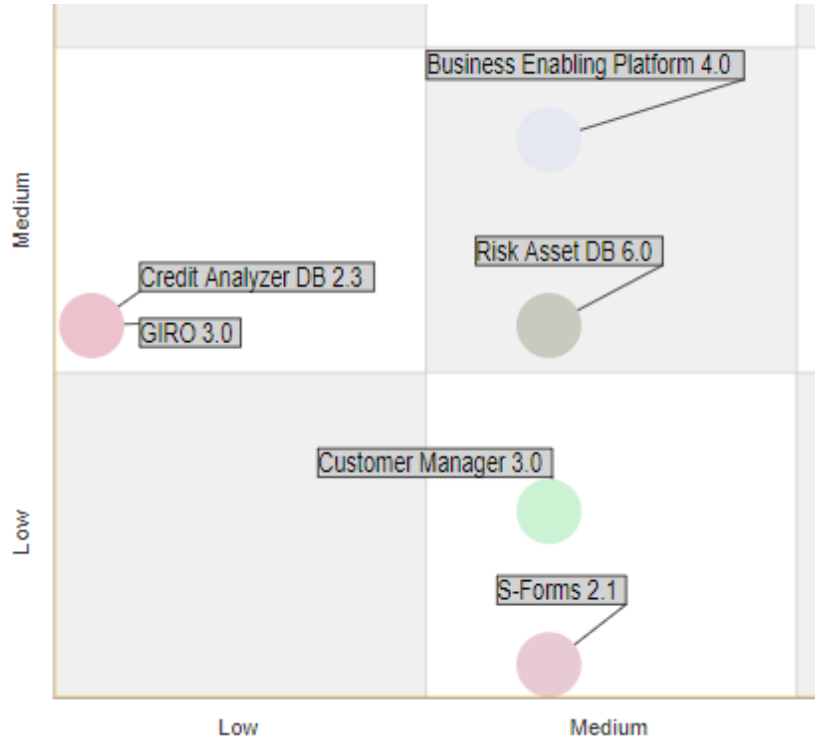


This configuration results in the following tooltip to be displayed:



Defining Labels for Objects in the Portfolio Area

Optionally, labels can be displayed in the portfolio.



Labels are only displayed if the label text is returned in the query of the portfolio report and the name of the column returning the text is mapped to the **Label Text** attribute in the expandable attribute **Label** in the root node of the report assistant.

Coloring and links from the label to the objects in the portfolio area can optionally be defined in the query and the child attributes of the expandable attribute **Label**:

Attribute	Description
Label Text	Defines the name of the dataset column that returns the label text. Labels are only displayed if this attribute is set.
Label Fore-color	Defines the name of the dataset column that returns the text color of the label in HTML compliant color code.
Label Background Color	Defines the name of the dataset column that returns the background color of the label in HTML compliant color code.
Label Border Color	Defines the name of the dataset column that returns the border color of the label in HTML compliant color code.
Label Connection Color	Defines the name of the dataset column that returns the color of the connecting line between the object and the label in HTML compliant color code. Connections will only be displayed if a connection color is defined.

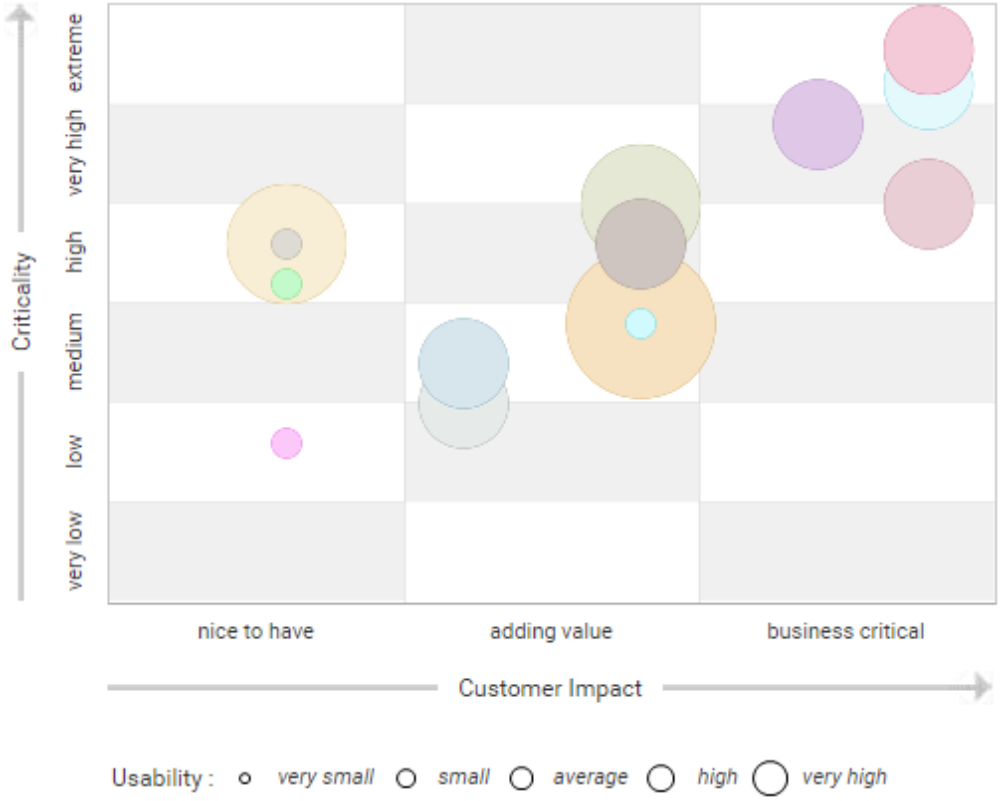
Attribute	Description
Label	Defines the relative height of the labels. the label height in pixel will be calculated for the available screen size and portfolio size and the relative height specification. Select one of the following from the drop-down list: • L = Large • M = Medium • S = Small

Defining a Legend for the Power Axis of the Portfolio

By default, a legend is available for the portfolio via the floating toolbar that lists all objects with the icon or shape and color combination used for the representation of the object. If background coloring is defined, the legend can also include the background coloring definition.

In addition to the legend that opens via the floating toolbar, a legend can optionally be added for the sizing defined by the power axis. The power axis legend will be displayed below the portfolio chart. It can distinguish between up to 5 different shape sizes. Shapes are displayed as bubbles only independent from the actual shapes used in the portfolio. The sizing is explained relative to each other and not in absolute sizes. The legend text displayed to explain the different sizing is configurable.

Optionally, a caption can be displayed in front of the size definitions.



The power axis caption and legend texts can be either defined statically for the report or dynamically via the query the report is based on. Dynamic definition may For example, be used to limit the number of entries to the number of different bubble sizes in the report. If both configurations are available, the dynamic design of the power axis legend supersedes the static design. It is possible to define the caption statically and the legend texts dynamically and vice versa.

To define a dynamic power axis legend, the data required to define the power axis must be provided via the dataset returned by the query defined in the ***Query*** element of the portfolio report described in the section Defining the Query the Portfolio is Based On. The dataset column names must then be defined in the attributes of the root node of the Report Assistant.

The power axis is defined with the following attributes of the root node and the **Power Axis** element in the **Report Assistant**:

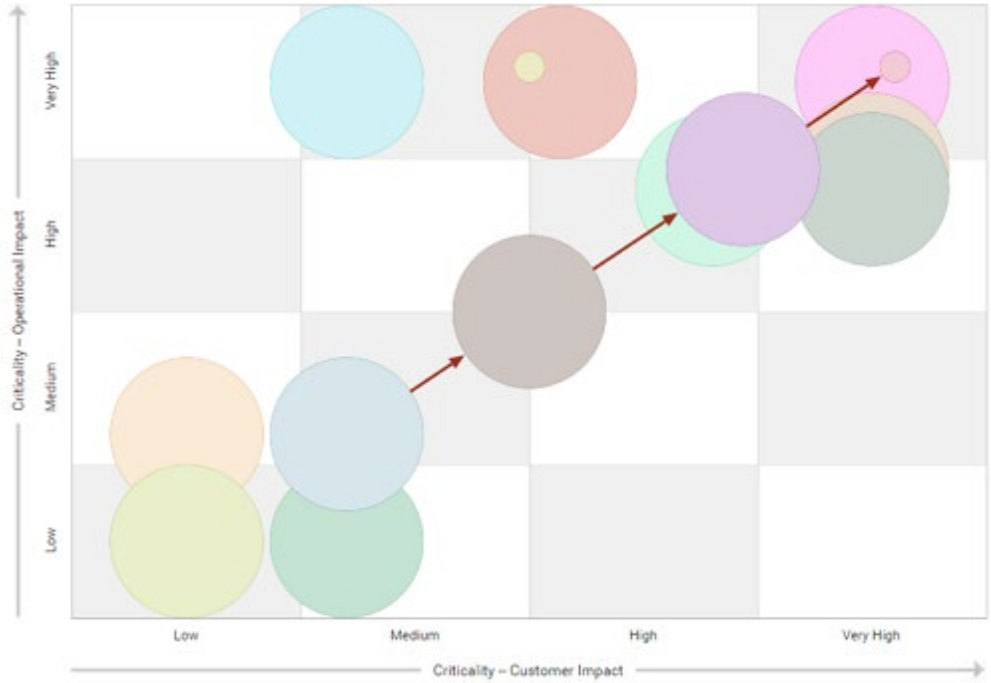
Attribute	Description
Root node	
Power Caption Column	Defines the name of the dataset column that returns the caption of the power axis displayed in front of the legend items. If the dataset column returns multiple different values, the first value that is not `NULL` returned in the dataset will be displayed. The caption for the power axis can alternatively be defined statically with the attribute **Caption** of the **Power Axis** element of the **Report Assistant**. If neither the **Power Caption Column** attribute nor the **Caption** attribute of the **Power Axis** element are defined, no caption will be displayed in front of the power axis legend. If both attributes are defined, the setting of the **Power Caption Column** attribute supersedes the setting of the static **Caption** attribute of the **Power Axis** element. If the attribute **Add Power to Tooltip** is set to `True` in the root node of the `Report Assistant`, the caption is also displayed in the tooltip that is visible if the user moves the cursor over an object in the portfolio report. For more information about the definition of the tooltip content, see Defining a Tooltip for Objects in the Portfolio Area.
Bubble Size Legend Column	Enter the name of the dataset column returning the power axis legend texts. The first result returned in the column that is not NULL will be used as legend text definition. The legend texts must be separated with CHAR(10). For example, to define an legend for three bubble sizes having the legend texts **Small**, **Medium** and **Large**, the column can be defined as follows in a native SQL query: `'Small'+CHAR(10)+'Medium'+CHAR(10)+'Large' AS 'PowerLegendTexts'` The legend text definition must start with the text for the smallest size and end with the text for the largest size. The legend definition is restricted to five different sizes. If more than five texts are defined, the legend will nevertheless display 5 different bubble sizes and will use texts from the beginning and the end of the definition as titles in the legend.

Attribute	Description
	If neither the **Bubble Size Legend** attribute nor the **Segment Names** attribute in the **Power Axis** element is defined, the power axis legend will not be displayed.
Power Axis element	
Caption	Optionally define the caption of the power axis legend. If defined, the caption will be displayed in front of the legend. The caption for the power axis can alternatively be defined dynamically with the attribute **Power Caption** of the root node of the **Report Assistant**. If neither the **Power Caption Column** attribute of the root node nor the **Caption** attribute of the **Power Axis** element are defined, no caption will be displayed for the power axis legend. If both attributes are defined, the setting of the **Power Caption Column** attribute of the root node supersedes the setting of the static **Caption** attribute of the **Power Axis** element. If the attribute **Add Power to Tooltip** is set to `True` in the root node of the `Report Assistant`, the caption is also displayed in the tooltip that is visible if the user moves the cursor over an object in the portfolio report. For more information about the definition of the tooltip content, see Defining a Tooltip for Objects in the Portfolio Area.
Segment Names	Click the Browse button in the attribute field and define up to five legend texts, each in a separate line in the editor. The text order from top to bottom is identical to the size order from small to big. If more than five texts are defined, the legend will nevertheless display 5 different bubble sizes and will use texts from the beginning and the end of the definition as titles in the legend. Please note that there must not be a line break after the last specification. An empty row at the end of the definition will create a bubble size specification without a text. Definition of the **Bubble Size Legend** attribute in the root node of the **Report Assistant** supersedes the definition in the **Segment Names** attribute. If neither the **Segment Names** attribute nor the **Bubble Size Legend** attribute in the root node of the **Report Assistant** is defined, the power axis legend will not be displayed.

Defining Connections between Objects Displayed in the Portfolio

Objects in a portfolio report can optionally be connected with lines indicating a configurable relationship.

In the following example, a portfolio report displays evaluation of applications in an application group. If a migration is defined to migrate from one application to the other, the migration is displayed as a red arrow:



To display connections between objects in the report, all relevant data for rendering the connections must be provided via th optional **Connections Query**. The **Connections Query** must at least return the `REFSTR` of the object the connection is coming from and the `REFSTR` of the object the connection is ending at. Both objects must be displayed in the portfolio.

The columns in the result dataset of the **Connections Query** must then be mapped to attributes in the root node of the report assistant that are available when expanding the attribute **Connection Query Columns** in the section **Data**.

In addition to the mandatory definition of the start object and end object of the connection, the query can return styling parameters, legend texts and a link target to activate navigation from the connection to an Alfabet view. The columns must be mapped to the following sub-attributes of the attribute **Connection Query Columns**:

Attribute	Column must return:	Mandatory
Start Ref Column	The `REFSTR` of the object the connection is starting at.	Yes
End Ref Column	The `REFSTR` of the object represented with by the item the connection is ending at.	Yes
Line Style Column	The style of the connecting line. Allowed values are `Solid`, `Dash`, `DashDot`, `DashDotDot` and `Dot`.	No, by default connections are displayed as solid lines (`Solid`).

Attribute	Column must return:	Mandatory
Line Weight Column	The width of the connecting line as integer.	No, the default value is 1.
Line color Column	The color of the connecting line as HTML compliant color code.	No, by default connections are displayed as black lines.
Start Arrow Style Column	The style of the start point of the connection. Allowed values are `None`, `Angle`, `Triangle`, `Square`, `Circle` and `Rhomb`. If you would like to activate navigation or the display of a tooltip and the line drawn for the connection is very thin, definition of a start arrow style will make it easier for the user to point to the connection.	No, by default connections are displayed with no extra decoration at the start point (`None`).
End Arrow Style Column	The style of the end point of the connection. Allowed values are `None`, `Angle`, `Triangle`, `Square`, `Circle` and `Rhomb`. If you would like to activate navigation or the display of a tooltip and the line drawn for the connection is very thin, definition of an end arrow style will make it easier for the user to point to the connection.	No, by default connections are displayed with no extra decoration at the end point (`None`).
Tooltip Column	A string that is displayed as tooltip when the user moves the cursor over the start or end point of the connection.	No, by default no tooltips are displayed.
Link Ref Column	The `REFSTR` of an object. If this column is specified, navigation from the connection is activated. When the user clicks and holds the connection, a preview of the object with the `REFSTR` returned in the **Link Ref Column** opens. The user can navigate to the object profile or object cockpit of the object by clicking the **Show Details** button in the preview. If **View Column** is defined, the defined navigation target opens instead of the object profile or object cockpit of the object.	No, if not set, navigation is deactivated.
View Column	Definition of a link target that shall open if a user double-clicks on a connection or clicks and holds the connection. The format is `View=`*`ViewType`*`:`*`ViewName`* The `ViewType` can be one of the following: • ***Report*** for a configured report • ***GraphicView*** for a standard Alfabet view	No, if not set, but **Link Ref Column** is set, navigation is opening the object profile or object cockpit of the object defined with **Link Ref Column**.

Attribute	Column must return:	Mandatory
	• ***ObjectView*** for an object profile `ViewName` is the name of the view. For detailed information about the required configuration to activate navigation, see Defining Navigation from the Report to Alfabet Views.	
Legend Group Column	A string that shall be displayed as heading for the legend entry defined with **Legend Text Column** in the same row of the dataset. All legend entries that have the same **Legend Group Column** are grouped and displayed under the heading defined with **Legend Group Column**. A legend is only displayed for connections if both **Legend Group Column** and **Legend Text Column** are defined.	No, if the attribute is not defined, the legend does not display entries for connection styles.
Legend Text Column	A string that shall be displayed next to the style of the connection in the legend of the report. A legend is only displayed for connections if both **Legend Group Column** and **Legend Text Column** are defined.	No, if the attribute is not defined, the legend does not display entries for connection styles.

Defining Background Coloring for the Portfolio

By default, the background of a portfolio report is divided into black and light grey areas in a checkered pattern. Optionally, the background of the portfolio can be colored in three different design types.

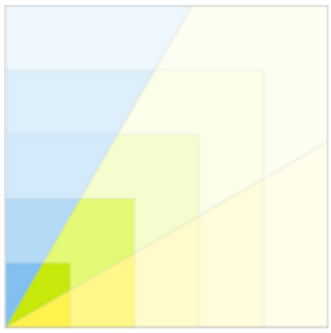
This feature is not available for portfolio reports displaying a quadrant layout. For portfolio reports with an X-axis and Y-axis definition, the portfolio background can be colored with user-defined colors. Color zones are applied diagonally applying to the portfolio background either from left to right or from right to left. The following design types are available:

- ***Wave***: The colors are displayed in circles of different diameters with the neutral position of the axes of the portfolio as the center of the circle with the portfolio displaying a quarter of each colored circle. The circles can have segments of different color. The center of the design can start on the lower left or upper right corner of the portfolio chart area.

Left to Right

Right to Left

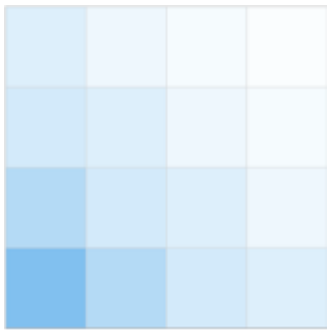
- ***NestedSquare***: The portfolio background displays nested squares. The solution designer can specify the number of squares and their relative size. The center of the design can start on the lower left or upper right corner of the portfolio chart area.



Left to Right

Right to Left

- ***CheckerBoard***: The portfolio background is a raster of squares with squares in a diagonal line colored in the same color. The solution designer can specify the number of colors to be used and the relative proportion of their sizes.



If a background design is defined, a legend is available explaining the meaning of the coloring in the configured report. The legend explains the coloring for each series in a `NestedSquare` and `Wave` design separately.

In the root node of the Report Assistant, the following attributes must be set to defined the background layout of the portfolio chart area:

Attribute	Description
Report Type	Select `DesignedBackground` from the drop-down list.
Background Design	Select the design type from the drop-down list. The design type can be `Wave`, `Nested-Square`, or `CheckerBoard`. The different design types are explained above in this section.
Default Background Color	Select the color that shall be displayed in all areas of the portfolio chart that are not colored by the background design definition.
Background Orientation	This attribute is set to `Left2Right` by default and the center of the design is the lower left of the portfolio chart. For `NestedSquare` and `Wave` design, you can optionally set this attribute to `Right2Left`. The center of the design will then be in the upper right corner of the portfolio chart.
Background Info	Click the **Browse** button in the attribute field to open the editor for the definition of the cell coloring. The way colors are defined is explained below.

The **Background Info** attribute displays a table for the definition of the cell coloring. Definition requires the following steps:

1. Defining of the colors to be used for the background

Color are defined in the **Series 1** - **Series 5** columns of the editor. Colors can be defined by entering the HTML color code into the field or by picking a color with the color picker on the right of the series column.

The orders of colors in the portfolio chart from left to right is identical to the order of colors within a series definition from top to bottom. This is independent from the setting of the **Background Orientation** attribute.

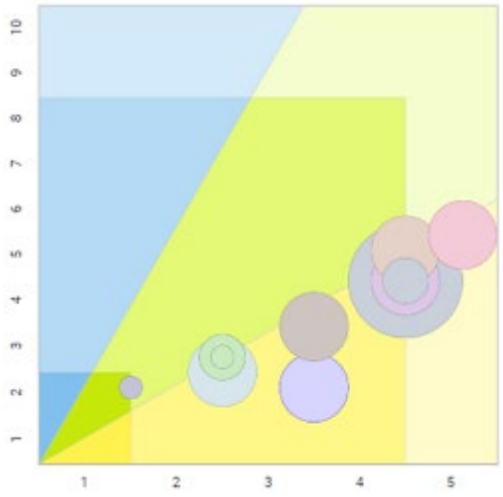
For a `CheckerBoard` design, colors are defined in the `Series 1` column only. For `Wave` and `NestedSquare` design, up to five color series can be defined. The number of color series defines the number of diagonal segments with different coloring. The order of colors is identical to the order of series. For `Left2Right` orientation, the series are displayed from left to right, while for `Right2Left` orientation, they are displayed from right to left.

It is recommended to read the part of about sizing of colored segments prior to defining the individual colors to ensure that enough colors are defined in the correct order for the intended design.

2. Defining the size of colored segments

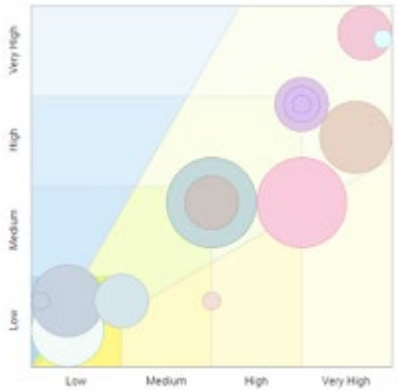
The size of colored segments is defined in the **Value** column based on the values of the X-axis of the portfolio report. The color distribution along the X-axis is repeated along the Y-axis with the same distribution pattern irrespective of the underlying values on the Y-axis.

The following portfolio has an x-axis with a range from 0 to 5 and a y-axis range from 0 to 10. Coloring is identical for both axes of the report. While the coloring is change on the x-axis at the end of segment 1 and 4, this corresponds on the y-axis to a color change at the end of segment 2 and 8:

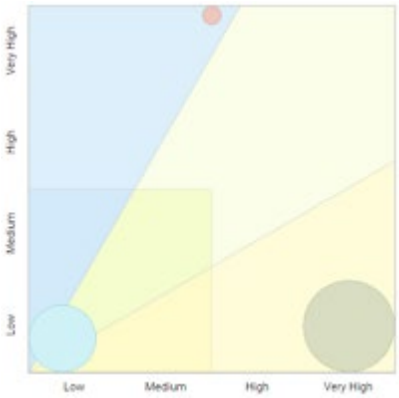


Coloring is based on the numerical values of the indicator of the X-axis definition. It is recommended to define minimum and maximum value for the X-axis as described in the section Defining the Range of the Portfolio X-Axis and Y-Axis. If no minimum and maximum value is defined for the X-axis, the scope of the x-axis depends on the lowest and highest X-axis indicator value returned for an object displayed in the portfolio. This might lead to different distribution of coloring dependent on the current scope of the X-axis.

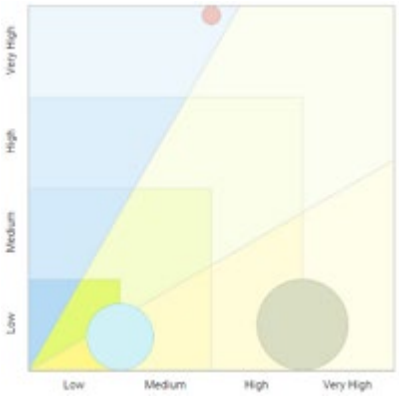
For example, the following configured report is defined for the evaluation of applications in an application group. The indicator on the X-axis can have values from 1 to 5. No minimum and maximum value are defined for the X-axis. A `NestedSquare` design is selected for the background of the portfolio chart. The background color is defined to change at the positions 1, 2, 3, 4 and 5 . If the indicator values for the applications in the group covers all available values, the X-axis will display the range of 1-5 and the portfolio will show all colors:



If the application group only contains applications with an X-indicator set to 2 and 4, the x-axis will display the range of 2 - 4 and only part of the colors are visible:



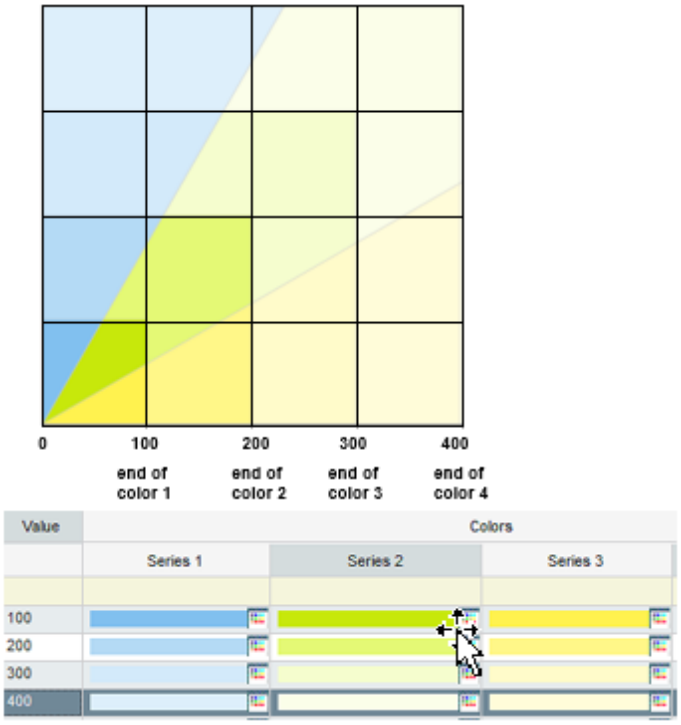
This is not necessarily wrong and might be the designed behavior. Setting of a minimum and maximum value for the x-axis will change the portfolio design to show the complete range of colors independent from the location of elements in the portfolio:



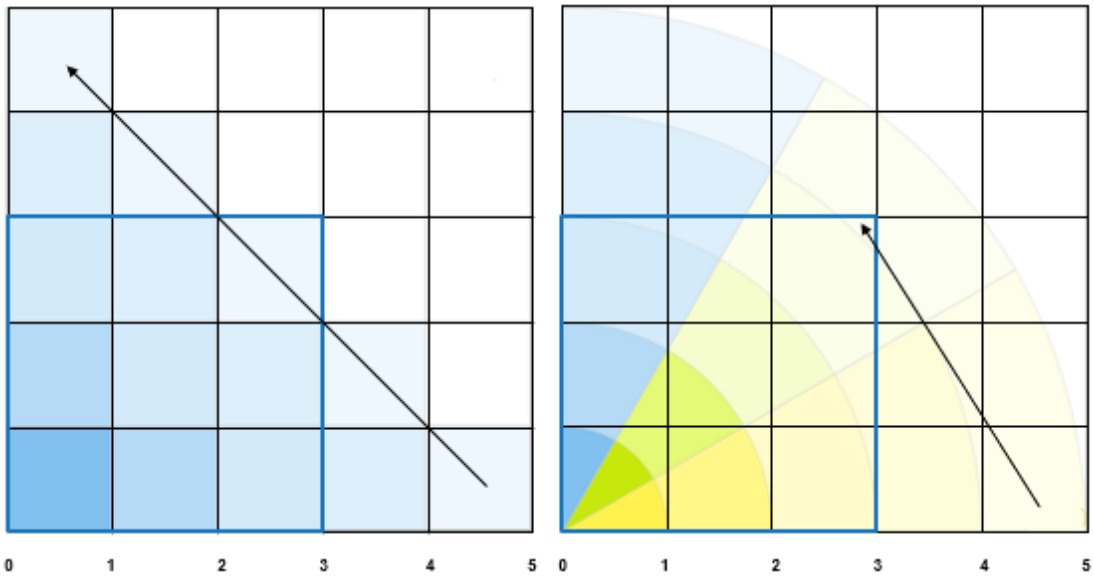
To define the size of a colored area, enter the X-axis end value for the colored block. The way the blocks are defined depends on the definition of the **Background Orientation** attribute of the portfolio chart:

- **Defining color positions for a Left2Right design**

Left centered design starts at the lower left corner of the portfolio chart with the first row for each series. The define the size of the color block, enter the value on the x-axis column where the coloring should end in the **Value** column for the respective color row in the editor. For example, to define a coloring for an X-axis with a range from 0 - 400 with each of the four default axis segments colored differently, the Value elements of the first four colors in a series must be 100, 200, 300 and 400, starting from top to bottom:

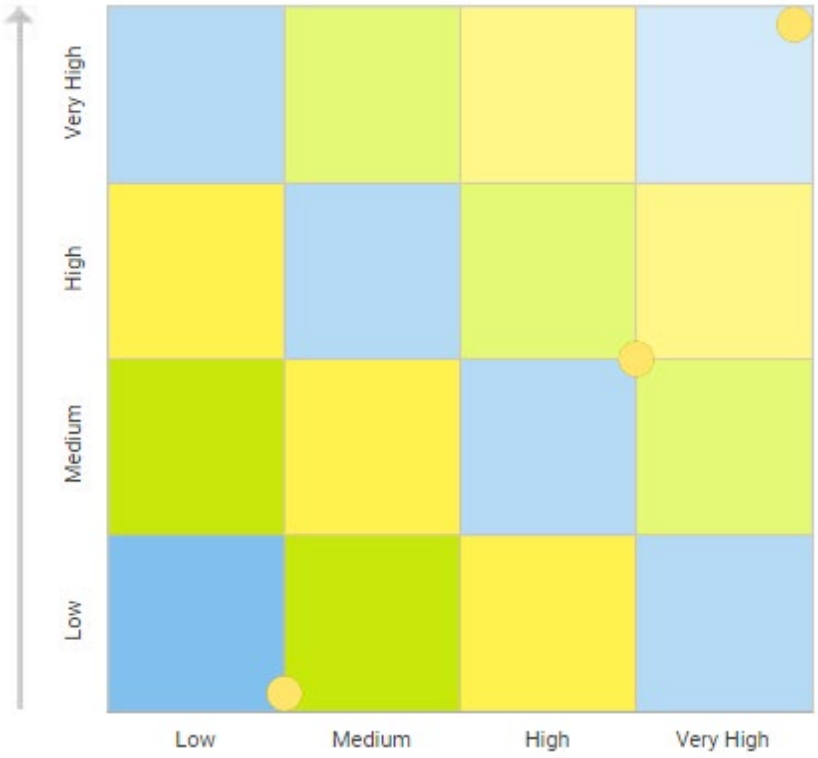


Please note that for coloring of the complete area of a `CheckerBoard` and `Wave` design, you must define additional colors for segments that are starting after the maximum of the x-axis. This can be best demonstrated by displaying the design of the actual portfolio chart as a part of a wider screen filled with the designed colors. The following graphics show a portfolio chart that has an X-axis range from 0 - 3 in a screen with a range from 0 - 5. As you can see from the graphic below, the colors defined for the segments ending with the virtual end position 4 and 5 are still part of the design of a `CheckerBoard` or `Wave` design:

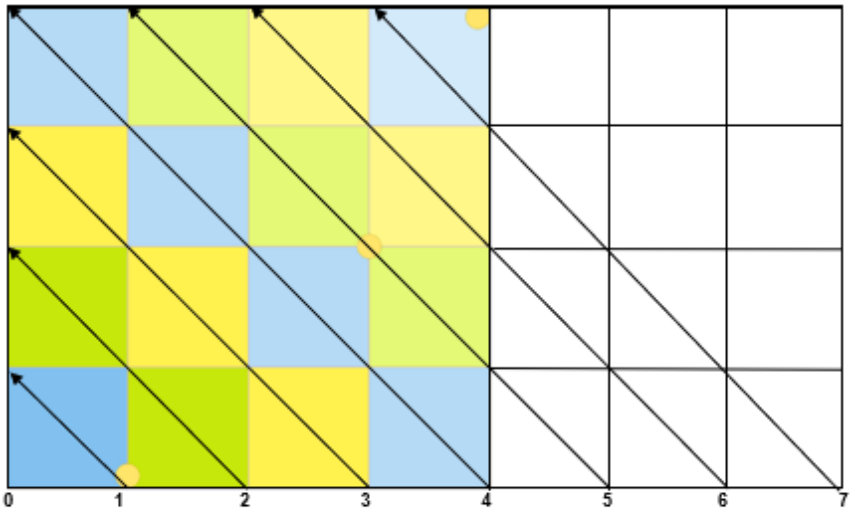


The width of the elements with a virtual end point is irrelevant for the design, because the size of the areas in the portfolio is only defined by the colors with visible end point definitions. The **Value** defined for virtual end points of color definitions must be higher than the maximum value of the x-axis.

For example, the following report is displaying a `CheckerBoard` background design:

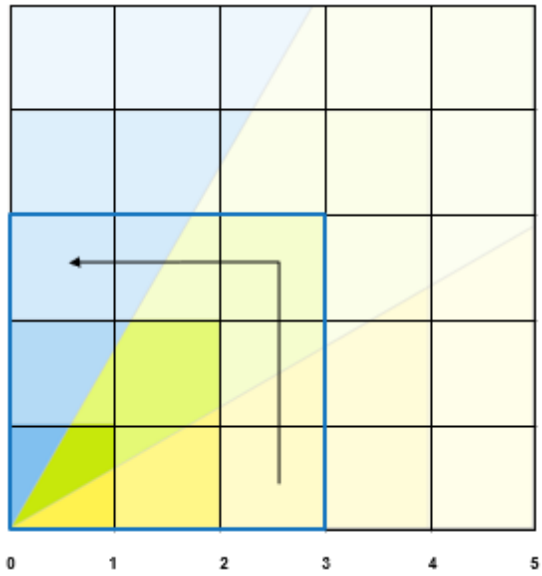


The x axis of the report has a range from zero to 5. 6 different colors are displayed in the report design. The color definition must Therefore, include 7 colors. The definition of where the coloring ends on the x-axis covers the range from 1 as end of the first color to 7 as end of the last color directly on the x-axis. The values from 5 to 7 are on the in-visible prolongation of the x-axis:



		Series 1
Series Caption		
First color defined (top of series)	1	
Second color defined	2	
Third color defined	3	
Fourth color defined	4	
Fifth color defined	5	
sixth color defined	6	
Seventh color defined	7	

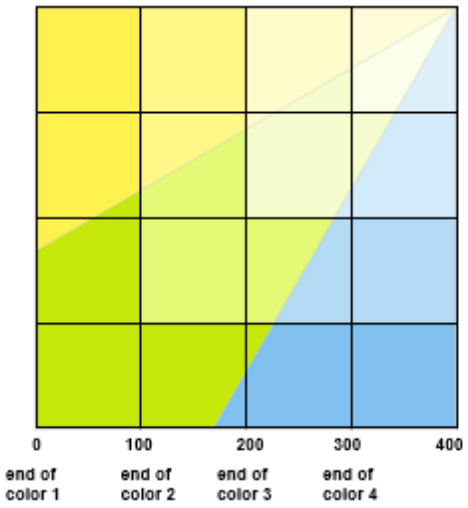
For `NestedSquare` design, it is not required to defined coloring for virtual segments. The last color displayed on the visible x-axis is the last color that is used in the report:



- **Defining color positions in a Right2Left design**

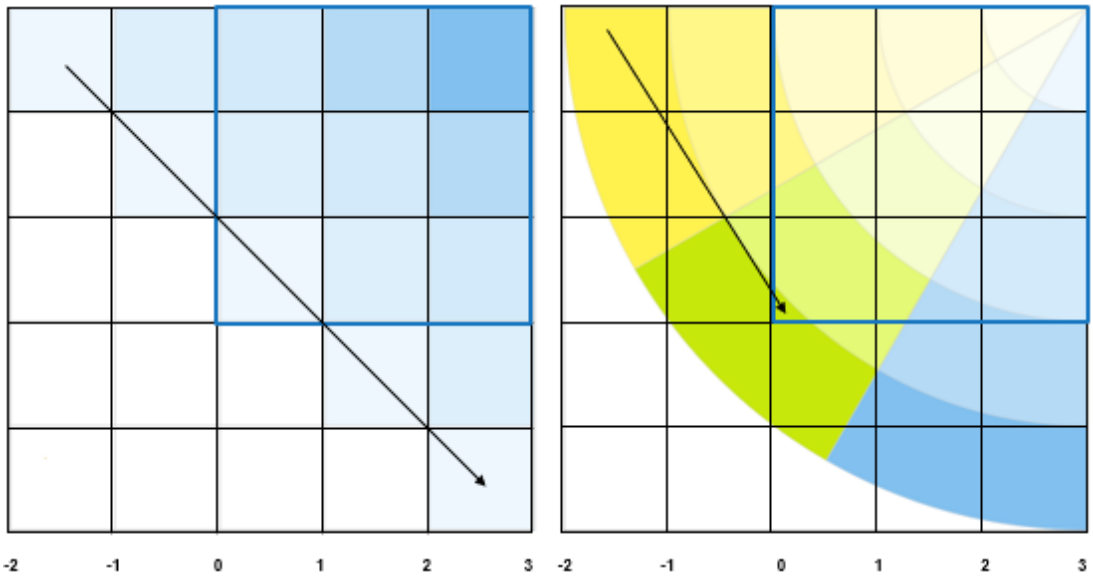
Right centered design starts at the upper right corner of the portfolio chart with the last row for each series. The define the size of the color block, enter the value on the x-axis column where the coloring should end, beginning at the start point of the design, in the **Value** column for the respective color row in the editor. For example, to define a coloring for an X-axis with a range from 0 - 400 with each of the four default

axis segments colored differently, the **Value** settings of the first four colors in a series must be 0, 100, 200 and 300, starting from top to bottom:



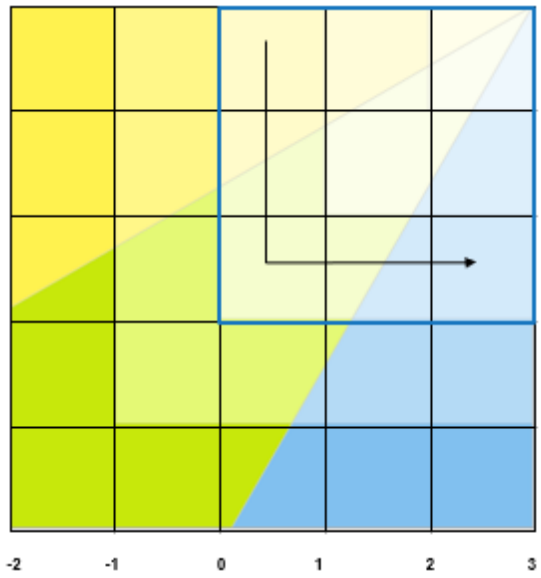
Value	Colors		
	Series 1	Series 2	Series 3
0			
100			
200			
300			
400			

Please note that for coloring of the complete area of a `CheckerBoard` and `Wave` design, you must define additional colors for segments that are having an end position before the minimum of the x-axis. This can be best demonstrated by displaying the design of the actual portfolio chart as a part of a wider screen filled with the designed colors. The following graphics show a portfolio chart that has an X-axis range from 0 - 3 in a screen with a range from -2 to 3. As you can see from the graphic below, the colors defined for the segments ending with the virtual end position -1 and -2 are still part of the design of a `CheckerBoard` or `Wave` design:



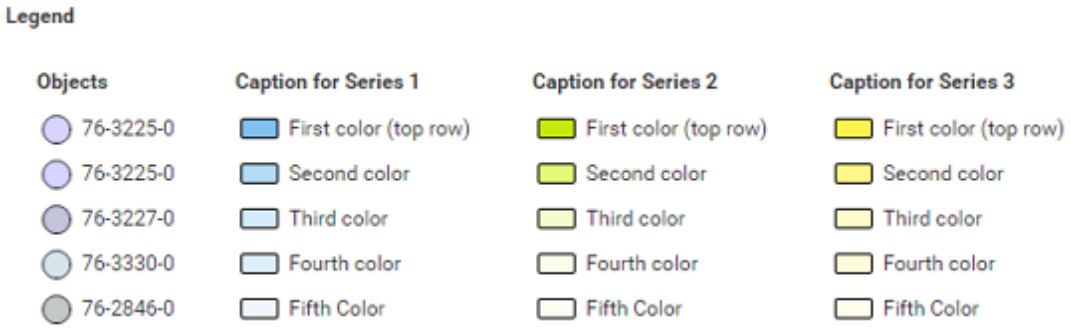
The width of the elements with a virtual end point is irrelevant for the design, because the size of the areas in the portfolio is only defined by the colors with visible end point definitions. The **Value** defined for virtual end points of color definitions must be lower than the minimum value of the x-axis.

For `NestedSquare` design, it is not required to defined coloring for virtual segments. The last color displayed on the visible x-axis is the last color that is used in the report:



3. Definition of the legend for the coloring

Colors defined for the background of the portfolio are automatically added to the legend of the report. The legend displays all colors grouped by series definition.



The text displayed for each color within the series is defined in the **Caption** column of the editor.

The title displayed for the legend of each series is defined in the **Series Caption** row of the editor in the columns for the definition of the series on top of the color definitions. The example legend displayed above is based on the following definition:

Caption	Value	Colors		
		Series 1	Series 2	Series 3
Series Caption		Caption for Series 1	Caption for Series 2	Caption for Series 3
First color (top row)	1			
Second color	4			
Third color	5			
Fourth color	6			
Fifth Color	7			

Defining Navigation from the Objects in the Portfolio to Alfabet Views

By default, a user can click and hold a shape in the configured portfolio report to open a preview of the object. Navigation to the object profile or object cockpit of the object is also available by default via the Show Details button in the preview or via double click on the object. The default behavior can be changed to open any other standard or configured view for the object.

To change the default navigation of the portfolio, the following attribute of the **Report Assistant** explorer's root node must be mapped to the column name in the dataset returned by the query defined in the ***Query*** element of the portfolio report described in the section Defining the Query the Portfolio is Based On:

Attribute	Description
View Column Name	The name of the database column in the query that returns the definition of an alternative link target for navigation from objects in the portfolio. By default, the standard object profile of the object opens when a user double clicks on a node or clicks a node and then clicks the **Navigate** button. The column must return a string with the following structure: `View=ViewType:ViewName` The `ViewType` can be one of the following: • ***Report*** for a configured report • ***GraphicView*** for a standard Alfabet view • ***ObjectView*** for an object profile `ViewName` is the name of the view. For more information about defining navigation to a target view including handing over of parameters, see Defining Navigation from the Report to Alfabet Views.

Defining the Caption and Size of the Portfolio Report Area

Optionally the size reserved for display of the graphic can be changed and a caption can be displayed on top of the portfolio.

The graphic is displayed in the defined size independent from the size of the client device's screen. If the screen size is not sufficient to display the whole graphic, scroll bars are added to the graphic. In order to

ensure that business graphics can be adequately viewed on a variety of devices, it is recommended that graphic reports and object cockpits are designed for a low screen resolution (such as 1280 x 800 or 1280 x 1024).

To define the caption of the portfolio, the string to be displayed as caption must be provided via the dataset returned by the query defined in the ***Query*** element of the portfolio report described in the section Defining the Query the Portfolio is Based On. The dataset column names must then be defined in the attributes of the root node of the Report Assistant.

To define the layout of the report area, you can optionally set the following attributes in the root node of the **Report Assistant**:

Attribute	Description
Portfolio Caption Column	Defines the name of the dataset column that returns the caption of the portfolio displayed on top of the portfolio chart area. If the dataset column returns multiple different values, the first value that is not `NULL` returned in the dataset will be displayed.
Width	Defines the width of the graphic in pixel. Allowed values are between 1 and 1000. The default value is 800.
Height	Defines the height of the graphic in pixel. Allowed values are between 1 and 1000. The default value is 600.

Defining Portfolio Diagnostics Reports

Portfolio diagnostics reports are designed in the **Report Assistant**. For information about how to open the **Report Assistant**, see Creating a New Graphic Report. For more information about how to work with the **Report Assistant**, see Configuring the Report with the Report Assistant.

The report assistant for portfolio diagnostics reports provides configuration options to specify the scope of analyzed objects and to fine-tune the analysis:

1) In the **Data Source Definition** tab of the **Report Assistant**, define one of the following:

   - A native SQL query selecting all properties of a single object class. To define the query, click the **Browse** button in either the **Alfabet Query** attribute to open the **Alfabet Query Builder** or in the **Query as Text** attribute to open a text editor.
   - An Alfabet query without **Show Properties**. To define the query, click the **Browse** button in either the **Native SQL Query** attribute to open the native SQL editor in Alfabet that provides a button for the definition of Alfabet query language instructions or in the **Query as Text** attribute to open a text editor.

   The definition of `WHERE` conditions and `JOIN`s (`FROM` clauses in Alfabet queries) is only allowed for the definition of filters for the report or to reduce the report's dataset to a subset of objects of the

selected object class. Objects of joint object classes are not included in the analysis. The configured report exclusively handles one object class.

To build a configured report for analysis of objects of the class **Component**, a native SQL query without `WHERE` conditions will read:

```
SELECT * FROM COMPONENT
```

2) Open the **Default Layout** tab. A table is displayed containing all object class properties of the selected object class that have the **Property Type** attribute set to `Boolean`, `Integer`, `Real`, `Reference`, or `String` and all indicators and role types that are available for the object class. The following information is displayed about each object class property/indicator:

- **Feature**: The name of the object class property, role type or indicator.
- **Type**: The property type of the object class property. For indicators, the type is `Indicator` if the indicator is manually defined, or `Indicator Computed` if the indicator is automatically calculated. For role types, the type is `RoleType`.
- **Caption**: The caption of the object class property, role type or indicator.

If new object class properties, role types, or indicators are available for an object class, these are not automatically added to the list in an already existing portfolio diagnostics report. To add new object class properties, role types, or indicators to the list, click the **Refresh Available Features** button.

3) In the **Exclude** column, set a checkmark for all object class properties, role types or indicators that you would like to exclude from the analysis via the augmented AI mechanism calculating the portfolios.

To remove all checkmarks from the column, click the **Include All** button in the toolbar. To set a checkmark in all cells of the column, click the **Exclude All** button in the toolbar.

4) Define the number of results that shall be displayed for the levels of the portfolio analysis by means of the parameters in the right pane of the table:

- **Factor Charts**: In the drop-down list, select the number of bar charts that shall be displayed for each of the drill-down levels created via factor analysis. Please note that the number must be selected via the drop-down list. It is not possible to change the number via text edit.
- **Variance Charts**: In the drop-down list, select the number of bar charts that shall be displayed on the root level of the analysis when a user opens the report. Please note that the number must be selected via the drop-down list. It is not possible to change the number via text edit.

5) When the user opens the portfolio diagnostics report, a dataset is generated based on the calculation. The dataset includes all objects found via the query defined in the **Data Source Definition** tab and all properties listed in the table of the **Default Layout** tab. Prior to starting the analysis, the software will check whether a configurable amount of data is available in each row or column of the dataset. If the percentage of allowed undefined values is higher than the allowed value, the row or column will be removed from the dataset. The resulting dataset is used for all subsequent analysis steps without further changes. Define the limits for availability of undefined data with the following parameters on the right of the table:

- **Feature Elimination %**: Define the percentage of undefined values that shall result in the indicator being excluded from the analysis.

- **Object Elimination %**: Define the percentage of undefined indicators that shall result in the object being excluded from the analysis.

6) Define the following default values in the right pane of the table. If a property of the respective value type is not set for an object, the specified value will be used in the analysis for the object. The value does not change the actual setting of the object class property for the object but is exclusively used in the analysis. It is entered in the dataset created for analysis after the removal of columns and rows with an exceedingly high number of undefined values.

- **Default Integer/Real**: Enter an integer value to be used as the default for numeric values.
- **Default Boolean**: Select 1 if the Boolean value should be `True` or select 0 if the Boolean value should be `False`.
- **Default Indicator**: Select a numeric default for the indicator.
- **Default String/Reference**: Enter a default string that is displayed for undefined indicators specified to have a semantic value, undefined object class properties of the type `String` or `Reference`, or role types that have no roles assigned.

7) If more than 20 values are available for a chart, the values are combines in buckets and the overall number of objects with one of the values in the bucket is shown as Y-Value for the bucket. The X-Value is then a range of values. The overall number of buckets to be formed from the values is configurable. Set the attribute **Bucket Number** to a number between 1 and twenty to specify how many buckets shall be available.

For example, analyzed applications have 40 different values for an indicator that returns the number of support requests per year. If the number of buckets is ten, each bucket will include 4 values. If the number of buckets is 5, each bucket will combine 8 values.

8) Open the **Additional Report Properties** tab and define the following attributes:

- **Portfolio Diagnostics Report Details View Caption**: Enter the caption that shall be displayed on all factor analysis views of the portfolio analytics report.
- **Portfolio Diagnostics Report Details View Description**: Enter the description that shall be displayed in the header of all factor analysis views of the portfolio analytics report.
- **Portfolio Diagnostics Report Object View Caption**: Enter the caption that shall be displayed on the object list views of the portfolio analytics report.
- **Portfolio Diagnostics Report Object View Description**: Enter the description that shall be displayed in the header of the object list views of the portfolio analytics report.
- **Chart Height**: Enter the height of the charts displayed in the report views as number of pixel. the default is 300.
- **Chart Width**: Enter the width of the charts displayed in the report views as number of pixel. the default is 400.

9) Click **OK** to save your settings and to close the report assistant.

10) In the toolbar, click the **Save**  button to save your changes.

Defining a Treemap Report

Treemap reports are designed in the **Report Assistant**. For information about how to open the **Report Assistant**, see Creating a New Graphic Report. For more information about how to work with the **Report Assistant**, see Configuring the Report with the Report Assistant.

The content and layout of a treemap report is defined by adding elements to the explorer of the **Report Assistant** and setting the attributes in the attribute pane of the **Report Assistant** window.

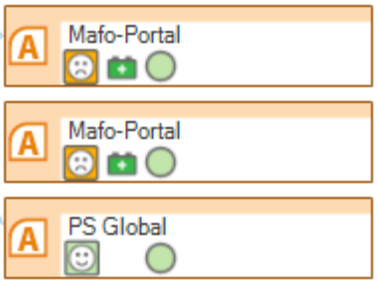
The following table lists the types of explorer node elements that can be added to the report configuration to specify the content of the report and provides information about the purpose of each explorer node element:

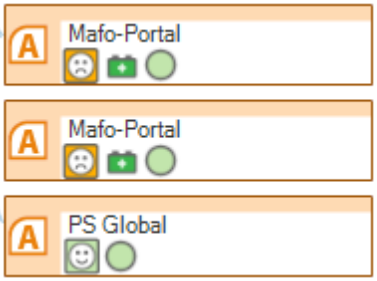
Explorer Node Element	Required to Configure:
Root Node	General layout of the report
Root Queries	Container for **Query** elements that define the column headers for the report.
Query	Definition of the column headers for the report when specified as a sub-element of the **Root Queries** node. Defines the choice of elements displayed in the columns when specified as a sub-element of the **Class Entry** node.
Class Entry	Definition of the object classes included in the report.
Color Rule	Definition of the colors used in the treemap report and the rules that define which objects are displayed with which color.
Size Rule	Definition of three different heights for the boxes representing the objects in the treemap report and the rules that define which objects are displayed with which height.
Indicator Rule	Definition of one or multiple icons displayed in the boxes representing the objects in the current cell of the lane report. The icons can either represent an indicator or any aspect defined via a query. For more information, see the section Defining Indicator Rules.

Defining the Basic Design of the Treemap Report

The basic design of the report is designed directly in the attributes of the root node element of the **Report Assistant** explorer.

Attribute	Description
Area Background Color	Defines the background color of the columns and color of the column header fields.
Layout	Defines whether this is a layered diagram, a rectangualr treemap or a treemap report. Set this attribute to `TreeMap` to define a treemap report.
Use Mandates	Defines whether the implementation of the mandate capability must be taken into consideration to view objects in the report. If set to `True`, a user can only view objects that have a mandate specification identical to the user's mandate.
Show Legend	Defines whether a legend is shown for the color coding, indicators, and size of the objects in the report.
Margin	Defines the space between the treemap and the border of the presentation object in pixel.
Space	Defines the spacing between the columns and between the boxes in the columns in pixel.
Item Width	Defines the width of the boxes in the report in pixel.
Medium Height	Defines the height of the boxes that are displayed for all objects when no size rules are specified in pixel. **Note**: If size rules are specified, additional box heights must be defined. The definition of these attributes is described in the section Displaying Evaluation Criteria By Size or Color Variation or Addition of Indicator Icons.
Item Border Width	Defines the width of the border of the boxes in the report. The default value is 1 point. when using the border color as object property indicator via a color rule, it is recommended to set the border width to a higher value.
Show Indicator	Select `True` to display an indicator icon in the upper right corner of the object boxes or multiple indicator icons at the bottom of the object boxes. Indicator icons are one displayed in the object boxes if an indicator rule is defined that specifies the dipslay of an indicator for the object. Indicator icons can represent an indicator value from the object evaluation or any other aspect that can be defined for the object via a query. Select `False` to exclude the display of the indicator icons for the object box column. For more information about the definition of indicator rules to assign icons to object boxes, see Defining Indicator Rules.

Attribute	Description
Show Multiple Indicators	Defines whether display of icons defined via indicator rules is limited to one icon per object box or whether multiple icons can be assigned to an object box. If **Show Multiple Indicators** is set to `False`, a single icon can be displayed on the right of the object box. If multiple icons are found by indicator rules, the first icon found is assigned and all other icons are ignored. If **Show Multiple Indicators** is set to `True`, one or multiple icons are displayed from left to right at the bottom of the object box. If the number of icons that are assigned to the object box via indicator rules exceeds the space available for display of icons in a single row, the first icons found are displayed and, if the object box space is exceeded, any other icons are ignored. The way icons are placed in the object box also depends on the attribute **Indicator Fill Policy** of the **Item** node. For more information about the placement of icons assigned via indicator rules, see the section Defining Indicator Rules.
Indicator Fill Policy	If **Show Multiple Indicators** is set to `True`, this attribute defines how icons assigned to an object box are placed in the row on the bottom of the object box: • `ByIndex`: This setting shall be used if multiple indicators are defined via multiple indicator rules, each returning a single icon. The position of the icon resulting from each indicator rule is defined by means of the attribute **Indicator Index** of the respective indicator rule that must be set to an unsigned integer starting with 0 for the left most position. If an **Indicator Index** attribute has been defined for each indicator rule and an indicator is not set for a specific object, the position of this indicator icon in the box will be left empty and the position of all other indicator icons will not be changed:  If the order of the **Indicator Index** attribute is not defined, the placement of indicators will be arbitrary. This method is designed for positioning only one icon per indicator rule. If an indicator rule returns multiple icons, only one of the icons is displayed. • `LeftAlign`: This setting shall be used if a single indicator rule is defined that returns multiple icons. It can also be applied when multiple indicator rules each return a single icon, but the position and order of icons is not important to understand the report. The Icons are displayed in the order of occurence within the indicator rule(s)

Attribute	Description
	from left to right. If a different number of indicator icons is returned, the remaining icons are moved to the left:  If the indicator icons are all found by a single indicator rule, the order of icons depends on the order of results in the result dataset of the query of the indicator rule and may be different for different objects if no sort order is defined in the query.

The design of the boxes representing objects in the configured report can be changed from the standard design with a white rectangle as background for text within the colored object box to fully colored object boxes. For more information see Defining the General Display of Object Boxes in Treemap, Diagram, Lane and Layered Diagram Reports.

Defining the Objects to Display in the Report

The objects displayed in the column headers and the columns of the report are found via queries defined for the elements of the report. The queries can either be native SQL queries or Alfabet queries.

If you are not familiar with Alfabet queries, see the chapter Defining Queries.

For special rules that apply to the definition of native SQL queries in the context of configurations for Alfabet, see the section Defining Native SQL Queries.

The information displayed in the object boxes of the report is identical to the column headers that result from a tabular output of the defined query.

Report columns that are defined for technical reasons are not displayed in the object boxes in the report. This includes:

- The first column of the output of a native SQL query. This column must specify the `REFSTR` of the object for technical reasons and is ignored in the visible output..

There are two methods to define a treemap report:

- The complete content of the report is defined in one query resulting in a grouped dataset.

- A separate query is added to the report configuration to find the objects in the column header and the objects beneath the column header.

  When separate queries are defined, a high number of queries must be executed by the database to display the report. This can lead to performance issues. Therefore, it is recommended to base the report on a grouped dataset defined in a single query.

Defining a Treemap Report With a Single Query

The columns in the treemap report including column headers and column content are specified via an Alfabet query or a native SQL query defined in an element node **Query** as sub-node of the **Root Queries** node. The query finds the objects that are displayed in the column header. It also determines the information that is displayed about the objects in the header cells and the sorting of the column headers in the first grouping level. The second grouping level defines the content of the columns.

One **Class Entry** child element must be added to the explorer root node for each object class that is included in the report. The attribute **Class Name** defines the name of the specified object class and is mandatory for the configuration. The result is as follows:

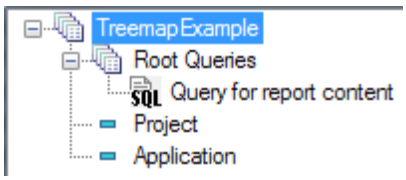
- One **Class Entry** element for the object class displayed in the column headers.
- One **Class Entry** element for each object class displayed in the columns. You can display objects of multiple classes in the report.

The **Class Entry** elements may not contain a sub-element **Query**.

The following example displays the position and structure of the elements **Class Entry** and the **Query** elements defining the column header and the objects in the report for an example report that displays projects in the column headers and the applications assigned to the project architecture in the columns.

The root element must contain two **Class Entry** elements. One for the object class Project displayed in the column headers and one for the applications displayed in the columns of the treemap.

A **Query** element is added to the **Root Queries** node to find the projects that are displayed in the column headers and the applications displayed in the columns in a grouped dataset.



The query defined for the **Query** element must define a grouped dataset with one grouping level:

```
ALFABET_QUERY_500
FIND Project
   InnerJoin ProjectArch ON Project.REFSTR = ProjectArch.Project
   InnerJoin Application ON ProjectArch.Object = Application.REFSTR
Instructions
```



```
    GroupBy_Ex("Project.REFSTR","Application.REFSTR","Project",0);
    RemoveColumns("Application.REFSTR");
EndOfInstructions
QUERY_XML
<QueryDef>
    <ShowProperty Type="Property" ClassName="Project" Name="REFSTR"
    />
    <ShowProperty Type="Property" ClassName="Project" Name="Name" />
    <ShowProperty Type="Property" ClassName="Application"
    Name="REFSTR" />
    <ShowProperty Type="Property" ClassName="Application" Name="Name"
    />
    <ShowProperty Type="Property" ClassName="Application"
    Name="Version" />
    <SortProperty Type="Property" ClassName="Project" Name="REFSTR"
    />
    <SortProperty Type="Property" ClassName="Application"
    Name="REFSTR" />
</QueryDef>
```

Optionally, an icon can be selected in the attributes of the **Class Entry** element. The icon is then displayed in the object boxes of the defined object class in the report.

Defining a Treemap Report With One Query Per Object Class

The columns in the treemap report are defined via an Alfabet query or a native SQL query defined in an element node **Query** defined as sub-node of the **Root Queries** node. The query finds the objects that are displayed in the column header and determines the information that is displayed about the objects in the header cells and the sorting of the column headers.

One **Class Entry** child element must be added to the explorer root node for each object class that is included in the report with the mandatory attribute **Class Name** that defines the name of the specified object class. The result is as follows:

- One **Class Entry** element for the object class displayed in the column headers
- One **Class Entry** element for each object class displayed in the columns. You can display objects of multiple classes in the report.

The object class defined for the column headers of the report is the base class for the objects displayed in the report. The Alfabet queries that define which objects are to be displayed in the report are specified as the child elements **Query** of the **Class Entry** element of the object class defined for the column header.

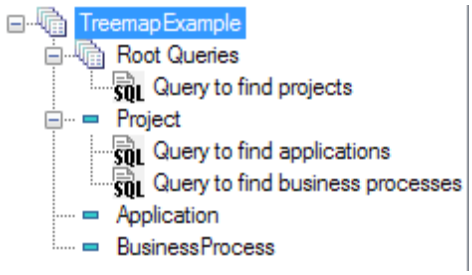
All other **Class Entry** elements may not contain a sub-element `Query`.

The Alfabet query defining an object class displayed in the columns must contain the parameter:`BASE` in a `WHERE` clause to map the objects to a column. `BASE` is identical to the `REFSTR` of the object in the column header.

The following example displays the position and structure of the elements **Class Entry** and the **Query** elements defining the column header and the objects in the report for an example report that displays projects in the column headers and the applications and business processes assigned to the project architecture in the columns.

The root element must contain three **Class Entry** elements. One for the object class Project displayed in the column headers and one for each object class displayed in the columns of the treemap.

A **Query** element is added to the RootQueries node to find the projects that are displayed in the column headers. The Alfabet queries that define which applications and business are displayed in which column must be specified as child elements of the **Class Entry** element for Project.



The queries defined for the **Query** element of the **Class Entry** for Project must contain a `WHERE` clause mapping applications and business processes to the project in the column header with the parameter `BASE`. For example,:

```
ALFABET_QUERY_500
FIND Application
   InnerJoin ProjectArch ON ProjectArch.Object = Application.REFSTR
   InnerJoin Project ON Project.REFSTR = ProjectArch.Project
WHERE
   Project.REFSTR CONTAINS:BASE
QUERY_XML
<QueryDef>
   <ShowProperty Type="Property" ClassName="Application" Name="Name"
   />
   <ShowProperty Type="Property" ClassName="Application"
   Name="Version" />
</QueryDef>
```

Optionally, an icon can be selected in the attributes of the `ClassEntry` element. The icon is then displayed in the object boxes of the defined object class in the report.

The table below lists all available attributes in the **Query** element.

Attribute	Description
Name	Defines the name of the **Query** element displayed in the explorer of the **Report Assistant**.

Attribute	Description
Alfabet Query/ Query as Text/ Native Sql	Defines a query to find the objects that are displayed in the report. The way the query must be defined depends on the location of the **Query** element in the report.
Cascading	Not relevant for treemap reports.

The table below lists all available attributes in the **Class Entry** element.

Attrib-ute	Description
Class Name	Defines the object class specified in this **Class Entry** element. For the object classes that allow stereotype specifications (For example, `Project` and `Domain`), the XML attribute `ClassName` can specify an object of a specific stereotype. To define an XML element ***Query*** for a stereotype, the XML attribute `ClassName` must be *`ClassName:StereotypeName`* For example,: *`Project:StatementOfWork`*
Icon	Defines the icon to be displayed for the object in the object boxes. An icon can be selected from a drop-down list.
Param Name	Not relevant for treemap reports.

Displaying Evaluation Criteria By Size or Color Variation or Addition of Indicator Icons

Evaluation criteria can be added to the report:

- Rules can be defined that assign colors on the basis of property values of the current object. For example, coloring can be different for boxes of objects that are planned, active, or retired. For more information about the definition of color rules, see Defining Color Rules.
- Rules can be defined that trigger the display of an indicator icon in the object boxes of selected objects. For more information, see Defining Indicator Rules.
- Rules can be defined that lead to objects with defined property values to be displayed in smaller or larger object boxes. Size rules are described below.

The boxes in the report can be displayed with three different heights. You can specify a size rule for each box height that allows you to display objects with different attribute values with different box sizes. For example, you can display application groups in a report in a different size determined by the number of applications assigned to the application group. If the number is lower than 5, a small box is displayed. If the number is higher than 20, a big box is displayed. Other application groups are displayed with a medium box height.

To define the size of the boxes in the report, you have to specify the following:

- The height in pixel for the small, medium, and large box size. These are defined by means of the attributes of the root node element.
- The size rules that define which objects are displayed with which box size. Size rules are defined in the element `Size Rule`, which is a child element of the root node element.

You can define two different types of size rules:

- `ObjectQuery`: The query defined for the size rule returns the `REFSTR` of the object that the box height shall be applied to. The size that shall be applied to the object boxes is defined separately via attributes of the size rule.

  When defining an Alfabet query, no Show property definition is required. The `REFSTR` of the `FIND` object class is returned.

  When defining a native SQL query, the `SELECT` statement of the query must have the `REFSTR` of the relevant object class defined as the first argument. No further arguments are required in the `SELECT` statement.

- `SizeQuery`: The query defined for the size rule not only returns the `REFSTR` of the objects that the size shall be applied to but also the size specification that shall be applied.

  SizeQuery rules require the specification of a native SQL query.

  The `SELECT` statement of the SQL query must return the following in the given order:

  - the `REFSTR` of the current object
  - a string that is used as the caption for the size assignment in the legend of the report
  - the size specification as string. Allowed values are `Small`, `Large` and `Medium`.

  For example,:

  ```
  SELECT DOMAIN.REFSTR, "Legend text", "Large"
  ```

  - The definition of a size query in a size rule has the following advantages as compared to an object query: The number of overall size rules required for a report can be reduced. For example, if you want to apply a different box size to applications in a different object state, you can define the size for all object states in one query while a separate size rule for each object state would be required when using object queries.
  - Dynamic size assignment can be applied to a report. You can, For example, define a query that sizes object boxes according to the indicator values assigned to the objects. In the report, you can define a filter that allows the user viewing the

report to choose an indicator type. The size of object boxes is then reassigned according to the values for the selected indicator.

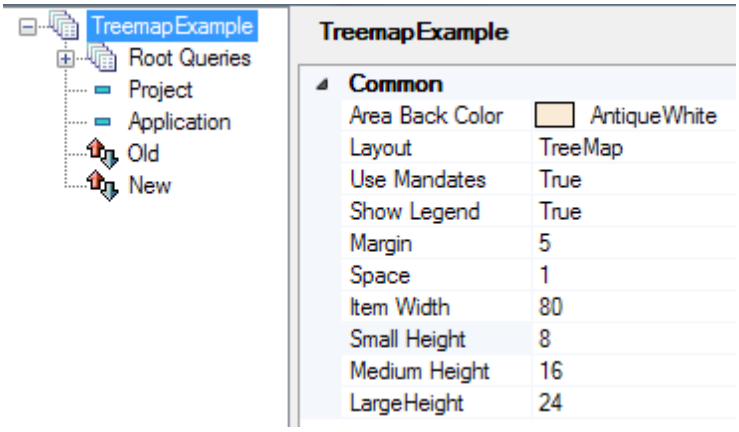
The specification of box sizes and size rules is optional. If no box sizes are defined, all boxes will be displayed in a default height of 12 for all size rules. If no size rules are specified, all objects will be displayed with the defined medium box size.

When you define the queries for the size rules of the report, make sure that all objects can be assigned explicitly to one size rule. If this is not the case, the following applies:

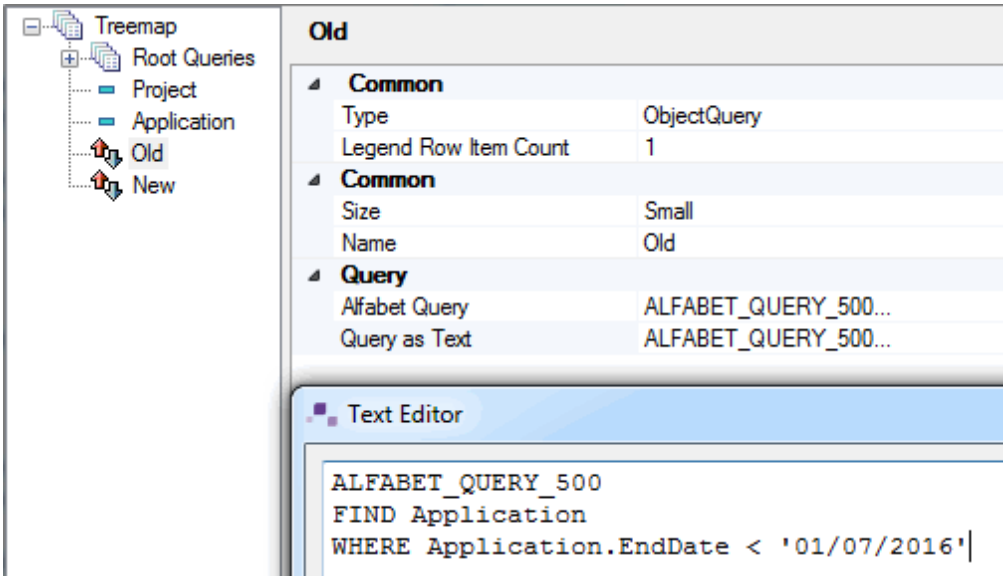
- Objects that correspond to more than one size rule are displayed in the size of the first rule written in the XML object that applies.
- Objects that do not correspond to any of the defined size rules are displayed in medium size. You cannot distinguish between these objects and objects corresponding to the medium size rule in the treemap report.

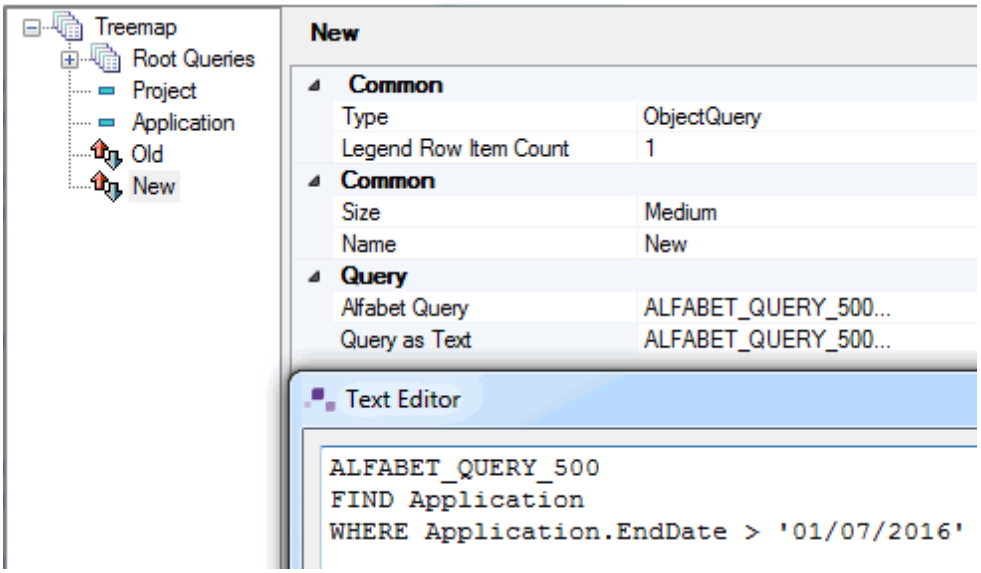
The following example displays a report where applications are displayed in different box height dependent on the start date definition. Old applications are displayed in a smaller height as new applications.

In the root node attributes, three different of object box sizes are defined:



Two size rules are defined, each specifying the box size with the attribute Size and the objects to that the size rule applies with an Alfabet query.





The table below lists all attributes for the specification of box sizes in treemap reports.

Attribute	Description
In the root node element	
Small Height	Defines the height of the boxes that are displayed for objects corresponding to the size rule with **Size** = `"Small"`.
Medium Height	Defines the height of the boxes that are displayed • for objects corresponding to the size rule with **Size** = `"Medium"`. • for all objects when no size rules are specified.
Large Height	Defines the height of the boxes that are displayed for objects corresponding to the size rule with **Size** =`"Large"`.
In the Size Rule element	
Name	Defines the title displayed for the size in the legend. For size rules of the **Type** `ObjectQuery`, also defines the the caption displayed for the size rule in the explorer of the **Report Assistant**. **NOTE**: The attribute **Show Legend** of the **Root** element must be set to true to display the legend. If this element is set to `false`, you do not need to specify a name for the size rule.
Type	Select `ObjectQuery` to define the objects that the size rules apply to with a query and the size specification with the attributes of the size rule. Select `SizeQuery` to define both the size specification and the objects that the size rule applies to with a query.
Size	Defines the size of the boxes of objects corresponding to this size rule if the Type of the size rule is ObjectQuery. The pixel size of the boxes is specified with the `Small Height, Medium Height` and `Large Height`attributes of the root node element.
Alfabet Query/ Native Sql/ Query as Text	If the **Type** of the size rule is `SizeQuery`, define a native SQL query in the **Native Sql** attribute or an Alfabet query in the **AlfabetQuery** or **Query as Text** attribute. The query must return the `REFSTR` of the objects that the size rule shall apply to. If the **Type** of the size rule is `SizeQuery`, define a native SQL query in the **Native Sql** attribute that returns: • the `REFSTR` of the current object

Attribute	Description
	• a string that is used as caption for the size assignment in the legend of the report • a string that defines which box size applies. Valid values are `Medium`, `Small` and `Large`.
Legend Row Item Count	Defines the number of sizes displayed in one row of the legend of the report. For size rules of the type `ObjectQuery`, only the value defined in the first size rule of the type `ObjectQuery` of the report is applied to the group. By default, 4 items are displayed in a row of the legend.

Defining a Rectangular Treemap Report

Rectangular treemap reports are designed in the **Report Assistant**. For information about how to open the **Report Assistant**, see Creating a New Graphic Report. For more information about how to work with the **Report Assistant**, see Configuring the Report with the Report Assistant.

The content and layout of a layered diagram report is defined by adding elements to the explorer of the **Report Assistant** and setting the attributes in the attribute pane of the **Report Assistant** window.

Rectangular Treemap reports can only be defined using native SQL. Alfabet query language does not provide the ability to include text into search results, that is required to define the report.

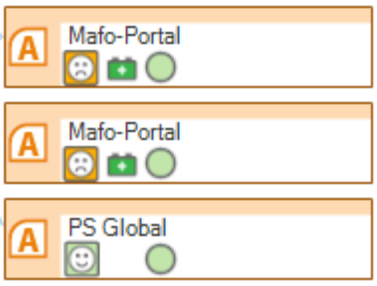
The following table lists the types of explorer node elements that can be added to the report configuration to specify the content of the report. Information is provided in the table about the purpose of each explorer node element:

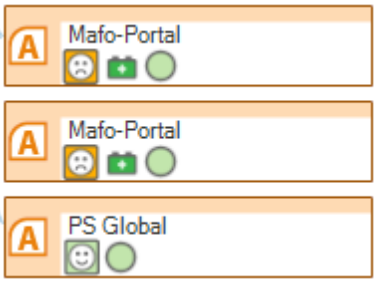
Explorer Node Element	Required to Configure:
Root Node	General layout of the report.
Root Queries	Container for the **Query** element that defines the content of the report.
Query	Definition of the content and design of the report.
Indicator Rule	Definition of one or multiple icons displayed in the boxes representing the objects in the current cell of the lane report. The icons can either represent an indicator or any aspect defined via a query. For more information, see the section Defining Indicator Rules.

Defining the Basic Design of the Rectangular Treemap Report

The basic design of the report is designed directly in the attributes of the root node element of the **Report Assistant** explorer. Only a subset of the available attributes for treemaps are relevant for rectangular treemap reports. The following table lists only attributes relevant for the overall design of a rectangular treemap report:

Attribute	Description
Layout	Defines whether this is a layered diagram, a rectangular treemap report or a treemap report. Set this attribute to `RectangularTreeMap` to define a rectangular treemap report.
Use Mandates	Defines whether the implementation of the mandate capability must be taken into consideration to view objects in the report. If set to `True`, a user can only view objects that have a mandate specification identical to the user's mandate.
Show Legend	Defines whether a legend is shown for the color coding and indicators in the report.
Margin	Defines the space between the object boxes of the rectangular treemap and the border of the presentation object in pixel.
Space	Defines the spacing between the boxes in pixel. The spacing is applied to boxes of both levels of the report hierarchy.
Report Height	Defines the overall height of the resulting report. The size of the object boxes is adjusted accordingly.
Report Width	Defines the overall width of the resulting report. The size of the object boxes is adjusted accordingly.
Item Border Width	Defines the width of the border of the boxes in the report. The default value is 1 point. When using the border color as object property indicator via a **Border Color Column** specification, it is recommended to set the border width to a higher value. For more information about defining colors in the report dependent on an object property or simply deviating from the default colors for the object, see Displaying Evaluation Criteria By Color Variation.
Show Class Icon	Select `True` to display the class icon in the object boxes. the class icon is defined in the class settings of the object class.
Show Indicator	Select `True` to display an indicator icon in the upper right corner of the object boxes or multiple indicator icons at the bottom of the object boxes. Indicator icons are one displayed in the object boxes if an indicator rule is defined that specifies the dipslay of an indicator

Attribute	Description
	for the object. Indicator icons can represent an indicator value from the object evaluation or any other aspect that can be defined for the object via a query. Select `False` to exclude the display of the indicator icons for the object box column. For more information about the definition of indicator rules to assign icons to object boxes, see Defining Indicator Rules.
Show Multiple Indicators	Defines whether display of icons defined via indicator rules is limited to one icon per object box or whether multiple icons can be assigned to an object box. If **Show Multiple Indicators** is set to `False`, a single icon can be displayed on the right of the object box. If multiple icons are found by indicator rules, the first icon found is assigned and all other icons are ignored. If **Show Multiple Indicators** is set to `True`, one or multiple icons are displayed from left to right at the bottom of the object box. If the number of icons that are assigned to the object box via indicator rules exceeds the space available for display of icons in a single row, the first icons found are displayed and, if the object box space is exceeded, any other icons are ignored. The way icons are placed in the object box also depends on the attribute **Indicator Fill Policy** of the **Item** node. For more information about the placement of icons assigned via indicator rules, see the section Defining Indicator Rules.
Indicator Fill Policy	If **Show Multiple Indicators** is set to `True`, this attribute defines how icons assigned to an object box are placed in the row on the bottom of the object box: • `ByIndex`: This setting shall be used if multiple indicators are defined via multiple indicator rules, each returning a single icon. The position of the icon resulting from each indicator rule is defined by means of the attribute **Indicator Index** of the respective indicator rule that must be set to an unsigned integer starting with 0 for the left most position. If an **Indicator Index** attribute has been defined for each indicator rule and an indicator is not set for a specific object, the position of this indicator icon in the box will be left empty and the position of all other indicator icons will not be changed:  If the order of the **Indicator Index** attribute is not defined, the placement of indicators will be arbitrary. This method is designed for positioning only one icon per indicator rule. If an indicator rule returns multiple icons, only one of the icons is displayed.

Attribute	Description
	• `LeftAlign`: This setting shall be used if a single indicator rule is defined that returns multiple icons. It can also be applied when multiple indicator rules each return a single icon, but the position and order of icons is not important to understand the report. The Icons are displayed in the order of occurence within the indicator rule(s) from left to right. If a different number of indicator icons is returned, the remaining icons are moved to the left:  If the indicator icons are all found by a single indicator rule, the order of icons depends on the order of results in the result dataset of the query of the indicator rule and may be different for different objects if no sort order is defined in the query.

Defining the Objects to Display in the Report

The object boxes in the first and second level in the rectangular treemap report, including the text displayed in the boxes, are specified via a native SQL query defined in an element node **Query** as sub-node of the `RootQueries` node. The query must define a grouped dataset via definition of Alfabet query language instructions. The grouped dataset finds the objects that are displayed in the column header. It also determines the information that is displayed about the objects in the cells of the report. The first grouping level defines the information about the object boxes of the superordinate level and the second grouping level defines the content of the object boxes of the subordinate level.

For information about how to define a grouped dataset using Alfabet query language instructions see the section Grouping Query Results in Expandable Reports in the chapter Defining Queries.

The native SQL query defined for a rectangle treemap report can be very complex. To make sure that your query corresponds to the above-mentioned requirements without taking the configuration of the report into account, you can first define a configured report of the **Type** `NativeSQL` and control that the output is a grouped table with the column joined in the correct way. When the output is as expected, paste the query into the rectangular treemap report and set the required attributes in the report assistant.

The native SQL query can return the following information about an object:

- The text to display in the object boxes.
- A link to a view that shall open when the user clicks the text in the object boxes.
- A size definition for the object boxes in the report. This definition must be an integer that is then used to calculate the relative size of the object boxes in relation to other boxes of the report.

- A color definition for the background of the object boxes.
- A color definition for the text of the object boxes.
- A color definition for the border of the object boxes.

After having defined the native SQL query, the name of the column containing the respective information must be defined in the attributes of the root node of the report assistant:

Attribute	Description
Image Column	Defines the name of the column containing the text to be displayed in the boxes.
Value Column	Defines the name of the column containing integer values that shall be used to calculate the sizing of the boxes in the report.
Aggregated Value	If set to `True`, the upper level of boxes is sized according to the aggregated sizing values defined for the subordinate level of boxes in the **Value Column**. Sizing values defined for objects on the superordinate objects within the dataset resulting from the query are ignored. If set to `False`, the upper level of boxes is sized according to the size specification for the objects in the superordinate level in the **Value Column**. If no values are defined for the superordinate level of boxes and **Aggregated Value** is set to `False`, the report cannot be executed. For more information about sizing of object boxes, see Displaying Evaluation Criteria By Size Variation.
Tooltip Column	Defines the content of the tooltip that is displayed when the user moves the mouse over the text in the boxes. If **Tooltip Column** is not set, the tooltip is identical with the text displayed in the boxes.
Background Color Column	Defines the name of the column containing a HTML compliant color definition for coloring of the background of the boxes.
Foreground Color Column	Defines the name of the column containing a HTML compliant color definition for coloring of the text in the boxes.
Border Color Column	Defines the name of the column containing a HTML compliant color definition for coloring of the border of the boxes.
Color Legend Column	Defines the name of the column containing a text to display in the legend for the coloring of the boxes.

Attribute	Description
View Column	Defines the name of the column containing the definition of an alternative link target for the double click action on the object boxes.

In the attributes of the root node of the report assistant, only one column can be defined for a type of information. For example, only one column can be defined to display the text of the object boxes. Therefore, the native SQL query must be defined to display information about both levels in the grouped dataset in the same column. This can be achived with `JoinColumn` instructions of the Alfabet query language.

In the following, the building of the report query is described stepwise on basis of an example. Each column definition is handled in a separate section. This section describes the basic definition of object boxes containing a name via an example. For the other aspects of the report, the following sections are providing information about the query definition:Displaying Evaluation Criteria By Size Variation

- Displaying Evaluation Criteria By Size Variation
- Displaying Evaluation Criteria By Color Variation
- Defining Indicator Rules
- Defining Navigation from the Report to Alfabet Views

To define the native SQL query for the report:

1) In the report assistant, right-click the node **Root Queries** and select **Add New Root Query**.
2) Click on the new **Query** node under the **Root Queries** node. The attributes for the **Query** are displayed on the right.
3) Define the native SQL query either in the attribute **Sql Query** or **Query As Text**.
4) Optionally, define a name for the query element that is displayed in the explorer of the report assistant with the attribute **Name**. All other attributes of the **Query** element are not relevant for rectangular treemaps.
5) Click on the root node of the report assistant.
6) Enter the name of the columns in the dataset defined by your query into the relevant attributes of the section **Data** of the attributes of the root node.

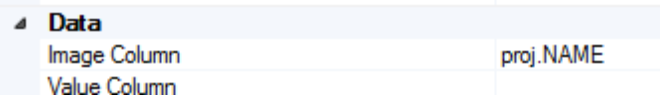
The example report shall display projects as superordinate objects and applications that are part of the project architecture as subordinate objects. In the select statement of the native SQL query of the report four columns are defined. Two columns define the RESTR of the projects and the applications respectively to identify the objects in the report. Two columns are added to return the name of the projects and the name and version of the applications. These names shall be displayed in the object boxes of the report.

```
SELECT proj.REFSTR AS 'proj.REFSTR', proj.NAME AS 'proj.NAME',
app.REFSTR AS 'app.REFSTR', app.NAME + app.VERSION AS 'Application'

FROM PROJECT proj

INNER JOIN PROJECT_ARCH pa ON pa.PROJECT=proj.REFSTR

INNER JOIN APPLICATION app ON app.REFSTR = pa.OBJECT

ORDER BY proj.NAME,app.NAME
```


Please note that the first argument of the `SELECT` statement must include an explicit definition of a column name to be used in the `GroupBy_Ex` instruction that is used to group the dataset. Otherwise the column name would be ignored. The Alfabet query language instructions defined for the native SQL query consist of one `GroupBy_Ex` instruction for grouping of the dataset and one `JoinColumns` instruction to join the columns `proj.NAME` and `Application` into one column named `proj.Name`:

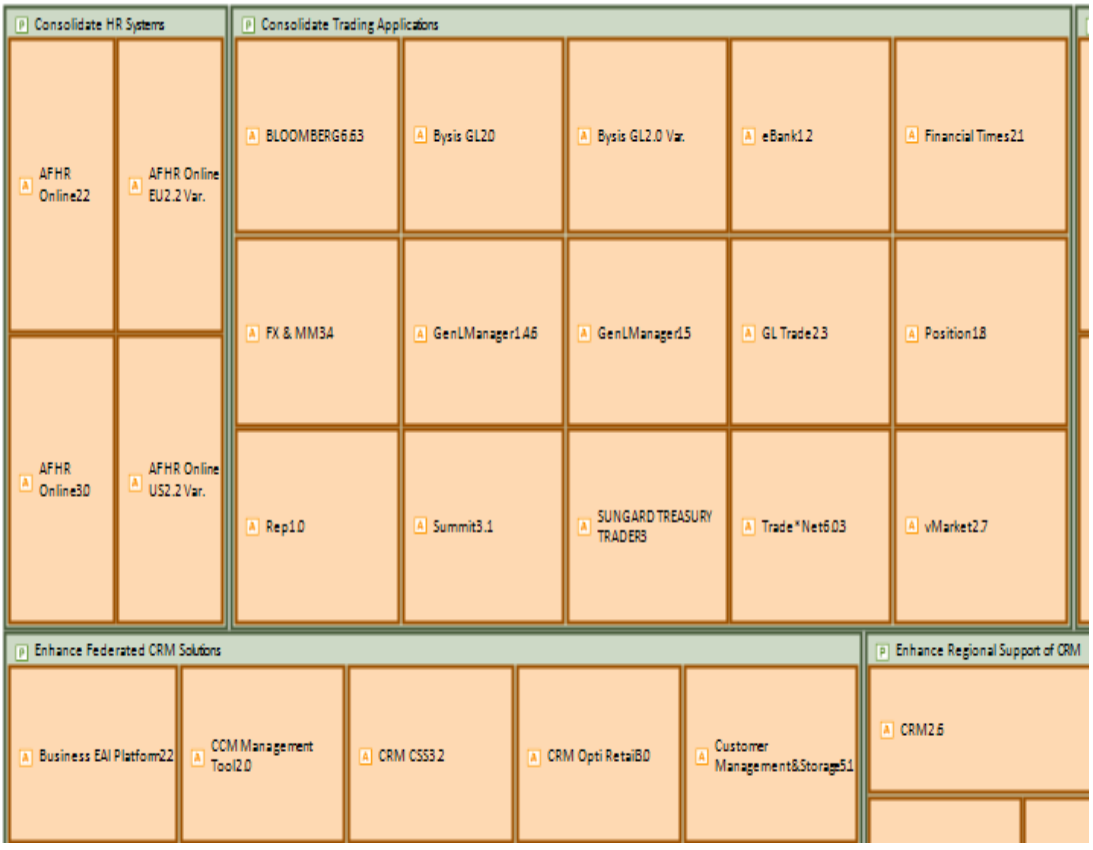
```
GroupBy_Ex("proj.REFSTR","app.REFSTR","proj",0);

JoinColumns("proj.NAME,Application","proj.NAME", "");
```

In the attributes of the root node of the report assistant, the attribute **Image Column** of the report is set to the name of the column containing the object names:



The resulting report shows reports as boxes with application as small boxes within with the name of the object displayed in each box. When the user double clicks a box, the object profile of the object represented by the box opens.

The colors of the boxes are the default colors defined for the objects in the class settings of the object class. The boxes for applications in the subordinate level all have the same size, because no size specification is included in the report. The size of the superordinate box reflects the number of applications that are part of the project's affected architecture:



In the following sections, evaluation criteria and navigation to objects is added to the report.

Displaying Evaluation Criteria By Size Variation

The object boxes can be sized relative to an integer value, For example, the number of subordinate objects or the value of an indicator cast as an integer.

Sizing of boxes can be performed for both levels of object boxes or for one level only. The value for the size specification of the object boxes is read from a column of the output of the native SQL query that the report is based on. This column must return an integer.

The column of the native SQL query that contains the integer values as basis for the relative sizing of the object boxes can contain values for both levels of object boxes. You can use a `Join Column` instruction to combine the values for the superordinate and subordinate level of boxes in one column.

In the attributes of the root node of the report, you must define the name of the column in the dataset containing the sizing values with the attribute **Value Column**.

In addition, you should define the attribute **Aggregated Value** to define how the sizing on the superordinate level of object boxes is performed. If **Aggregated Value** is set to `True`, the sizing of the object boxes in the superordinate level is exclusively performed by aggregating all sizing values of the subordinate objects of the respective box and using the aggregated value as sizing value. Sizing values for the superordinate level defined in the native SQL query are ignored.

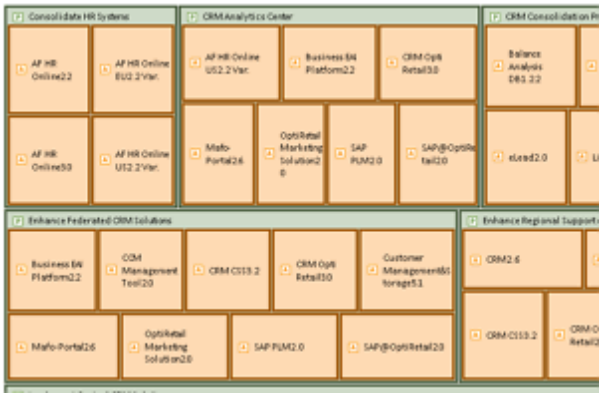
If **Aggregated Value** is set to `False`, the sizing of both levels of object boxes is done according to the sizing values in the column defined with the attribute **Value Column**. If the **Value Column** does not contain values for the superordinate boxes, the report cannot be executed.

If **Value Column** is set, the configured report can only be executed if values are defined for the subordinate level of object boxes. If you want a report to show box sizing exclusively dependent on values from the superordinate level, you can define a fixed integer as value for the subordinate level.

The following examples show the same report defined with all kind of settings for the **Value Column** and **Aggregated Value** attributes:

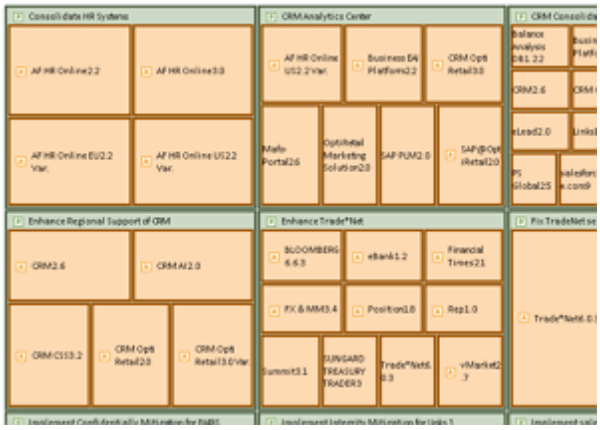
- **Value Column is not set and Aggregated Value is set to True**

All object boxes on the subordinate level have the same size. The size of the superordinate object boxes depend on the number of subordinate object boxes displayed in the superordinate box.



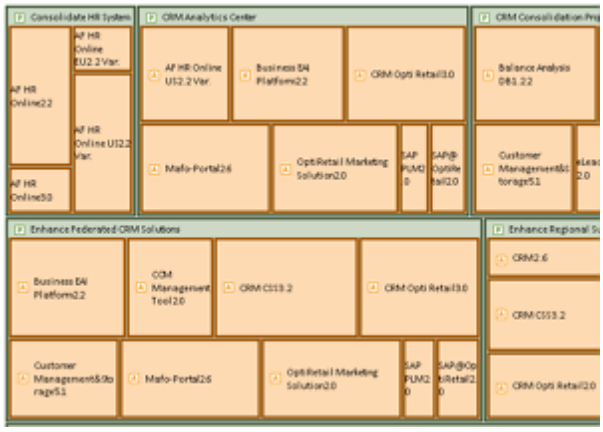
- **Value Column is not set and Aggregated Value is set to False**

All object boxes on the superordinate level have the same size. The size within the superordinate box is equally shared by all subordinate object boxes. That means that subordinate boxes are bigger if less object boxes are in the superordinate object box.



- **Value Column is set and contains only values for subordinate object boxes and Aggregated Value is set to True**

The size of the object boxes on the subordinate level depend on the sizing value defined in the **Value Column**. The size of the superordinate object boxes is calculated from the aggregated values of the subordinate object boxes. This means that the superordinate box size depends on the size of the object boxes displayed within the box. This setting allows to compare all objects on the subordinate level, even if they are displayed within different superordinate object boxes.

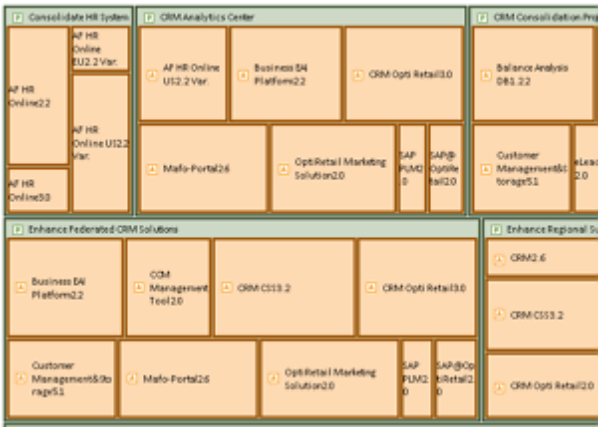


- **Value Column is set and contains only values for subordinate object boxes and Aggregated Value is set to False**

The report cannot be executed.

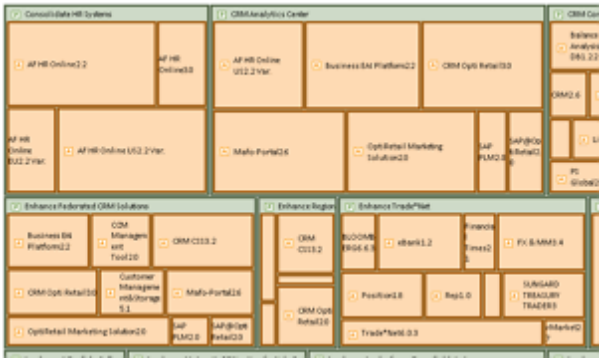
- **Value Column is set and contains values for both levels of object boxes and Aggregated Value is set to True**

The size of the object boxes on the subordinate level depend on the sizing value defined in the **Value Column**. The size of the superordinate object boxes is calculated from the aggregated values of the subordinate object boxes. This means that the superordinate box size depends on the size of the object boxes displayed within the box. This setting allows to compare all objects on the subordinate level, even if they are displayed within different superordinate object boxes.



- **Value Column is set and contains values for both levels of object boxes and Aggregated Value is set to False**

The size of the object boxes on the superordinate level depends on the sizing value defined in the **Value Column**. Within one superordinate box, the sizing of the subordinate boxes depend on the sizing values defined in the **Value Column**. The absolute sizing of the subordinate boxes in the different superordinate object boxes depends on the size of the superordinate box. The size of two subordinate boxes with the same sizing value, but in different superordinate boxes can be different.



The example report displaying projects and applications that are part of the project architecture shall show applications sized according to the setting of an indicator that informs about the criticality of the application for the customer. At the same time, the size of the boxes representing projects shall depend on an indicator that reflects the strategic importance of the project in the enterprise. Both indicators are added to the query results. Indicator vlaues are stored as real values in the Alfabet. Therefore, a `CAST` expression is required in the `SELECT` statement of the native SQL query to convert the value to an integer. The column name for the size speicification of the superordinate project object boxes must follow the naming convention for the superordinate level and start with `proj.`.

```
SELECT proj.REFSTR AS 'proj.REFSTR', proj.NAME AS 'proj.NAME',
app.REFSTR AS 'app.REFSTR', app.NAME + app.VERSION AS
'Application',CAST(indproj.VALUE AS Integer) AS 'proj.BoxSize',
CAST(ind.VALUE AS Integer) AS 'BoxSize'

FROM PROJECT proj

INNER JOIN PROJECT_ARCH pa ON pa.PROJECT=proj.REFSTR

INNER JOIN APPLICATION app ON app.REFSTR = pa.OBJECT

INNER JOIN INDICATOR ind ON ind.OBJECT=app.REFSTR
```



```
INNER JOIN INDICATORTYPE indtype ON ind.INDICATORTYPE =
indtype.REFSTR

INNER JOIN EVALUATIONTYPE evtype ON ind.EVALUATIONTYPE =
evtype.REFSTR

INNER JOIN INDICATOR indproj ON indproj.OBJECT=proj.REFSTR

INNER JOIN INDICATORTYPE indprojtype ON indproj.INDICATORTYPE =
indprojtype.REFSTR

INNER JOIN EVALUATIONTYPE evprojtype ON indproj.EVALUATIONTYPE =
evprojtype.REFSTR

WHERE evprojtype.NAME = 'Business Value'

   AND indprojtype.NAME = 'Strategic Value'

   AND evtype.NAME = 'Criticality'

   AND indtype.NAME = 'Criticality -- Customer Impact'

ORDER BY proj.NAME,app.NAME
```

An additional `JoinColumns` instruction must be added to the Alfabet query language instructions defined for the native SQL query to join the columns `proj.BoxSize` and `BoxSize`, which contain the indicator values, into one column named `BoxSize`:

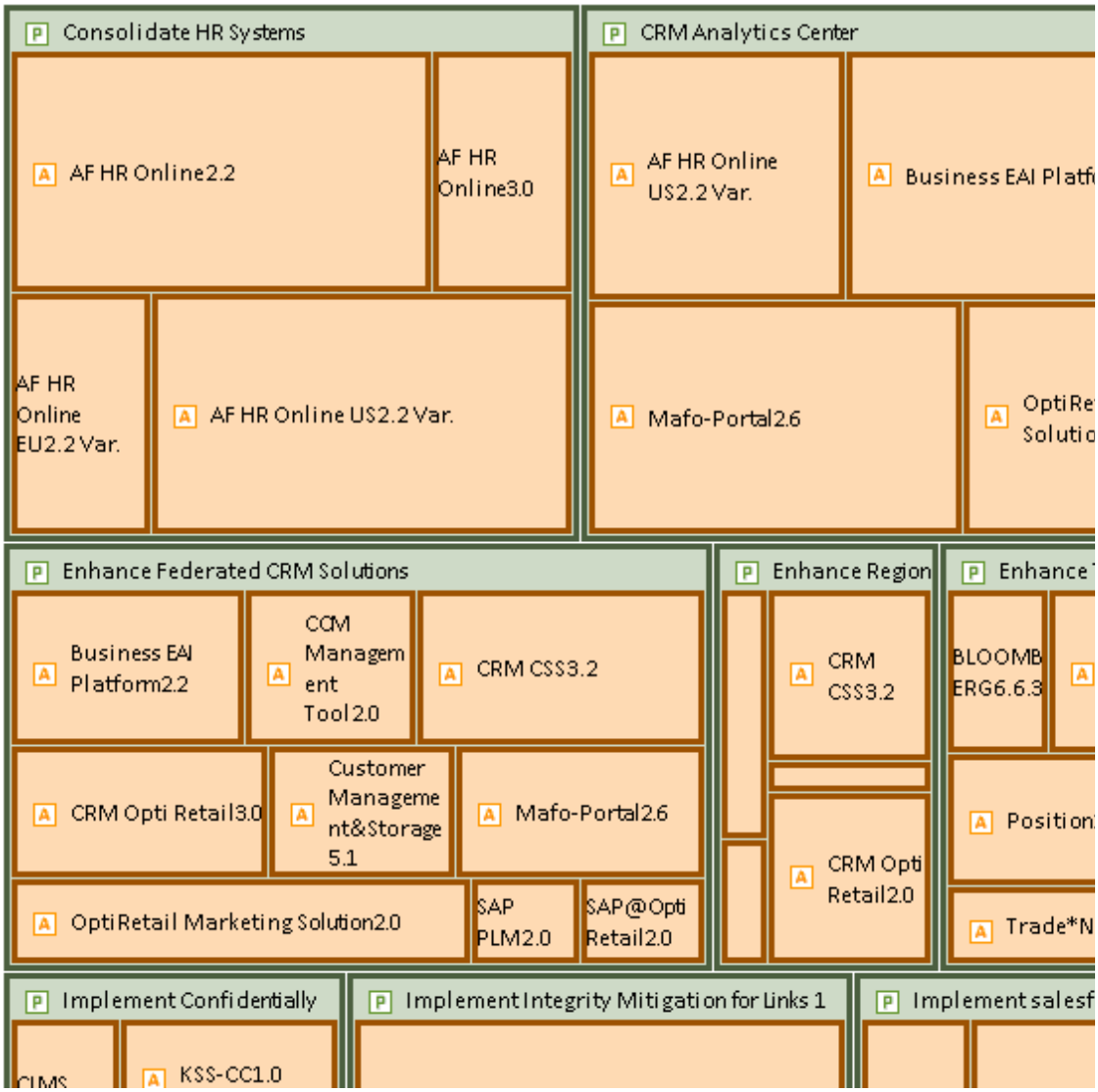
```
GroupBy_Ex("proj.REFSTR","app.REFSTR","proj",0);

JoinColumns("proj.NAME,Application","proj.NAME", "");

JoinColumns("proj.BoxSize,BoxSize","BoxSize","");
```

In the attributes of the root node of the report assistant, the attribute **Value Column** of the report is set to the name of the column containing the object names and the attribute Aggregated Value is set to False to size both level of boxes according to the individual indicator defined for the respective level:

Aggregated Value	False
Show Class Icon	True
Show Indicator	True
Data	
Image Column	proj.NAME
Value Column	BoxSize

The resulting report shows reports as boxes with application as small boxes within with the name of the object displayed in each box. When the user double clicks a box, the object profile of the object represented by the box opens.

The colors of the boxes are the default colors defined for the objects in the class settings of the object class. The boxes for applications in the subordinate level all have the same size, because no size specification is included in the report. The size of the superordinate box reflects the number of applications that are part of the project's affected architecture:



Displaying Evaluation Criteria By Color Variation

By default the object boxes in rectangular treemap reports are colored according to the colors defined for the displayed objects in the object's class settings. The colors of the report can be customized to either display other static colors or to display objects in different colors dependent on defined criteria. The color of the background, the border and the text of the object boxes can be changed. Coloring can be changed for a single level or both level of boxes in the report. It is also possible to define For example, only the background color without changing border or text color.

If the color definition depend on defined criteria, a legend text can be defined and displayed in the legend that is available for graphic reports via the floating toolbar. The legend can contain a string for each combination of background, border and text color.

The following configuration is required to define coloring of the report:

1) In the native SQL defined as root query of the rectangular treemap report, define a column in the output of the query for each color that you want to change. The column must contain HTML compliant color specifications (HTML color name or HEX code).

   The background color, the text color and the border color must each be defined in a separate, single column. If you want to define coloring for both level of object boxes, The columns containing the coloring information for one type of coloring for the superordinate and for the subordinate level must be combined by a `JoinColumns` instruction.

2) If your color definitions give information about the object, because the color depends on an attribute of the object, add an additional column to the query that returns an explanatory text for the color definition in the respective row.

3) in the root node of the rectangular treemap report, enter the name of the columns containing the color definition in the following attributes:

   - **Background Color Column**: Enter the name of the column defining the background color of the object boxes.
   - **Foreground Color Column**: Enter the name of the column defining the text color of the object boxes.
   - **Border Color Column**: Enter the name of the column defining the border color of the object boxes.
   - **Color Legend Column**: Enter the name of the column containing the text that explains the meaning of the combination of colors for the object.

The example report displaying projects and applications that are part of the project architecture shall show applications colored according to their object state. A legend shall be displayed that explains which state is represented by each color. In the select statement of the native SQL query, the color definitions are added with a condition that relates the coloring with the object state. Background color, text color and border color are all changed for the subordinate level of object boxes in the example. The column that is used for the legend text returns the object state of the current object.

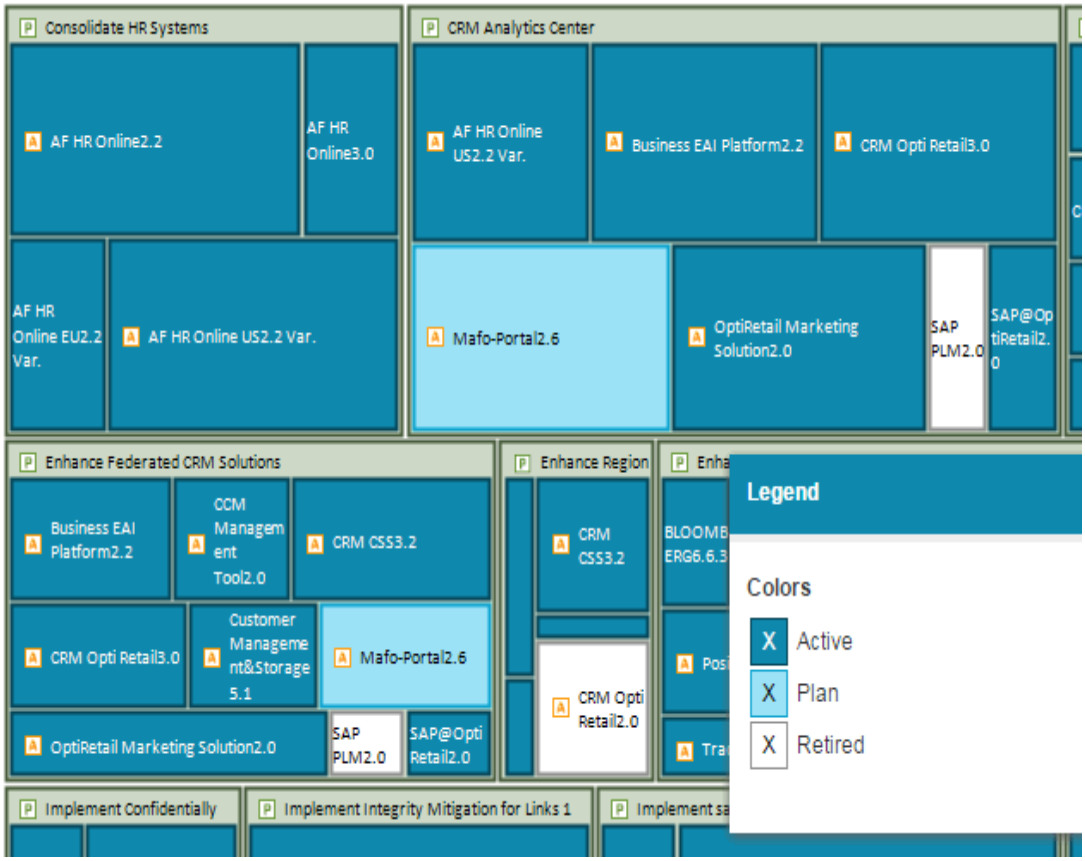
The color of the object boxes of the superordinate level is not changed.

```
SELECT proj.REFSTR AS 'proj.REFSTR', proj.NAME AS 'proj.NAME',
app.REFSTR AS 'app.REFSTR', app.NAME + app.VERSION AS 'Application',
CAST(indproj.VALUE AS Integer) AS 'proj.BoxSize', CAST(ind.VALUE AS
Integer) AS 'BoxSize',

app.OBJECTSTATE As 'Legend',

CASE app.OBJECTSTATE WHEN 'Plan' THEN '#9DE2F7' WHEN 'Retired' THEN
'#FFFFFF' ELSE '#0E89AD' END As 'BackColor',

CASE app.OBJECTSTATE WHEN 'Plan' THEN '#11A6D0' WHEN 'Retired' THEN
'#919191' ELSE '#07475A' END As 'BorderColor',

CASE app.OBJECTSTATE WHEN 'Active' THEN '#FFFFFF' ELSE '#000000' END
As 'ForeColor'

FROM ....
```

In the attributes of the root node of the report assistant, the attributes **Background Color Column**, **Foreground Color Column**, **Border Color Column**, and **Color Legend Column** of the report are set to the names of the columns containing the respective information:

Data	
Image Column	proj.NAME
Value Column	BoxSize
Back Color Column	BackColor
Fore Color Column	ForeColor
Border Color Column	BorderColor
Color Legend Column	Legend

In the resulting report, the object boxes for applications are displayed in different colors according to their object state and the color coding is explained in the legend:



Defining Navigation from the Report to Alfabet Views

By default, the object profile of the object represented by the box opens when the user double-clicks a box in the report or uses the Navigate button in the toolbar of the report. Alternatively, the rectangular treemap report can be configured to open another configured report, an object profile or a standard Alfabet view with the object represented by the box as base object. You can define an alternative navigation target for both or only for a single level of the object boxes. If no navigation target is defined for one level, the default is used and the object profile of the object opens.

Navigation to a defined configured report from the configured chart report requires the following configurations:

1) In the native SQL query defined as root query of the report, a column must be added for the definition of the navigation target. If you want to define the navigation target for both levels of

object boxes in the report, you must specify two columns, one with the navigation target definition of the superordinate level and one with the navigation target definition if the subordinate level and then combine the two columns into one with a `JoinColumns` instruction.

The syntax required for definition of the view is the following:

`View=`*`ViewType`*`:`*`ViewName`*

The `ViewType` can be one of the following:

- ***Report*** for a configured report
- ***GraphicView*** for a standard Alfabet view
- ***ObjectView*** for an object profile

`ViewName` is the name of the view.

If the navigation target is a configured report, parameters values can be handed over to the view that opens.

The query of the report must contain each parameter definition as a variable defined as

`@`*`ParameterName`*

Please note that the old syntax of the Alfabet query language defining parameters as:`ParameterName` is not supported.

The parameters can be used in `WHERE` clauses of the query without defining a filter for the report, because the parameter values are already defined in the navigation link.

Optionally, filters can be defined for the report to allow the user opening the report to alter the parameter values. In this case the parameter values in the navigation link are used as default values for the filter when opening the report. The filter settings are also stored in the user context settings for the report. That means, when the user first opens the report via drilldown from a chart report and then afterwards opens the report in any other context, For example, via the **Reports** functionality, the report will open with the last filter settings derived from the drill-down.

For each parameter the following string must be added to the navigation target definition:

`&`*`ParameterName=ParameterValue`*

The setting of three parameters would result in the following navigation link:

`View=`*`ViewType`*`:`*`ViewName`*`&`*`ParameterName1=ParameterValue1`*`&`*`ParameterName2=ParameterValue2`*`&`*`ParameterName1=ParameterValue2`*

2) In the attributes of the root node of the report assistant, enter the name of the column containing the navigation target specification into the attribute **View Column**.

In the example report, navigation from a box representing an application shall open the **Evaluations** page view instead of the application's object profile. The **Evaluations** page view is a standard Alfabet view with the name `ObjectEvaluation`. A new column for the definition of the target is added to the `SELECT` statement of the native SQL query defined as root query:

```
SELECT proj.REFSTR AS 'proj.REFSTR', proj.NAME AS 'proj.NAME',
app.REFSTR AS 'app.REFSTR', app.NAME + app.VERSION AS 'Application',
CAST(indproj.VALUE AS Integer) AS 'proj.BoxSize', CAST(ind.VALUE AS
Integer) AS 'BoxSize', app.OBJECTSTATE As 'Legend', CASE
```



```
app.OBJECTSTATE WHEN 'Plan' THEN '#9DE2F7' WHEN 'Retired' THEN
'#FFFFFF' ELSE '#0E89AD' END As 'BackColor', CASE app.OBJECTSTATE
WHEN 'Plan' THEN '#11A6D0' WHEN 'Retired' THEN '#919191' ELSE
'#07475A' END As 'BorderColor', CASE app.OBJECTSTATE WHEN 'Active'
THEN '#FFFFFF' ELSE '#000000' END As
'ForeColor','View=GraphicView:ObjectEvaluation' As 'ViewDefinition'

FROM ....
```

In the attributes of the root node of the report assistant, the attribute **View Column** of the report is set to the name of the column containing the navigation target definition:

Defining a Layered Diagram Report

Layered diagram reports are designed in the **Report Assistant**. For information about how to open the **Report Assistant**, see Creating a New Graphic Report. For more information about how to work with the **Report Assistant**, see Configuring the Report with the Report Assistant.

The content and layout of a layered diagram report is defined by adding elements to the explorer of the **Report Assistant** and setting the attributes in the attribute pane of the **Report Assistant** window.

The following table lists the types of explorer node elements that can be added to the report configuration to specify the content of the report. Information is provided in the table about the purpose of each explorer node element:

Explorer Node Element	Required to Configure:
Root Node	General layout of the report
Root Queries	Container for **Query** elements that define the first layer or the overall content of the report depending on the type of query definition selected for the report. For more information about the query definitions, see Defining the Objects to Display in the Report.
Query	Definition of the content of the first layer or the overall content of the report when specified as a sub-element of the **Root Queries** node. Defines the choice of elements displayed in the layer below the current layer when specified as a sub-element of the **Class Entry** node.
Class Entry	Definition of the object classes included in the report.

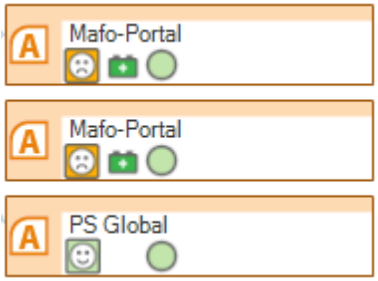
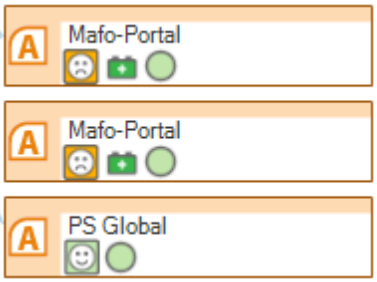
Explorer Node Element	Required to Configure:
Color Rule	Definition of the colors used in the layered diagram report and the rules that define which objects are displayed with which color.
Size Rule	Definition of three different heights for the boxes representing the objects in the layered diagram report and the rules that define which objects are displayed with which height.
Indicator Rule	Definition of one or multiple icons displayed in the boxes representing the objects in the current cell of the lane report. The icons can either represent an indicator or any aspect defined via a query. For more information, see the section Defining Indicator Rules.

Defining the Basic Design of the Layered Diagram Report

The basic design of the report is configured directly in the attributes of the root node element. The table below lists all attributes that specify the general layout of the report in the root node element.

Attribute	Description
Layout	Defines whether this is a layered diagram or a treemap report. Set this attribute to `LayerDiagram` to define a layered diagram report.
Show Legend	Defines whether a legend is shown for the color coding, indicators, and the size of the objects in the report.
Use Mandates	Defines whether the implementation of the mandate capability must be taken into consideration to view objects in the report. If set to `True`, a user can only view objects that have a mandate specification identical to the the user's mandate.
Margin	Defines the space between the layered diagram and the border of the presentation object and between the layers of the diagram in number of pixels.
Area Background Color	Defines the background color of the layers in the report. For more information about defining colors, see Defining Color Attributes.
Space	Defines the spacing between the boxes representing the objects in a layer in number of pixels.

Attribute	Description
Medium Height	Defines the height of the boxes that are displayed for all objects in number of pixels.
Item Border Width	Defines the width of the border of the boxes in the report. The default value is 1 point. When using the border color as object property indicator via a color rule, it is recommended to set the border width to a higher value.
Show Indicator	Select `True` to display an indicator icon in the upper right corner of the object boxes or multiple indicator icons at the bottom of the object boxes. Indicator icons are one displayed in the object boxes if an indicator rule is defined that specifies the dipslay of an indicator for the object. Indicator icons can represent an indicator value from the object evaluation or any other aspect that can be defined for the object via a query. Select `False` to exclude the display of the indicator icons for the object box column. For more information about the definition of indicator rules to assign icons to object boxes, see Defining Indicator Rules.
Show Multiple Indicators	Defines whether display of icons defined via indicator rules is limited to one icon per object box or whether multiple icons can be assigned to an object box. If **Show Multiple Indicators** is set to `False`, a single icon can be displayed on the right of the object box. If multiple icons are found by indicator rules, the first icon found is assigned and all other icons are ignored. If **Show Multiple Indicators** is set to `True`, one or multiple icons are displayed from left to right at the bottom of the object box. If the number of icons that are assigned to the object box via indicator rules exceeds the space available for display of icons in a single row, the first icons found are displayed and, if the object box space is exceeded, any other icons are ignored. The way icons are placed in the object box also depends on the attribute **Indicator Fill Policy** of the **Item** node. For more information about the placement of icons assigned via indicator rules, see the section Defining Indicator Rules.
Indicator Fill Policy	If **Show Multiple Indicators** is set to `True`, this attribute defines how icons assigned to an object box are placed in the row on the bottom of the object box: • `ByIndex`: This setting shall be used if multiple indicators are defined via multiple indicator rules, each returning a single icon. The position of the icon resulting from each indicator rule is defined by means of the attribute **Indicator Index** of the respective indicator rule that must be set to an unsigned integer starting with 0 for the left most position. If an **Indicator Index** attribute has been defined for each indicator rule and an indicator is not set for a specific object, the position of this indicator icon in the box will be left empty and the position of all other indicator icons will not be changed:

Attribute	Description
	 If the order of the **Indicator Index** attribute is not defined, the placement of indicators will be arbitrary. This method is designed for positioning only one icon per indicator rule. If an indicator rule returns multiple icons, only one of the icons is displayed. • `LeftAlign`: This setting shall be used if a single indicator rule is defined that returns multiple icons. It can also be applied when multiple indicator rules each return a single icon, but the position and order of icons is not important to understand the report. The Icons are displayed in the order of occurence within the indicator rule(s) from left to right. If a different number of indicator icons is returned, the remaining icons are moved to the left:  If the indicator icons are all found by a single indicator rule, the order of icons depends on the order of results in the result dataset of the query of the indicator rule and may be different for different objects if no sort order is defined in the query.

The design of the boxes representing objects in the configured report can be changed from the standard design with a white rectangle as background for text within the colored object box to fully colored object boxes. For more information see Defining the General Display of Object Boxes in Treemap, Diagram, Lane and Layered Diagram Reports.

Defining the Objects to Display in the Report

Each layer in the layered diagram report displays objects of an object class that is related to the object class of the layer displayed above the current layer.

The objects displayed in the layers of the report are found on the basis of queries defined for the elements of the report. The queries can either be native SQL queries or Alfabet queries.

If you are not familiar with Alfabet queries, see the chapter Defining Queries.

For special rules that apply to the definition of native SQL queries in the context of configurations for Alfabet, see the section Defining Native SQL Queries.

The information displayed in the object boxes of the report is identical to the column headers that result from a tabular output of the defined query.

Report columns that are defined for technical reasons are not displayed in the object boxes in the report. This includes:

- The first column of the output of a native SQL query. This column must specify the `REFSTR` of the object for technical reasons and is ignored in the visible output.

There are two methods to define a layered diagram report:

- The complete content of the report is defined in one query resulting in a grouped dataset.
- A separate query is added to the report configuration to find the objects in each layer of the report.

  When separate queries are defined, a high number of queries must be executed by the database to display the report. This can lead to performance issues. Therefore, it is recommended to built the report on a grouped dataset defined in a single query.

Defining a Layered Diagram Report With a Single Query

The content of all layers in the layered diagram is defined in one query in a **Query** element that is defined as sub-node of the `RootQueries` node. The query finds the objects that are displayed in the first layer and all subordinate layers in a grouped dataset with each layer corresponding to a grouping level in the query.

The `FIND` class of the Alfabet query or the first column of the output of the native SQL query must find objects of the class displayed in the first layer of the report.

One **Class Entry** child element must be added to the explorer root node for each object class that is included in the report with the mandatory attribute **Class Name** that defines the name of the specified object class.

The **Class Entry** elements may not contain a sub-element **Query**.

Optionally, an icon can be selected in the attributes of the **Class Entry** element. The icon is then displayed in the object boxes of the defined object class in the report.

Defining a Layered Diagram Report With One Query Per Object Class

For each layer, the objects in the layer must be specified by the following:

- An element **Class Entry** that specifies the object class in the layer and the default display of object boxes.
- An element **Query** that is child element of the **Class Entry** element for the objects in the layer above. In the element **Query**, an Alfabet query or a native SQL query specifies which objects are displayed in the next layer below the layer defined by the **Class Entry** element.

For the first and last layer, the following must be taken into account:

- To define which objects are displayed in the top layer of the diagram, an element **Query** is specified directly as child element of the `Root Queries` element.
- The **Class Entry** element for the bottom layer of the diagram does not contain a child element **Query**, because no objects for a next layer must be specified. Nevertheless, the **Class Entry** element must be specified for the bottom layer.

The object class defined for a layer of the report is the same as the base class for the objects displayed in the next layer. The Alfabet query or native SQL query defining the object class displayed in the next layer must contain the parameter `BASE` in a `WHERE` clause to map the objects to the objects of the current layer. `BASE` is identical to the `REFSTR` of the objects in the current layer.

This specification is not mandatory for the top layer objects. The attribute `BASE` can only be used in the Alfabet query or native SQL query defining the top layer objects if the report is assigned to an object class with the attribute **Apply To Class**.

The use of the parameter `BASE` depends on the setting of the attribute **Cascading** in the element **Query**:

- If **Cascading** is set to `True`, a query in a layer cannot only refer to an object of the directly superordinate layer, but also to objects of all other superordinate layers. The parameters to refer to objects of the superordinate layers are then `BASE0` to refer to the directly superordinate layer, `BASE1` to refer to the layer above the directly superordinate layer and so on.
- If **Cascading** is set to `False`, a query in a layer can only refer to an object of the directly superordinate layer using the parameter `BASE`.

To refer to objects of any superordinate layer, you can either set **Cascading** to `True` and user the `BASE`<number> parameters or you can define the attribute **Param Name** of the **Class Entry** elements and refer to the objects defined by a class entry element with the Param Name string as parameter. For example, if a Class Entry element for applications has a Param Name attribute set to APP, you can define, For example, the following `WHERE` clause in an Alfabet query of a sub-ordinate layer to refer to the `REFSTR` of the current application:

```
WHERE Application.Name =:BASE
```

or in native SQL queries:

```
WHERE APPLICATION.NAME = @BASE
```

The table below lists all available attributes in the **Query** element.

Attribute	Description
Name	Defines the name of the **Query** element displayed in the explorer of the **Report Assistant**.
Alfabet Query/ Query as Text/ Native Sql	Defines a query to find the objects that are displayed in the report. The way the query must be defined depends on the location of the **Query** element in the report.
Cascading	Only relevant for layered diagram reports defined with one query per layer: If **Cascading** is set to `True`, a query in a layer cannot only refer to an object of the directly superordinate layer, but also to objects of all other superordinate layers. The parameters to refer to objects of the superordinate layers are then `BASE0` to refer to the directly superordinate layer, `BASE1` to refer to the layer above the directly superordinate layer, and so on. **Note**: It is not possible to refer to the object class that the report is assigned to with the property **Apply To Class** of the report with the parameter `BASE`<number>. Only the query of the first layer in the report can refer to the base object the report is applied to with the parameter `BASE`. **Cascading** must be set to `"false"` for that layer. If **Cascading** is set to `false`, a query in a layer can only refer to an object of the directly superordinate layer using the parameter `BASE`.

The table below lists all available attributes in the **ClassEntry** element.

Attribute	Description
Class Name	Defines the object class specified in this ***ClassEntry*** element. For the object classes that allow stereotype specifications (For example, `Project` and `Domain`), the XML attribute `ClassName` can specify an object of a specific stereotype. To define an XML element ***Query*** for a stereotype, the XML attribute `ClassName` must be *`ClassName:StereotypeName`* For example,: *`Project:StatementOfWork`*
Icon	Defines the icon to be displayed for the object in the object boxes. An icon can be selected from a drop-down list.
Param Name	Defines the string that can be used as parameter in queries of subordinate layers to return the `REFSTR` of the current object of this layer.

Displaying Evaluation Criteria By Size or Color Variation or Addition of Indicator Icons

Evaluation criteria can be added to the report:

- Rules can be defined that assign colors on the basis of property values of the current object. For example, coloring can be different for boxes of objects that are planned, active, or retired. For more information about the definition of color rules, see Defining Color Rules.
- Rules can be defined that trigger the display of an indicator icon in the object boxes of selected objects. For more information, see Defining Indicator Rules.
- Rules can be defined that lead to objects with defined property values to be displayed in smaller or larger object boxes. Size rules are described in the following.

The boxes in the report can be displayed with three different heights. You can specify a size rule for each box height that allows you to display objects with different attribute values with different box sizes. For example, you can display application groups in a report in a different size determined by the number of applications assigned to the application group. If the number is lower than 5, a small box is displayed. If the number is higher than 20, a big box is displayed. Other application groups are displayed with a medium box height.

To define the size of the boxes in the report, you have to specify the following:

- The height in pixel for the small, medium, and large box size. These are defined by means of the attributes of the root node element.
- The size rules that define which objects are displayed with which box size. Size rules are defined in the element **Size Rule**, which is a child element of the root node element.

You can define two different types of size rules:

- `ObjectQuery`: The query defined for the size rule returns the `REFSTR` of the object to that the box height shall be applied. The size that shall be applied to the object boxes is defined separately via attributes of the size rule.

  When defining an Alfabet query, no Show property definition is required. The `REFSTR` of the `FIND` object class is returned.

  When defining a native SQL query, the `SELECT` statement of the query must have the `REFSTR` of the relevant object class defined as the first argument. No further arguments are required in the `SELECT` statement.

- `SizeQuery`: The query defined for the size rule not only returns the `REFSTR` of the objects that the size shall be applied to but also the size specification that shall be applied.

  `SizeQuery` rules require the specification of a native SQL query.

  The `SELECT` statement of the SQL query must return the following in the given order:

  - the `REFSTR` of the current object
  - a string that is used as caption for the size assignment in the legend of the report
  - the size specification as string. Allowed values are `Small`, `Large` and `Medium`.

  For example,:


```
SELECT DOMAIN.REFSTR, "Legend text", "Large"
```

- The definition of a size query in a size rule has the following advantages as compared to an object query: The number of overall size rules required for a report can be reduced. For example, if you want to apply a different box size to applications in a different object state, you can define the size for all object states in one query while a separate size rule for each object state would be required when using object queries.
- Dynamic size assignment can be applied to a report. You can for, example, define a query that sizes object boxes according to the indicator values assigned to the objects. In the report, you can define a filter that allows the user viewing the report to choose an indicator type. The size of object boxes is then reassigned according to the values for the selected indicator.

The specification of box sizes and size rules is optional. If no box sizes are defined, all boxes will be displayed in a default height of 12 for all size rules. If no size rules are specified, all objects will be displayed with the defined medium box size.

When you define the queries for the size rules of the report, make sure that all objects can be assigned explicitly to one size rule. If this is not the case, the following applies:

- Objects that correspond to more than one size rule are displayed in the size of the first rule written in the XML object that applies.
- Objects that do not correspond to any of the defined size rules are displayed in medium size. You cannot distinguish between these objects and objects corresponding to the medium size rule in the treemap report.

The table below lists all attributes for the specification of box sizes in layered diagram reports.

Attribute	Description
In the root node element	
Small Height	Defines the height of the boxes that are displayed for objects corresponding to the size rule with **Size** ="Small".
Medium Height	Defines the height of the boxes that are displayed • for objects corresponding to the size rule with **Size** ="Medium". • for all objects when no size rules are specified.
Large Height	Defines the height of the boxes that are displayed for objects corresponding to the size rule with **Size** ="Large".

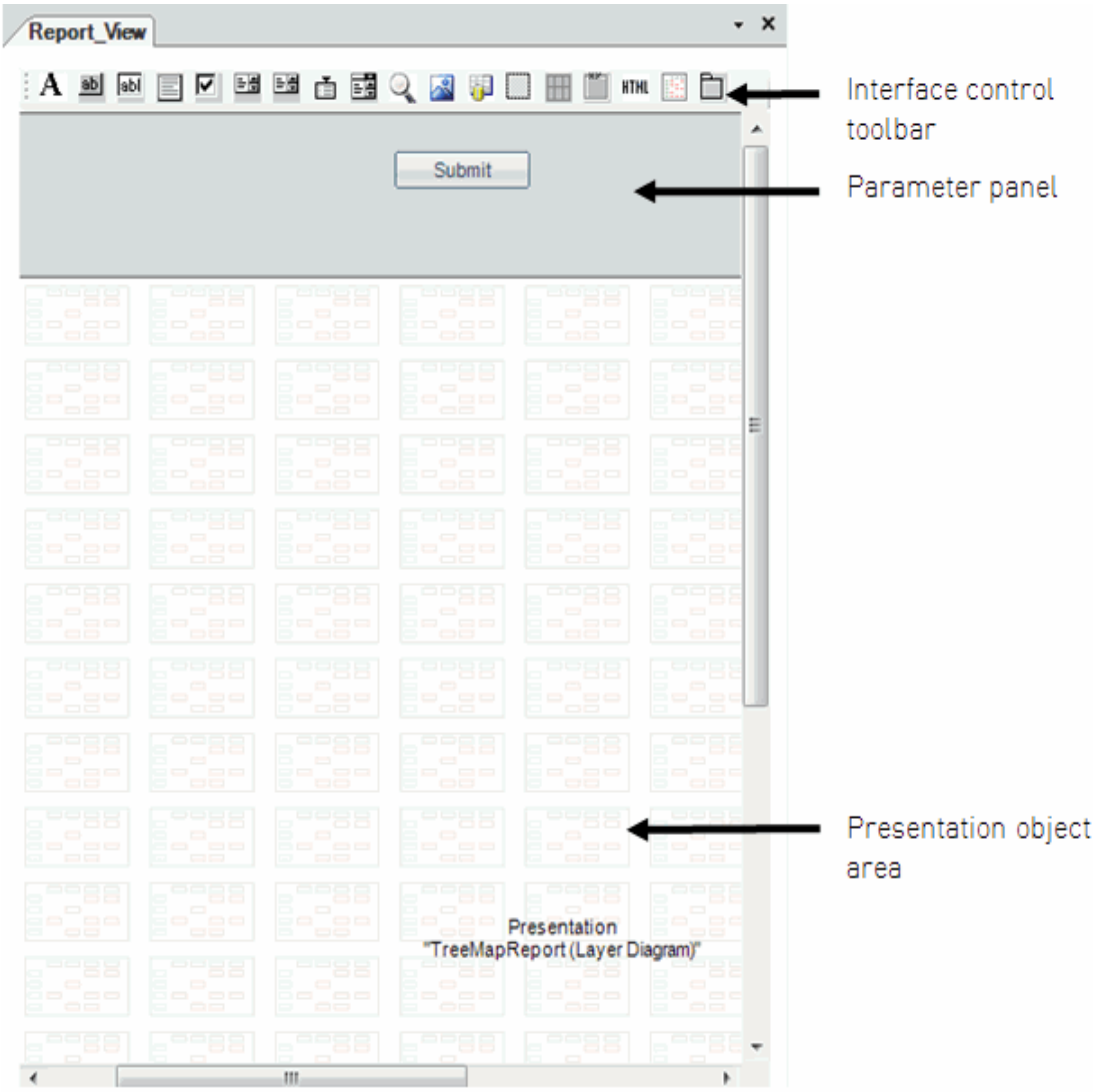
Attribute	Description
In the Size Rule element	
Name	Defines the title displayed for the size in the legend and for size rules of the `TypeObjectQuery` the caption displayed for the size rule in the explorer of the **Report Assistant**. **NOTE**: The attribute **ShowLegend** of the Root node element must be set to `true` to display the legend. If this element is set to `false`, you do not need to specify a name for the size rule.
Type	Select `ObjectQuery` to define the objects that the size rules apply to with a query and the size specification with the attributes of the size rule. Select `SizeQuery` to define both the size specification and the objects to that the size rule applies with a query.
Size	Defines the size of the boxes of objects corresponding to this size rule if the `Type` of the size rule is `ObjectQuery`. The pixel size of the boxes is specified with the **Small Height**, **Medium Height** and **Large Height** attributes of the root node element.
Alfabet Query/ Native Sql/ Query as Text	If the `Type` of the size rule is `SizeQuery`, define a native SQL query in the Native Sql attribute or Alfabet query in the AlfabetQuery or Query as Text attribute. The query must return the `REFSTR` of the objects that the size rule shall apply to. If the `Type` if the size rule is `SizeQuery`, define a native SQL query in the Native Sql attribute that returns: • the `REFSTR` of the current object • a string that is used as caption for the size assignment in the legend of the report • a string that defines which box size applies. Valid values are `Medium`, `Small` and `Large`.

Defining HTML Reports

The **Report Assistant** is not available for HTML reports. To define or edit the HTML report, you must define an XML object in the attribute window of the presentation object of the report.

To edit the presentation object in the report view of the report:

1) In the explorer, double-click the custom report view of the report. The view editor opens.



2) In the editor, click the **Presentation** area. You will see the attributes of the presentation object in the attribute window.

3) In the **SubType** field of the attribute window, select `HTMLView`.

4) In the **Source** field select `Object_Indicators_Html_Report`.

5) In the **Parameters** field, paste the following code:

```
<!DOCTYPE HTML>
<html>
<head>
    <meta content="text/html; charset=utf-8" http-equiv="Content-Type" />
    <style type="text/css">
    table#HTML_REPORT_TABLE
    {
    border: Solid 1px #cecfce;
    border-collapse: collapse;
    border-spacing: 0px;
```



```
    font-family: Verdana;
    color: Black;
    font-size: 7pt;
    }
    td#HTML_REPORT_TD
    {
    border: Solid 1px #cecfce;
    }
    </style>
</head>
<body>
    <div id="IndicatorReport" style="overflow-x:auto;overflow-y:visible">
    <table id='HTML_REPORT_TABLE' cellpadding="4" cellspacing="0"
    style="width: 100%;">
    <col width='50%' />
    <col width='30%' />
    <col width='30%' />
    <!--INDICATOR ROW-->
    <tr height="20px">
    <td id="HTML_REPORT_TD" align="left" style="background-
    color:#d6dfde;color:#161530;">
    <b><!--INDICATOR NAME--></b>
    </td>
    <td id="HTML_REPORT_TD" style="border-right:0px;">
    <!--INDICATOR PICTURE-->
    </td>
    <td id="HTML_REPORT_TD" align="right" style="border-left:0px;">
    <!--INDICATOR VALUE-->
    </td>
    </tr>
    <!--INDICATOR ROW END-->
    </table>
    </div>
</body>
</html>
```

6) In the attribute window, click the **Browse** button in the **XML Def** field. An editor opens.

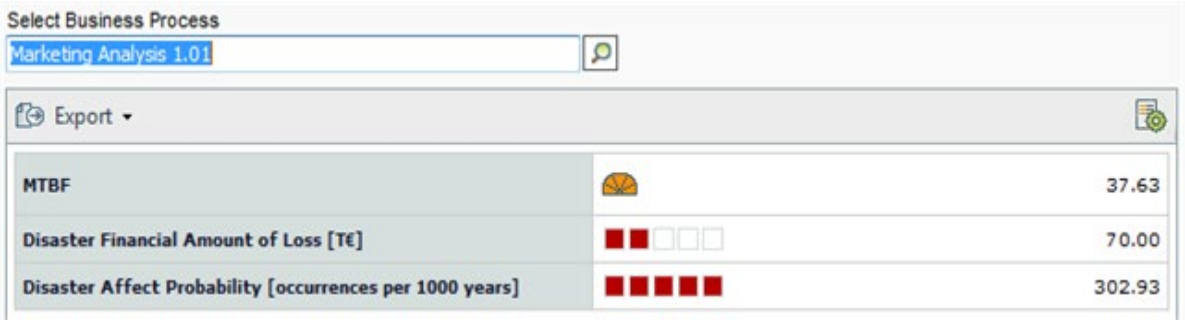
7) Define the content and layout of the report in the editor as described below.

Please note the following when configuring the XML definition of an HTML report:

- It is not possible to specify XML elements or attributes other than the ones described in the section below.
- XML is case-sensitive.
- Some definitions may be left out if they are not required in the configured report. For example, it is possible to specify configured reports that do not display an indicator in the object boxes.

The content and layout of an HTML report is defined in the XML element ***ReportDef***. The general design of the grid is defined via the attributes of the ***ReportDef*** element. The ***ReportDef*** element can contain one or multiple ***Indicator*** elements. For each ***Indicator*** element, a new row is added to the HTML table that is generated for the report. Each row of the report displays a caption for the indicator, a symbol that allows a quick overview over the criticality and the numerical value of the indicator. The ***Indicator*** element defines which indicator is displayed in a row and which caption and symbol are used for display.

The following example displays an example for an HTML report that displays indicators for a selected business process.



The filter for the selection of the business process displayed in the report is generated automatically because the report is applied to the class Business Process with the attribute **Apply To Class** in the attribute window of the report.

The indicators listed in the report and the caption and image used for each indicator is configured in the XML object of the presentation object for the report. The root element `ReportDef` of the XML object contains one child ***Indicator*** element for each indicator displayed in the report.

```
<ReportDef
   GaugeBackColor="#b50000"
   GaugeBorderColor="#d6dfde"
   GaugeWidth="10"
   GaugeHeight="10"
   GaugeSpace="2"
   GaugeCount="5">
      <Indicator
         Type="Indicator"
         Name="MTBF/Max. Mean Time Between Failures"
         Caption="MTBF"
         ReportType="Icon" />
      <Indicator
```



```
            Type="Indicator"
            Name="Risk Management (Business Process)/Disaster
            Financial Amount of Loss [T€]"
            Caption="Disaster Financial Amount of Loss [T€]"
            ReportType="Gauge"
            MaxValue="200"/>
        <Indicator
            Type="Indicator"
            Name="Risk Management (Business Process)/Derived Disaster
            Affect Probability [occurrences per 1000 years]"
            Caption="Disaster Affect Probability [occurrences per 1000
            years]"
            ReportType="Gauge"
            MaxValue="200"/>
</ReportDef>
```

Please note the following when configuring the images representing the indicators in the report:

- An icon can only be displayed in the report if the indicator type has been configured to display icons rather than numerical values. Indicator types are specified in the **Evaluations and Portfolios** functionality of Alfabet. For information about the configuration of indicator types and the icon gallery used to visualize their values, see the section *Configuring Indicator Types for an Evaluation Type* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.
- All gauge graphics in the report have the same coloring and graduation. The look of the gauge element is directly defined in the ***ReportDef*** root element of the XML object. The maximum value of the gauge is defined separately for each indicator in the respective Indicator element.
- The report can display up to 5 indicators. If you want a higher number of indicators to be displayed directly in an object view, you must display indicators in any of the other types of configured reports, assign the report to the object view, and define it to be expanded in the object profile.

The table below lists all XML elements (bold) and their attributes for the specification of indicators in HTML reports and effect thereof.

Element/ Attribute	Allowed Values or Data Types	Default Values / Mandatory	Description
ReportDef			
`GaugeBack-Color`	HTML valid color codes (e.g.	`grey`	Defines the color of the boxes representing shares of the maximum value of the gauge reached by the indicator value displayed in the gauge. The boxes representing shares of the maximum value of the gauge not reached by the indicator are white.

Element/ Attribute	Allowed Values or Data Types	Default Values / Mandatory	Description
	hexadecimal)		
`GaugeBorderColor`	HTML valid color codes (e.g. hexadecimal)	`black`	The border colors of the boxes representing shares of the maximum value of the gauge.
`GaugeWidth`	width in mm	12	Specifies the width of each gauge field in mm (1 mm = 4 pixel)
`GaugeHeight`	height in mm	12	Specifies the height of each gauge field in mm (1 mm = 4 pixel)
`GaugeSpace`	distance in mm	2	Specifies the distance between two gauge fields in mm (1 mm = 4 pixel)
`GaugeCount`	number of fields	5	Specifies the number of fields of a gauge. Each field represents a share of the maximum value of the gauge that is specified with the attribute **MaxValue** of the **Indicator** element. For example, if GaugeCount is 6, each box represents 1/6 of the maximum value. For a maximum value of 600, the first field represents the numerical values between 1 and 100, the second field represents the values between 101 and 200 and so on. A field is colored if the indicator value is within or higher as the range of values represented by the box.
Indicator			
`Type`	`"Indicator"`	`Indicator`	The indicator type must be set to `"Indicator"`.
`Name`	Name of indicator	**MANDATORY**	The name of the indicator. **Note**: The name of the indicator is the value of the property Name of the class `Indicator`. It is identical to the name of the evaluation type and the name of the indicator type of the indicator separated with a slash:

Element/ Attribute	Allowed Values or Data Types	Default Values / Mandatory	Description
			EvaluationTypeName/IndicatorTypeName
Caption	String	**MANDATORY**	The caption displayed for the indicator in the table.
ReportType	"Icon" "Gauge"	Icon for indicators configured to use an icon gallery. No graphic display for indicators without icon gallery specification.	Defines the graphic display of the indicator value in the report: Select Icon to use the icon gallery of the indicator or select Gauge to display the fulfilled share of a configured maximum value for the indicator. **Note**: If you select Icon, the indicator must be configured to use an icon gallery to display its value in evaluations. This is done in the **Evaluations and Portfolios** functionality in Alfabet. For more information, see the section *Configuring Indicator Types for an Evaluation Type* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*. **Note**: If you select Gauge, you must also specify the ***MaxValue*** attribute of the Indicator element. Optionally, you can specify the coloring and size of the gauge with the attributes of the ***ReportDef*** element.
MaxValue	Integer	MANDATORY for reports of the ReportTypeGauge	For reports of the ReportTypeGauge only: The maximum numeric value displayed in the gauge. The gauge displays fields that represent equal shares of the maximum numeric value. Shares that are covered by the indicator value are colored in the gauge.

Defining Widget Reports

For widget reports, the definition of design and content is partly decoupled. Widget reports are based on a layout that is defined in a separate configuration object widget . The widget configuration object contains placeholders for images and text. The widget configuration object must be defined prior to defining the widget report. It is then referenced by the widget reports and the placeholders are filled with data via the widget report.

A widget configuration object can be referenced by multiple widget reports, each defining a different content for the same layout.

Do the following to define a widget report:

- Defining a Widget

Defining a Widget

Widgets are configured in the **Presentation** tab of Alfabet Expand:

1) In the **Presentation** tab of Alfabet Expand, right-click the **Widgets** node and select **New Widget** from the context menu.
2) In the attribute window of the new widget that opens on the right, define the following attributes according to demand:
   - **Name**: Change the default name to a unique, meaningful name. The name is used to identify the widget in technical processes and shall not contain whitespaces or special characters. It is also displayed in the widget selector of the widget report assistant.
   - **Description**: Enter a description that explains the purpose of the widget for other solution designers. The description is displayed in the widget selector of the widget report assistant next to the widget's name to ease widget selection.

Widgets are technically views and have multiple standards view attributes that are also available for configured reports and standard Alfabet views. Most of the available attributes are of no relevance for widgets and can be ignored. The respective settings are done on the level of the configured report.

3) Right-click the new widget in the explorer and select **Edit Widget** from the context menu. The widget opens in the middle pane of Alfabet Expand. Design elements are displayed in the toolbar of the widget.

4) Click into the widget area. The attribute window opens.

5) In the attribute window, define the overall size of the widget with the attributes **Height** and **Width**. The height and width are defined as an integer defining the number of pixels.

Please note that there are three different places where a size can be defined for a widget report:

- The widget can be resized.
- The configured widget report referencing the widget has a defined height and width.
- If the configured report is included in an object cockpit, guide page or guide view, the area reserved for displaying the report is configurable in size.

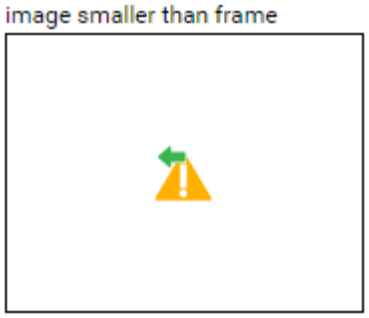
All height and width definitions should be identically to ensure that the widget is fully displayed.

- 6) Design the widget as described in the following sections: General Instruction for Adding and Positioning Elements on the Widget
  - Changing the Size of a Single Widget Element
  - Changing the Location of a Single Widget Element
  - Aligning the Size and Position of Multiple Widget Elements
  - romefnsfmDocking A Widget Element to the Borders of the Widget Area
- Adding Background Color and Borders to the Widget
  - Defining a Default Background Color on the Widget Definition
  - Defining a Panel Element for the Widget
- Using a Flow Panel or Table to Structure the Design Elements in the Widget
- Adding a Widget Picture Field to the Widget
- Adding a Widget Text Field to the Widget
- Changing a Widget Definition Already Used In Configured Widget Reports

Note the following about designing the widget:

- If you want to add a **Panel**, **Flow Panel** or **Table Layout Panel** as background element to the Widget, you must do this before placing images or text elements. The elements added to the widget cannot be moved to back or front. Elements added first are hidden under by overlapping elements added later.
- The widget designer does not display the widget elements exactly as displayed on the user interface.

  The font size is larger and the positioning of text and images within the elements is not always matching the position on the user interface. For example, if you add an image smaller than the **Widget Picture** element, the image is displayed in the upper left of the field, while in the report, the image is displayed centered.



- The design that you add to the widget is a default that may be overwritten with design specifications in the configured report using the widget.
- Not all attributes that are visible for elements added to the widget or for the widget itself have a visible effect on the widget design. For example, a background color defined for the widget itself will be ignored. Background colors are only taken over if defined on elements added to the widget. The documentation describes the valid settings. Attributes not described below may not have any effect on the widget layout.

7) In the toolbar, click **Save**  to save your changes.

The first of the following sections is giving information about how to generate and position elements. The subsequent section inform about the styling options available via the elements:

- General Instruction for Adding and Positioning Elements on the Widget
  - Changing the Size of a Single Widget Element
  - Changing the Location of a Single Widget Element
  - Aligning the Size and Position of Multiple Widget Elements
  - romefnsfmDocking A Widget Element to the Borders of the Widget Area
- Adding Background Color and Borders to the Widget
  - Defining a Default Background Color on the Widget Definition
  - Defining a Panel Element for the Widget
- Using a Flow Panel or Table to Structure the Design Elements in the Widget

- Adding a Widget Picture Field to the Widget
- Adding a Widget Text Field to the Widget
- Changing a Widget Definition Already Used In Configured Widget Reports

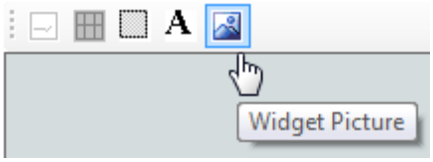
General Instruction for Adding and Positioning Elements on the Widget

The toolbar in the design area of the widget displays all elements that can be added to the widget:

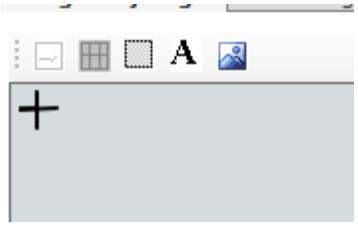
- **Panel** : The **Panel** element adds a background element to the widget to define background coloring and borders. For information about the layout configuration options available via **Panel** elements, see Adding Background Color and Borders to the Widget.
- **Flow Panel** : The **Flow Panel** elements can be used to define a layout that adapts element sizes to the size of the element content. For information about the layout configuration options available via **Flow Panel** elements, see Using a Flow Panel or Table to Structure the Design Elements in the Widget.
- **Table Layout Panel** : The **Table Layout Panel** elements can be used to structure elements on the widget in a tabular layout. For information about the layout configuration options available via **Table Layout Panel** elements, see Using a Flow Panel or Table to Structure the Design Elements in the Widget.
- **Widget Picture** : The **Widget Picture** element adds a placeholder for images and optionally a default image to the widget. The actual image displayed to the user on the Alfabet user interface is added via the configured widget report. For information about the layout configuration options available via **Widget Picture** elements, see Adding a Widget Picture Field to the Widget.
- **Widget Text** : The **Widget Text** element adds a placeholder for text to the widget. The actual text displayed to the user on the Alfabet user interface is added via the configured widget report. For information about the layout configuration options available via **Widget Text** elements, see Adding a Widget Text Field to the Widget.

To add any of the elements to the widget:

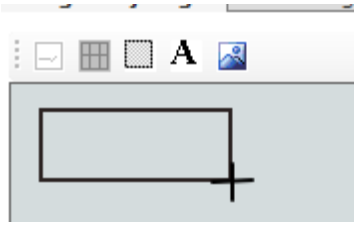
1) Click the element's icon in the toolbar of the design view:



2) In the widget area, click and hold the position where you want to place one corner of the element.



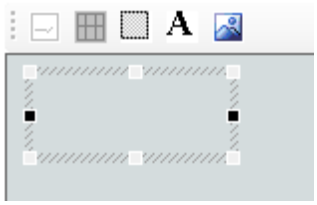
3) Hold the mouse key while you move the mouse pointer to the opposite corner position.



4) After having placed the element box, the attributes of the element are displayed in the attribute window and the design can be changed for the element by setting the attributes.

5) In the toolbar, click the **Save** button to save your changes.

After releasing the mouse click when placing the widget element, the field is displayed highlighted in the widget area:



Clicking anywhere into the widget area will move the focus away from the element. For **Panel**, **Flow Panel** and **Table Layout Panels** dotted lines will be displayed on non focused elements. For **Widget Text** and **Widget Picture** elements, the field itself is not visible, but a caption is displayed by default that allows you to locate the field. Please note that the caption of the **Widget Text** element is displayed in the text field and the caption of the **Widget Picture** element is displayed above the image field. You must click on the caption of a **Widget Text** element to activate the text field and on the empty area below the caption of a **Widget Picture** element to activate the image field.

The position and size of placed widget elements can be changed anytime with one of the following mechanisms:

- Changing the Size of a Single Widget Element
- Changing the Location of a Single Widget Element
- Aligning the Size and Position of Multiple Widget Elements
- romefnsfmDocking A Widget Element to the Borders of the Widget Area

Changing the Size of a Single Widget Element

There are two methods to change the size of a widget element in the widget area:

- Click the element and pull the handles to resize it.

Please note that the frames are not displayed while you pull the handles.

- Click the element in the panel to open the attribute window and change the following attributes in the section **Coordinates**:
  - **Height**: Define the height of the element as an integer.
  - **Width**: Define the width of the element as an integer.

  Please note that borders are drawn outside the element. If you define a width of 10 for an element with a left and right border with a width of 2, the overall width of the element is 14.

After changing the size, click the **Save** button in the toolbar to save your changes.

You can combine both methods for easy and correct sizing of elements in the widget area. After having resized the elements using the handles to define the basic layout of the widget, correct the size of the elements in relation to each other using the attributes in the section **Coordinates**. For example, you can define an equal **Width** for elements that are placed below each other or an equal **Height** for elements displayed in the same row.

Changing the Location of a Single Widget Element

There are two methods to change the location of a widget element in the widget area:

- Drag and drop the widget element to the correct location and click the **Save** button in the toolbar to save your changes.
- Click the element in the widget area to open the attribute window and change the following attributes in the section **Coordinates**:
  - **Left**: Define the distance from the left of the widget area.
  - **Top**: Define the distance from the top of the widget area.

  If the element is placed on one of the panel elements, the values for **Left** and **Top** are marking the distance to the border of the panel element instead to the border of the widget area.

After changing the field location, click the **Save** button in the toolbar to save your changes.

You can combine both methods for easy and correct placement of widget element in the widget area. After having used drag and drop to define the basic layout of the widget, correct the location of the widget element in relation to each other using the attributes in the section **Coordinates**. Widget elements that are

placed below each other should have the same value for the attribute **Left**. Widget element that are placed next to each other in the same row should have the same value for the attribute **Top**.

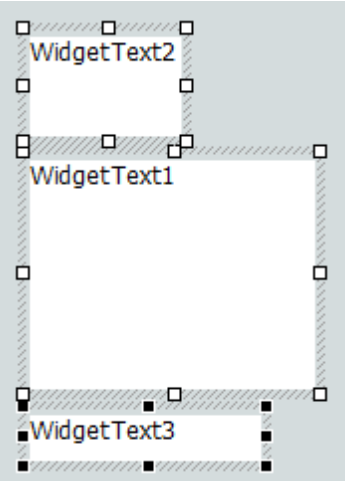
Aligning the Size and Position of Multiple Widget Elements

When you click a widget element on a widget area, you can see a menu **Format** in the menu bar of Alfabet Expand. Via the options in the **Format** menu the design of multiple widget in the widget area can be aligned to each other with regard to the position of the widget elements in the widget area and the widget size:

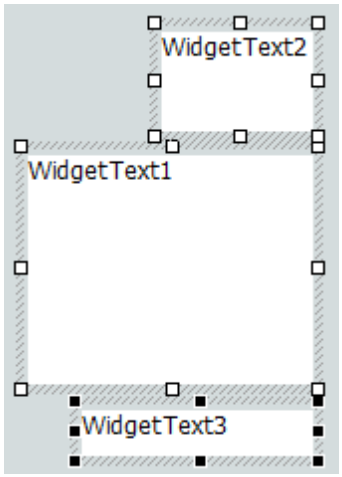
Aligning the Position of Multiple Widget Element

You can align multiple widget elements with one another to have an equal position for one of the borders of the widget area. For example, if you select multiple widget elements that are located next to each other in the same row, you can align the position of the top border to ensure that all widget elements are on the same horizontal axis.

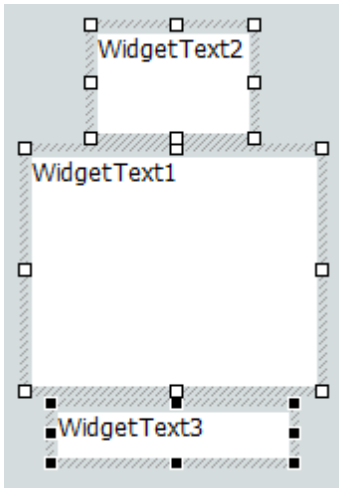
1) In the widget area, place one of the widget element in the correct position.
2) Select all widget element that you want to align with each other holding CTR + SHIFT while clicking on the widget element. The last widget element that you select is the widget element that determines the position.
3) In the toolbar, select:
   - **Format** > **Align Left** to align the position of the left border of all selected widget elements with the position of the last selected widget element.



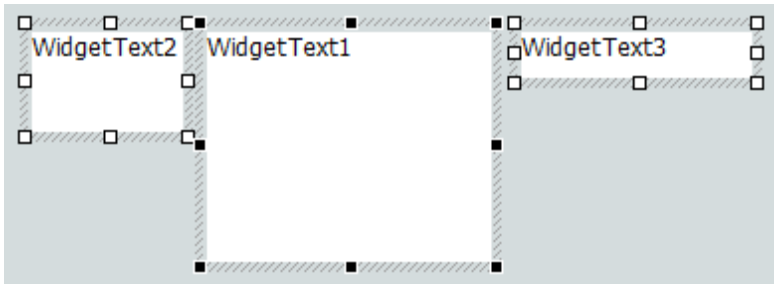
   - **Format** > **Align Right** to align the position of the right border of all selected widget elements with the position of the last selected widget element.



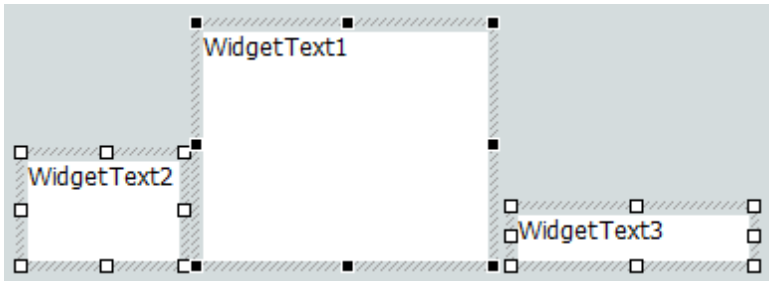
- **Format** > **Align Center** to align the horizontal middle position of all selected widget elements with the horizontal middle position of the last selected widget element.



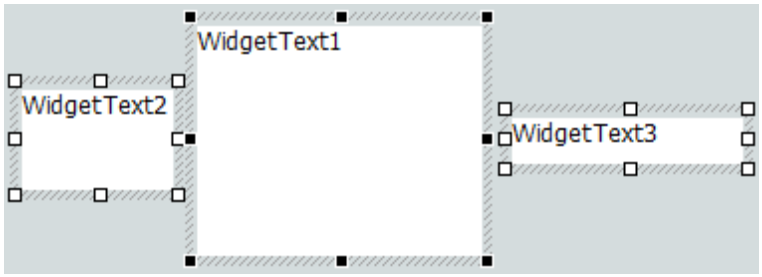
- **Format** > **Align Top** to align the position of the top border of all selected widget element with the position of the last selected widget element.



- **Format** > **Align Bottom** to align the position of the bottom border of all selected widget elements with the position of the last selected widget element.



- **Format** > **Align Middle** to align the vertical middle position of all selected widget elements with the vertical middle position of the last selected widget element.



4) In the toolbar, click the **Save** button to save your changes.

Aligning the Size of Multiple Widget Elements

The width or height or one or multiple selected widget elements can be aligned to the width or height of a selected widget element.

1) In the widget area, select all widget elements that you want to resize holding CTR + SHIFT while clicking on the widget elements. The last widget element that you select is the widget element that determines the resulting width or height of all widget elements.
2) In the toolbar, select:
   - **Format** > **Make Widths Equal** to adapt the widths of all selected widget elements to the width of the last selected widget element.
   - **Format** > **Make Heights Equal** to adapt the heights of all selected fields to the height of the last selected widget element.
3) In the toolbar, click the **Save** button to save your changes.

Aligning Spacing between Widget Elements

If multiple widget elements are placed in a horizontal row or a vertical column you can define an equal spacing between all fields in a single row or in a single column. The position of the first and the last widget element in the row or column is maintained and the fields in between are distributed equally in between.

1) In the widget area, select all filter fields that you want to distribute equally along a vertical or horizontal axis. Please note that only fields that are located next to each other on an axis shall be selected.
2) In the toolbar, select:

- **Format** > **Make Vertical Spacing Equal** to distribute the widget elements equally along the vertical axis.
- **Format** > **Make Horizontal Spacing Equal** to distribute the widget elements equally along the horizontal axis.

3) In the toolbar, click the **Save** button to save your changes.

Aligning the Widget Elements Along a Grid

The widget elements can be aligned along a grid that is invisibly drawn on the widget area. This method allows to align the position to the widget elements horizontally and vertically.

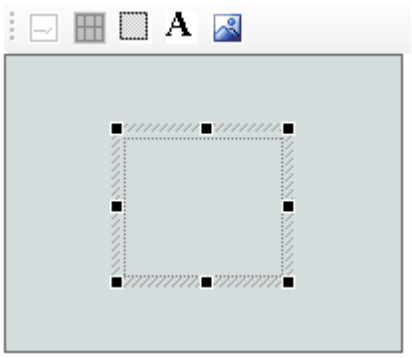
The top left corners of the widget elements are aligned with the respective crossing point of the grid lines left and above the respective widget element.

1) Select the widget elements that you want to align with along the grid by holding CTR + SHIFT while clicking the widget elements.
2) In the toolbar, select **Format** > **Snap to Grid**. The selected widget elements are positioned with the upper left corner matching the nearest crossing point of grid lines.
3) If the results are not as expected, you can alter the spacing between grid lines by selecting **Format** > **Grid Size...** in the toolbar and setting the spacing between grid lines in the window that opens. The default value is 2. It is recommended to select a number between 2 and 10 for a start and check the resulting effect on the widget first before moving on to higher numbers. If the spacing between grid lines is too high, the widget elements might be placed outside the widget area.
4) In the toolbar, click the **Save** button to save your changes.

romefnsfmDocking A Widget Element to the Borders of the Widget Area

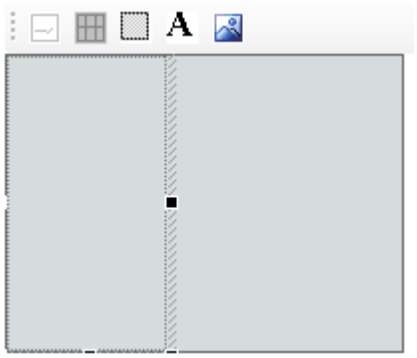
Widget elements can be configured to be docked to one or all borders of the widget area. If a widget element is docked to a border, the size changes to stretch along the whole border and the position of the widget element is fixed for directly starting with the attached border.

For example, a **Panel** element is placed in the middle of the widget area:

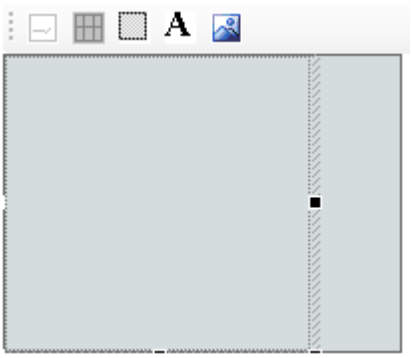


Configuring the widget element to be docked to the left border will change the size and position. The height is changed. The widget element is now stretching along the whole left border. The

position is changed. The widget element is now starting directly at the left border of the widget. The width of the widget element is kept.



After having docked the panel to the left border, the panel cannot be moved to a position further to the right any longer and the height cannot be changed. Only the width of the panel element is resizable:



This method for placement is especially useful for placement of **Panel**, **Tabular Layout Panel** and **Flow Panel** elements.

To dock a widget element to a single or all borders of the widget area:

1) Click the widget element in the widget area. The attribute window of the widget element opens.
2) In the attribute **Dock** a docking option from the dropdown list:
   - **Right**, **Top**, **Bottom**, **Left** will dock the widget element to the right, top, bottom or left border of the widget respectively.
   - **Fill** will dock the widget element to all borders. The element will cover the whole widget area and all resizing and re-positioning options will be deactivated.
   - **None** shall be selected if you do not want the element to be docked at all.
3) In the toolbar, click the **Save** button to save your changes.

If you set the **Dock** attribute, the attributes in the section **Coordinates** are altered according to your settings. The width and height of the element are not exactly matching the size defined for the widget area but are reduced by two. For example, if the widget area has a height of 400, a widget element with **Dock** set to **Left**, **Right** or **Fill** will have a height of 398.

This allows for a frame to be defined for the elements. Lines defined for the widget element borders are drawn outside of the element. The actual size of the widget element is then the widget size plus the width defined for the borders. The reduction of size during docking allows a border with a width 1 to be displayed around the widget element.

If you would like to define border lines with a higher width, you must reset **Dock** to **None** and reduce the defined widget element height and width accordingly. For example, if the widget area width is 400 and your element has a left and right border line with a width of 3, the size must be defined as: widget area width minus sum of line width, which is 400 - (3 + 3) = 394 in the example.

Adding Background Color and Borders to the Widget

By default, a widget has a transparent background and no borders. The following options are available to define a border and background color for the widget:

- A background color can be defined for the widget area. This is then used as default if no background color is defined in the configured widget report. Borders cannot be defined in the widget, but border lines can be defined in the configured widget report for the report area.
- A background color and border can be defined for the complete widget or part of the widget by adding an element **Panel** to the widget. Background color and borders defined with panel elements are fixed for all configured widget reports based on the widget. They cannot be changed via the report definition. Multiple panel elements can be defined for the widget to structure the background, For example, by defining a heading area with one color and a content area with a different color.

Defining a Default Background Color on the Widget Definition

To define a background color for the complete widget:

1) Click into the background of the widget area. The attribute window of the widget area opens.
2) Click into the field for the attribute **Background Color**, click the arrow that is displayed on the right of the field and select a color from the color picker.

   Please note that the background color is not visible in the design area. Nevertheless it will be visible in any configured widget report based on this widget and not including any background color definition in the cponfigured report.

3) In the toolbar, click the **Save** button to save your changes.

Defining a Panel Element for the Widget

Please note the following about the relation between the **Panel** element and other elements placed on the widget:

- **Panel** elements must be defined first. The visibility of elements depends on the order of definition. If you have defined For example, **Widget Picture** and **Widget Text** elements in a widget and then add a Panel element covering the whole widget area to define a background color for the widget, the

already placed **Widget Picture** and **Widget Text** elements are hidden behind the **Panel** element. There is no functionality to move elements to front or back.

- Elements placed on a **Panel** become part of the **Panel**. If you delete the **Panel**, all elements placed on the **Panel** are also deleted.

To add a **Panel** element to the widget area:

1) In the toolbar, click the **Panel** icon and add the **Panel** element to the widget area with the size and location matching your demands.

   For detailed information about placing elements on the widget area and sizing them, see the section General Instruction for Adding and Positioning Elements on the Widget.

2) Click the **Panel** element that you have added to the widget area to open the attribute window.
3) In the attribute window, expand the **Style** attribute and set the following subordinate attributes according to demand:

   - **Background Color**: Select a color from the color picker to add a background color to the panel. By default, **Panel** elements are transparent.
   - **Border Bottom**, **Border Left**, **Border Right** and **Border Top**: To define a line to be displayed on the top, left, right or bottom of the panel, expand the respective attribute and set the following attributes:
     - **Border**: Define the line width for the border as a positive integer. If the line width is zero, the line will not be displayed.
     - **Color**: Select the color for the line from the color picker.

     Border lines are drawn outside of the **Panel** element. The size required for display of the **Panel** element is the size defined for the borders plus the width of the border lines. This is of special importance if the panel shall fill the whole widget area. For example, if the widget area width is 400 and your element has a left and right border line with a width of 3, the size of the panel must be reduced to: widget area width minus sum of line widths, which is 400 - (3 + 3) = 394 in the example.

     The top and left position defined for a panel element include border lines. Lines are drawn right of the defined left position and under the defined top position. If the **Panel** shall fill the complete widget area, the **Top** and **Left** attributes must be defined as zero.

4) In the toolbar, click the **Save** button to save your changes.

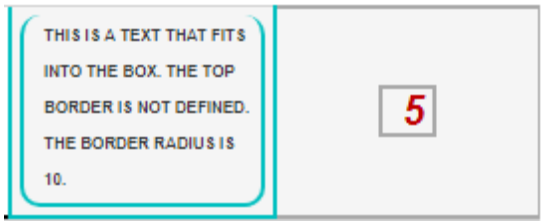
Using a Flow Panel or Table to Structure the Design Elements in the Widget

Flow Panel and **Table Layout Panel** elements that are the basic for designing object cockpits are also available to structure the content of a widget. This design is only relevant for specific use cases, For example, if you are not sure whether the dynamically generated widget content will match the sizing of the widget and would like to make sure that scrollbars will be available if the content exceeds the widget size. But usually widgets are small windows containing only a limited number of text and image elements and it is recommended to define them without using a table or a flow panel.

The visibility of elements depends on the order of definition. If you have defined For example, **Widget Picture** and **Widget Text** elements in a widget and then add a Table Layout Panel element covering the whole widget area, the already placed **Widget Picture** and **Widget Text** elements are hidden behind the **Panel** element. There is no functionality to move elements to front or back.

Using a **Table Layout Panel** will have the following effect on design:

- A table grid layout will be created. **Widget Text** and **Widget Picture** elements can be added to the cells in the table grid and will always span the whole table cell. Resizing of **Widget Text** and **Widget Picture** elements themselves is no longer available, but the **Widget Text** or **Widget Picture** elements can be configured to span multiple rows or columns with the attributes **Column Span** and **Row Span**, that are only visible on the **Widget Text** or **Widget Picture** elements if added to a **Table Layout Panel**.
- The table grid is created with equal column width. If an image is wider than the column width, only the left part of the image fitting the column width is displayed. If a text does not fit into one row in the column width, line breaks are used.
- The row height depends on the row content. If the content of all rows requires a size that is larger than the widget area size, a scroll bar is displayed. The row height depends on the biggest image or text element in the row. Images are always fully displayed in height.
- Captions defined for **Widget Picture** elements are ignored and not displayed in the configured widget report.
- Text defined in **Widget Text** elements is always displayed in all upper case letters.
- **Border Radius** definitions for **Widget Picture** elements are ignored.
- Border lines defined for **Widget Text** elements are displayed doubled:



  - One border is displayed around the field and one is displayed around the text. If the text is For example, centered and small, the distance between the two borders will be large.
  - The availability of lines depends on the definition of the borders in the **Widget Text** element. Border color and border width are used for both border lines.
  - **Border Radius** definitions are only used for the inner border.
  - If two text fields are located next to each other, the border between the elements is displayed in the design of the thicker border line.

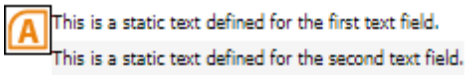
Using a **Flow Panel** will have the following effect on design:

- The size and location specifications of the elements in your widget will be ignored.
- The size of the elements matches the size of content of the element. Text is displayed without line breaks.

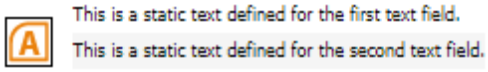
- The elements are all located one after the other without a spacing. If you would like to add a spacing between elements, you can set the attribute **Margin** of the **Widget Text** and **Widget Image** elements to define the spacing between the element and the next elements beside it or the border of the Flow Panel element. The attribute is expandable and allows to define the margin for each side of the element separately. If two elements are located next to each other and both have a margin defined, the distance between the elements is the sum of the **Margin** settings.

For example, one **Widget Picture** element and two **Widget Text** elements are added with a spacing and in a row on the **Flow Panel** added to the widget area:

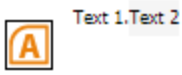
In the final configured report, the design will look like this without margin definition:



Defining a margin for the image will change the design to:



If the configured report adds shorter texts to the Widget Text elements, the two text elements are displayed side by side in the same row instead of below each other:



To add a **Flow Panel** or **Table Layout Panel** element to the widget area:

1) In the toolbar, click the **Flow Panel** icon or the **Table Layout Panel** icon and add the element to the widget area with the size and location matching your demands.

   For detailed information about placing elements on the widget area and sizing them, see the section General Instruction for Adding and Positioning Elements on the Widget.

2) If you have added a **Table Layout Panel** element, define the attribute **Table Layout** attribute of the element as <number of rows>x<number of columns>. For example, a definition of 2x4 will create a table grid with two rows and four columns.

3) In the toolbar, click the **Save** button to save your changes.

Adding a Widget Picture Field to the Widget

A **Widget Picture** element is a placeholder for any image that can be defined in the configured widget report referencing the widget. Only images uploaded to the icon gallery via Alfabet Expand can be displayed in the **Widget Picture** element.

For information about how to upload images to the icon gallery, see Adding and Maintaining Icons for the Alfabet Interface.

Optionally, you can add an image to the **Widget Picture** element that will then be displayed as default if no image is provided via the configured report. The **Widget Picture** element can be designed via its attributes to have a caption, borders or a background color.

In the configured report, only the image to be displayed in the **Widget Picture** element is specified. Borders, background color and caption as well as the size of the **Widget Picture** element cannot be changed within the configured widget report.

Please note the following about the display of images within the **Widget Picture** element:

- Images are not resized in the final configured widget report rendered on the Alfabet user interface.
- Images that are smaller than the **Widget Picture** element will be displayed centered in the **Widget Picture** element.
- If the image is bigger than the **Widget Picture** element, only the upper left corner of the image is displayed.
- If you add a default image to the **Widget Picture** element, the default image is displayed if the configured report does not provide any image definition for the **Widget Picture** element. If the configured report provide image definitions via a query and the query does not return a valid image definition, the icon for missing images is displayed in the configured widget report rendered on the Alfabet user interface.
- If you add a default image to the **Widget Picture** element, the default image is not displayed in the position and size it is displayed in the configured widget report rendered on the Alfabet user interface. In the widget area in Alfabet Expand, images that are smaller than the **Widget Picture** element are displayed in the upper left corner of the **Widget Picture** element and images that are bigger than the **Widget Picture** element are resized to match the element size.

If you would like to design or structure the background of the widget via a **Panel**, **Flow Panel** or **Table Layout Panel**, you must add these elements prior to adding **Widget Picture** elements. The elements cannot be moved to back or front and a panel added to a widget hides all already defined **Widget Picture** elements that it covers.

To define a **Widget Picture** element:

1) In the toolbar, click the **Widget Picture**  icon and add the **Widget Picture** element to the widget area with the size and location matching your demands.

   For detailed information about placing elements on the widget area and sizing them, see the section General Instruction for Adding and Positioning Elements on the Widget.

2) In the **Name** attribute, change the default name to a short, meaningful name. The name is displayed in the explorer of the report assistant for configured widget reports referencing the

widget. The name must be unique within the widget specification and shall not contain special characters.

If you are changing the name of a **Widget Picture** element after having used the widget in a configured report, the **Widget Picture** element is regarded as a new element and the element with the prior name is regarded as removed. The existing definition in the configured report is removed and substituted with an unspecified element.

3) In the **Caption** attribute, either delete the default caption to display the **Widget Picture** element without a caption, or define a caption to be displayed in the widget on top of the **Widget Picture** element.

Please note the following about the caption of the **Widget Picture** element:

- The caption cannot be overwritten or removed by a specification of the configured widget report.
- The font style and size and the location of the caption, that is displayed on the top left of the Widget Picture element cannot be altered neither in the **Widget Picture** element definition nor in the configured widget report. The caption is displayed in the same design as filter and editor fields.

4) If you want to add a default image that shall be displayed if no image definition is provided in the configured report, expand the **Picture Source Definition** attribute and define the following attributes in the order given below:

- **Icon Type**: Select the icon gallery that the image is available in.
- **Icon Name**: Select the image from the dropdown list displaying all images in the icon gallery selected with the attribute **Icon Type**.

5) In the attribute window, expand the **Style** attribute and set the following subordinate attributes according to demand. The style definitions are fixed for the **Widget Picture** element and cannot be altered via the configured widget report:

- **Background Color**: Select a color from the color picker to add a background color to the **Widget Picture** element. By default, **Widget Picture** elements are transparent.
- **Border Bottom**, **Border Left**, **Border Right** and **Border Top**: To define a line to be displayed on the top, left, right or bottom of the **Widget Picture** box, expand the respective attribute and set the following attributes:
  - **Border**: Define the line width for the border as a positive integer. If the line width is zero, the line will not be displayed.
  - **Color**: Select the color for the line from the color picker.

  Border lines are drawn outside of the **Widget Picture** element. The size required for display of the **Widget Picture** element is the size defined for the borders plus the width of the border lines.

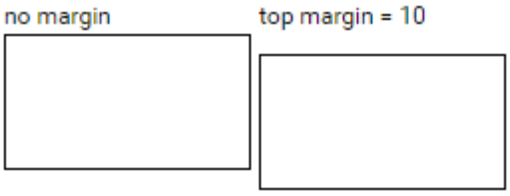
  The top and left position defined for a **Widget Picture** element is the position outside the lines.

- **Border Radius**: Define the positive integer to cause border edges to be rounded. The higher the integer, the more intense is the rounding of the border. Zero means that a straight corner

is displayed as edge. The rounding of borders is also visible if no border lines, but only a background is defined.

The **Border Radius** setting is not visible in the design area for the widget, but only in the configured report.

- **Margin**: Setting a margin will define a spacing between the **Widget Picture** element and all other elements. The **Margin** will also be applied to the spacing between the defined caption and the box itself:



To define a margin, expand the **Margin** attribute and define the margin as a positive integer either separately for each side of the box in the attributes **Bottom**, **Left**, **Right**, **Top**, or for all sides equally in the attribute **All**.

Setting a **Margin** is only useful if the **Widget Picture** element is located in a **Flow Panel**. For **Widget Picture** elements on a **Panel** or directly added to the widget area, the **Margin** will alter the position of the **Widget Picture** element in a way that is hard to control. If you do not need more spacing to be added between caption and box, it is recommended to use the positioning options described in the section General Instruction for Adding and Positioning Elements on the Widget to define spacing between elements.

The **Margin** setting is not visible in the design area for the widget, but only in the configured report.

6) In the toolbar, click the **Save** button to save your changes

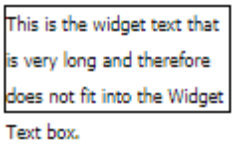
Adding a Widget Text Field to the Widget

A **Widget Text** element is a placeholder for text defined in the configured widget report referencing the widget. In the widget itself no text can be defined as default. Widget Text elements are empty elements that are exclusively filled via the configured widget report.

The layout of the **Widget Text** element, like For example, background color, borders and font style and size are both configurable in the widget and in the configured widget report referencing the widget. The configuration provided via the widget definition is used as default if no styling is provided in the configured widget report.

Please note the following about the sizing of **Widget Text** elements:

- The size and position of the **Widget Text** element that you define in the widget is fixed. It cannot be changed via the configured report.
- If text is smaller that the text box, you can define the location of the text in the box via the attributes of the **Widget Text** element as default. The default can be overwritten by a definition in the configured widget report.
- Line breaks are added automatically to keep the text within the left and right borders of the **Widget Text** element. If the text added via the configured report is longer than the place in the text box allows to display, the text is nevertheless completely displayed, with part of the text being displayed outside the box:



Text box.

If you would like to design or structure the background of the widget via a **Panel**, **Flow Panel** or **Table Layout Panel**, you must add these elements prior to adding **Widget Text** elements. The elements cannot be moved to back or front and a panel added to a widget hides all already defined **Widget Text** elements that it covers.

To define a **Widget Text** element:

1) In the toolbar, click the **Widget Text** icon and add the **Widget Text** element to the widget area with the size and location matching your demands.

   For detailed information about placing elements on the widget area and sizing them, see the section General Instruction for Adding and Positioning Elements on the Widget.

2) In the **Name** attribute, change the default name to a short, meaningful name. The name is displayed in the explorer of the report assistant for configured widget reports referencing the widget. The name must be unique within the widget specification and shall not contain special characters.
3) In the **Caption** attribute, either delete the default caption, or define a caption to be displayed in the **Widget Text** element in the widget design area as a placeholder text. The caption is not displayed in the configured widget report, even if no text is provided for the text box in the configured widget report definition. It may be useful to define a caption with a typical text to be added via the configured widget report prior to defining the standard layout of the text box to see the effect of your design on the text in the final report. Font style, size and color that you define for text in the text box will also be applied to the caption text.
4) In the attribute window, expand the **Style** attribute and set the following subordinate attributes according to demand. Style definitions for the **Widget Text** element are default values that can be altered via the configured widget report:
   - **Background Color**: Select a color from the color picker to add a background color to the **Widget Text** element. By default, **Widget Text** elements are transparent.
   - **Border Bottom**, **Border Left**, **Border Right** and **Border Top**: To define a line to be displayed on the top, left, right or bottom of the **Widget Text** box, expand the respective attribute and set the following attributes:

- **Border**: Define the line width for the border as a positive integer. If the line width is zero, the line will not be displayed.
- **Color**: Select the color for the line from the color picker.

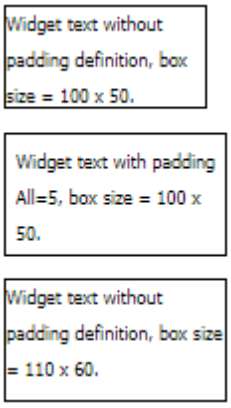
Border lines are drawn outside of the **Widget Text** element. The size required for display of the **Widget Text** element is the size defined for the borders plus the width of the border lines.

The top and left position defined for a **Widget Text** element is the position outside the lines.

- **Border Radius**: Define the positive integer to cause border edges to be rounded. The higher the integer, the more intense is the rounding of the border. Zero means that a straight corner is displayed as edge. The rounding of borders is also visible if no border lines, but only a background is defined.

The **Border Radius** setting is not visible in the design area for the widget, but only in the configured report.

- **Padding**: Define the spacing between the border of the text box and the text. Please note that the definition of a padding resizes the text box. The resulting size is the box size plus the defined padding. For example, if you have a text box with a width of 100 and a height of 50, and you define padding for all sides as 5, the actual width of the box will be 100 + 5 + 5 = 110, because the left and right padding must be added, and the resulting height will be 50 + 5 + 5 = 60, because the top and bottom padding must be added.



To define a padding, expand the **Padding** attribute and define the padding as a positive integer either separately for each side of the box in the attributes **Bottom**, **Left**, **Right**, **Top**, or for all sides equally in the attribute **All**.

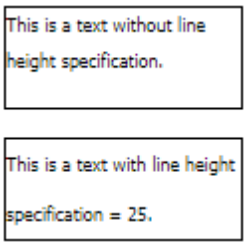
- **Margin**: Setting a margin will define a spacing between the **Widget Text** element and all other elements. To define a margin, expand the **Margin** attribute and define the margin as a positive

integer either separately for each side of the box in the attributes **Bottom**, **Left**, **Right**, **Top**, or for all sides equally in the attribute **All**.

Setting a **Margin** is only useful if the **Widget Picture** element is located in a **Flow Panel**. For **Widget Picture** elements on a **Panel** or directly added to the widget area, the **Margin** will alter the position of the **Widget Picture** element in a way that is hard to control. If you do not need more spacing to be added between caption and box, it is recommended to use the positioning options described in the section General Instruction for Adding and Positioning Elements on the Widget to define spacing between elements.

The **Margin** setting is not visible in the design area for the widget, but only in the configured report.

- **Line Height**: If a text spans multiple lines, the **Line Height** defines the space reserved for each text line. Make sure to define a number higher than the one defined for the font size. Otherwise the text will overlap. If you define `-1`, the line height is automatically adapted to the text size. The line height can be used to define additional spacing between lines of text:



- **Font**: Expand the attribute and define the style of the text font according to your demands with the following subordinate attributes:
  - **(Name)**: Select the font type from the drop-down list.
  - **Bold**: Set to `True` if you want the text to be displayed bold.
  - **Color**: Select a text color from the color picker.
  - **Horizontal Alignment**: Select the horizontal alignment of text either to the right (`Right`), to the left (`Left`) or to the center (`Center`) of the text field. If you select `Unknown`, the horizontal text alignment is `Right`, which is the default.
  - **Italic**: Set to `True` if you want the text to be displayed italic.
  - **Size**: Define the font size as a positive integer.
  - **Underline**: Set to `True` if you want the text to be underlined.
  - **Vertical Alignment**: Select the horizontal alignment of text either to the top (`Top`), to the bottom (`Bottom`) or to the center (`Center`) of the text field. If you select `Unknown`, the vertical text alignment is `Top`, which is the default.

5) In the toolbar, click the **Save** button to save your changes.

Changing a Widget Definition Already Used In Configured Widget Reports

You can also edit an existing widget or delete it even if it has been used in one or multiple configured reports. The following applies for changes to a widget that is already in use:

- If you delete a widget that is used in configured widget reports, the widget report definition is completely removed from the configured widget reports even if they are having the **Report Status** set to **Active**. When you try to delete the widget, a warning will pop up informing you that the widget is used in configured reports.
- If you change the name of a widget, the widget name is automatically updated in all configured reports using the widget. The existing configuration is maintained.
- If you change the name of a widget element within the widget, the definition of this widget element is removed from all configured reports using the widget and a new, undefined element with the new name is added to the configured widget report definition. Renaming of widget elements Therefore, require a re-definition of the renamed widget element in all configured reports referencing the widget. Definitions of all other widget elements in the configured widget report are not affected.
- If you change the default layout defined for a widget element, the change is immediately applied to all configured reports that reference the widget. If the default layout has been changed via the configured report definition, the configuration in the configured report definition will be maintained and you will not see any changes in the configured widget report rendered on the user interface.

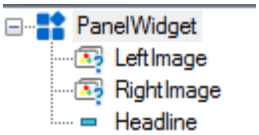
Prior to changing or deleting a widget, you should right-click the widget and select **Show Usage** from the context menu to see a list of configured reports the widget is assigned to.

Defining the Configured Widget Report

Widget reports are designed in the **Report Assistant**. For information about how to open the **Report Assistant**, see Creating a New Graphic Report. For more information about how to work with the **Report Assistant**, see Configuring the Report with the Report Assistant.

Widget reports are based on widgets. A widget must be created and defined prior to creating the configured widget report. Definition of the widget is described in detail in the section Defining a Widget.

In the report assistant, you must first select a widget in the attribute **Widget Source** of the root node of the report assistant. After you have selected the **Widget Source**, the **Widget Picture** and **Widget Text** elements in the widget are added as sub-nodes to the explorer tree: The elements are displayed with an icon followed by the string defined in the **Name** attribute of the **Widget Text** or **Widget Picture** element:



You can see from the icon which kind of widget elements and report definitions are used:

- : A **Widget Text** element that is filled with a static text. This is the default that is displayed for all **Widget Text** elements if no further specification is provided via the configured widget report.
- : A **Widget Text** element that is filled via a query.
- : A **Widget Picture** element.

To create the widget report, you must fist define the basic design in the root node of the widget report assistant by setting the following attributes:

Attribute	Description
Widget Source	Click the **Browse** button on the right of the attribute field to open the **Select Widget Source** window. Select the widget the configured widget report shall be based on in the table and click **OK**. **This attribute is mandatory and must be set first.**
Value Column Name	If your widget includes **Widget Text** elements and you would like to define the content of one or multiple **Widget Text** elements via a query, enter the name of the column in the result dataset of the query that returns the text. The column name must be identical in all queries defined for **Widget Text** elements in the configured widget report. For more information about the definition of text via a query, see Defining Dynamic Content for a Widget Text Element.
Picture Name Column Name	If your widget includes **Widget Picture** elements and you would like to define the content of one or multiple **Widget Picture** elements via a query, enter the name of the column in the result dataset of the query that returns the image name. The column name must be identical in all queries defined for **Widget Picture** elements in the configured widget report. For more information about the definition of images via a query, see Defining the Content of a Widget Picture Element.
Font Size Column Name	If your widget includes **Widget Text** elements and you would like to define the content and the font size of one or multiple **Widget Text** elements via a query, enter the name of the column in the result dataset of the query that returns the font size. The column name must be identical in all queries defined for **Widget Text** elements in the configured widget report. For more information about the definition of text via a query, see Defining Dynamic Content for a Widget Text Element.
Font Color Column Name	If your widget includes **Widget Text** elements and you would like to define the content and the font color of one or multiple **Widget Text** elements via a query, enter the name of the column in the result dataset of the query that returns the font color. The column name must be identical in all queries defined for **Widget Text** elements in the configured widget report. For more information about the definition of text via a query, see Defining Dynamic Content for a Widget Text Element.
Object Reference Column Name	If you would like to enable navigation from an image or text in the configured widget report to the object view or object cockpit of an object represented by the graphic or text, the image or text must be defined via a query. The `REFSTR` of the object must be returned in a column of the result dataset and the name of the column must be entered into this attribute.

Attribute	Description
	If you would like to navigate to any other view instead of the object view or object cockpit of the object, the **Object View Column Name** must also be defined. For more information about the definition of text or images via a query, see Defining Dynamic Content for a Widget Text Element or Defining the Content of a Widget Picture Element. For detailed information about activating navigation from a configured report, see Defining Navigation from the Report to Alfabet Views.
Object View Column Name	If you have enabled navigation from the widget report as described for the **Object Reference Column Name** attribute, you can define a view that shall be opened instead of the object view or object cockpit of the reference object. The view is opened with the `REFSTR` of the reference object handed over as parameter `BASE`. `View= ViewType: ViewName` The view must be defined in a column of the result dataset of the query and the column name must be defined in this attribute. The column name must be identical in all queries defined for widget elements in the configured widget report. In the result dataset, the view must be defined as: The view type can be `Report`, `GraphicView` or `ObjectView`. For more information about the definition of text or images via a query, see Defining Dynamic Content for a Widget Text Element or Defining the Content of a Widget Picture Element. For detailed information about activating navigation from a configured report, see Defining Navigation from the Report to Alfabet Views.
Tooltip Column Name	A tooltip can be displayed in the configured widget report if the user moves the mouse over an image or text defined via a query. The tooltip must be returned in a column of the result dataset and the name of the column must be entered into this attribute. The column name must be identical in all queries defined for widget elements in the configured widget report. For more information about the definition of text or images via a query, see Defining Dynamic Content for a Widget Text Element or Defining the Content of a Widget Picture Element.
Width	The width of the configured widget report is automatically set to the width of the widget when a **Widget Source** is selected. It can be changed if required, but usually should be left unchanged to ensure complete visibility of the widget design on the interface.
Height	The height of the configured widget report is automatically set to the width of the widget when a **Widget Source** is selected. It can be changed if required, but usually should be left unchanged to ensure complete visibility of the widget design on the interface.

Attribute	Description
Style > **Background Color**	Select a color from the color picker to change the background color of the widget background. This setting will overwrite a default background color setting of the widget. If the widet background is filled with a panel element defined to have a background color, this setting will not have any effect, because it is overlaid by the panel element.
Style > **Border Bottom**, **Border Left**, **Border Right** and **Border Top**	To define whether and how a line will be displayed on the top, left, right or bottom of the configured widget report, expand the respective attribute and set the following attributes: • **Border**: Define the line width for the border as a positive integer. If the line width is zero, the line will not be displayed. • **Color**: Select the color for the line from the color picker. Border lines are drawn outside of the configured widget report area defined with the attributes **Width** and **Height**. This alters the size required for display of the configured widget report.
Style > **Border Radius**	Define the positive integer to cause border edges to be rounded. The higher the integer, the more intense is the rounding of the border. Zero means that a straight corner is displayed as edge. The rounding of borders is also visible if no border lines, but only a background is defined. 

After having created the basic design, you can add text and images to the **Widget Picture** and **Widget Text** elements and change the styling of text fields:

- Defining the Content of a Widget Picture Element
- Defining The Content and Design for a Widget Text Element
  - Defining Static Content for a Widget Text Element
  - Defining Dynamic Content for a Widget Text Element
  - Defining the Layout of the Widget Text Element in the Subordinate Attributes of the Style Attribute
  - Revert all Layout to the Default Layout of the Underlying Widget

Defining the Content of a Widget Picture Element

The content of a **Widget Picture** element is dynamically defined via a query.

Please note the following about the display of images within the **Widget Picture** element:

- Images are not resized in the configured widget report rendered on the Alfabet user interface.
- Images that are smaller than the **Widget Picture** element will be displayed centered in the **Widget Picture** element.
- If the image is bigger than the **Widget Picture** element, only the upper left corner of the image is displayed.
- If a default image is defined in the **Widget Picture** element in the widget, the default image will be displayed if the configured report does not provide any image definition for the **Widget Picture** element. If the configured report provide image definitions via a query and the query does not return a valid image definition, the icon for missing images is displayed in the configured widget report rendered on the Alfabet user interface.
- Background color and borders as well as a caption defined for the **Widget Picture** element cannot be changed via the configured report.

Only images that are uploaded to the icon gallery can be displayed in configured widget reports. For information about uploading images to the icon gallery, see Adding and Maintaining Icons for the Alfabet Interface.

If you define the content of the **Widget Picture** in the configured widget report, you must define a query that returns at least the image source from the icon gallery. Optionally, a tooltip and navigation from the image to an Alfabet view can be defined.

To define dynamic content for a **Widget Picture** element, click the **Widget Picture** node in the explorer and define a native SQL query in either the attribute **Native SQL** or **Query as Text**, or define an Alfabet query in either the attribute **Alfabet Query** or **Query as Text**.

The query must return a result dataset with a single row and the columns required to define the content. The column names in the dataset must match the column names defined in attributes in the root node of the report assistant. The following table lists all content that can be defined in the query together with the attribute in the root node that must contain the column name:

Definition of	Required return value	Column name to be defined in attribute
Image to be displayed.	The image source must be defined as `ImageSource:ImageName` *ImageSource* defines the icon gallery the image is located in. Allowed values are: • `SmallIcon` for images from the 22x22 icon gallery and images assigned to object classes via the class settings.	**Picture Name Column Name**

Definition of	Required return value	Column name to be defined in attribute
	• `LargeIcon` for images from the Large Icons (30x30) icon gallery. • `FreeIcon` for images from the Free Icons gallery. *ImageName* is the name of the icon in the icon gallery. `SmallIcon:Application`	
Tooltip to be displayed if the user moves the mouse over the image.	The data to be displayed, For example, a string, text, date, integer or real number.	**Tooltip Column Name**
Activation of navigation and specification of the navigation target object.	The `REFSTR` of an object that shall be the target of the navigation. Please note that this specification is required even if the view that opens does not require a base object specification. If the column is not defined or not returning a valid `REFSTR`, navigation is disabled.	**Object Reference Column Name**
Definition of the navigation target view.	If no navigation target view is defined, the object view or object cockpit that is the standard for the object in the current user profile opens. To alter the navigation target, the target view must be defined as: `View=`*`ViewType`*`:`*`ViewName`* The `ViewType` can be one of the following: • ***Report*** for a configured report • ***GraphicView*** for a standard Alfabet view • ***ObjectView*** for an object profile `ViewName` is the name of the view. For more information about defining navigation to a target view, see Defining Navigation from the Report to Alfabet Views.	**Object View Column Name**

Defining The Content and Design for a Widget Text Element

The text to be displayed in a **Widget Text** element is exclusively defined via the configured widget report. The caption that is specified in the Widget Text element in the widget is a placeholder that is only displayed in the widget design area and never displayed in the configured reports based on the widget.

In the configured widget report, either static text or dynamic text evaluated via a query can be defined for a **Widget Text** element. If you do not define any text, the **Widget Text** element is displayed empty if it has a border and/or background color defined. Otherwise it is not displayed at all. There is not default provided via the widget configuration.

The styling of the underlying **Widget Text** elements can also be changed.

The **Widget Text** element in the report assistant has an attribute **Style** containing all styling option as default. The default values for all styles are identical to the values configured in the **Widget Text** element in the referenced widget. The styling can then be changed according to demand.

If the text is defined via a query, the font size and font color can be defined in the query instead of the attribute. Apart from that, the **Style** attributes are the same for **Widget Text** elements defining a static text or a text via a query.

The following sections provide an overview of the required settings for dynamic and static text definition:

- Defining Static Content for a Widget Text Element
- Defining Dynamic Content for a Widget Text Element
- Defining the Layout of the Widget Text Element in the Subordinate Attributes of the Style Attribute
- Revert all Layout to the Default Layout of the Underlying Widget

Defining Static Content for a Widget Text Element

A static text can be defined within a **Widget Text** element.

To define a static text for a **Widget Text** element, click the **Widget Picture** node in the explorer and define the following attributes:

Attribute	Description
Content Type	Select `StaticText` from the dropdown list. **This attribute must be set first.**
Caption	Enter the text to be displayed.
Style	If applicable, change the default layout of the **Widget Text** field and the text font in the subordinates attributes of the attribute **Style**. Details are given in the section Defining the Layout of the Widget Text Element in the Subordinate Attributes of the Style Attribute.

Defining Dynamic Content for a Widget Text Element

The content of a **Widget Text** element can be dynamically defined via a query.

If you define the content of the **Widget Text** in the configured widget report, you must define a query that returns at least the text to be displayed. Strings, text and numeric values can be displayed as text. For numeric values, the number of decimal values displayed in the widget can be specified as well as the display of a currency symbol or a percentage sign next to the number.

Optionally, text color and font size, a tooltip and navigation from the text to an Alfabet view can be dynamically defined via the query.

To define dynamic content for a **Widget Text** element, click the **Widget Text** node in the explorer and define the following attributes:

Attribute	Description
Content Type	Select `QueryResult` from the dropdown list. **This attribute must be set first.**
Style	If applicable, change the default layout of the **Widget Text** field and the text font in the subordinates attributes of the attribute **Style**. Details are given in the section Defining the Layout of the Widget Text Element in the Subordinate Attributes of the Style Attribute.
Numeric Format	If the query returns a numeric value, this attribute specifies how the numeric value is displayed. Select one of the following: • `None`: The number is displayed in the shortest possible format without any number grouping symbols. • `Generic`: The number is displayed with number grouping symbols. The number of decimal digits can then be specified with the attribute **Decimals**. • `Percentage`: The % sign is displayed behind the number. The number of decimal digits can then be specified with the attribute **Decimals**. • `Currency`: The symbol of the currency defined as the standard currency in the Alfabet database is displayed in front of the number. The number of decimal digits can then be specified with the attribute **Decimals**. The standard currency is defined in the **Reference Data** functionality. For more information, see *Configuring Currencies and Currency Exchange Rates for Cost Management Capabilities* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.
Decimals	If the query returns a numeric value and the **Numeric Format** attribute is set to `Generic`, `Percentage` or `Currency`, this attribute specifies how many decimals are

Attribute	Description
	displayed. By default, no decimals are displayed and values with decimals are rounded.
Native SQL / **Query as Text** / **Alfabet Query**	Define a native SQL query in either the attribute **Native SQL** or **Query as Text**, or define an Alfabet query in either the attribute **Alfabet Query** or **Query as Text**. Details about the required output of the query are given below.

The query must return a result dataset with a single row and the columns required to define the content. The column names in the dataset must match the column names defined in attributes in the root node of the report assistant. The following table lists all content that can be defined in the query together with the attribute in the root node that must contain the column name:

Definition of	Required return value	Column name to be defined in attribute
Text or number to be displayed.	The data to be displayed, For example, a string, text, date, integer or real number. Please note that a display value is required and if no column name is defined in the **Value Column Name** attribute, the value in the last column returned by the query is displayed as fallback value.	**Value Column Name**
Tooltip to be displayed if the user moves the mouse over the image.	The data to be displayed, For example, a string, text, date, integer or real number.	**Tooltip Column Name**

The font size of the text.	The text size returned as integer.
	
	Please note that a number returned as string will not be evaluated.
	
	  For example, in the SELECT statement of a native SQL query, the font size must be defined as:
	`20 AS 'FontSize'`
	and not as
	`'20' AS 'FontSize'`
		Font Size Column Name

The text color.	An HTML compliant color code.	**Font Color Column Name**

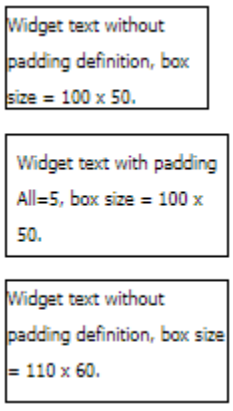
Definition of	Required return value	Column name to be defined in attribute
Activation of navigation and specification of the navigation target object.	The `REFSTR` of an object that shall be the target of the navigation. Please note that this specification is required even if the view that opens does not require a base object specification. If the column is not defined or not returning a valid `REFSTR`, navigation is disabled.	**Object Reference Column Name**
Definition of the navigation target view.	If no navigation target view is defined, the object view or object cockpit that is the standard for the object in the current user profile opens. To alter the navigation target, the target view must be defined as: `View=`*`ViewType`*`:`*`ViewName`* The `ViewType` can be one of the following: • ***Report*** for a configured report • ***GraphicView*** for a standard Alfabet view • ***ObjectView*** for an object profile `ViewName` is the name of the view. For more information about defining navigation to a target view, see Defining Navigation from the Report to Alfabet Views.	**Object View Column Name**

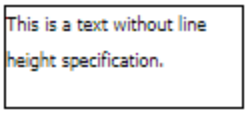
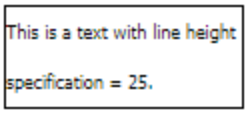
Defining the Layout of the Widget Text Element in the Subordinate Attributes of the Style Attribute

To change the layout of the **Widget Text** element defined in the widget to any other styling, expand the **Style** attribute of the **Widget Text** element in the report assistant and change the style with the following attributes:

Attributes that are not described in the following table currently do not have any effect on the layout.

Attribute	Description
Background Color	Select a color from the color picker to change the background color of the **Widget Text** element.
Border Bottom, **Border Left**, **Border**	To define whether and how a line will be displayed on the top, left, right or bottom of the **Widget Text** box, expand the respective attribute and set the following attributes:

Attribute	Description
Right and **Border Top**	• **Border**: Define the line width for the border as a positive integer. If the line width is zero, the line will not be displayed. • **Color**: Select the color for the line from the color picker. Border lines are drawn outside of the **Widget Text** element. This alters the size required for display of the **Widget Text** element. The top and left position defined for a **Widget Text** element is the position outside the lines and are not changed by changing the border width.
Border Radius	Define the positive integer to cause border edges to be rounded. The higher the integer, the more intense is the rounding of the border. Zero means that a straight corner is displayed as edge. The rounding of borders is also visible if no border lines, but only a background is defined. 
Padding	Define the spacing between the border of the text box and the text. Please note that the definition of a padding resizes the text box. The resulting size is the box size plus the defined padding. For example, if you have a text box with a width of 100 and a height of 50, and you define padding for all sides as 5, the actual width of the box will be 100 + 5 + 5 = 110, because the left and right padding must be added, and the resulting height will be 50 + 5 + 5 = 60, because the top and bottom padding must be added.  To define a padding, expand the **Padding** attribute and define the padding as a positive integer either separately for each side of the box in the attributes **Bottom**, **Left**, **Right**, **Top**, or for all sides equally in the attribute **All**.
Margin	Define a spacing between the text box and other elements. Setting a **Margin** is only useful if the **Widget Text** element is located in a **Flow Panel**. For **Widget Text** elements on a **Panel** or directly added to the widget area, the definition of a margin changes the

Attribute	Description
	location of the text box in a way that is not easy to predict. It is recommended to change the location of text boxes via the underlying widget. To define a margin, expand the **Margin** attribute and define the margin as a positive integer either separately for each side of the box in the attributes **Bottom**, **Left**, **Right**, **Top**, or for all sides equally in the attribute **All**.
Line Height	If a text spans multiple lines, the **Line Height** defines the space reserved for each text line. Make sure to define a number higher than the one defined for the font size. Otherwise the text will overlap. If you define `-1`, the line height is automatically adapted to the text size. The line height can be used to define additional spacing between lines of text:  
Font	Expand the attribute and define the style of the text font according to your demands with the following subordinate attributes: • **(Name)**: Select the font type from the drop-down list. • **Bold**: Set to `True` if you want the text to be displayed bold. • **Color**: Select a text color from the color picker. Please note that this setting will be ignored if a font color is defined via a query in a **Widget Text** element with the **Content Type** set to `QueryResult`. • **Horizontal Alignment**: Select the horizontal alignment of text either to the right (`Right`), to the left (`Left`) or to the center (`Center`) of the text field. If you select `Unknown`, the horizontal text alignment is `Right`, which is the default. • **Italic**: Set to `True` if you want the text to be displayed italic. • **Size**: Define the font size as a positive integer. Please note that this setting will be ignored if a font color is defined via a query in a **Widget Text** element with the **Content Type** set to `QueryResult`. • **Underline**: Set to `True` if you want the text to be underlined.

Attribute	Description
	• **Vertical Alignment**: Select the horizontal alignment of text either to the top (`Top`), to the bottom (`Bottom`) or to the center (`Center`) of the text field. If you select `Unknown`, the vertical text alignment is `Top`, which is the default.

Revert all Layout to the Default Layout of the Underlying Widget

The layout of the **Widget Text** elements defined in the subordinate attributes of the attribute **Style** can be removed from the widget report to fall back to the default layout defined in the widget.

This does not include the text definitions itself. They will still be available after removal of the layout. Styling options defined via a query are also still available.

To revert to the layout defined in the widget:

1) In the report assistant, click the button **Remove Field Styles**.

Defining a Words Cloud Report

Words cloud reports are designed in the **Report Assistant**. For information about how to open the **Report Assistant**, see Creating a New Graphic Report. For more information about how to work with the **Report Assistant**, see Configuring the Report with the Report Assistant.

The report assistant has a single node in the explorer with the attributes displayed in the area on the right.

The content of the words cloud report is defined via a query that returns the data in the required formats. The names of the columns in the dataset resulting from the query must then be mapped to the attributes in the **Report Assistant** to define which information required to build the report is returned in which column of the query.

It is recommended to use native SQL for creating words cloud reports. Alfabet query language does not provide the ability to include anything else than data from the tables in the Alfabet database into the search results, and the data is usually not corresponding to meaningful text size or text orientation values for the words cloud report.

The following table lists the attributes available to define the query and the mapping of data from the query to the respective functionality. The description informs about the data type to return in the column mapped to the respective attribute:

Attribute	Description
Native SQL / **Query as Text** /	Define a native SQL query in either the attribute **Native SQL** or **Query as Text**, or define an Alfabet query in either the attribute **Alfabet Query** or **Query as Text**. Details about the required output of the query are given below with the description of the column mapping attributes.

Attribute	Description
Alfabet Query	
Text Column Name	The name of the database column in the query that returns the text to be displayed. This attribute is mandatory.
Font Size Column Name	The name of the database column in the query that returns the font size for the text in pixel. This attribute is mandatory.
Rotation Column Name	The name of the database column in the query that returns the text orientation. Allowed values are 0 - 180. 0 will display the text in normal horizontal orientation, while 180 will display the text upside down. The text is turned clockwise. This attribute is mandatory.
Tooltip Column Name	The name of the database column in the query that returns a string to be displayed as tooltip on mouse over on the text. This attribute is optional. By default, no tooltip is displayed.
Text Color Column Name	The name of the database column in the query that returns the color of the text in HTML compatible format. This attribute is optional. The default text color is black (#000000).
Legend Group Column Name	A legend for cell coloring can be defined for the configured report if the coloring is defined via the **Text Color Column Name** attribute. A legend will only be displayed if the **Show Legend** attribute is set to `True` and both the **Legend Caption Column Name** attribute and the **Legend Group Column Name** attribute are defined. Optionally, the legend can be grouped in sections. This can be useful For example, in case coloring has a different impact for different font sizes. The legend will be grouped in sections if the **Legend Group Column Name** attribute is set to the name of the database column in the query that returns the caption of the legend group that the text color returned in the same row shall be displayed in. If **Legend Group Column Name** is not set, all coloring will be listed under the heading **Legend** without grouping.
Legend Caption Column Name	A legend for cell coloring can be defined for the configured report if the coloring is defined via the **Text Color Column Name** attribute. A legend will only be displayed if the **Show Legend** attribute is set to `True` and both the **Legend Caption Column Name** attribute and the **Legend Group Column Name** attribute are defined. To display the legend, the **Legend Caption Column Name** attribute must be set to the name of the database column in the query that returns an explanatory string for the text color returned in the same row. Please note that the legend will contain one entry for each different text. If the color is equal for different texts, colors will be listed twice. This can be used For example, in

Attribute	Description
	combination with different text returned in the column defined with the **Legend Group Column Name** attribute to create different legend groups per text size and list each color in each group with a different text. If the same legend caption text and legend group text is returned for multiple colors, only the color of the first result returned for the legend caption text will be displayed in the legend.
View Column Name	The name of the database column in the query that returns the definition of an alternative link target for navigation from words. By default, the standard object profile of the object opens when a user double clicks on a node or clicks a node and then clicks the **Navigate** button. The column must return a string with the following structure: `View=`*`ViewType`*`:`*`ViewName`* The `ViewType` can be one of the following: • ***Report*** for a configured report • ***GraphicView*** for a standard Alfabet view • ***ObjectView*** for an object profile `ViewName` is the name of the view. For more information about defining navigation to a target view including handing over of parameters, see Defining Navigation from the Report to Alfabet Views.
Reference Column Name	The name of the database column in the query that returns the `REFSTR` of the objects represented by the words. If a user clicks and holds on a word, the preview will open for the corresponding object and navigation will be enabled for the object. By default, this is the object defined via the `REFSTR` returned in the first `SELECT` statement of the native SQL query or the object of the `FIND` class of the Alfabet query. If the first `SELECT` statement of the native SQL query does not return a `REFSTR` or if any other object shall be used as navigation and preview target, an additional column returning a `REFSTR` value can be added to the query and mapped to the **View Column Name** attribute.

In addition to the dynamic content, static formatting can be applied to the report with the following attributes. All these attributes are optional.

Attribute	Description
Width	Defines the width of the area that can be filled with text in pixel. The default value is 800 pixel.
Height	Defines the height of the area that can be filled with text in pixel. The default value is 600 pixel.

Attribute	Description
Background Color	Defines the background color of the area reserved for the report. By default, the background is white. The color can be changed via the color picker. For more information about working with the color picker, see Defining Color Attributes.
Border Color	Defines the color of the border drawn around the report area. By default, the border color is white. As the default for the background color is also white, no border will be visible per default for the report. The color can be changed via the color picker. For more information about working with the color picker, see Defining Color Attributes.
Border Thickness	Defines the thickness of the border drawn around the report area. The default value is 1 pixel. Please note that the border is drawn within the area filled with text by the report algorithm. Therefore, if the report shows a very high number of results, the text may overlap the border.
Padding	Defines the space between texts in the report. The default is 2 pixel.
Show Legend	A legend for cell coloring can be defined for the configured report if the coloring is defined via the **Text Color Column Name** attribute. A legend will only be displayed if the **Show Legend** attribute is set to `True` and both the **Legend Caption Column Name** attribute and the **Legend Group Column Name** attribute are defined.

Creating Geo Map Reports

The geo map reports are based on the interactive FusionMaps® provided by FusionMaps. These maps are not part of the Alfabet database by default. Therefore, configuration of geo map reports require import of FusionMaps® data prior to configuring the configured report.

Do the following to configure geo map reports:

- Importing FusionMaps® Data
  - Importing Marker Positions Not Included In Standard Marker Import
- Configuring the Relevant Object Classes in the Alfabet database for Use in Geo Map Reports
- Creating a New Geo Map Report
- Configuring the Content of the Geo Map Report
  - Defining the Basic Design of the Geo Map Report
  - Defining Coloring of Map Regions Dependent on Data in the Alfabet database

Importing FusionMaps® Data

Import of data for FusionMaps and markers to be set on the map is imported from a Microsoft® Excel® file delivered by Software AG via predefined ADIF schemes. ADIF schemes and the import file are available on request only. Please contact Software AG Support if the files and schemes are not included in your product delivery.

Two ADIF schemes are available for data import:

Name of the ADIF scheme	Name of the import file	Description	Import required for
`FUSIONMAPINFO_IMPORT`	`fc_150302.xslx`	This import adds information about the FusionMaps and the different regions that a map is divided into to the Alfabet database.	Activation of the functionality. The execution of this ADIF scheme is mandatory as basis for any geo map report configuration.
`FusionMaps-Markers`	`FusionMaps-Markers.xlsx`	This import adds information about main cities and their location on different map. For each combination of map and location, the x-axis and Y-axis position of the location is given.	Definition of geo map reports containing markers on locations. If only geo map reports displaying coloring of regions are configured, this import is not required.

For information about how to execute an ADIF import, see *Configuring ADIF Schemes*.

As a result of the import, data is added to the database tables of the object classes ALFA_FUSIONMAPINFO and ALFA_FUSIONMAPMARKER in the Alfabet database.

For each available FusionMap, information about the map elements is stored in the database table ALFA_FUSIONMAPINFO. A map element can be either a country or a region or a city. It is an area on the map that can be colored independent from the rest of the map. The following properties of the object class ALFA_FUSIONMAPINFO are relevant for the configuration of the geo maps:

Property Name	Property Caption	Description
MAPNAME	Geographic Map Name	The name of the FusionMap this map element is part of.
ID	ID	The identifier of the map element on the map. The ID explicitly identifies the map element as a combination of map and region information. For example, France has different IDs on a world map or on a map of Europe.
LABEL	Label	The label assigned to the map element on the map. Usually this is the name of a continent, a country, a region or a city.
SHORTLABEL	Short Label	A short label used for display of map element labels in the geo map report if the geo map report is configured to display a label and no label is defined in the map element query of the geo map report.

For each available FusionMap, information about the markers on the map is stored in the database table ALFA_FUSIONMAPMARKER. Markers are defined locations on the map identified by there x-axis and Y-axis position on the map that can be colored and configured separately independent from the map element that they are located in. Markers can be connected via lines that represent an interconnection between the two locations the markers are defined for.

Property Name	Property Caption	Description
MAPNAME	Geographic Map Name	The name of the FusionMap this marker is part of.
ID	ID	The identifier of the marker on the map. The ID explicitly identifies the marker as a combination of map and position information.
XPOS	X Position	The horizontal position of the marker on the map.

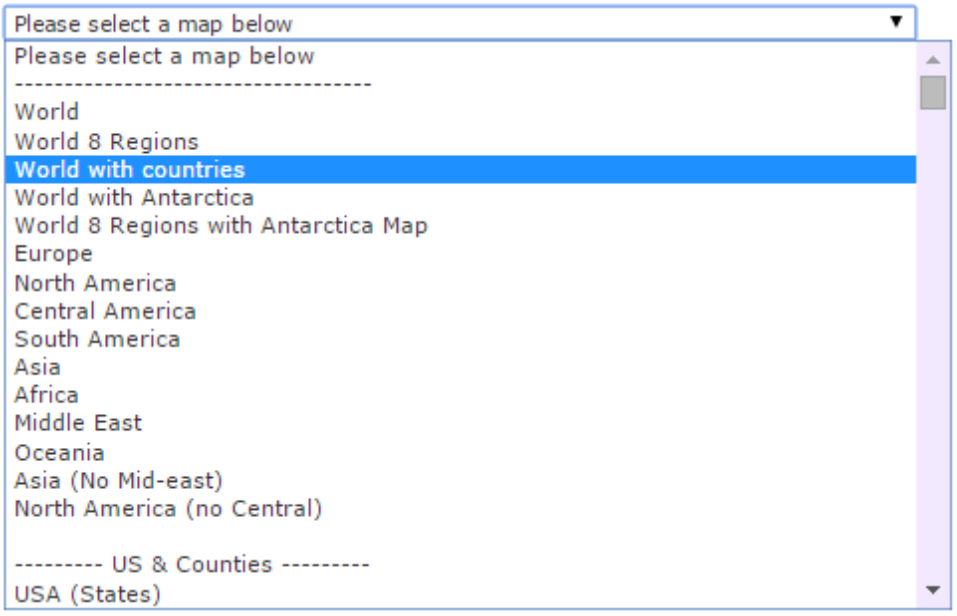
Property Name	Property Caption	Description
YPOS	Y Position	The vertical position of the marker on the map.
LABEL	Label	The label assigned to the marker on the map. Usually this is the name of a city.
TOOLTIP	Marker Tooltip	The tooltip displayed when a user moves with the mouse over the marker position on the map.

Importing Marker Positions Not Included In Standard Marker Import

Each position on the map can potentially be defined as a marker. The import file delivered by Software AG imports markers for prominent locations that are most likely to be used as markers. Markers can be added to the import file any time and imported via the ADIF scheme `FusionMapsMarkers`.

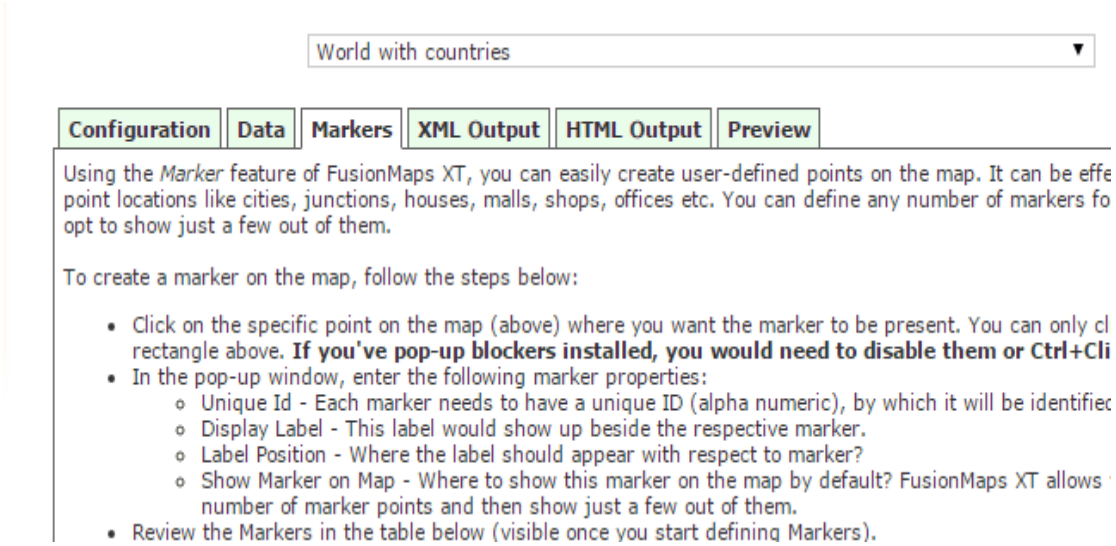
To add locations, that are not included in the standard markers, to the marker information in the Application architecture:

1) In your browser, open the URL http://docs.fusioncharts.com/maps/Tools/GUI/FusionMapsGUI.html.
2) Select the map for that you want to define markers in the drop-down list at the bottom of the page:



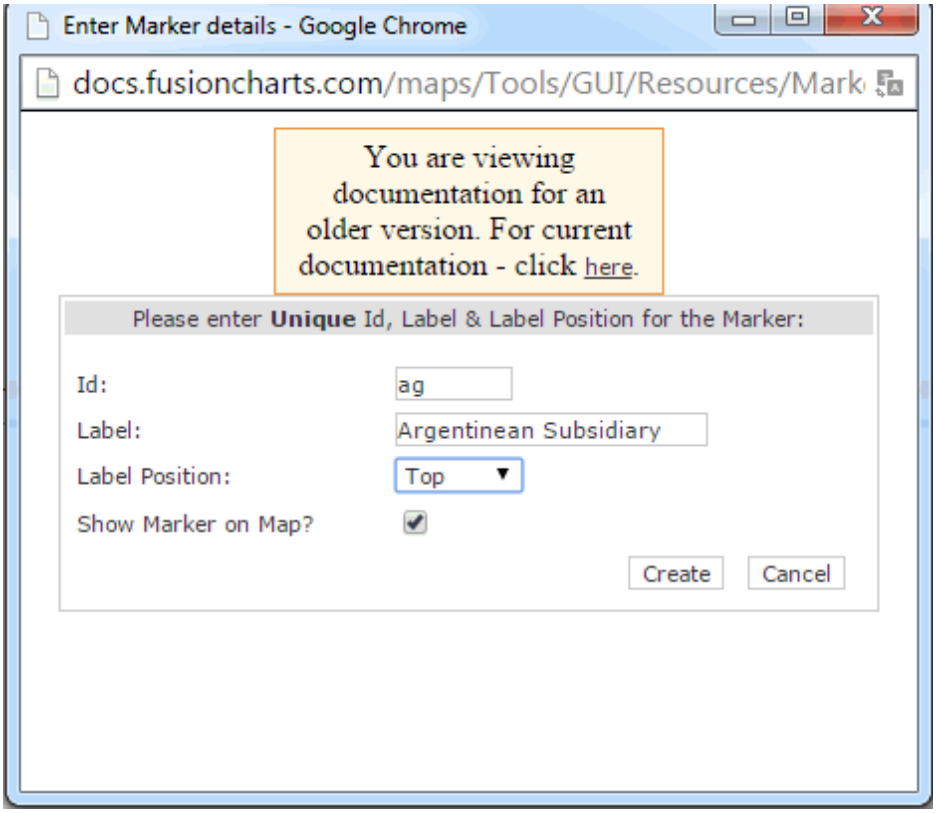
The map opens.

3) Below the map, a number of tabs for map content is displayed. Click the `Markers` tab to open it:



4) Click the location for that you want to define a marker in the map:

A new window opens with an editor for the definition of the marker:



5) Define the following attributes of the marker:

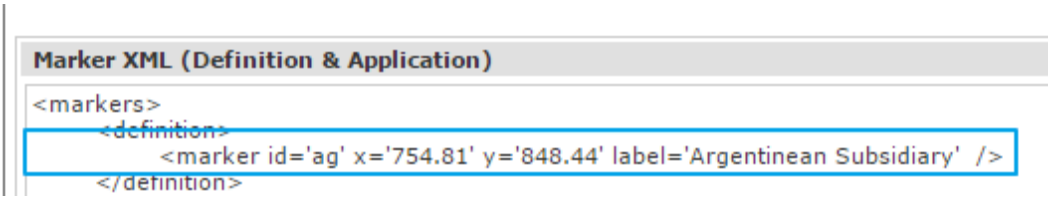
- `Id`: Define a short unique id for the marker.
- `Label`: Define a name for the marker.

> If you set markers for the same location, For example, a subsidiary, on different maps, you should define the same label for the location markers on all maps. This will ease the configuration of geo map reports because the assignment of objects to a location is independent form the map currently displayed.

The settings for `Label Position` and `Show Marker on Map` are irrelevant because they are not used in Alfabet geo map reports.

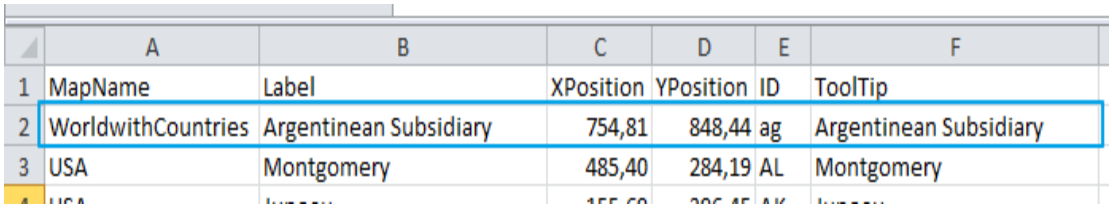
6) Click `Create` to create the marker. The new marker is displayed in the markers tab.

7) Click the `XML Output` tab to open it and scroll down to the section `Marker XML (Definitions & Application)`. The data for your marker including x-axis and Y-axis position is displayed:



8) Open the Alfabet marker import file `FusionMapsMarkers.xlsx`.

9) Add a new row to the Microsoft® Excel® sheet containing the marker definitions and copy the information about the marker into the respective fields in the columns for the definition of the marker properties:



	A	B	C	D	E	F
1	MapName	Label	XPosition	YPosition	ID	ToolTip
2	WorldwithCountries	Argentinean Subsidiary	754,81	848,44	ag	Argentinean Subsidiary
3	USA	Montgomery	485,40	284,19	AL	Montgomery

Please note that the `XPosition` and `YPosition` must be included with a comma instead of a dot as decimal separator.

10) Save your changes and import the markers via ADIF using the `FusionMapsMarkers` import scheme.

Configuring the Relevant Object Classes in the Alfabet database for Use in Geo Map Reports

In order to display data from the Alfabet database in the FusionMaps, a connection between IT related objects and the marker and map element information in the Alfabet database must be defined. There is no standard property on an object class for the definition of such a relation, because the way that data shall be represented in a map is customer specific.

If you want to display information in a map, you must first determine the object classes that shall be used to bound the information to a location on a map and configure custom properties for them that enable mapping between the objects of the class and the map elements or markers on the FusionMaps.

In FusionMaps, the same region, country or city normally has the same label name in all maps. The example below displays a part of the import file for map elements that displays the data for the country Iran in different maps. The Entity Label, that is store in the property `LABEL` of the object class `ALFA_FUSIONMAPINFO` is identical for all maps:

A	B	C	D	
Map Name	Entity Original	Entity ID	Entity Short Lab	Entity Label
bahia	2914307	2914307	IM	Iramaia
asia	016	016	IR	Iran
middleeast	03	03	IN	Iran
southernasia	106	106	IA	Iran
worldwithcountries	106	106	IA	Iran
amazonas	1301852	1301852	IR	Iranduba
santacatarina	4207809	4207809	IN	Irani

Therefore, it is recommended to define custom properties of the Type `String` that are identical to For example, a label of a map element rather than defining a custom property that contains a reference to an object of the object class `ALFA_FUSIONMAPINFO`. The reference would combine one object with a region on one single map while a string that is identical to a label name can be used in queries to combine the object with a region on all maps that display that region.

The following table lists objects of the Alfabet object class Location with three custom properties `SC_CONTINENT`, `SC_COUNTRY` and `SC_REGION` defined to couple the locations with the map elements on FusionMaps. The Name property of the location is also displayed:

	NAME	SC_CONTINENT	SC_COUNTRY	SC_REGION
1	Albany	North America	United States	New York
2	Argentina	South America	Argentina	NULL
3	Atlanta	North America	United States	Georgia
4	Berlin	Europe	Germany	Berlin
5	Bordeaux	Europe	France	Aquitaine
6	Boston, MA	North America	United States	Massachusetts
7	Brazil	South America	Brazil	Acre

Creating a New Geo Map Report

To create a new geo map report, create a configured report of the type `Custom` in Alfabet Expand:

1) In the **Reports** tab, right-click the **Reports** folder or a sub-folder thereof and select **Create New Report**. You will see the new configured report listed below the selected report folder.

As an alternative to defining all attributes for a new configured report, you can copy and paste an existing configured report and make modifications, as needed. To do so, right-click the existing configured report that you want to copy and select **Copy**. Right-click the new configured report and select **Paste**. Modify the necessary attributes described below.

2) In the attribute window, select `Custom` in the **Type** field.

3) In the attribute window, define the following attributes for the geo map report.

- **Name**: Optionally, change the technical name of the configured report. The technical name of the configured report must be unique.

The technical name is used for identification of the configured report in the context of technical processes such as merging the report configuration with a report configuration saved as an XML file. Therefore, the following is required when defining the name of a configured report:

- The name must be unique. The name must be unique regardless of the report folder it is structured in.
- The name may not be changed once a configured report has been created.
- A validation mechanism checks for correct syntax when defining a technical name. Please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.

- If the technical name of the configured report is changed, the name will be correctly updated in other configuration objects referencing the changed report during design time.

  Please note that the name of a changed configuration object will not be updated in guide pages or guide views. If you plan to change the name of a configuration object, the reference in the navigation page should be changed prior to changing the configuration object name. You can use the **Show Usage** functionality in the context menu of the configured report to evaluate whether the configured report has been added to a guide view or guide page.

- **Caption**: Enter a caption for the configured report. The caption you define here will be displayed in the **Reports Administration** functionality in the **Administration** module and the **Configured Reports** views of the Alfabet interface. If the configured report is assigned to an object view as a page view, the text will be displayed as the page view caption in the object view. The caption of the configured report may exceed the conventional 64 character limit.
- **Description**: Provide a short information about the configured report that is useful to Alfabet users. This comment will be displayed on the Alfabet user interface in the header of the configured report in a single line under the report caption and the **Description** field for the configured report in the table of all views listing configured reports, like the **Configured Reports** views of the **Search** functionalities. If the configured report is assigned to an object view as a page view, the text will be displayed under the page view caption as short description in the object view.

  In the configured report, the description will be truncated if it is longer than one line on the screen. Therefore, the description should be short to ensure complete display in the configured report header.

- **Usage Guideline**: Provide information that helps users to execute and interpret the configured report. This information will not be displayed directly on the user interface. The user can access the information using the following mechanisms:
  - If the user moves the cursor over the description provided for the configured report with the attribute **Description**, a tooltip opens that includes both the description and the usage guideline.
  - The **Options**  button will be displayed on the upper right with an option **Help On Filter Fields** to open a new window displaying the usage guideline in addition to any filter field hints, if configured.

    For views that have filters defined, the **Options**  button will only be displayed if the filter panel either allows batch clearing of filter fields or collapsing and expanding of the filter panel. For more information about the configuration of these functionalities, see Configuring the Complete Filter Area to be Collapsible and Allowing the User to Clear the Filter Area.

- **Technical Comment**: Provide information that is useful for solution designers working with Alfabet Expand in order to better understand the configured report from a technical perspective. The **Technical Comment** will not be displayed in the user interface. Nevertheless, you can configure the interface to display the information in, For example, the attribute section of the configured report's object profile in the **Reports Administration** functionality.

- **Template**: Select `MapChartReport`.
- **Report State**: The **Report State** attribute is view only and is set to `Plan` for new configured reports. The configured report can only be edited when the **Report State** attribute is set to `Plan`. After finishing all configuration steps described in the following, the **Report State** attribute must be set to `Active` as described in the section General Guidelines for Creating Configured Reports. The configured report is then visible to users in the Alfabet user interface.
- **Help Index**: Specify the location of the external Help file using the following syntax:(For example,:). The external help file will be displayed when the user clicks the context-sensitive Help link in the standardHelp menu.

  Server variables can be used in the Help links. Server variables allow you to define all or part of the URL definition in the server alias configuration of the Alfabet Server instead of directly defining it in the configuration in Alfabet Expand. Use of server variables is useful, For example, in order to change a Web server name defined in multiple URL definitions in Alfabet Expand. Instead of changing all URL definitions, only the variable value in the server alias configuration must be updated. The use of server variables is described in the section Using Server Variables in Web Link and Database Server-Related Specifications in the chapter Configuring Reports.

- **Apply to Class**: When the configured report is applied to a class, a base object can be selected for the configured report. A filter for selection of a base object will automatically be set on top of the configured report. When configuring queries for the configured report, you can use the parameter `BASE` to refer to the object selected in the filter and define search conditions dependent on the base object. To apply the configured report to a base object class or object class stereotype, click the **Browse** button to open a dialog box and select the object class or object class stereotype to which you want to apply the configured report and click **OK**.

  Note the following:

  - If the configured report is assigned to the object view for the relevant object class, the configured report will automatically open with the current object selected as the base object even if the **Apply to Class** attribute is not set.
  - If the configured report is applied to an object class stereotype, the filter field for selection of the base object will use the caption of the object class stereotype as caption of the filter field and the selector defined in the class settings for the object class stereotype for selection of the base object.
  - Alternatively, the base object of a configured report can be set with the **Base Query** attribute. Whereas the **Apply to Class** attribute enables the user to determine the current base object by setting a filter, the **Base Query** attribute evaluates the base object by executing a query when the configured report is opened. A filter field is not available and the user cannot change the base object.
  - The **Apply to Class** attribute also influences the availability of the configured report in the Alfabet interface. Select a class to make the configured report available in the **Configured Reports** page view in the object view for the selected object class. If you do not set the attribute, the configured report will only be available in the **Configured Reports** functionality. The configured

report can be assigned to object views and wizards independent of the attribute.

- **Base Object Query**: If a base object query is defined for the configured report, a base object will be evaluated for the configured report at runtime. When configuring the other queries for the configured report, you can use the parameter `BASE` to refer to the object returned by the base object query and define search conditions dependent on the base object. To apply the configured report to a base class, click the **Browse** button to open a text editor, define either a native SQL query or an Alfabet query that returns a single object, and click **OK**.

  Note the following:

  - Alternatively, the base object of a configured report can be set by means of the **Apply To Class** attribute. Whereas the setting of the **Base Object Query** attribute triggers the evaluation of the base object by executing a query when the configured report t is opened, the **Apply to Class** attribute enables the user to determine the current base object by setting of an automatically-generated filter.
  - The query defined for the base object query is executed when the user opens the configured report to evaluate the base object at runtime. Parameters referring to the current working environment can be used in the query to find a base object dependent on the user opening the configured report or the user profile used for login, For example,. A base object query may also find, For example, the ICT object that a specific application is currently assigned to.
  - If the **Base Object Query** attribute is set and the configured report is assigned as a view to an object profile, the Alfabet parameter `BASE` referring to the `REFSTR` of the object that the user is working with when the configured report is opened can be used in the base object query.

- **Selector Behavior**: The attribute can be used to exclude configured reports that are defined for special purposes, like For example, triggering of data export via the RESTful API, from being selectable in the **Configured Reports** table. By default, the attribute is set to `Visible` and the configured report is visible in the selector for adding configured reports to the **Configured Reports** table if all other conditions for access to the configured report apply. If the attribute is set to `NotVisible`, the configured report is excluded from the selector for adding configured reports to the **Configured Reports** functionality. Please note however, that this setting is not excluding the configured report from any standard views or customer configured views and selectors, but exclusively from the standard selector for configured reports implemented in Alfabet.

The attribute cannot be edited in the attribute window directly. To change the setting, right-click the configured report and select **Set Report Selector Behavior to 'Not Visible'** or **Set Report Selector Behavior to 'Visible'** respectively.

- **Applicable for AlfaBot**: Specify whether the configured report is accessible via the AlfaBot. By default, the attribute will be set to `True` for new reports. Set the attribute to `False` if the configured report should not be opened outside of a specific context. For example, this may be relevant for reports that are specifically designed to be embedded in an object cockpit.

This attribute is only relevant if the AlfaBot is implemented. For information about the implementation of the AlfaBot, see Implementing the AlfaBot for Navigation via a Full-Text Command.

- **Business Problem Statement**: Optionally enter a description that will help users to search for the configured report in the AlfaBot. The text is not displayed in the user interface, but it is of special relevance for the search mechanisms implemented for the AlfaBot. If the report caption entered by the user in the AlfaBot for a request to open a configured report does not match the caption of one of the available configured reports, the AlfaBot will split the caption entered by the user into keywords and will perform a keyword search on the **Caption**, **Description** and the **Business Problem Statement** attributes of the configured reports that are accessible via the AlfaBot. In addition to a keyword search, a synonym search is performed on the text provided in the **Business Problem Statement** attribute.

  This attribute is only relevant if the AlfaBot is implemented. For information about the implementation of the AlfaBot, see Implementing the AlfaBot for Navigation via a Full-Text Command.

- **Base Classes**: Define the object classes that must be available to run the configured report. If this attribute is set, the Alfabet Server checks whether the object classes specified as base classes are excluded from the view scheme used to access the configured report. If the user does not have access permissions to one of the base classes, the configured report will be disabled. To set the attribute, click the **Browse**  button to select the object classes that are required to run the configured report and click **OK**.

- **Can Create Express View**: Select `True` if an express view may be created for the report. Select `False` if an express view may not be created for the report. Please note that for reports that have a custom report view, the setting must be consistent with the setting of the attribute **Can Create Express View** of the custom report view.

  An express view is a link to a specific Alfabet view displaying data that allows individuals inside or outside of the user community to view information in Alfabet. When the express view is created, an email notification is automatically mailed to a specified recipient. The recipient receives a URL that allows him/her to access the current page view in Alfabet. For more information, see the section *Sending and Receiving Express Views* in the reference manual *Getting Started with Alfabet*.

4) In the explorer, right-click the configured report and select **Create Custom Report View**. The view editor opens. In the explorer, the custom report view is displayed as a subordinate object of the configured report. The attributes for the custom report view are displayed in the attribute window. You can optionally edit the following attributes:

   - **Name**: Optionally you can change the name for the custom report view. The **Name** must be unique for custom report views.

   - **Can Create Bookmark**: Select `True` if a bookmark may be created for the configured report. Select `False` if a bookmark may not be created for the configured report.

   - **Can Create Express View**: Select `True` if an express view may be created for the configured report. Select `False` if an express view may not be created for the configured report. Please note that the setting must be consistent with the setting of the attribute **Can Create Express View** of the configured report.

An express view is a link to a specific Alfabet view displaying data that allows individuals inside or outside of the user community to view Alfabet information. When the express view is created, an e-mail notification is automatically mailed to the specified recipient who receives a URL that allows him/her to access the current page view in Alfabet. For more information, see *Sending and Receiving Express Views* in the reference manual *Getting Started with Alfabet*.

The custom report view initiates the **Report Assistant**. A custom report view shall never be deleted from the configured report after the **Report Assistant** was already used to configure the report. When you delete the custom report view of an existing template-based configured report and create a new one, the **Report Assistant** is reset to the example specified by default and your configuration is lost.

5) In the explorer, right-click the configured report and select **Start Alfabet Report Assistant**. The **Report Assistant** opens.
6) Define the geo map report as described below in the section Configuring the Content of the Geo Map Report. For general information about working with the **Report Assistant**, see Configuring the Report with the Report Assistant.
7) Delete or edit the parameter panel for the geo map report. For more information, see Configuring Filter Layout and Defining Filters for Configured Reports and Selectors.
8) Click **OK** to save the configuration.
9) In the toolbar, click the **Save** button to save your changes.
10) In the explorer, right-click the configured report and select **Review Report**. The configured report opens in a web browser. If the results are not as expected, you can edit the configured report until the output of the configured report meets your expectations.

Configuring the Content of the Geo Map Report

Geo map reports are designed in the **Report Assistant**. For information about how to open the **Report Assistant**, see Creating a New Geo Map Report. For more information about how to work with the **Report Assistant**, see Configuring the Report with the Report Assistant.

The content and layout of a geo map report is defined by setting the attributes of the explorer element in the attribute pane of the **Report Assistant** window.

Geo map reports can in general be defined using Alfabet query language or native SQL. But drill down to other configured reports from a geo map element, For example, opening continent maps from a world map, requires a native SQL query.

The following table lists the types of explorer node elements that can be added to the report configuration to specify the content of the geo map report. Information is provided in the table about the purpose of each explorer node element:

Explorer Node Element	Required to Configure:
Root Node	General layout of the geo map report and mapping of the information provided by the queries of the geo map report to the relevant graphic display in the geo map report.
Map Element Query	Definition of a query that defines the display of map elements depending on data in the Alfabet database.
Marker Query	Definition of a query that defines the display of markers depending on data in the Alfabet database.
Marker Connection Query	Definition of a query that defines the display of connections between markers depending on data in the Alfabet database.

Defining the Basic Design of the Geo Map Report

The basic design of the geo map report is designed directly in the attributes of the root node element of the **Report Assistant** explorer. The following table lists only the attributes related to basic settings. Attributes that are relevant for the configuration of map elements, markers or marker connections are not included.

The basic design includes the specification of the map to be displayed and the general display of colors, labels and tooltips on the map:

Section/Attribute	Description
Hover Color	Defines the color that shall be used to color a region on mouse-over.
Title	Defines the title that is displayed on top of the map. This attribute is optional. If it is not set, the graphic is displayed without a title.
Map Name	Defines the FusionMap that shall be displayed in the geo map report. Enter the name of the map as defined in the property `MAPNAME` of the object classes `ALFA_FUSIONMAPMARKER`Sand `ALFA_FUSIONMAPINFO`.
Number Suffix	This feature is not yet documented.
Show Labels	Defines whether labels are shown on colored regions of the map.

Section/Attribute	Description
Show Tooltips	Defines whether a tooltip is displayed on colored regions of the map.
Include Values in Labels	This feature is not yet documented.
Width	Defines the overall width of the map graphic.
Height	Defines the overall width of the map graphic.

A geo map report with only the basic design settings cannot be displayed using For example, the **Review Report** functionality in the context menu of the custom report view. The geo map report must contain either marker or map element definitions to be displayed.

Defining Coloring of Map Regions Dependent on Data in the Alfabet database

Regions of the maps can be colored depending on a numeric value derived from data in the Alfabet database.

The colored regions can display a label, that means a text in the colored map element, and a tooltip that is displayed when a user moves the mouse over the colored region. The label text is either derived from the data in the map element query or the short name of the map element stored in the object class property `SHORTLABEL` of the `ALFA_FUSIONMAPINFO` object class storing the map element data is displayed either with or without information about the value the coloring is based on.

To color the map elements of a geo map report, the following configuration steps are required:

- A query must be defined that returns the data that shall be displayed.
- The columns in the query containing the relevant information must be mapped to attributes in the root node of the geo map report assistant to enable the graphic display of the data returned by the query.
- Optionally the range of colors for coloring of the regions according to data in the query and the display of labels can be defined in the attributes of the root node of the report assistant.

To color the map elements, a map element query has to be defined in the explorer node **Map Element Query**. To define an Alfabet query, use the attribute **Alfabet Query** to open the **Alfabet Query Builder** or the attribute **Query as Text** to define the Alfabet query in a text editor. To define a native SQL query, use the attribute ***Native Sql*** to define the query.

The query must return a tabular dataset providing the data required for display of the map. The information which column contains which information must then be added to the attributes of the root nodes of the report assistant. The relevant attributes are available in the section **Data**. For each relevant attribute, a column name of the map element query must be defined.

For information about the column names in Alfabet queries and native SQL queries see Defining Column Names and Captions in the chapter Defining Queries.

The following table lists the attributes of the root node and informs about the kind of data that must be returned in the defined column of the query:

Attribute of the root node of the report assistant returning the column name	Required data to be returned for the defined column in the query	Specification is
Id Column	The identifier of the map element on the map. The column must contain the values of the ID property of existing objects of the object class `ALFA_FUSIONMAPINFO`.	Mandatory
Label Column	A string to be displayed as label on the map element. For more information about the definition of labels see Defining the Display of Labels and Tooltips in the Geo Map Report.	Optional. If no column is defined in the Label Column attribute, the `SHORTLABEL` object class property of the current map element is displayed as label.
Tooltip Column	A string to be displayed as tooltip when the user moves the cursor over the map element in the geo map report. For more information about the definition of tooltips see Defining the Display of Labels and Tooltips in the Geo Map Report.	Optional. If no column is defined in the Tooltip Column attribute, the `LABEL` object class property of the current map element and the value defined in the value column is displayed as tooltip.
Color Column	A HTML compliant color specification for coloring of the map element.	Optional. If no column is defined in the **Color Column** atttibute, the coloring of the map element must be defined by defining a color range in the **Color Range** attribute of the root node. Please note that a legend is only available for definition of colors as color range.
Value Column	An integer or real value that the coloring of the map elements depends on.	Mandatory.

Attribute of the root node of the report assistant returning the column name	Required data to be returned for the defined column in the query	Specification is
View Column	The definition of a link to another configured report that opens when the user clicks on the label in the map element. The required syntax for the link is: `View=Report:`*`ReportName`* For detailed information about the definition of the link including parameter values handed over to the configured report that opens see the section Defining Navigation from the Geo Map Report to Alfabet Views.	Optional.

The color displayed for each map element depending on the numeric value in the dataset column defined with the **Value Column** attribute can be set in three ways:

- The color can be read from the respective row of the query in the column defined with the attribute **Color Column** of the root node of the report assistant.
- In the **Color Range** attribute of the root node of the report assistant, a color can be defined for different ranges of values that might be returned in the dataset column defined as **Value Column**.
- In the **Color Range** attribute of the root node of the report assistant, a color gradient can be defined with a fixed color setting for the lowest and the highest possible value that might be returned in the dataset column defined as **Value Column**. The colors for the map elements then correspond to the gradient color matching their position in the range of values.

A legend for the coloring is only available for the definitions via the **Color Range** attribute.

Defining the Display of Color in the Map Element Query

To define the coloring of map elements in the query defined as **Map Element Query**:

1) In the query defined as **Map Element Query**, add a column to the tabular output of the query that returns an HTML compliant color code (For example, #336699 for dark blue).
2) In the attributes of the root node of the report assistant, set the following attributes in the section **Data**:
   - **Color Column**: Enter the name of the column in the dataset containing the label text.

Defining the Display of Colors via Color Definitions in the Root Node of the Report Assistant

To define the colors for regions by defining fixed colors for ranges of values:

1) In the root node of the report assistant, expand the attribute **Color Range**.
2) Set the attribute **Type** to `Simple`.
3) Define the following attributes of the **Color Range**:
4) Click into the value field of the attribute **Colors** and click on the button that is displayed on the right of the field. The **Edit Object Collection** window opens.
5) To define coloring for a range of values, click the button **Add** and define the following in the **Properties** window:
   - **Caption**: Enter a caption that is displayed for the color in the legend.
   - **Min Value**: Enter the lowest numeric value for the range.
   - **Max Value**: Enter the highest numeric value for the range.

     Max Value must be higher than **Min Value**. If you want to define a range with only one value, you must define the value as **Min Value** of this range and for **Max Value** of this range, you must enter the same numeric value as for the **Min Value** of the next range. The **Min Value** setting of the next range supersedes the **Max Value** of the current range.

   - **Color**: Select a color from the color selector.
6) Repeat the previous steps for all ranges that you want to define.
7) Click **OK** to save your changes.

Defining the Display of Colors via a Color Range Definition in the Root Node of the Report Assistant

To define the colors for regions by defining a range of colors:

1) In the root node of the report assistant, expand the attribute **Color Range**.
2) Define the following attributes of the **Color Range**:
   - **Type**: Set to Gradient.
   - **Min Value**: Enter the lowest numerical value defined for a result.
   - **Start Label**: Enter the label to be displayed on the lowest end of the gradient in the legend.
   - **End Label**: Enter the label to be displayed on the highest end of the gradient in the legend.
   - **Start Color**: Enter the color that shall be used for the lowest value of the range of values used for coloring.
3) Click into the value field of the attribute **Colors** and click on the button that is displayed on the right of the field. The **Edit Object Collection** window opens.
4) Click the button **Add** and define the following in the **Properties** window:
   - **Min Value**: Enter the highest numeric value for the range of values used for coloring.
   - **Max Value**: Enter the highest numeric value for the range of values used for coloring.

Max Value must be identical to **Min Value** to define a color gradient.

- **Color**: Select a color from the color selector.

5) Click **OK** to save your changes.

Defining Markers to be Displayed in the Map

A marker is a spot on the map that can either be marked with a circle, a polygon or an icon. Markers can only be set for locations that are stored in the database table `ALFA_FUSIONMAPMARKERS`.

The shape of the markers as well as the colors of polygons and circles can be defined in the geo map report depending on a numeric value derived from data in the Alfabet database.

The markers can display a label, that means a text in the colored map element, and a tooltip that is displayed when a user moves the mouse over the colored region. The label text is derived from the data in the marker query.

To define the markers of a geo map report, three configuration steps are required:

- A query must be defined that returns the data that shall be displayed.
- The columns in the query containing the relevant information must be mapped to attributes in the root node of the geo map report assistant to enable the graphic display of the data returned by the query.
- Optionally the shapes of markers can be defined in the attributes of the root node of the report assistant.

To define markers, a marker query has to be defined in the explorer node **Marker Query**. To define an Alfabet query, use the attribute **Alfabet Query** to open the **Alfabet Query Builder** or the attribute **Query as Text** to define the Alfabet query in a text editor. To define a native SQL query, use the attribute **Native Sql** to define the query.

The query must return a tabular dataset providing the data required for display of the marker. The information which column contains which information must then be added to the attributes of the root nodes of the report assistant. The relevant attributes are available in the sections **Data** and **Special Marker Data**. For each relevant attribute, a column name of the marker query must be defined.

For information about the column names in Alfabet queries and native SQL queries see Defining Column Names and Captions in the chapter Defining Queries.

The following table lists the attributes of the root node and informs about the kind of data that must be returned in the defined column of the query:

Attribute of the root node of the report assistant returning the column name	Required data to be returned for the defined column in the query	Specification is
Data		
Id Column	The identifier of the marker on the map. The column must contain the values of the ID property of existing objects of the object class `ALFA_FUSIONMAPMARKER`.	Mandatory
Label Column	A string to be displayed as label on the marker. For more information about the definition of labels see Defining the Display of Labels and Tooltips in the Geo Map Report.	Optional. If no column is defined in the **Label Column** attribute, the marker is displayed without a label.
Tooltip Column	A string to be displayed as tooltip when the user moves the cursor over the marker in the geo map report. For more information about the definition of tooltips see Defining the Display of Labels and Tooltips in the Geo Map Report.	Optional. If no column is defined in the **Tooltip Column** attribute, no tooltips are displayed.
View Column	The definition of a link to another configured report that opens when the user clicks on the label in the map element. The required syntax for the link is: `View=Report:`*`ReportName`* For detailed information about the definition of the link including parameter values handed over to the configured report that opens see the section Defining Navigation from the Geo Map Report to Alfabet Views.	Optional.
Special Marker Data		
X Pos Column	The column must contain the values of the `XPOS` object class property of the current object of the object class `ALFA_FUSIONMAPMARKER`.	Mandatory
Y Pos Column	The column must contain the value of the `YPOS` object class property of the current object of the object class `ALFA_FUSIONMAPMARKER`.	Mandatory

Attribute of the root node of the report assistant returning the column name	Required data to be returned for the defined column in the query	Specification is
Shape Column	The column must return a string that is identical to the value of the `Name` attribute of one of the defined shapes of the geo map report.	Optional. If no shape column is defined, all markers will be displayed as circles in a light color.

Defining the Display of Marker Shapes Depending on Results Returned by the Marker Query

To define the shaping of markers:

1) In the root node of the report assistant, click into the value field of the attribute **Colors** and click on the button that is displayed on the right of the field. The **Edit Object Collection** window opens.
2) Click the button **Add** and define the following in the **Properties** window:
   - **Type**: Select the type of marker that you want to define from the drop down list. A marker can either be displayed as circle, polygon or image.
   - **Name**: Enter a unique name for the marker shape. This value is used to identify the marker in the marker query.
   - **Radius**: Enter the radius of the marker. For images this attribute is not relevant.
   - **Sides Count**: Enter the number of sides for a polygon. For images and circles this attributes is not relevant..
   - **Background Color**: Select the background color of the marker. For images this attribute is not relevant.
   - **Border Color**: Select the border color of the marker. For images this attribute is not relevant.
   - **Legend Text**: Select the text that shall be displayed for the shape in the legend.
   - **Icon Type**: If the **Type** attribute is set to **Image**, select an icon gallery from the drop-down list.
   - **Icon**: If the **Type** attribute is set to **Image**, select an icon from the icon gallery selected with the **Icon Type** attribute.
3) Repeat the previous step for all shapes that you want to define.
4) Click **OK** to save your changes.
5) In the query defined as **Marker Query**, add a column to the tabular output of the query that returns the value of the Name attribute of one of the defined shapes.
6) In the attributes of the root node of the report assistant, set the following attributes in the section **Special Marker Data**:

- **Shape Column**: Enter the name of the column in the dataset containing the name of the shape.

Defining Connections Between Markers in the Map

Markers in the map can be connected by lines. The color and the thickness of the marker connections can be configured in the query defining each connection.

To define connections between the markers of a geo map report, two configuration steps are required:

- A query must be defined that returns the data that shall be displayed.
- The columns in the query containing the relevant information must be mapped to attributes in the root node of the geo map report assistant to enable the graphic display of the data returned by the query.

To define marker connections, a marker connection query has to be defined in the explorer node **Marker Connection Query**. To define an Alfabet query, use the attribute **Alfabet Query** to open the **Alfabet Query Builder** or the attribute **Query as Text** to define the Alfabet query in a text editor. To define a native SQL query, use the attribute **Native Sql** to define the query.

The query must return a tabular dataset providing the data required for display of the marker connection. The information which column contains which information must then be added to the attributes of the root nodes of the report assistant. The relevant attributes are available in the sections **Data** and **Special Marker Connection Data**. For each relevant attribute, a column name of the marker query must be defined.

For information about the column names in Alfabet queries and native SQL queries see Defining Column Names and Captions in the chapter Defining Queries.

The following table lists the attributes of the root node and informs about the kind of data that must be returned in the defined column of the query:

Attribute of the root node of the report assistant returning the column name	Required data to be returned for the defined column in the query	Specification is
Data		
Color Column	A HTML compliant color specification for coloring of the map element.	Optional. If no column is defined in the **Color Column**, black connections are displayed.

Attribute of the root node of the report assistant returning the column name	Required data to be returned for the defined column in the query	Specification is
View Column	The definition of a link to another configured report that opens when the user clicks on the label in the map element. The required syntax for the link is: `View=Report:`*`ReportName`* For detailed information about the definition of the link including parameter values handed over to the configured report that opens see the section Defining Navigation from the Geo Map Report to Alfabet Views.	Optional.
Special Marker Data		
From Column	The column must contain the value of the ID object class property of the object of the object class **ALFA_FUSIONMAP-MARKER** in the map where the connection starts.	Mandatory
To Column	The column must contain the value of the ID object class property of the object of the object class **ALFA_FUSIONMAP-MARKER** in the map where the connection ends.	Mandatory
Thickness Column	The column must return a number that is used to define the thickness of the connection.	Optional. If no thickness column is defined, all connections have the same thickness.

Defining the Display of Labels and Tooltips in the Geo Map Report

For each region that is colored in the map, that means a map element, and for each marker, a label and a tooltip can be displayed. The content of the labels and tooltips can either be configured in the query defined as **Map Element Query** or **Marker Query**.

The display of labels and tooltips for map elements only can alternatively be configured via the attributes of the root node of the report assistant. A definition in the query overwrites settings in the attributes of the root node. You can also define the labels via the root nodes and the tooltips via the query or vice versa.

Defining the Display of Labels and Tooltips in the Query

To define the text displayed in the labels and tooltips in a query:

1) In the query defined as **Map Element Query** or **Marker Query**, add the following to the output of the query:
   - To define a label text, add a column to the tabular output of the query that returns a string to be displayed as label text.
   - To define a tooltip text, add a column to the tabular output of the query that returns a string to be displayed as label text.

   Please note the following:
   - If you want to define the same text as tooltip and label, it is sufficient to define the text once and map it to both the label and tooltip in step 2.
   - If you want to define the label or tooltip in **Map Element Query** and **Marker Query** the column names in the tabular output of both queries must be identical.

2) In the attributes of the root node of the report assistant, set the following attributes in the section **Data**:
   - **Label Column**: Enter the name of the column in the dataset containing the label text.
   - **Tooltip Column**: Enter the name of the column in the dataset containing the tooltip text.

Defining the Display of Labels and Tooltips of Map Elements in the Root Node of the Report Assistant

If the attribute **Label Column** for the root node of the geo map report assistant is not set and the attribute **Show Labels** is set to `True`, the short label of the map element is displayed as label on colored regions in the map. For markers, no label is displayed.

If the attribute **Tooltip Column** for the root node of the geo map report assistant is not set and the attribute **Show Tooltips** is set to `True`, the label of the map element is displayed as tooltip on colored regions in the map.

Optionally, the numeric value that the coloring is based on can be displayed. The following attributes of the root node of the geo map report assistant must be set to define display of values in labels and tooltips:

Section/Attribute	Description
Common	
Number Suffix	Optionally a suffix can be defined that is displayed after the value to explain the meaning of the value. For example, if the numeric value refers to the number of market products that are available in a region, the suffix " products" may be defined. Please note that the

Section/Attribute	Description
	suffix is displayed directly after the number. If you want a whitespace to be displayed between numeric value and suffix, you must add the whitespace to the suffix specification.
Include Values in Labels	Numeric values are displayed in the labels and tooltips if this attribute is set to `True`. Please note that this attribute is only applied for tooltips and labels not based on a value in the query.

Deactivating the Display of Labels and Tooltips

By default labels as well as tooltips are displayed for the map elements and markers of the geo map report. You can hide labels and/or tooltips by setting the following attributes of the root node of the report assistant to `False`:

Section/Attribute	Description
Common	
Show Labels	Defines whether labels are shown on colored regions of the map.
Show Tooltips	Defines whether a tooltip is displayed on colored regions of the map.

Defining Navigation from the Geo Map Report to Alfabet Views

The geo map report can be configured to open another configured report.

Navigation to a defined configured report from the geo map report requires the following configurations:

1) The labels of the map elements or markers that shall provide the link must be defined in the query.
2) In the native SQL query defined as **Map Element Query** or **Marker Query** of the geo map report, a column must be added for the definition of the navigation target.

   The syntax required for definition of the view is the following:

   `View=Report:`*`ReportName`*

   `ReportName` is the name of the configured report.

   Optionally, values can be handed over to the configured report that opens.

The query of the configured report that opens must contain each parameter definition as an Alfabet parameter defined as

`@ParameterName`

Please note that the old syntax of the Alfabet query language defining Alfabet parameters as:`ParameterName` is not supported.

The Alfabet parameters can be used in `WHERE` clauses of the query without defining a filter for the configured report, because the Alfabet parameter values are already defined in the navigation link.

Optionally, filters can be defined for the configured report to allow the user opening the configured report to alter the Alfabet parameter values. In this case the Alfabet parameter values in the navigation link are used as default values for the filter when opening the configured report. The filter settings are also stored in the user context settings for the configured report. That means, when the user first opens the configured report via drill-down from a chart report and then afterwards opens the configured report in any other context, For example, via the **Configured Reports** functionality, the configured report will open with the last filter settings derived from the drill-down.

For each Alfabet parameter the following string must be added to the navigation target definition:

`&ParameterName=ParameterValue`

The setting of three Alfabet parameters would result in the following navigation link:

`View=Report:ViewName&ParameterName1=ParameterValue1&ParameterName2=ParameterValue2&ParameterName1=ParameterValue2`

3) In the attributes of the root node of the report assistant, enter the name of the column containing the navigation target specification into the attribute **View Column**.

- Coloring of countries/regions
- Setting markers
- Configuring connections between markers

Creating Configured Reports With Editing Capabilities

Software AG offers a number of configured reports that allow object editing:

- A tabular report for multi-editing of object class properties including indicator values and role assignments.
- A Kanban report to maintain and change object relations or enumeration based properties in a tabular or matrix based interface.
- A matrix for multi-editing of business supports.
- A matrix for multi-editing of object relations that provides editability via an editor.
- A matrix for multi-editing of object relations that provides editability directy in the matrix without opening an editor.
- A tabular report for multi-editing of indicators.

- A tabular report for multi-editing of questionaire questions.
- A tabular report providing the functionality available in the standard data capture environments.
- A tabular report providing the functionality of the user administration functionality.

Configured reports with editing capabilities are based on a template.

Configured reports providing a standard data capture environment or access to user administration functionality are of the type Query or NativeSQL and are described in the context of the definition of these types of reports. For more information, see and

- Defining a Data Capture Environment for a Configured Report of the Type Query for the definition of a data capture environment based on an Alfabet query.
- Defining a Data Capture Environment for a Configured Report of the type NativeSQL for the definition of a data capture environment based on a native SQL query.
- Defining a User Management Functionality via a Configured Report of the Type Query for the definition of a user administration functionality based on an Alfabet query.
- Defining a User Management Functionality via a Configured Report of the Type NativeSQL for the definition of a user administration functionality based on a native SQL query.

All other types of configured reports with editing capability are of the type Custom. For these configured reports, a **Report Assistant** is available that allows easy definition of configured report visualization by setting predefined attributes and defining queries. The definition of these configured reports is described in this section.

To generate a configured report with editing capabilities:

- In the **Reports** tab in Alfabet Expand, create a new configured report and define the general attributes for the configured report and the attributes specific for template-based configured reports. For more information about creating a configured report and defining the configured report's attributes, see Creating a New Report for Multi-Editing of Objects below.
- Define a custom report view for the configured report. For more information, see Creating a New Report for Multi-Editing of Objects below.
- Open the **Report Assistant** and configure the configured report.
  - For more information about defining a tabular report for multi-editing of object class properties in an editor, see Defining Tabular Configured Reports for Multi-Editing of Object Class Properties.
  - For more information about defining a Kanban report to maintain and change object relations or enumeration based properties in a tabular or matrix based interface, see Configuring Kanban Reports for Maintenance of Relationships between Objects.
  - For more information about defining a matrix for multi-editing of business supports or object relations, see Configuring Matrices for Multi-Editing of Object Relations or Business Supports.

        - For more information about defining an editable matrix for the specification of the affected architecture for object classes like projects, IT policies, or value nodes, or for the specification of business data usage, see XXX.
        - For more information about defining an editable tabular report for the definition of aspect indicators, see XXX.
        - For more information about defining a tabular report for multi-editing of indicators, see Configuring a Report for Multi-Editing of Indicators.
        - For more information about defining a tabular report providing the functionality available via the **Job Schedule** functionality, see XXX.
    - If the configured report has filters defined, filter fields must be added manually to the custom report view. For more information, see Defining Filters for Configured Reports and Selectors in the chapter Defining Queries.
    - Define access rights for the configured report in the **Reports Administration** functionality of Alfabet. By default, access is possible for all users. For more information, see the chapter *Defining and Managing User Access to Configured Reports* in the reference manual *User and Solution Administration*.
    - Set the state of the configured report to `Active`. For information on how to change theattribute of the configured report, see.

The following information is available:

- Creating a New Report for Multi-Editing of Objects
- Defining Tabular Configured Reports for Multi-Editing of Object Class Properties
    - Configuring a Report to Restrict Drop-Down List Content Via a Query
    - Configuring a Report to Restrict Drop-Down List Content Depending on A Master Control
    - Considering Access Permissions in Configured Reports for Multi-Editing of Object Class Properties
        - Edit Permissions for Objects
        - Edit Permissions for Object Class Properties
- Configuring Kanban Reports for Maintenance of Relationships between Objects
    - Defining the Content and Structure of the Kanban Report
    - Defining the Toolbar Buttons Available for the Kanban Report
    - Defining the Layout of the Kanban Report
    - Defining Tooltips for the Kanban Report
    - Displaying Evaluation Aspects in Kanban Reports
    - Defining Connections between Objects in the Cells of a Kanban Report
    - Defining Navigation from the Kanban to Other Views
- Configuring Matrices for Multi-Editing of Object Relations or Business Supports

Creating a New Report for Multi-Editing of Objects

To create a new configured report with editing capabilities in Alfabet Expand:

1) In the **Reports** tab, right-click the **Reports** folder or a sub-folder thereof and select **Create New Report**. You will see the new configured report listed below the selected folder.

   As an alternative to defining all attributes for a new configured report, you can copy and paste an existing configured report and make modifications, as needed. To do so, right-click the existing configured report that you want to copy and select **Copy**. Right-click the new configured report and select **Paste**. Modify the necessary attributes described below.

2) In the attribute window select `Custom` in the **Type** attribute.
3) In the **Template** attribute, select the template the report shall be based on:
   - `EditableClassViewReport` to create a tabular report for multi-editing of object class properties including indicator values and role assignments.
   - `KanbanReport` to maintain and changing object relations or enumeration based properties in a tabular or matrix based interface.
   - `ITMAP_TableReport` to create a matrix for multi-editing of business supports.
   - `Relationships_TableReport` to create a matrix for multi-editing of object relations that provides editability via an editor.

- `Relationships_EditableTableReport` to create a matrix for multi-editing of object relations that provides editability directy in the matrix without opening an editor.
- `EvaluationReport` to create a tabular report for multi-editing of indicators.

4) In the attribute window, you can define the following attributes according to demand:

- **Name**: Optionally, change the technical name of the configured report. The technical name of the configured report must be unique.

  The technical name is used for identification of the configured report in the context of technical processes such as merging the report configuration with a report configuration saved as an XML file. Therefore, the following is required when defining the name of a configured report:

  - The name must be unique. The name must be unique regardless of the report folder it is structured in.
  - The name may not be changed once a configured report has been created.
  - A validation mechanism checks for correct syntax when defining a technical name. Please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.
  - If the technical name of the configured report is changed, the name will be correctly updated in other configuration objects referencing the changed report during design time.

  Please note that the name of a changed configuration object will not be updated in guide pages or guide views. If you plan to change the name of a configuration object, the reference in the navigation page should be changed prior to changing the configuration object name. You can use the **Show Usage** functionality in the context menu of the configured report to evaluate whether the configured report has been added to a guide view or guide page.

- **Caption**: Enter a caption for the configured report. The caption you define here will be displayed in the **Reports Administration** functionality in the **Administration** module and the **Configured Reports** views of the Alfabet interface. If the configured report is assigned to an object view as a page view, the text will be displayed as the page view caption in the object view. The caption of the configured report may exceed the conventional 64 character limit.
- **Description**: Provide a short information about the configured report that is useful to Alfabet users. This comment will be displayed on the Alfabet user interface in the header of the configured report in a single line under the report caption and the **Description** field for the configured report in the table of all views listing configured reports, like the **Configured Reports** views of the **Search** functionalities. If the configured report is assigned to an object view as a page view, the text will be displayed under the page view caption as short description in the object view.

  In the configured report, the description will be truncated if it is longer than one line on the screen. Therefore, the description should be short to ensure complete display in the configured report header.

- **Usage Guideline**: Provide information that helps users to execute and interpret the configured report. This information will not be displayed directly on the user interface. The user can access the information using the following mechanisms:
  - If the user moves the cursor over the description provided for the configured report with the attribute **Description**, a tooltip opens that includes both the description and the usage guideline.
  - The **Options** button will be displayed on the upper right with an option **Help On Filter Fields** to open a new window displaying the usage guideline in addition to any filter field hints, if configured.

    For views that have filters defined, the **Options** button will only be displayed if the filter panel either allows batch clearing of filter fields or collapsing and expanding of the filter panel. For more information about the configuration of these functionalities, see Configuring the Complete Filter Area to be Collapsible and Allowing the User to Clear the Filter Area.

- **Technical Comment**: Provide information that is useful for solution designers working with Alfabet Expand in order to better understand the configured report from a technical perspective. The **Technical Comment** will not be displayed in the user interface. Nevertheless, you can configure the interface to display the information in, For example, the attribute section of the configured report's object profile in the **Reports Administration** functionality.
- **Report State**: The **Report State** attribute is view only and is set to `Plan` for new configured reports. The configured report can only be edited when the **Report State** attribute is set to `Plan`. After finishing all configuration steps described in the following, the **Report State** attribute must be set to `Active` as described in the section General Guidelines for Creating Configured Reports. The configured report is then visible to users in the Alfabet user interface.
- **Help Index**: Specify the location of the external Help file using the following syntax:(For example,:). The external help file will be displayed when the user clicks the context-sensitive Help link in the standardHelp menu.

  Server variables can be used in the Help links. Server variables allow you to define all or part of the URL definition in the server alias configuration of the Alfabet Server instead of directly defining it in the configuration in Alfabet Expand. Use of server variables is useful, For example, in order to change a Web server name defined in multiple URL definitions in Alfabet Expand. Instead of changing all URL definitions, only the variable value in the server alias configuration must be updated. The use of server variables is described in the section Using Server Variables in Web Link and Database Server-Related Specifications in the chapter Configuring Reports.

  For matrix based report for data maintenance, Software AG provides a standard help file that can be linked to configured reports to provide help for the user. To specify a link to a page in the standard help in Alfabet, the syntax specified above is not required. The help will be available when the name of the HTML file is entered without any prefix. The following file names can be specified in the **Help Index** for help on reports for multi-editing of objects:

- `Understand_RelationshipsMatrixReport.html` (for `Relationships_TableReport`)
- `Understand_BSMatrixReport.html` (for `ITMPAP_TableReport`)

- **Apply to Class**: When the configured report is applied to a class, a base object can be selected for the configured report. A filter for selection of a base object will automatically be set on top of the configured report. When configuring queries for the configured report, you can use the Alfabet parameter `BASE` to refer to the object selected in the filter and define search conditions dependent on the base object. To apply the configured report to a base object class or object class stereotype, click the **Browse** button to open a dialog box and select the object class or object class stereotype to which you want to apply the configured report and click **OK**.

  Note the following:

  - This parameter is mandatory for configured reports displaying an HTML report about indicators. It is optional for all other configured reports of the type `Custom`.
  - If the configured report is applied to an object class stereotype, the filter field for selection of the base object will use the caption of the object class stereotype as caption of the filter field and the selector defined in the class settings for the object class stereotype for selection of the base object.
  - If the configured report is assigned to the object view of an object of the object class it is applied to, the configured report will automatically open with the current object selected as base object, even if **Apply to Class** is not set.
  - Alternatively, the base object of a configured report can be set with the attribute **Base Query**. While the attribute **Apply to Class** enables the user to determine the current base object by setting a filter, the **Base Query** evaluates the base object by execution of a query when opening the configured report. A filter field is not available and the user cannot change the base object.
  - The attribute **Apply to Class** also influences the availability of the configured report on the Alfabet interface. Select a class to make the configured report available in the **Configured Reports** page view on the object view of objects of the selected object class. If you do not set the attribute, the configured report is only available in the **Configured Reports** functionality. The configured report can be assigned to object views and wizards independent from the parameter.

- **Base Object Query**: When a base object query is defined for the configured report, a base object is evaluated for the configured report at runtime. When configuring the other queries for the configured report, you can use the Alfabet parameter `BASE` to refer to the object returned by the base object query and define search conditions dependent on the base object. To apply the configured report to a base class, click the **Browse** button to open a text editor, define either a native SQL query or an Alfabet query that returns a single object, and click **OK**.

Note the following:

- Alternatively, the base object of a configured report can be set with the attribute **Apply To Class**. While the setting of **Base Object Query** triggers the evaluation of the base object by execution of a query when opening the configured report, the attribute **Apply to Class** enables the user to determine the current base object by setting of an automatically generated filter.
- The query defined for base object query is executed when the user opens the configured report to evaluate the base object at runtime. Alfabet parameters referring to the current working environment can be used in the query to find a base object dependent For example, on the user opening the configured report or the user profile used for login. A base object query may also find For example, the ICT object that a specific application is currently assigned to.
- If **Base Object Query** is set and the configured report is assigned as a view to an object profile, the Alfabet parameter `BASE` referring to the `REFSTR` of the object the user is working with when opening the configured report can be used in the base object query.

- **Selector Behavior**: The attribute can be used to exclude configured reports that are defined for special purposes, like For example, triggering of data export via the RESTful API, from being selectable in the **Configured Reports** table. By default, the attribute is set to `Visible` and the configured report is visible in the selector for adding configured reports to the **Configured Reports** table if all other conditions for access to the configured report apply. If the attribute is set to `NotVisible`, the configured report is excluded from the selector for adding configured reports to the **Configured Reports** functionality. Please note however, that this setting is not excluding the configured report from any standard views or customer configured views and selectors, but exclusively from the standard selector for configured reports implemented in Alfabet.

The attribute cannot be edited in the attribute window directly. To change the setting, right-click the configured report and select **Set Report Selector Behavior to 'Not Visible'** or **Set Report Selector Behavior to 'Visible'** respectively.

- **Applicable for AlfaBot**: Specify whether the configured report is accessible via the AlfaBot. By default, the attribute will be set to `True` for new reports. Set the attribute to `False` if the configured report should not be opened outside of a specific context. For example, this may be relevant for reports that are specifically designed to be embedded in an object cockpit.

  This attribute is only relevant if the AlfaBot is implemented. For information about the implementation of the AlfaBot, see Implementing the AlfaBot for Navigation via a Full-Text Command.

- **Business Problem Statement**: Optionally enter a description that will help users to search for the configured report in the AlfaBot. The text is not displayed in the user interface, but it is of special relevance for the search mechanisms implemented for the AlfaBot. If the report caption entered by the user in the AlfaBot for a request to open a configured report does not match the caption of one of the available configured reports, the AlfaBot will split the caption entered by the user into keywords and will perform a keyword search on the **Caption**, **Description** and the **Business Problem Statement** attributes of the configured reports that are accessible via the AlfaBot. In addition to a keyword search, a synonym search is performed on the text provided in the **Business Problem Statement** attribute.

This attribute is only relevant if the AlfaBot is implemented. For information about the implementation of the AlfaBot, see Implementing the AlfaBot for Navigation via a Full-Text Command.

- **Base Classes**: Define the object classes that must be available to run the configured report. If this attribute is set, the Alfabet Server checks whether the object classes specified as base classes are excluded from the view scheme used to access the configured report. If the user does not have access permissions to one of the base classes, the configured report will be disabled. To set the attribute, click the **Browse** button to select the object classes that are required to run the configured report and click **OK**.
- **Can Create Express View**: Select `True` if an express view may be created for the report. Select `False` if an express view may not be created for the report. Please note that for reports that have a custom report view, the setting must be consistent with the setting of the attribute **Can Create Express View** of the custom report view.

  An express view is a link to a specific Alfabet view displaying data that allows individuals inside or outside of the user community to view information in Alfabet. When the express view is created, an email notification is automatically mailed to a specified recipient. The recipient receives a URL that allows him/her to access the current page view in Alfabet. For more information, see the section *Sending and Receiving Express Views* in the reference manual *Getting Started with Alfabet*.

5) In the explorer, right-click the configured report and select **Create Custom Report View**. The view editor opens. In the explorer, the custom report view is displayed as a subordinate object of the configured report. The attributes for the custom report view are displayed in the attribute window. You can optionally edit the following attributes:
   - **Name**: Optionally you can change the name for the custom report view. The **Name** must be unique for custom report views.
   - **Can Create Bookmark**: Select `True` if a bookmark may be created for the configured report. Select `False` if a bookmark may not be created for the configured report.
   - **Can Create Express View**: Select `True` if an express view may be created for the configured report. Select `False` if an express view may not be created for the configured report. Please note that the setting must be consistent with the setting of the attribute **Can Create Express View** of the configured report.

     An express view is a link to a specific Alfabet view displaying data that allows individuals inside or outside of the user community to view Alfabet information. When the express view is created, an e-mail notification is automatically mailed to the specified recipient who receives a URL that allows him/her to access the current page view in Alfabet. For more information, see *Sending and Receiving Express Views* in the reference manual *Getting Started with Alfabet*.

   The custom report view initiates the **Report Assistant**. A custom report view shall never be deleted from the configured report after the **Report Assistant** was already used to configure the report. When you delete the custom report view of an existing template-based configured report and create a new one, the **Report Assistant** is reset to the example specified by default and your configuration is lost.

6) In the explorer, right-click the configured report and select **Start Alfabet Report Assistant**. The **Report Assistant** opens.

7) Define the configured report as described below in the sections for the configuration of the different configured report types:

   - Defining Tabular Configured Reports for Multi-Editing of Object Class Properties (for configured reports based on the template `EditableClassViewReport`)
   - XXX (for configured reoprts based on the template `KanbanReport`)
   - Configuring Matrices for Multi-Editing of Object Relations or Business Supports (for configured reports based on the templates `Relationships_TableReport` and `Relationships_EditableTableReport`)
   - Configuring a Report for Multi-Editing of Indicators (for configured reports based on the template `ITMAP_TableReport`)

8) Edit the parameter panel for the configured report according to demand. For more information, see Defining Filters for Configured Reports and Selectors.

   Please note that the **Submit** button must not be deleted for configured reports based on the templates `ITMPAP_TableReport` or `Relationships_EditableTableReport` if these reports are not included into a wizard but opened as separate view on the Alfabet user interface. The **Submit** button is required to submit changes made in the dataset.

9) Click **OK** to save the configuration.

10) In the toolbar, click the **Save** button to save your changes.

11) In the explorer, right-click the configured report and select **Review Report**. The configured report opens in a web browser. If the results are not as expected, you can edit the configured report until the output of the configured report meets your expectations.

    Please note the following:

    - In the **Review Report** functionality the ability to edit objects is disabled.
    - The **Configure Report** functionality allows the visibility of toolbar buttons of the configured report to be defined for different view schemes. For more information, see Refining Visibility Issues in the View Scheme.

Defining Tabular Configured Reports for Multi-Editing of Object Class Properties

The report is configured in the report assistant opening for configured reports based on the template `EditableClassView`. For information about creating the report and opening the report assistant, see Creating a New Report for Multi-Editing of Objects.

The **Report Assistant** allows you to define both the content of the tabular dataset and the editor of the report:

1) In the **Data Source Definition** tab of the **Report Assistant**, define one of the following:

   - A native SQL query selecting all properties of a single object class. To define the query, click the **Browse** button in either the **Alfabet Query** attribute to open the **Alfabet Query Builder** or in the **Query as Text** attribute to open a text editor.

- An Alfabet query without **Show Properties**.

  To open the **Alfabet Query Builder**, click the **AQL Builder** button. To define the query, click the **Browse** button in either the **Native SQL Query** attribute to open the Alfabet specific native SQL editor with enhanced usability for the definition of Alfabet query language instructions or in the **Query as Text** attribute to open a text editor.

The definition of `WHERE` conditions and `JOIN`s (`FROM` clauses in Alfabet queries) is only allowed for the definition of filters for the report or for reducing the dataset in the report to a subset of objects of the selected object class. Object class properties of joint object classes are not included into the column definitions of the configured report. The configured report is exclusively handling one object class.

It is recommended to define the query to return only objects editable for the current user, taken the implemented access permission concepts into account.

To build a configured report view for editing of the object class properties of the class **Information Flow**, a native SQL query without filter definitions will read:

```
SELECT * FROM INFORMATIONFLOW
```

2) Go to the **Default Layout** tab. The table displays all object class properties of the selected object class as well as indicators and role types assigned to the selected object class. The property caption is displayed in the column **Property** and the property type is displayed in the column **Type**. Clicking a column header sorts the table rows according to the values in the column.

Please note:

- Aspect indicators and project indicators are not editable via the configured report and not included into the dataset.
- Computed indicators are configured to be non-editable for users. They are included into the dataset with a checkmark in the column **Computed** to mark them as computed. For these editors, the definition as **View Only** cannot be changed.

3) Define the visibility and editability of each property in the following columns by clicking into a table cell to set a checkmark or remove a set checkmark:

- **Exclude**: Set a checkmark to exclude the property from both the table in the view and the editor.

  If you want to exclude most properties, you can use the button **Exclude all** in the toolbar above the table to first set a checkmark in all columns and then only remove the checkmark in the columns you want to view. The button **Include all** removes all checkmarks in the **Exclude** column.

- **Show in Editor**: Remove the checkmark if the property shall not be included into the editor.

- **View Only**: Set a checkmark if the property shall be displayed non editable in the editor, For example, to provide additional information required for editing to the user. The setting is only taken into account if **Show in Editor** is also checked for the property.

If you would like to set identical values in the columns for all properties of a defined data type, you can use the **Group Settings** button on top of the table to perform the changes. The **Group Settings** button opens an editor. Specify the following and then click **OK** to change the visibility and editability settings according to your specification:

- **Attribute Types**: Select the checkbox of all property types the changes shall be applied to.
- **Role in Editable View**: Select the checkbox of all columns in the table the changes shall be applied to.
- **Include All** / **Exclude All**: Select whether you would like to exclude or include all of the selected attribute types in the selected columns of the table.

4) In the table, change the order of rows in the dataset to define the column order in the table of the resulting configured report and editor. The row order from top to bottom in the **Report Assistant** corresponds to the order of columns in the report from left to right, or from right to left when rendering the interface in Arabic language.

   To change the position of one or multiple rows, select the rows in the table and use the following buttons in the toolbar to change the position of the selected rows:

- **Move down**  : Click to move the selected row(s) one position down.
- **Move up**  : Click to move the selected row(s) one position up.
- **Move to top**  : Click to move the selected row(s) directly to the top of the dataset.

If you click one of the header cells of the dataset, the columns are resorted according to the values in the column that you clicked and you will loose your defined sort order. It is recommended to click the **OK** button close the **Report Assistant** after having defined the sort order of columns for presentation in the configured report and editor. Reopen the **Report Assistant** again. If you then change the sort order by chance, you can click **Cancel** to revert to the version stored before last opening of the **Report Assistant**.

5) If you want the first columns in the editor and in the result dataset to stay visible during horizontal scrolling, you can select the number of columns that shall be frozen during scrolling in the field **Number of Frozen Columns** above the table. If the first columns in the dataset include columns that are excluded from display in the editor, the number of frozen columns in the editor will be reduced by the number of excluded columns. This ensures that the same information remains visible on scrolling in the result dataset and in the editor.

The value defined here can be overwritten per user profile with the **Configure**  button available to the users working with the configured report and to solution designers opening the Alfabet user interface via the **Configure User Profile** option in the context menu of a user profile in the **Admin** tab. For more information, see Configuring User Profiles for the User Community. For more information about the handling of columns in views in the Alfabet user interface, see the section *Working with Data in Tabular Datasets* in the reference manual *Getting Started with Alfabet*.

Please note that the option to change the sort order of columns usually available via the **Configure** button is not available for this type of configured report.

6) If you want the rows displayed in the configured report to be sorted by the values in a column of the configured report, select the row in the dataset that represents the column in the configured report and click the **Add to Sort By** button in the toolbar above the dataset. The column is added to the field **Sort by** below the dataset. You can add multiple rows to the **Sort by** field either one by one or by selecting multiple rows in the datset prior to clicking the the **Add to Sort By** button.

   Sorting is performed in the order of the sort options in the **Sort by** field. The values in the column in the configured report that corresponds to the first row in the **Sort by** field is sorted alphanumerically in ascending order. The values of the column in the configured report that corresponds to the next row in the **Sort by** filed is then only sorted within the identical results for values of the already sorted column. You can change the sort order of the rows in the Sort by field with the buttons **Move down** and **Move up** in the toolbar above the **Sort by** field.

   You can remove a sort option by selecting it in the **Sort by** field and click the **Delete** button in the toolbar above the **Sort by** field.

   The user can change the sorting at runtime by clicking in the header of the column he/she want to sort by.

7) Go to the **Options** tab. An explorer is displayed with an attribute window on the right that lists the attributes of the node selected in the explorer. The explorer displays all relevant properties and role types available for the object class to be edited. Object class properties are excluded from display if they are marked as **Exclude** or **View Only** in the **Default Layout** tab. Only object class properties of the data type `String` or `Reference` are displayed. Object class properties of other data types and indicator types are not available in the explorer, because no attributes can be set for them.

8) In the attribute window of the root node, define the following for the processing of changes to the meta-model and to the reference data configuration:

   - **Add New Class Properties**: If set to `True`, a new object class property created for the object class in Alfabet Expand is automatically added to the bottom of the dataset in the tab **Default Layout** of the configured report. If set to `False`, the new role type must be added explicitly to the dataset in the configured report by clicking the button **Refresh Available Properties** in the **Default Layout** tab. For both methods, the object class property added to the dataset will be configured to be both visible and editable in the configured report. The default is `False`.
   - **Automatically Add New Indicator Types**: If set to `True`, and a new evaluation type is assigned to the object class in the **Class Configuration** functionality, the indicator(s) of the evaluation type are automatically added to the bottom of the dataset in the tab **Default Layout** of the configured report. If set to `False`, the new indicator(s) must be added explicitly to the dataset in the configured report by clicking the button **Refresh Available Properties** in the **Default Layout** tab. For both methods, the indicator added to the dataset will be configured to be both visible and editable in the configured report. The default is `True`.
   - **Automatically Add New Role Types**: If set to `True`, a new role type is assigned to the object class in the **Class Configuration** functionality is automatically added to the bottom of the dataset in the tab **Default Layout** of the configured report. If set to `False`, the new role type

must be added explicitly to the dataset in the configured report by clicking the button **Refresh Available Properties** in the **Default Layout** tab. For both methods, the role type added to the dataset will be configured to be both visible and editable in the configured report. The default is `True`.

9) In the attribute window of the root node, define the following for the behaviour of the edit capabilities in the configured report:

- **Validate Edit Permissions**: If set to `True`, access permissions of the user are evaluated when the user opens the editor. If objects selected by the user are not editable for the user because of any of the implemented access permission concepts, the objects are not displayed in the editor. If none of the objects is editable, an empty editor might open. Therefore, it is recommended to define the query of the configured report to take access permission concepts of the user into account and display only editable objects in the dataset. If **Validate edit permissions** is set to `False`, the user can edit any object in the editor even if the access permission concepts implemented would not allow him/her to edit the object. The default is `True`.

- **Enforce Property Editability Rules**: If set to `True`, the editability of the properties in the editor will depend on the editability of the property in the editor or wizard for the base object class used in the class settings for the object profile:

  - If the **Read-Only** attribute of the editor interface control is set ot `True`, the property will not be editable for all objects, even if explicitly defined as editable in the configured report.

  - If a condition defined in the **Read-Only Condition** attribute of the editor inhibits editing for a specific object, the property will not be editable for the specific object. The default is `False`.

  - If a validator is defined with the attribute Validator of an object class property, the validation is also performed in the editor of the configured report.

  - If the editor for the base object class checks whether start dates are set to a date prior to end dates, the validation of start and end dates is also performed in the editor of the configured report.

  Please note the following:

  - Evaluation of conditions requires a lot of computing capacity. Therefore, the number of objects that are simultaneously editable should be set to a small number with the attribute **Max Number of Records** if **Enforce Property Editability Rules** is set to `True`.

  - The restrictions based on editability in editors and wizards are not applied to editing in the **Set All** functionality.

  - Wizard Step conditions are not evaluated for the editor in the configured report.

- **Max Number of Records**: Defines the maximum number of objects that can be edited simultaniously in the editor. If the user selects more objects for editing, some of the selected objects are not displayed for editing and the user will be informed about that via a message on top of the editor.

- **Restrict Properties to Editor/Wizard Definition**: If set to `True`, properties not ediable to the user via an editor or wizard will also be automatically set to view only in the editor of the configured report. This is independent of the editability setting in the **Default Layout** tab.
- **Enable 'Set All' for Class Properties**: If set to `True`, the user can select multiple objects in the table of the configured report, select **Properties** from the drop-down list of the button **Set all** and open an editor to set identical values for properties of all selected objects. The user first selects the properties to be reset. Then he/she assignes one value to each selected property and these values are then set for all selected objects when the user closes the editor. If set to `False`, the option **Properties** is not visible in the dropdown list of the **Set all** button.
- **Enable 'Set All' for Indicators**: If set to `True`, the user can select multiple objects in the table of the configured report, select **Indicators** from the drop-down list of the button **Set all** and open an editor to set identical values for indicators of all selected objects. The user first selects the indicators to be reset. Then he/she assignes one value to each selected indicator and these values are then set for all selected objects when the user closes the editor. If set to `False`, the option **Indicators** is not visible in the dropdown list of the **Set all** button.
- **Enable 'Set All' for Role Types**: If set to `True`, the user can select multiple objects in the table of the configured report, select **Role types** from the drop-down list of the button **Set all** and open an editor to set identical values for roles of all selected objects. The user first selects the role types to be reset. Then he/she assignes one value to each selected role type and these values are then set for all selected objects when the user closes the editor. If set to `False`, the option **Role types** is not visible in the dropdown list of the **Set all** button.

10) If your data includes properties that are returning a color, click the object class property returning the color in the explorer and set the attribute **Is Color** to `True`. The table of the configured report will then show the colors defined with the attributes as background color of the respective cells for the property and a color picker will be available in the editor to change the color. If the attribute is set to `False`, the hexadecimal color code is displayed in the table of the configured report and the user must enter the hexadecimal color code for the color into a text field in the editor.

11) Optionally define how the selection of objects for properties of the type `Reference` or the selection of objects of the class `Person` for role types shall be handled in the editor. Click on the node of the object class property or role type in the explorer and configure the attributes according to the descriptions below.

    By default, the editor in the configured report will display a drop-down list with all available objects of the object class the property allows to add. For example, the dropdown list for the property Application Group of the object class Application will list all application groups in the database. For performance reasons, the number of objects displayed in the combo box is limited to 300. This might cause objects in your database to be unavailable for selection.

    There are three methods to change the content of the dropdown list to a limited number of available objects:

    - The field can be configured to display a text box with a autocomplete functionality and, optionally, a selector button. If the user starts typing into the field, a dropdown list with objects is displayed with the name starting with the already typed in text. The user can stop typing after the dropdown list is conveniently small and select the object from the remaining values in the dropdown list. Set the following attributes:
        - **Property enumeration type**: Select `LookAhead` from the drop-down list.

- **Selector Name**: Optionally, select a custom selector from the drop-down list. Selection of a custom selector provides the following functionality:
  - The user can open the selector via a button displayed at the end of the field and select the object from the selector.
  - The autocomplete functionality searches for matches only within the subset of objects that match the query defined in the **Lookahead** attribute of the selector.

  A custom selector must be available for the object class to attach a selector to the autocomplete field. Standard selectors are not considered. For information about creating a custom selector, including the definition of the query for the auto-fill function in selectors, see the section Configuring a Custom Selector for Search Functionalities.

- The field can be configuring to display a dropdown list filled only with objects found by a configured report that defines a subset of the object that are available in the database. For example, the users displayed in a dropdown list for defining a role can be limited to the users that are in the same user group than the authorized user of the currently edited object. The configured report can reference the objects currently selected by the user for editing or which filters have been set for the configured report for multi-editing of objects when opening the editor. After having defined the configured report for the dropdown list content as described below in the section Configuring a Report to Restrict Drop-Down List Content Via a Query, set the following attributes:
  - **Property enumeration type**: Select `Simple` from the dropdown-list.
  - **Enumeration provider name**: Select the name of the configured report providing information about the objects to be found.
  - **Parameters mapping**: f your secondary configured report contains filters, click the Browse button of the **Parameters mapping** attribute to open an editor that allows mapping of the parameters in the secondary report with the filter fields or selected objects of the configured report for multi-editing of objects.

The column **Query Parameter Name** lists all parameters defined in the query of the secondary report.

In the column **Control Name**, select the filter field defined for the configured report for multi-editing of properties that the parameter refers to. Please note that filter fields are only listed here if they are already defined in the report's filter panel and not only as parameter in the query of the configured report for multi-editing of properties. If you select:`SELECTION`, the value of the `REFSTR` of the objects that are currently displayed for editing in the editor of the report are returned to fill the parameter.

- The field can be configured to display a dropdown list filled with values that depend on the setting of another property of the type `Reference` of the object class edited in the configured report for multi-editing of objects. The information about which objects are permissable to select in the dropdown list for which setting of the master property is read from a linked, secondary report of the type `Query` or `NativeSQL`. After having defined the configured report for the dropdown list content as described below in the section Configuring a Report to Restrict Drop-Down List Content Depending on A Master Control, set the following attributes:
  - **Property enumeration type**: Select `Dependent` from the dropdown-list.

- **Enumeration provider name**: Select the name of the configured report providing information about the objects to be found.
- **Master Field**: Select the master property from the dropdown list of the attribute. The list also includes properties that are excluded from display and editability of the configured report for multi-editing of properties. The dependency is also evaluated for non-visible properties.
- **Parameters mapping**: f your secondary configured report contains filters, click the Browse button of the **Parameters mapping** attribute to open an editor that allows mapping of the parameters in the secondary report with the filter fields or selected objects of the configured report for multi-editing of objects.

The column **Query Parameter Name** lists all parameters defined in the query of the secondary report.

In the column **Control Name**, select the filter field defined for the configured report for multi-editing of properties that the parameter refers to. Please note that filter fields are only listed here if they are already defined in the report's filter panel and not only as parameter in the query of the configured report for multi-editing of properties. If you select:`SELECTION`, the value of the `REFSTR` of the objects that are currently displayed for editing in the editor of the report are returned to fill the parameter.

12) Click **OK** to close the report assistant and locally save your changes.

13) Click the **Save** button in the toolbar of Alfabet Expand to save your changes to the Alfabet database.

Configuring a Report to Restrict Drop-Down List Content Via a Query

The drop-down lists that are displayed for properties of the type `Reference` and for role types in the editor of a configured report for multi-editing of objects by default display all objects available for the selectable object class. for roles, this is the object class `Person`, for references, this is the target object class of the reference.

It is possible to restrict the number of objects that can be selected in the dropdown list. the selection is defined in a separate configured report of the type `Query` or `NativeSQL` that is then linked to the respective dropdown list in the **Options** tab of the configured report for multi-editing of object data.

The subordinate report must return the following information in the given order:

- The `FIND` class of the Alfabet query or the first `SELECT` argument of the native SQL query must return the `REFSTR` of the object class for that objects shall be listed in the dropdown list.
- The first column in the resulting visible dataset must return the `REFSTR` of an object edited by the user. This column can alternatively return `NULL` if the selection in the combobox shall not depend on the object that is currently edited.
- The second column in the resulting visible dataset must return the property of the object permitted as a reference that will be displayed in the dropdown list to identify the object. Make sure that the property is set for all objects. If it is not set, an empty row will be displayed for the object in the dropdown list.

Additional columns in the report will be ignored.

Note the following about the definition of the secondary report:

- For each drop-down list in the editor of the configured report for multi-editing of object data, results in the secondary report to fill the dropdown list are displayed if the first column in the report returns the REFSTR of the object that is edited via the dropdown list or it returns NULL.
- Results that return NULL in the first column of the report are displayed in the dropdown lists of all edited objects.
- If none of the rows of the secondary report returns either NULL or the REFSTR of the current object, the drop-down list is empty.
- The drop-down list is filled with the results as returned in the secondary report. It is recommended to sort the result dataset according to demands.
- If an object is listed twice in the result dataset of the secondary report, it will also be listed twice in the result dataset.

The report can include parameters in `WHERE` conditions. The parameter values can then be mapped to filter values set in the report for multi-editing of objects and to the current selection of objects in the editor. The secondary configured report must not have an own filter panel defined.

Please note the following:

- The current selection of objects to be edited returns a comma separated list of strings.

  This requires the following synthax of a `WHERE` condition refering to the current selection via a parameter:

  - In native SQL:

    ```
    WHERE CLASSNAME.REFSTR IN (@CurrentSelection)
    ```

  - In Alfabet query language:

    ```
    WHERE ClassName.REFSTR IN (:CurrentSelection)
    ```

- The parameter that refers to the currently selected objects must be mapped to the keyword:`SELECTION` in the configuration of the configured report for multi-editing of objects.
- The parameters in the secondary report must not exceed the number of filter fields plus one (for the selection) defined for the configured report for multi-editing of object properties.
- The names used for the parameters do not have to match the names of the filter fields in the report for multi-editing of objects. The mapping is done in the configuration of the configured report for multi-editing of objects.
- Alfabet query language parameters refering to the curent environment, like `CURRENT_USER`, can be used in the secondary report. The parameter `BASE` can be used in the secondary configured report to refer to the base object selection of the configured report for multi-editing of objects. this is a standard parameter and does not require any mapping in the configuration of the configured report for multi-editing of objects. The secondary report itself must not be applied to a class.

After having defined the secondary configured report and having set the **Report State** of the report to **Active**, the configured report for multi-editing of objects must be defined to use the report for object selection of the target drop-down list:

1) In the report assistant of the configured report for multi-edition of objects, go to the **Options** tab.

2) In the explorer, click the property of the type `Reference` or the role for that you would like to define the dropdown-list content.

3) Set the following attributes:

- **Property enumeration type**: Select `Simple` from the dropdown-list.
- **Enumeration provider name**: Select the name of the configured report providing information about the objects to be found.
- **Parameters mapping**: If your secondary configured report contains filters, click the Browse button of the **Parameters mapping** attribute to open an editor that allows mapping of the parameters in the secondary report with the filter fields or selected objects of the configured report for multi-editing of objects.

The column **Query Parameter Name** lists all parameters defined in the query of the secondary report.

In the column **Control Name**, select the filter field defined for the configured report for multi-editing of properties that the parameter refers to. Please note that filter fields are only listed here if they are already defined in the report's filter panel and not only as parameter in the query of the configured report for multi-editing of properties. If you select:`SELECTION`, the value of the `REFSTR` of the objects that are currently displayed for editing in the editor of the report are returned to fill the parameter.

4) Click **OK** to close the report assistant and locally save your changes.

5) Click the **Save** button in the toolbar of Alfabet Expand to save your changes to the Alfabet database.

Configuring a Report to Restrict Drop-Down List Content Depending on A Master Control

The drop-down lists that are displayed for properties of the type `Reference` and for role types in the editor of a configured report for multi-editing of objects by default display all objects available for the selectable object class. for roles, this is the object class `Person`, for references, this is the target object class of the reference.

It is possible to restrict the number of objects that can be selected in the dropdown list. the selection is defined in a separate configured report of the type `Query` or `NativeSQL` that is then linked to the respective dropdown list in the **Options** tab of the configured report for multi-editing of object data.

The subordinate report must return the following information in the given order:

- The first column in the resulting visible dataset must return the `REFSTR` of the object class for that objects shall be listed in the dropdown list.
- The second column in the resulting visible dataset must return the `REFSTR` of an object edited by the user. This column can alternatively return `NULL` if the selection in the combobox shall not depend on the object that is currently edited.

Please note:

- For each drop-down list in the editor of the configured report for multi-editing of object data, results in the secondary report to fill the dropdown list are displayed if the first column in the report returns the REFSTR of the object that is edited via

the dropdown list or it returns NULL and the second column in the report returns the REFSTR of the object selected in the master field.

- Results that return NULL in the first column of the report are displayed in the dropdown lists of all edited objects.
- If none of the rows of the secondary report returns either NULL or the REFSTR of the current object, the drop-down list is empty.

- The third column in the resulting visible dataset must return the REFSTR of an object that may be or may have been selected in the master field.

  Please note:

  - The master property does not have to be editable or visible in the dataset of the configured report for multi-editing of objects. Any property of the type Reference can be selected as master property.
  - The secondary report is only executed once on opening of the editor. If the master property is editable, the second column of the secondary report shall return not only the REFSTR of objects currently selected in the of the master field, but also take the REFSTR of all objects into account that might be selected by the user editing the master field. If the first column of the secondary report returns the REFSTR of a currently edited object, the combination of the currently edited object with all objects selectable in the master fields must be available in the configured report. It is recommended to set the first column to NULL in case of editable master fields to keep the number of rows in the secondary configured report low.

- The fourth column in the resulting visible dataset must return the property of the object permitted as a reference that will be displayed in the dropdown list to identify the object. Make sure that the property is set for all objects. If it is not set, an empty row will be displayed for the object in the drop-down list.

  Please note:

  - The drop-down list is filled with the results as returned in the secondary report. It is recommended to sort the result dataset according to demands.
  - If an object is listed twice in the result dataset of the secondary report, it will also be listed twice in the result dataset.

Additional columns in the report will be ignored.

The report can include parameters in **WHERE** conditions. The parameter values can then be mapped to filter values set in the report for multi-editing of objects and to the current selection of objects in the editor. The secondary configured report must not have a own filter panel defined.

Please note the following:

- The current selection of objects to be edited returns a comma separated list of strings.

  This requires the following synthax of a `WHERE` condition refering to the current selection via a parameter:

- In native SQL:

  ```
  WHERE CLASSNAME.REFSTR IN (@CurrentSelection)
  ```

- In Alfabet query language:

  ```
  WHERE ClassName.REFSTR IN (:CurrentSelection)
  ```

- The parameter that refers to the currently selected objects must be mapped to the keyword:`SELECTION` in the configuration of the configured report for multi-editing of objects.
- The parameters in the secondary report must not exceed the number of filter fields plus one (for the selection) defined for the configured report for multi-editing of object properties.
- The names used for the parameters do not have to match the names of the filter fields in the report for multi-editing of objects. The mapping is done in the configuration of the configured report for multi-editing of objects.
- Alfabet query language parameters refering to the current environment, like `CURRENT_USER`, can be used in the secondary report. The parameter `BASE` can be used in the secondary configured report to refer to the base object selection of the configured report for multi-editing of objects. this is a standard parameter and does not require any mapping in the configuration of the configured report for multi-editing of objects. The secondary report itself must not be applied to a class.

After having defined the secondary configured report and having set the **Report State** of the report to **Active**, the configured report for multi-editing of objects must be defined to use the report for object selection of the target drop-down list:

1) In the report assistant of the configured report for multi-editing of objects, go to the **Options** tab.
2) In the explorer, click the property of the type `Reference` or the role for that you would like to define the dropdown-list content.
3) Set the following attributes:
   - **Property enumeration type**: Select `Dependent` from the dropdown-list.
   - **Enumeration provider name**: Select the name of the configured report providing information about the objects to be found.
   - **Master Field**: Select the master property from the dropdown list of the attribute. The list also includes properties that are excluded from display and editability of the configured report for multi-editing of properties. The dependency is also evaluated for non-visible properties.
   - **Parameters mapping**: f your secondary configured report contains filters, click the Browse button of the **Parameters mapping** attribute to open an editor that allows mapping of the parameters in the secondary report with the filter fields or selected objects of the configured report for multi-editing of objects.

The column **Query Parameter Name** lists all parameters defined in the query of the secondary report.

In the column **Control Name**, select the filter field defined for the configured report for multi-editing of properties that the parameter refers to. Please note that filter fields are only listed here if they are already defined in the report's filter panel and not only as parameter in the query of the configured report for multi-editing of properties. If you select:`SELECTION`, the value of the `REFSTR` of the objects that are currently displayed for editing in the editor of the report are returned to fill the parameter.

4) Click **OK** to close the report assistant and locally save your changes.

5) Click the **Save**  button in the toolbar of Alfabet Expand to save your changes to the Alfabet database.

Considering Access Permissions in Configured Reports for Multi-Editing of Object Class Properties

Various settings in the configured report as well as in the Alfabet meta-model are available to ensure that the edit permissions of a user for objects and for individual properties are taken into account:

- Edit Permissions for Objects
- Edit Permissions for Object Class Properties

Edit Permissions for Objects

Edit permissions of the user are only taken into account if the configured report is configured to evaluate edit permissions.

In the dataset, this is done via the query that may For example, restrict display to objects for that the current user is the authorized user. If the query of the configured report does not take editability of the object for the current user into account, the table will also display objects that are not editable.

In the editor, access permissions are taken into account by default. If the user selects objects in the dataset and opens the editor, only the subset of the selected objects that are editable by the user are displayed in the editor. If the user selects only objects that are not editable for him/her, an empty editor opens. If no objects are selected, the **Edit** button is deactivated.

In the **Options** tab of the report assistant of the configured report, the validation of edit permissions for the current user can be deactivated by setting the attribute **Validate Edit Permissions** to `True`. The user can then edit objects that he/she has no edit permissions for according to the implemented access permission concepts in Alfabet.

Edit Permissions for Object Class Properties

By default, all object class properties, indicators and roles of a selected object class are visible and editable in the configured report for multi-editing of object class properties. The following methods are available to exclude individual object class properties from beeing editable:

- In the **Default Layout** tab of the report assistant, a checkmark can be set in the column Exclude to exclude the property from the report's dataset and the editor and the **Set All** functionalities.
- In the **Default Layout** tab of the report assistant, the checkmark in the column **Show in Editor** can be deselected to exclude the property from the editor and the **Set All** functionalities without excluding it from the dataset.
- In the **Default Layout** tab of the report assistant, the checkmark **View Only** can be selected in combination with the **Show in Editor** checkmark to include the property view only into the editor. The property will not be available in the the **Set All** functionalities.

- By default, object class properties that are defined in the **Default Layout** tab of the report assistant to be editable are editable in the configured report even if the user has no edit permissions for the object class property via an editor or wizard. This behavior can be changed by setting one of the following attributes of the root node of the **Options** tab of the report assistant. the options:
  - **Enforce Property Editability Rules**: If set to `True`, the editability of the properties in the editor will depend on the editability of the property in the editor or wizard for the base object class used in the class settings for the object profile. If the **Read-Only** attribute of the editor interface control is set ot `True`, the property will not be editable for all objects, even if explicitly defined as editable in the configured report. If a condition defined in the **Read-Only Condition** attribute of the editor inhibits editing for a specific object, the property will not be editable for the specific object. The default is `False`.

    Please note the following:
    - Evaluation of conditions requires a lot of computing capacity. Therefore, the number of objects that are simultaneously editable should be set to a small number with the attribute **Max Number of Records** if **Enforce Property Editability Rules** is set to `True`.
    - The restrictions based editability in editors and wizards are not applied to editing in the **Set All** functionality.
  - **Restrict Properties to Editor/Wizard Definition**: If set to `True`, properties not editable to the user via an editor or wizard will also be automatically set to view only in the editor of the configured report. This is independent of the editability setting in the **Default Layout** tab. The restrictions based editability in editors and wizards are not applied to editing in the **Set All** functionality.
- If a custom object class property shall not be editable in any configured report for multi-editing of object class properties, even if the user can edit the property in an editor and wizard, the attribute object class property **Disable Mass Update** of the object class property can be set to `True`. The column **View Only** cannot be deselected for the property in any report assistant for configured reports for mass update of object class properties. The property will not be editable in the editor an not be available in the **Set All** functionalities. The **Disable Mass Update** attribute can only be set for custom properties. For private and protected standard Alfabet object class properties, this attribute cannot be set and is `False` by default.

Configuring Kanban Reports for Maintenance of Relationships between Objects

The report is configured in the report assistant opening for configured reports based on the template `KanbanReport`. For information about creating the report and opening the report assistant, see Creating a New Report for Multi-Editing of Objects.

The content and layout of a Kanban report is defined by adding elements to the explorer of the **Report Assistant** and setting the attributes in the attribute pane of the **Report Assistant** window.

The following table lists the explorer node elements that can be configured to specify the content of the report as well as provides information about the purpose of each sub-element:

Explorer Node Element	Required to Configure:
Root Node	Definition of the functionality and layout of the report.
Cells	Definition of the cell content. The element is a container for one or multiple elements ***Query*** that define the objects displayed in the cells of the report. Only one object class can be used for display in cells. If you define multiple queries, they must all return objects of the same class.
Columns	Definition of the column header. The element is a container for one or multiple elements ***Query*** that define the objects displayed in the column header. Only one object class can be used for display in column headers. If you define multiple queries, they must all return objects of the same class.
Rows	Definition of the row header. The element is a container for one or multiple elements ***Query*** that define the objects displayed in the row header. Only one object class can be used for display in row headers. If you define multiple queries, they must all return objects of the same class.

The following sections guide you through the definition of various features of the Kanban report. The definition in the sections XXX and XXX are mandatory to define a Kanban report. All other features are optional.

- Defining the Content and Structure of the Kanban Report
- Defining the Toolbar Buttons Available for the Kanban Report
- Defining the Layout of the Kanban Report
- Defining Tooltips for the Kanban Report
- Displaying Evaluation Aspects in Kanban Reports
- Defining Connections between Objects in the Cells of a Kanban Report
- Defining Navigation from the Kanban to Other Views

Defining the Content and Structure of the Kanban Report

Kanban reports can be defined either to display a tabular layout or a matrix layout.

It is mandatory for the definition of a Kanban report to define at least the column headers of the Kanban and the objects to be displayed in the cells of the Kanban report. The definition of the row headers, generating a matrix layout, is optional.

The cells of the Kanban report can contain objects of a single object class only.

The columns and rows in a Kanban report can represent either of the following:

- Values of an object class property of the objects in the cells.
- Objects referenced by the objects in the cells via an object class property of the type `Reference` or `ReferenceArray` of the objects in the cells.
- Objects of an object class that the objects in the cell are assigned to via an intermediate object class. The following use cases are supported:
  - Alfabet users can be assigned a role for an object. The relation between object and user is established via the intermediate object class Role.
  - The relation between business data and an object that reads, creates, updates, processes, or deletes it is established via the intermediate object class BusinessDataUsage.
  - Objects are related to organizations, market products, business processes and domains via the business support they are providing. This relation can be established via the following intermediate object classes: BusinessSupport, TacticalBusinessSupport, StrategicBusinessSupport, and BusinessAppraisal.
  - Objects can be assigned as relevant architecture to a project, demand, policy, risk mitigation, or value node. These relations are established via the intermediate object classes ProjectArch, DemandArch, PolicyArch, AffectedArch, and ValueNodeArch.

The cell content of the Kanban report is defined via a query. The column and row headers of the Kanban report are defined via a query with the exception of Kanban reports that manage the values of the release status or an enumeration based object class property of the objects in the cells. A mechanism is implemented that fills the header fields without a query definition for the release status and enumeration based object class properties.

Do the following to define the content of the Kanban report:

1) In the explorer of the **Report Assistant**, right-click the ***Cells*** element and select **Add New Query** from the drop-down list.
2) Click the ***Query*** node and define the following attributes:

Attribute	Description
Name	Define the name of the ***Query*** element displayed in the explorer of the **Report Assistant**.
Alfabet Query / **Query as Text** / **Native Sql**	Define a query to find the objects that are displayed in the report, the indicator values that are used to calculate the location of the object shapes in the portfolio, and all optional dynamic design elements of the portfolio.

The query must provide the following information:

- In Alfabet Queries, the `FIND` statement of the Alfabet Query must find objects of the object class to be displayed in the cells of the configured report. In native SQL queries, the first argument of the `SELECT` statement must return the `REFSTR` of the object to be displayed in the cells.

Please note that for objects that are related to the objects in the column header or row header via an intermediate object class, objects of the intermediate object class must be displayed in the cells of the Kanban

report. For example, if you want to define a Kanban report that displays the users that are assigned to an object via a role, the `Role` object class must be selected as object class for the cell.

- A column of the dataset from the query must return the text that shall be displayed in the object boxes placed into the cells of the Kanban.
- A column in the dataset must define the position of the object with respect to the columns of the Kanban. If the column headers represent objects, the `REFSTR` of the object in the column header must be returned. If the column headers represent property values, the property value must be returned.
- If the Kanban shall have row headers, a column in the dataset must define the position of the object with respect to the rows of the Kanban. If the row headers represent objects, the REFSTR of the object in the row header must be returned. If the row headers represent property values, the property value must be returned.

The following information can be returned by the query to define optional features for the portfolio report:

Feature to be implemented	For more information see:
Displaying a tooltip when the customer moves the cursor over the object in a cell.	
Defining the height of the object boxes dynamically	
Coloring of object box background and border dynamically	
Defining a navigation target for navigation from objects in the portfolio	

3) If the column headers shall represent objects in the Alfabet database, right-click the ***Columns*** element in the explorer and select **Add New Query** from the drop-down list.

4) Click the ***Query*** node and define the following attributes:

Attribute	Description
Name	Define the name of the ***Query*** element displayed in the explorer of the **Report Assistant**.
Alfabet Query / **Query as Text** / **Native Sql**	Define a query to find the objects that are displayed in the column headers.

If the headers represent objects referenced by the objects in the cells, the query must provide the following information:

- In Alfabet Queries, the `FIND` statement of the Alfabet Query must find objects of the object class to be displayed in the column headers. In native SQL queries, the first argument of the `SELECT` statement must return the `REFSTR` of the objects to be displayed in the column headers.
- All show properties defined in the Alfaber Query or all other `SELECT` statement arguments defined in the native SQL query are displayed in the column headers as text.

5) If the Kanban report shall have row headers that represent objects in the Alfabet database, right-click the ***Rows*** element in the explorer and select **Add New Query** from the drop-down list.

6) Click the ***Query*** node and define the following attributes:

Attribute	Description
Name	Define the name of the ***Query*** element displayed in the explorer of the **Report Assistant**.
Alfabet Query / **Query as Text** / **Native Sql**	Define a query to find the objects that are displayed in the row headers.

The query must provide the following information:

- In Alfabet Queries, the `FIND` statement of the Alfabet Query must find objects of the object class to be displayed in the row headers. In native SQL queries, the first argument of the `SELECT` statement must return the `REFSTR` of the objects to be displayed in the row headers.
- All show properties defined in the Alfaber Query or all other `SELECT` statement arguments defined in the native SQL query are displayed in the row headers as text.

7) In the root node of the **Report Assistant**, define the following attributes in the given order:

Attribute	Description
Enable Object Creation	This attribute must be set to `True` if the Kanban shall be used to display and manage a relationship via an intermediate object class. The attribute must be set to `False` for all other types of Kanban content.
Item Class	Select the object class that is displayed in the cells of the Kanban report. This attribute is mandatory.
Column Property	Select the object class property of the object class in the cells of the Kanban report that establishes the connection to the objects in the column headers or that returns the release status or enumeration-based string value in the column header. This attribute is mandatory.
Row Property	Select the object class property of the object class in the cells of the Kanban report that establishes the connection to the objects in the column headers

Attribute	Description
	or that returns the release status or enumeration based string value in the column header. This setting is ignored if **Row Reference Column** is not set.
Row Header Source	Select `Query` if the row headers are defined via a query. Select `Property` if the row headers represent the release status or an enumeration based string property of the object class in the cells of the Kanban report. This setting is ignored if **Row Reference Column** is not set.
Column Header Source	Select `Query` if the column headers are defined via a query. Select `Property` if thecolumn headers represent the release status or an enumeration based string property of the object class in the cells of the Kanban report. This attribute is mandatory.
Image Column	Enter the name of the dataset column in the query defined in the ***Cells*** > ***Query*** element that returns the string to be displayed in the object boxes in the cells of the Kanban report. This attribute is mandatory.
Column Reference Column	Enter the name of the dataset column in the query defined in the ***Cells*** > ***Query*** element that returns the `REFSTR` of objects represented by the column headers or the values of the release status or enumeration based string property represented by the column headers. This attribute is mandatory.
Row Reference Column	Enter the name of the dataset column in the query defined in the ***Cells*** > ***Query*** element that returns the `REFSTR` of objects represented by the row headers or the values of the release status or enumeration based string property represented by the row headers.
Header Value Column	
Lexicographic Sorting of Column Header	
Lexicographic Sorting of Row Header	

Defining the Toolbar Buttons Available for the Kanban Report

Kanban reports can be configured to permit users to change the relation of the object displayed in the reports with respect to the information in the column and row headers. If the column and row headers

represent objects, the reference of the object in the cell to the objects in the column and row headers is editable via the Kanban report. If the column and row headers represent the value of an object class property of the objects in the cells of the Kanban report, the value for the object class properties can be changed for the object.

When a user changes the position of an object in the Kanban report, this leads to a change of the data stored about the object in the Alfabet database. You should Therefore, consider carefully which of the available editing options should be activated for the Kanban report.

To activate editing of objects via the Kanban report, set the following attributes in the root node of the **Report Assistant**:

Attribute	Description
Can Create Item	Set this attribute to `True` to allow users to create new objects of the object class displayed in the Kanban cells. This action will create a new object in the Alfabet database. Please note that objects shall only be created via a Kanban if they can be created without a content. If the creation of new objects requires setting of references to other object classes that are usually evaluated from the context the object is created in, the respective drop-down lists for selection of the correct references are empty. The **Can Create Item** attribute should be set to `False` for these object classes. For example,: Creating a feature within a Kanban report that displays features defined for all applications in an application group will fail. The object class property `Object` of the object class `Feature` that is storing the reference to the application the feature belongs to is not set via the **Feature** editor but filled automatically during feature creation with the reference to the application the user is currently working with. In a Kanban, the user is not working in the context of an application and Therefore, the required object class property setting will fail. This only applies to the creation of new objects. Existing references can be changed and features can be moved from one application to the other within a Kanban report.
Can Add Item	Set this attribute to `True` to allow users to add existing objects of the object class displayed in the Kanban cells. This action will change the data of the object managed in the Kanban in the Alfabet database.
Can Delete Item	Set this attribute to `True` to allow users to delete existing objects displayed in the Kanban cells. This action will delete the object irrevocably from the Alfabet database.
Can Detach Item	Set this attribute to `True` to allow users to detach an object from the Kanban report. The object is no longer displayed in the Kanban but still exists in the Alfabet database. This action will change the data of the object managed in the Kanban in the Alfabet database.
Can Move Item	Set this attribute to `True` to allow users to move an object to another cell of the Kanban report. This action will change the data of the object managed in the Kanban in the Alfabet database.

Defining the Layout of the Kanban Report

The color

Defining Tooltips for the Kanban Report

Displaying Evaluation Aspects in Kanban Reports

Defining Connections between Objects in the Cells of a Kanban Report

Defining Navigation from the Kanban to Other Views

Configuring Matrices for Multi-Editing of Object Relations or Business Supports

The **Report Assistant** allows you to define the queries for the specification of the row and column headers.

The **Report Assistant** has four tabs:

- **Relationship Table** / **IT MAP Table**: The first tab displays an example of a configured report. The tabs for row and column definition are populated with code for the queries required to build the example.
- **Rows Definition**: The **Rows Definition** tab allows to specify the query to define the objects displayed in the rows of the configured report with either an Alfabet query or a native SQL query:
  - **To define an Alfabet query**: Write the complete Alfabet query, including instructions, if required, into the **Query for Rows** field or click the **Browse** button to open the **Alfabet Query Builder** to specify your Alfabet query.
  - **To define a native SQL query**: Write the native SQL query into the **Query for Rows** field. If you need to define Alfabet instructions in combination with the native SQL query, For example, to create expandable rows or columns, define the instructions separately in the **Query Instructions (for native SQL only)** field.

More information about the specification of rows is given in the section General Rules for the Specification of Rows and Columns.

- **Columns Definition**: The **Columns Definition** tab allows to specify the query to define the objects displayed in the columns of the configured report with either an Alfabet query or a native SQL query as described above for the definition of rows.

By default, column headers are as long as the longest content of a field. If the column headers defined with the query are very long, this leads to column headers requiring great part of the screen space. You can restrict the size of vertically displayed the column header to avoid oversized column headers. If the content of the column header cell is longer than the allowed size, it is cropped. To restrict the length of a column,

define the maximum number of characters for the column header in the field **Max. Length of Vertical Column Names**. Enter -1 for unrestricted display of text.

More information about the specification of rows is given in the section General Rules for the Specification of Rows and Columns.

- **Relationship Definition/Business Support**: The last tab allows you to specify the content of the matrix cells. You can define which information in the rows and which information of the columns is related to each other and how the relationship is defined in Alfabet. Detailed information about the definition of relationships is given in the separate sections Defining Matrices for Multi-Editing of Object Relationships and Defining Matrices for Multi-Editing of Business Supports.

General Rules for the Specification of Rows and Columns

The displayed row as well as the column headers are both the output result of a query based tabular report. All options that are available for a single configured report are also available for the definition of the matrix rows or columns:

- The text in the row/column headers is defined by the Show properties of the Alfabet query or the `SELECT` properties of the native SQL query. Each Show/`SELECT` property defined in the query leads to a new field in the row/column headers. You can use the `JOINCOLUMNS` Alfabet instruction to display multiple properties in one field, For example, to combine information about Name and Version of an Application in one field.

  The text for column and row headers should be short. The header fields have a size limit to ensure readability of the configured report. Header texts that are too long are truncated.

- The size of the row/column header fields depends on the content of the cells in the row/column. If you want to restrict the size of the header cells For example, because of long object names, you must define the configured report using native SQL and use the truncate functionality available for native SQL or display any other property, like For example, the short name of an object, to identify the object.
- The rows and columns can be defined as an expandable report table. In the rows, the configured report than looks like a usual expandable report table and rows can be displayed or hidden using the + sign in front of a row.

In the column headers, grouping of report results lead to grouping of the column sub-headers in sections that correspond to the objects of the superordinate class in the `GROUPBY_EX` Alfabet instruction grouping the configured report. It is not possible to expand or collapse sections, but sections and the sub-headers thereof can be hidden by right-clicking a section header. If a section is hidden, the right-click menu of any of the other sections allows to unhide that section header.

- Filters can be defined for the row and column specifications. Above the matrix table, all filter fields for both row and column specification are displayed in a combined filter. Filters that are implied by the use of the Alfabet parameter `BASE` are created automatically. For filter fields resulting from conditions with parameters, the filters must be added to the custom report view manually. For more information about defining filters see the section Defining Filters for Configured Reports and Selectors in the chapter Defining Queries.

The method to create filter fields automatically by creating a new custom report view must not be used for this type of configured report. Deleting of the custom report view to create a new one after the configured report was specified will delete all specifications made with the **Report Assistant**. Therefore, filters can only be defined manually after including parameters into the conditions of the query.

- If the configured report is applied to a class (it depends on a base object of the selected class) and the user does not have edit rights to the base object, the **Edit** button in the matrix report will be disabled. If the user has edit rights to the object or the configured report is not applied to a class, the user will be able to create or delete business supports or object relationships for all objects displayed in the matrix. The query defined to find the objects displayed in the rows or columns of the configured report should Therefore, take access permissions for the current user into account.

Defining Matrices for Multi-Editing of Object Relationships

The configured report can handle two types of relationships:

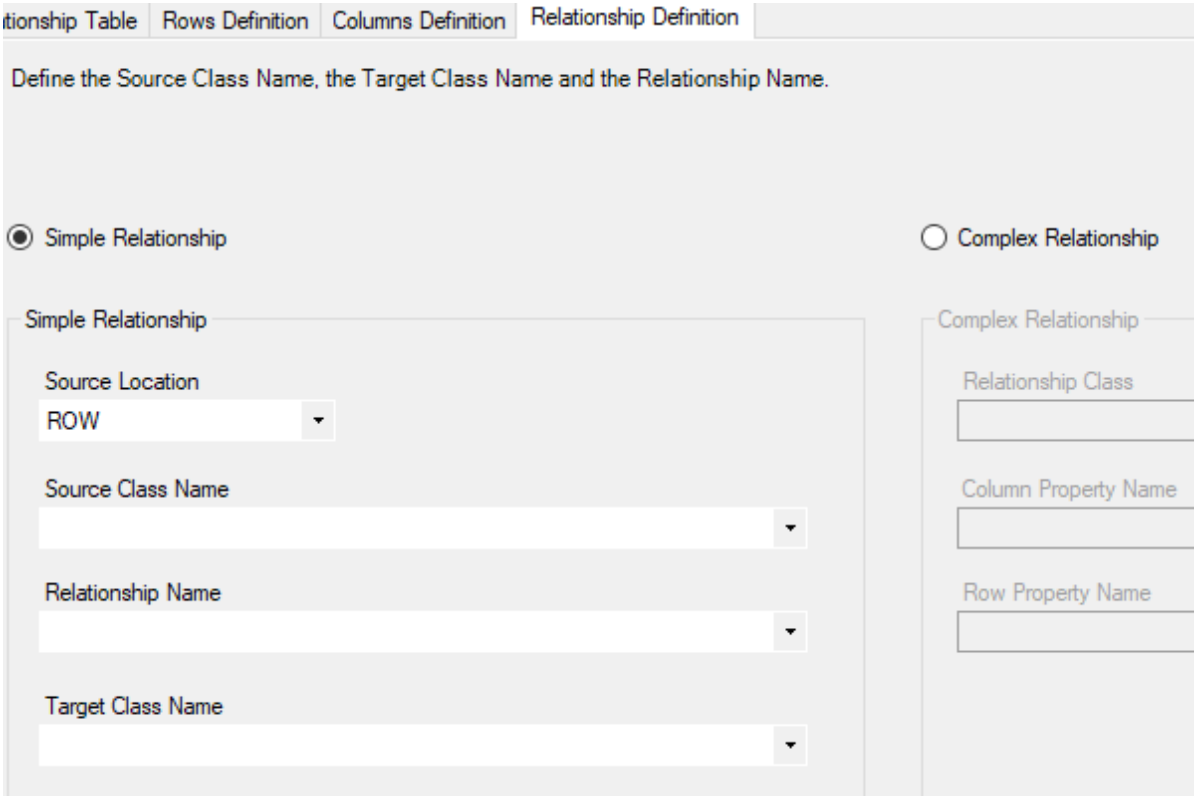
- **Simple Relationships**: Relationships between objects are defined in properties of the type `Reference` or `ReferenceArray`. The configured report for multi-editing of relationships is a two-dimensional matrix that displays the objects that have the property establishing the relationship in one dimension and the objects to that the relationship is built in the second dimension.
- **Complex Relationships**: The relation between two objects is stored in a third, intermediate object, that is exclusively used to store a relation between two objects in two properties of the type `Reference`. The configured report for multi-editing of relationships is a two-dimensional matrix that displays the objects the intermediate object class is referencing with one property in one dimension and the object class that the intermediate object is referencing in the second property in the other dimension.

Intermediate object classes that are designed to define relationships between two other object classes are:

Object class storing the relation	Relation of ... (property storing the relation)	to ... (property storing the relation)
`BusinessDataUsage`	Business data, business object (Data)	Object using the business data/business object (Object)
`DemandArch`	Demand (Demand)	Objects assigned to the demand as affected architecture (Object)
`PlatformElementDependency`	Platform element (Element)	Another platform element depending on the platform element (Dependent Element)
`ProjectArch`	Project (Project)	Objects assigned to the project as affected architecture (Object)

Object class storing the relation	Relation of ... (property storing the relation)	to ... (property storing the relation)
ValueNode	Value node (ValueNode)	Objects assigned to the value node (Object)

The content of the rows and columns of the relationship matrix is customizable and is defined by an Alfabet query or a native SQL query in the **Row** and **Column** tab of the **Report Assistant** as specified above. The relationship defined by the user when clicking a field in the matrix is specified in the fourth tab of the **Report Assistant**:



To define a simple relationship, edit the following fields:

- **Simple Relationship**: Select the checkbox to activate the configuration of a simple relationship.
- **Source Class Name**: Select the object class with the property establishing the solution from the drop-down list.
- **Source Location**: Define, whether objects of the Source Class are specified in the rows or columns of the configured report by selecting the respective location form the drop-down list.
- **Relationship Name**: Select the property that establishes the relationship from the drop-down list. The drop-down list includes all properties of the object class selected on the field **From Class Name**.
- **Target Class Name**: Select the object class to that a relation is specified from the drop-down list. The drop-down list displays the names of all object classes to that the property selected in the **Relationship Name** field specifies a relation.

To define a complex relationship, edit the following fields:

- **Complex Relationship**: Select the checkbox to activate the configuration of a complex relationship.
- **Relationship Class**: Select the intermediate object class that establishes the relation between the objects defined for the rows and columns of the configured report from the dropdown list.
- **Column Property Name**: Select the property of the intermediate object class that stores the information about the relation to the object class in the column definition.
- **Row Property Name**: Select the property of the intermediate object class that stores the information about the relation to the object class in the row definition.

Please note that the columns and rows of the matrix can show information about multiple object classes, but a relation can only be established to the class of the row or column definition that is the base class. The following rules apply:

- In a simple tabular output, the base class is the `FIND` class of the Alfabet query or the class for that the `REFSTR` is defined as first argument in the `SELECT` statement of the native SQL query.
- In a grouped dataset, the base class is the object class defined as lowest level of the dataset.

These rules should be regarded carefully because otherwise relations to object classes that are not allowed as relationship target are established. For example, you want to define a relationship between applications and business processes via the property Applications of the business process, and you want to display information about the ICT object the application is assigned to in the rows defining the applications. Make sure to define the applications as the base class in the respective query for a simple dataset or as subordinate class in a grouped dataset. Otherwise the setting of a relation within the matrix will store the `REFSTR` of the ICT object in the Applications property of the business process.

Defining Matrices for Multi-Editing of Business Supports

A business support describes the use of a specific application by an organization in support of a specific business process or domain. Therefore, a business support has three dimensions.

The configured report for multi-editing of business supports is a two-dimensional matrix. One dimension is specified in the row and one in the column definition. To specify the third dimension, there are two possible options:

- either the row or the column definition can include the third dimension.

For example, the column of the configured report shows business processes, while the row definition shows an expandable table with organizations as a first level and ICT objects owned by the organizations on the second level. When a user clicks a cell in the matrix, a business support is created that defines the use of the selected ICT object by the organization owning the ICT object to support the selected business process.

- the object the user is currently working with when opening the matrix is the third dimension of the business support. If the configured report includes the current object as the third dimension, a filter will be created automatically at the top of the configured report so that the user can select another object of the object class as the base object.

For example, business support shall be specified that is provided by the application the user is currently working with. The matrix based report for multi-editing of objects is added as page view to the object profile of the class application. The matrix is designed to display organizations on one axis and business processes on the other axis. When clicking a cell in the matrix, a business support is created that defines the use of the current application by the selected organization to support the selected business process. The user can use the filter on top of the configured report to select another application and define business support for the selected application.

The content of the rows and columns is customizable and is defined by an Alfabet query or a native SQL query in the **Row** and **Column** tab of the **Report Assistant** as specified above. The row and column header definition is generated based on a query. Each Show property of the Alfabet query or `SELECT` property of the native SQL query creates a column in the header definitions. Object relationships can be reflected in the column and row definitions and can be structured by creating an expandable result table in the row or column definition.

In the example below, the row definition displays organizations, sub-organizations thereof and ICT objects owned by the sub-organizations in a three-level expandable table. The column definitions display business processes grouped by subordinate business processes. In the **Report Assistant**, the root level of a column definition is named `COL1`, the second level `COL2` and so on. Rows are named accordingly as `ROW<number>`.

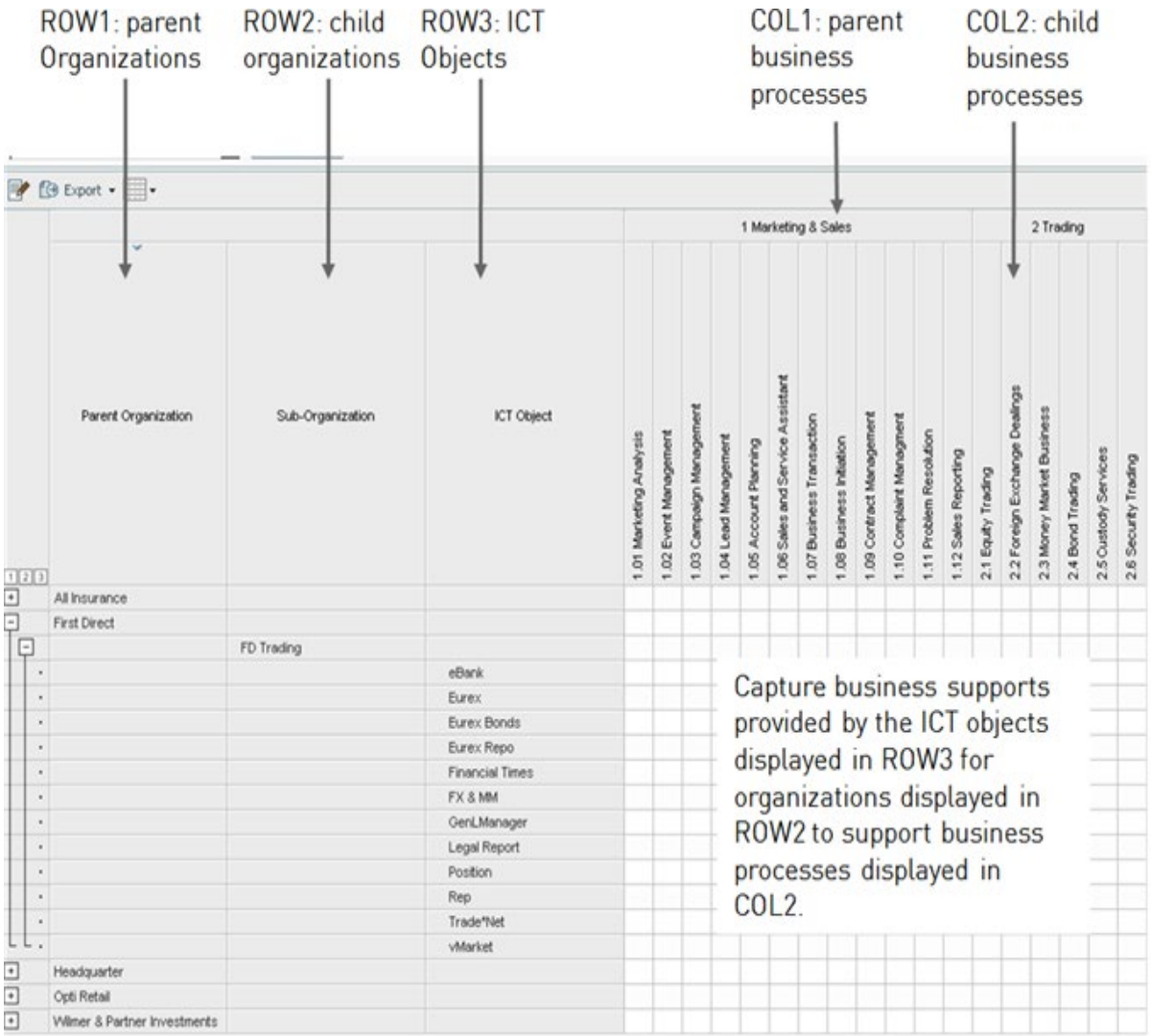
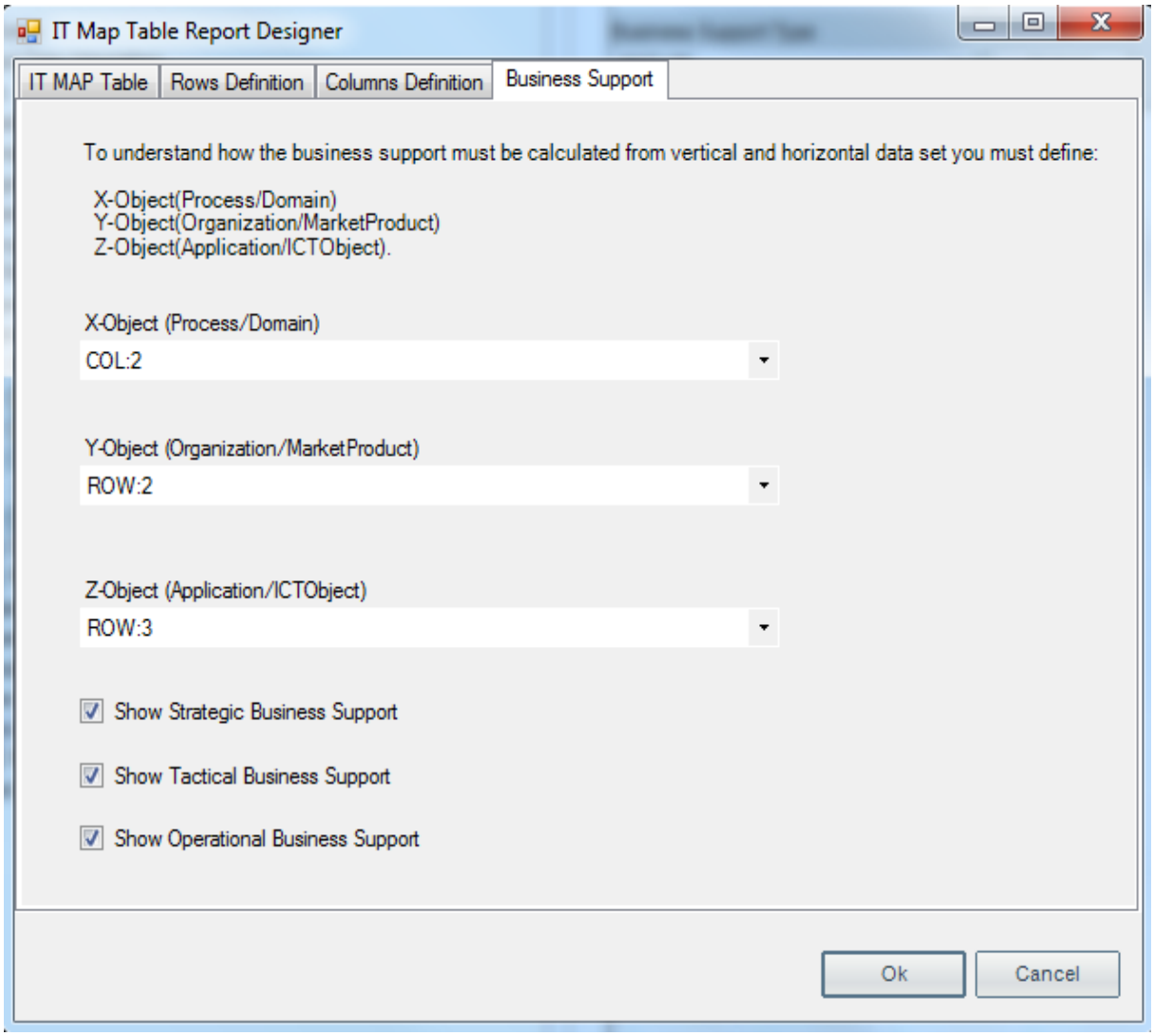


FIGURE: Example of a matrix for multi-editing of business supports

The business support defined by the user when clicking a cell in the matrix is specified in the fourth tab of the **Report Assistant**:



To define the relationship, edit the following fields:

- **X-Object (Process/Domain)**: Define where the business process or domain supported by the business support are specified in the matrix report. If business processes or domains are specified in one of the row or column header fields, select the respective row or column. If the base object is the supported business process or domain, select `BASE`.

  The term X-Object refers to the dimension of the business support and not to the display of objects in the matrix report. For this and the following settings, objects can be specified in any of the row and column specifications. It is even possible to select two dimensions of the business support from a row specification or a column specification. In the example above, organizations (Y-Object) and ICT objects (Z-object) are both specified in the column definition of the matrix.

- **Y-Object (Organization/Market Product)**: Define where the organization or market product using the business support are specified in the matrix report. If organizations or market products are

specified in one of the row or column header fields, select the respective row or column. If the base object is the supported organization or market product, select `BASE`.

- **Z-Object (Application/ICTObject)**: Define where the application or ICT object providing the business support are specified in the matrix report. If applications or ICT objects are specified in one of the row or column header fields, select the respective row or column. If the base object is the application or ICT object providing the business support, select `BASE`.
- **Show <Strategic/Tactical/Operational > Business Support**: Select which type of business support can be created in the matrix. You can select more than one type of business support. If more than one type is selected, a filter on top of the configured matrix allows the user to decide which type of business support he wants to define when working with the matrix.

Filters are created automatically on top of the matrix that allow the user to define the additional parameters to define the selected type of business support. For example, filter fields for the selection of master plan maps or strategy maps must be set prior to working with the matrix. The filter fields defined depend on the selection of business support types in the **Report Assistant**.

Configuring a Report for Multi-Editing of Indicators

The **Report Assistant** displays an explorer and an attribute window on the right of the explorer. The name of the explorer root node is identical to the name of the configured report you are currently designing. The attribute window displays the attributes of the selected element in the explorer. The attributes allow to define the look and content of elements of the configured report.

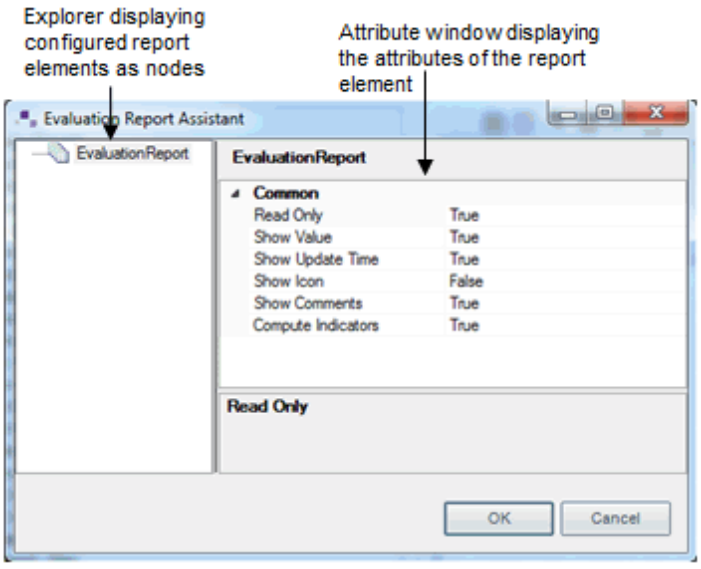


FIGURE: Interface of the Report Assistant for an evaluation report

When you right-click a node in the explorer, the context menu of the selected element opens and allows you to add new elements as child elements of the current element, if applicable.

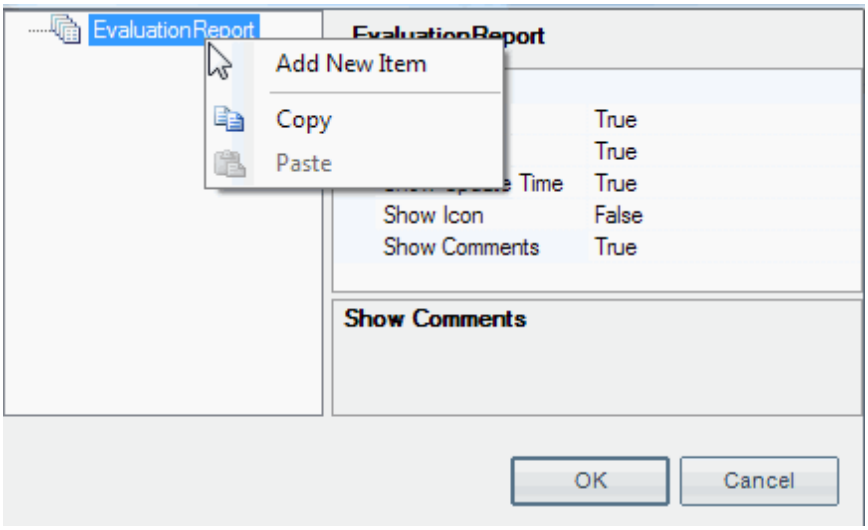


FIGURE: The context menu of the explorer root node of an evaluation report

The context menu also offers a copy and paste functionality to ease re-use of elements within a configured report or between configured reports.

If you copy an element, the element, its attributes and all child elements are copied. You can paste the copied element into another element of the same or a parallel configured report using the paste functionality of the context menu of the proposed parent element.

Defining the Column Display and Editability of the Evaluation Report

The general design of the configured report is defined via the attributes of the root node element. The root node attributes define which columns are displayed in the configured report and whether indicator values can be edited in the configured report.

To edit the attributes of the root node, click the root node in the explorer and edit the attributes described below in the attribute window:

Attribute	Description
Read Only	If set to `false`, the configured report allows to change indicator values and comments. If set to `true`, the configured report is view only. Please note that the report will be ReadOnly if embedded in object cockpits, regardless of the setting of the attribute.
Show Value	Defines whether the indicator value is shown in a column **Value** of the configured report. This attribute must be set to `true` if the user shall be able to edit indicator values directly in the columns of the configured report.
Show Comments	Defines whether the indicator comment is shown in a column **Comments** of the configured report. This attribute must be set to `true` if the user shall be able to edit indicator comments directly in the columns of the configured report.

Attribute	Description
Show Icon	Defines whether the indicator icon is shown in a column **Icon** of the configured report. **Note:** An icon can only be displayed in the configured report if the indicator type has been configured to display icons rather than numerical values. Indicator types are specified in the **Evaluations and Portfolios** functionality of Alfabet. For information about the configuration of indicator types and the icon gallery used to visualize their values, see *Defining an Indicator Type for an Evaluation Type* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.
Show Update Time	Defines whether the time the indicator was last changed is displayed in the configured report in a column.
Compute Indicators	Defines how computed indicators that are included in the configured report are updated. Computed indicators are calculated automatically and cannot be edited by the user. If **Compute Indicators** is set to `False`, computed indicator are updated when the user selects the option **Action** > **Compute Indicators for All Objects in the Report** in the configured report. If **Compute Indicators** is set to `True`, computed indicator are updated each time the user clicks the **Save** button in the toolbar of the configured report. The option **Action** > **Compute Indicators for All Objects in the Report** is not available in configured reports that are configured to recalculate computed indicators on each saving of indicator values.
Compute All Indicators	If set to `True`, re-calculation of computed indicators triggered via the configured evaluation report includes all computed indicators in the Alfabet database. If set to `False`, only computed indicators visible in the configured evaluation report are re-calculated. For performance reasons it is recommended to set this attribute to `False` if the database contains a high number of computed indicators.
Input Controls Limit	This setting is only taken into account if **Read Only** is set to `False`. It limits the number of input controls that may be displayed in the configured evaluation report. If the configured report exceeds the maximum number of input controls, the user is informed that he/she shall refine the search to reduce the tabular display to the allowed number of input controls. If this attribute is set, the configured evaluation report should have filters for refining the display.
Report Row Limit	This setting is only taken into account if **Read Only** is set to `False`. It limits the number of rows that may be displayed in the configured evaluation report. If the configured report exceeds the maximum number of rows, the user is informed that he/she shall refine the search to reduce the tabular display to the allowed number of rows. If this attribute is set, the configured evaluation report should have filters for refining the display.

Defining the Hierarchy of the Objects and Indicators Displayed in the Evaluation Report

The root node element can contain one or multiple **Item** elements. These first level **Item** elements must have a query definition in their attribute specifications to find the objects displayed in the configured report.

Only objects of artifact bject classes representing an artifact in the IT landscape can be selected to build evaluation reports.

The query can either be a native SQL query or an Alfabet query.

If you are not familiar with Alfabet queries, see the chapter Defining Queries.

For special rules that apply to the definition of native SQL queries in the context of configurations for Alfabet, see the section Defining Native SQL Queries.

The information displayed in the **Object** column of the evaluation report is identical to the column headers that would result from a tabular output of the defined query.

Report columns that are defined for technical reasons are not displayed in the object boxes in the configured report. This includes:

- The first column of the output of a native SQL query. This column must specify the `REFSTR` of the object for technical reasons and is ignored in the visible output.

The configured report can either display a flat list of objects and their indicators, or a grouped report can be configured. There are two methods to configure a grouped report:

- The complete content of the configured report is defined in one query resulting in a grouped dataset.
- A separate query is added to the report configuration for each object class added to one of the levels of the configured report.

When separate queries are defined for each level, a high number of queries must be executed by the database to display the configured report. This can lead to performance issues. Therefore, it is recommended to built the configured report on a grouped dataset defined in a single query. Nevertheless for some use cases, like For example, defining different indicators for objects assigned to the same level, it is required to define multiple queries for the configured report.

Defining a Grouped Report on Basis of a Single Query

A query resulting in a grouped tabular report can be defined for the first level **Item** element to define a grouped report. For a grouped report, the **Object** column of the evaluation report displays the information defined for the column headers of the respective group.

Grouped reports are defined via `GroupBy_Ex` Alfabet instructions defined in Alfabet query language. Alfabet instructions can also be combined with native SQL queries.

A detailed description about how to define a grouped report using the `GroupBy_Ex` c Alfabet instruction is given in the section Grouping Query Results in Expandable Reports in the chapter Defining Queries.

If the query output results in a grouped report, an **Item** element without a query specification must be added to the explorer tree for each grouping level in a hierarchical structure, that means each subordinate level **Item** element as child of the superordinate **Item** element. The hierarchy of **Item** elements correspond to the grouping levels. To enhance the overview over the report structure, an attribute **Name** is available for the item elements that allow to rename the **Item** explorer nodes.

Each element **Item** contains an attribute **Evaluation Types** that defines the evaluation types for that indicators are displayed and optionally can be edited in the configured report.

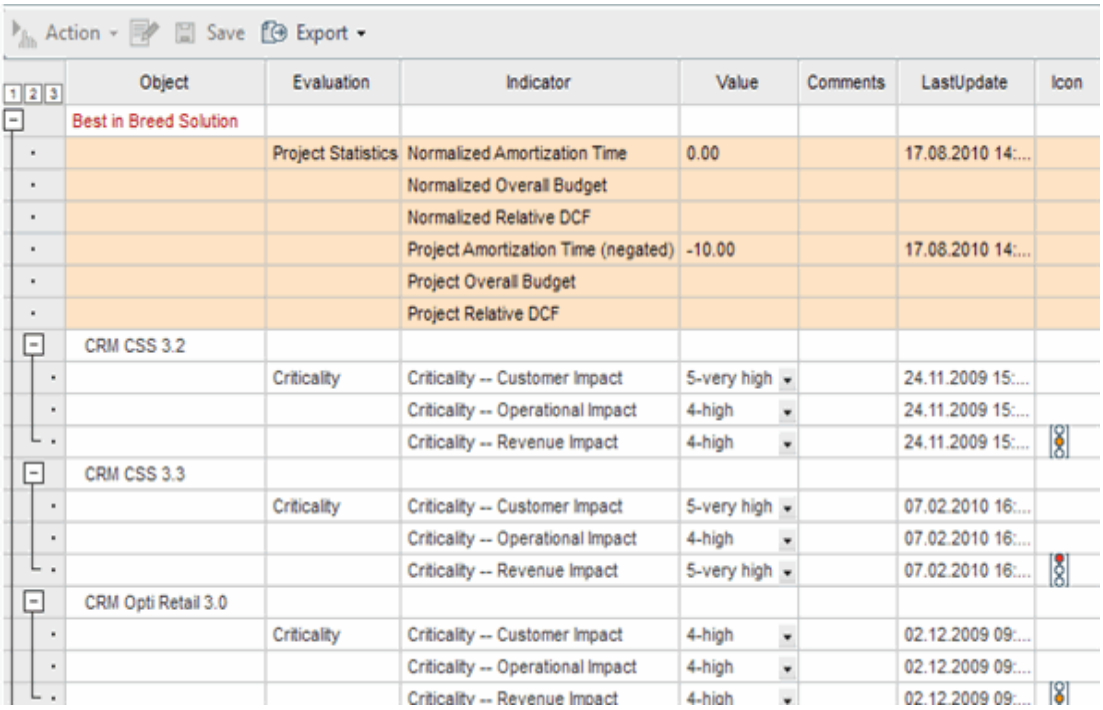
Please note the following when defining evaluation types for **Item** elements.

- The evaluation type must be assigned to the object class displayed in the level of the grouped report the **Item** element is defined for. For information about the assignment of evaluation types to object classes, see the chapter *Configuring Evaluations, Prioritization Schemes, and Portfolios* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.
- **Item** elements are not defined per object class but per grouping level. You can define grouped datasets with more than one object class per level, but in that case, the evaluation types selected in the **Item** element for the level must be assigned for all object classes.

The attribute **Visible** of the **Item** element allows you to define rows that are not displayed in the configured report. This is useful to display the indicators of objects that are related to the first level object via an intermediate object without displaying the information about the intermediate object class.

The following example shows a configured report that allows selected indicators to be edited for applications assigned to projects via the project architecture. The configured report also displays indicators for the project that the application is assigned to. The indicators on the project level are aggregated indicators and cannot be edited in the configured report and are Therefore, displayed as colored rows. To recompute the aggregated indicators, click **Action** > **Compute indicators for all objects in report**.

Please note that only one of the indicators for an application is configured to display an icon. Therefore, the column **Icon** will only display an icon in the row of that indicator.



Object	Evaluation	Indicator	Value	Comments	LastUpdate	Icon
Best in Breed Solution						
	Project Statistics	Normalized Amortization Time	0.00		17.08.2010 14:...	
		Normalized Overall Budget				
		Normalized Relative DCF				
		Project Amortization Time (negated)	-10.00		17.08.2010 14:...	
		Project Overall Budget				
		Project Relative DCF				
CRM CSS 3.2						
	Criticality	Criticality -- Customer Impact	5-very high		24.11.2009 15:...	
		Criticality -- Operational Impact	4-high		24.11.2009 15:...	
		Criticality -- Revenue Impact	4-high		24.11.2009 15:...	
CRM CSS 3.3						
	Criticality	Criticality -- Customer Impact	5-very high		07.02.2010 16:...	
		Criticality -- Operational Impact	4-high		07.02.2010 16:...	
		Criticality -- Revenue Impact	5-very high		07.02.2010 16:...	
CRM Opti Retail 3.0						
	Criticality	Criticality -- Customer Impact	4-high		02.12.2009 09:...	
		Criticality -- Operational Impact	4-high		02.12.2009 09:...	
		Criticality -- Revenue Impact	4-high		02.12.2009 09:...	

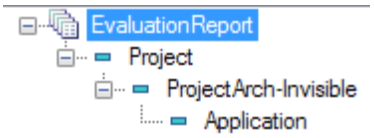
Applications are assigned to projects via the object class `ProjectArch` that defines the relation between a project and an object. The grouped dataset of the example Therefore, has three levels of related object classes: `Project`, `ProjectArch` and `Application`. The configured report can For example, be created on basis of the following Alfabet query:

```
ALFABET_QUERY_500
FIND Project
   InnerJoin ProjectArch ON ProjectArch.Project = Project.REFSTR
   InnerJoin Application ON ProjectArch.Object = Application.REFSTR
Instructions
   JoinColumns("Application.REFSTR,BusinessProcess.REFSTR","Applicat
   ion.REFSTR"," ");
   GroupBy_Ex("Project.Name","ProjectArch.REFSTR","Project",0);
   GroupBy_Ex("ProjectArch.REFSTR","Application.REFSTR","ProjectArch
   ",1);
   RemoveColumns("Application.REFSTR,ProjectArch.REFSTR");
EndOfInstructions
AUTODSINFO
QUERY_XML
<QueryDef>
   <ShowProperty Type="Property" ClassName="Project" Name="Name" />
   <ShowProperty Type="Property" ClassName="ProjectArch"
   Name="REFSTR" />
   <ShowProperty Type="Property" ClassName="Application" Name="Name"
   />
```

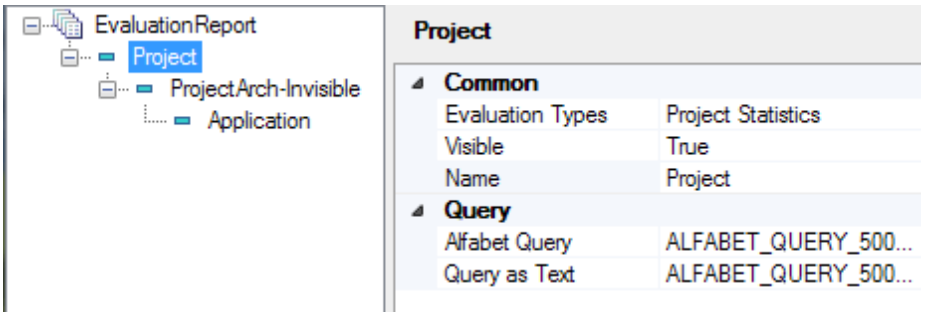


```
        <ShowProperty Type="Property" ClassName="Application"
        Name="Version" />

        <ShowProperty Type="Property" ClassName="Application"
        Name="REFSTR" />

        <SortProperty Type="Property" ClassName="Project" Name="Name" />

        <SortProperty Type="Property" ClassName="ProjectArch"
        Name="REFSTR" />

    </QueryDef>
```

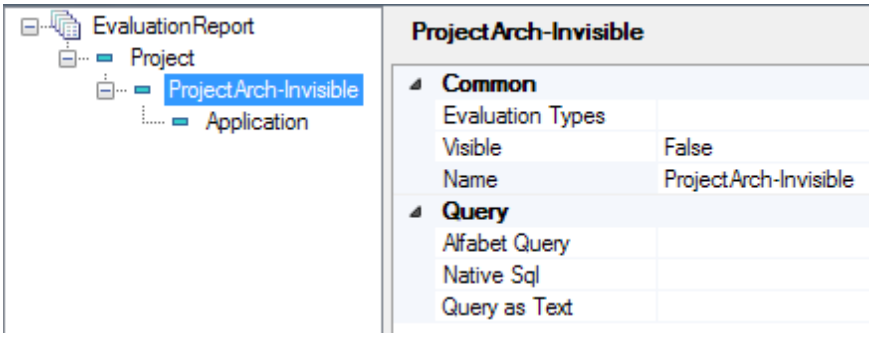
In the **Report Assistant** of the configured report, one Item element must be added to the root node for the definition of the first level. The first level Item element contains a second level Item element which contains the third level Item element:



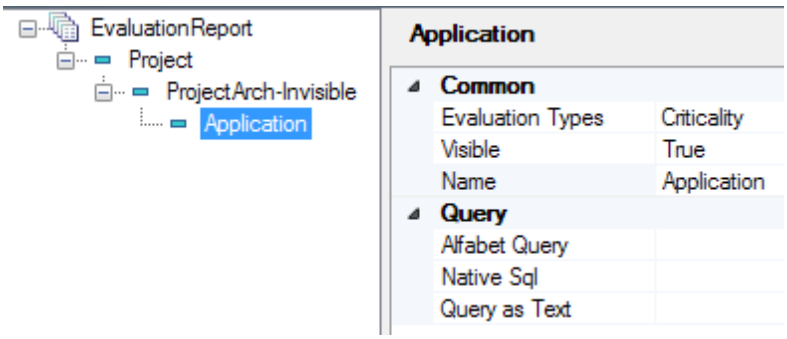
The Alfabet query defining the grouped dataset is defined in the attributes of the first level **Item** element. The first level element also contains the definition of evaluation types that shall be displayed for the projects displayed as first level:



The second grouping level of the configured report displays information about the objects of the object class `ProjectArch` that establish the relation between projects and applications. This level shall not be displayed in the configured report. The attribute **Visible** of the **Item** element in the second level is Therefore, set to `False`.



The third grouping level display applications. The evaluation type for that indicators shall be added in the configured report is defined in the third level **Item** element.



Defining a Grouped Report on Basis of One Query Per Object Class

Item elements can be structured hierarchically. If an element **Item** contains a child element **Item**, for each row added to the configured report on basis of the definition in the parent **Item** element, subordinate rows are added to the configured report on basis of the definition in the child element **Item**. To enhance the overview over the report structure, an attribute **Name** is available for the **Item** elements that allow to re-name the **Item** explorer nodes.

For each element **Item** a query defined in the attributes of the **Item** element specifies for which objects rows are added to the configured report on basis of an Alfabet query or a native SQL query.

The information displayed in the **Object** column of the evaluation report is identical to the column headers that would result from a tabular output of the defined query.

Report columns that are defined for technical reasons are not displayed in the object boxes in the configured report. This includes:

- The first column of the output of a native SQL query. This column must specify the `REFSTR` of the object for technical reasons and is ignored in the visible output.

The attribute **Visible** of the **Item** element allows you to define rows that are not displayed in the configured report. This is useful to display the indicators of objects that are related to an object via an intermediate object without displaying the information about the intermediate object class.

For each element **Item** the display of the indicators of the objects can be defined by setting the attribute ***Evaluation Types*** of the **Item** element.

Note the following when defining the query for the **Item** element:

- If an element **Item** is child of a parent **Item** element, the query defined for the **Item** element must contain the Alfabet parameter:`BASE` in a `WHERE` clause to map the objects of this layer to the objects in the parent layer. `BASE` is identical to the `REFSTR` of the object in the parent layer.

  This specification is not mandatory for the top layer objects. The Alfabet parameter `BASE` can only be used in the Alfabet query or native SQL query defining the top layer objects if the configured report is assigned to an object class with the attribute **Apply To Class**.

- The information displayed in the **Object** column of the evaluation report is identical to the column headers that would result from a tabular output of the defined query.

Report columns that are defined for technical reasons are not displayed in the object boxes in the configured report. This includes:

- The first column of the output of a native SQL query. This column must specify the `REFSTR` of the object for technical reasons and is ignored in the visible output.

Defining the Attributes of the Item Element

The table below lists all attributes of the **Item** element: .

Attribute	Description
Evaluation Types	Click in the field to display the **Browse** button and click the button to open a selector. Select evaluation types to add all indicators based on indicator types that are assigned to the listed evaluation types to the configured report. **NOTE**: The evaluation type must be assigned to the relevant object class. For information about the assignment of evaluation types to object classes, see the chapter *Configuring Evaluations, Prioritization Schemes, and Portfolios* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.
Visible	Select `True` if you want objects to be visible in the configured report. Select `False` if you do not want the objects to be visible in the configured report.
Name	Define a name for the **Item** element. The name is used in the explorer tree as name for the **Item** node.
Alfabet Query / **Query as Text** / **Native Sql**	Defines the objects displayed in the configured report via an Alfabet query or a native SQL query. For each object found by the query a line is added to the configured report. If you want to define an Alfabet query, you can either use the **Alfabet Query Builder** available via the **Alfabet Query** attribute or write the Alfabet query into a text editor available via the **Query as Text** attribute. If you want to define a native SQL query, use the text editor available via the **Native Sql** attribute. The text editor has separate tabs for the definition of the native SQL query and the definition of Alfabet query language instructions that can be used in combination with native SQL queries to built a grouped dataset. To open an editor, click in the respective attribute field to display the **Browse** button and click the button.

Configuring a Report for Multi-Editing of Aspect Indicators

This configured report provides a matrix view on aspect indicators and an editor for changing indicator values and comments. An aspect indicator is an indicator for the evaluation of an object in a specific context. For example, an application may be evaluated differently in the context of different business processes the application is providing services for. The business process is then the aspect for the evaluation.

An aspect indicator stores a value for a specific combination of evaluated object and aspect object for a defined indicator type and evaluation type.

Standard views for the definition of aspect indicators are only available for the object classes application, and component and project. For all other object classes the configured report is required to set the aspect indicators on the Alfabet user interface.

For general information about aspect indicators, see in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.

The report is configured in the report assistant opening for configured reports based on the template `Report`. For information about creating the report and opening the report assistant, see Creating a New Report for Multi-Editing of Objects

The content and layout of configured report is defined by setting the attributes in the attribute pane of the elements in the explorer of the **Report Assistant** window.

The following table lists the explorer node elements that can be configured to specify the content of the report as well as provides information about the purpose of each sub-element:

Explorer Node Element	Required to Configure:
Root Node	Definition of the layout for the report.
Aspects	Definition of the aspects that indicators shall be defined for.
Indicators	Definition of the indicator typess for that indicators shall be edited for each combination of aspect and object.
Objects	Definition of the objects the indicators shall be defined for.

The following information is available about the configuration of the configured report:

- Defining the Objects, Aspects and Indicators for the Evaluation
- Defining the Layout of the Aspect Indicator Report

Defining the Objects, Aspects and Indicators for the Evaluation

The report displays a matrix for setting combinations of aspects and indicators for either a single base object or a defined set of objects.

The aspect indicator is stored in the object class `Indicator`. The following object class properties of the object class indicator are relevant for the report:

- ***Object***: A reference to the object which is evaluated. In the configured report, either a number of objects or a single object can be evaluated:
    - If indicators shall be set for multiple objects, a query must be defined for the report as sub-node of the ***Objects*** explorer node of the report assistant. The objects returned in the query will be displayed as row headers in the matrix of the aspect indicator report. Details of the query definition are provided below.
    - If indicators shall be set for one object only, the **Apply to Class** attribute of the configured report must be set to the object class of the object that shall be evaluated. The report can then be added to object cockpits or a wizard for the object class it is assigned to and will allow the user to set indicators for the object he/she is currently working with. If the configured report is opened without a context providing a base object, a filter for selection of a base object will be automatically added above the configured report.
- ***Aspect***: A reference to the object that is the aspect of the evaluation. In the configured report, a query must be defined as sub-node of the ***Aspects*** node of the report assistant to specify the aspect objects. The aspect objects will be displayed as column headers or additional level of row headers in the matrix, dependent on the configuration of the aspect indicator report. Details of the query definition are provided below.
- ***IndicatorType***: A reference to the indicator type of the indicator. In the configured report, a query must be defined as sub-node of the ***Indicators*** node to specify the indicator types for that indicators shall be defined in the configured report. The indicator types will be displayed as column headers or additional level of row headers in the matrix, dependent on the configuration of the aspect indicator report. Details of the query definition are provided below.
- ***Value***/***SemanticValue***: The semantic value is displayed in the configured report and the editor. Both attributes are set if a change is made via the editor of the configured report.
- ***Comments***: The comment a user has entered when setting the indicator. The comment can optionally be shown in the report. It is always shown in the editor and editable for the user.
- UpdateTime: The time and date of the last update of the indicator. The information can optinally be shown in the report and is always shown read-only in the editor.

To define objects, aspects and indicators in the report assistant:

- Right-click the node ***Objects***, ***Aspects***, or ***Indicators***and select **Add New Query**.
- Define the query in the attributes of the configured report. The attribute **Query As Text** opens a simple editor to type in either a native SQL query or an Alfabet query. The attributes **Native SQL Query** and **Alfabet Query** open editors specific for the respective query language.

Note the following about the query:

- Depending on the parent node, the query must find objects of the classes `IndicatorType`, the evaluated object class, or the object class that represents the aspect. That means that

this class must be the `FIND` class of an Alfabet query. In native SQL queries, the first `SELECT` argument must return the `REFSTR` values of objects of the respective class.

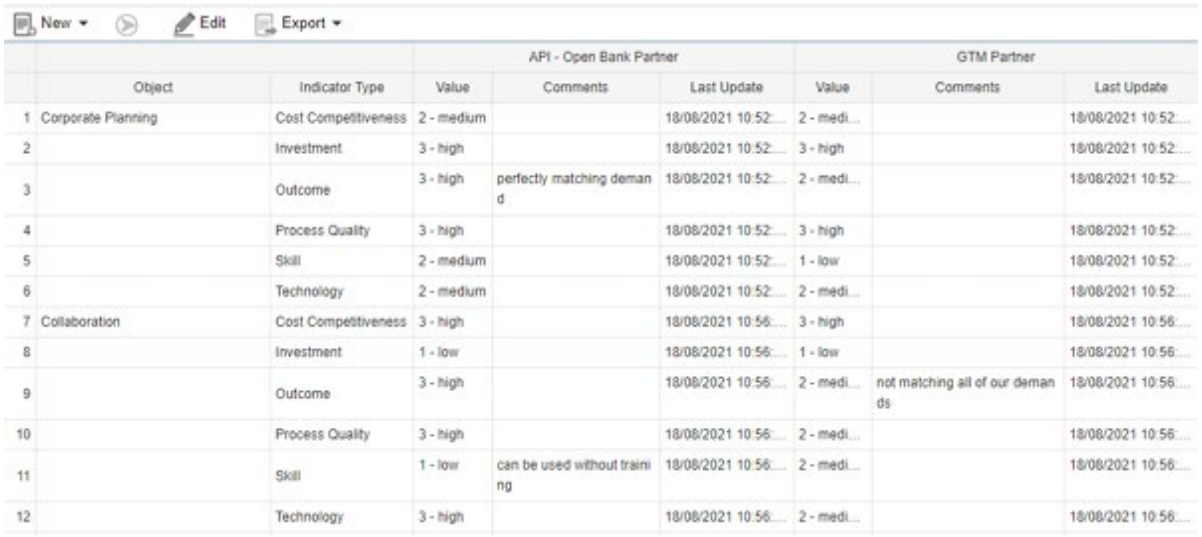
- The text that shall be displayed in the row headers or column headers must be returned in a single column of the query dataset. The column name for that column must be identical in all return datasets of all queries defined for the aspect indicator report.

For more information about defining queries, see Defining Queries.

- After having defined all queries for the aspect indicator report, click the root node of the report assistant and enter the name of the column of the query dataset returning the text to be displayed in the **Image Column** attribute.

Defining the Layout of the Aspect Indicator Report

The following image shows the default layout of an aspect indicator report:

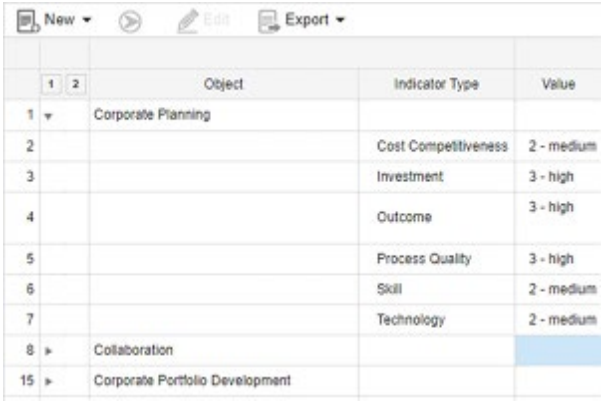


			API - Open Bank Partner			GTM Partner		
	Object	Indicator Type	Value	Comments	Last Update	Value	Comments	Last Update
1	Corporate Planning	Cost Competitiveness	2 - medium		18/08/2021 10:52:...	2 - medi...		18/08/2021 10:52:...
2		Investment	3 - high		18/08/2021 10:52:...	3 - high		18/08/2021 10:52:...
3		Outcome	3 - high	perfectly matching demand	18/08/2021 10:52:...	2 - medi...		18/08/2021 10:52:...
4		Process Quality	3 - high		18/08/2021 10:52:...	3 - high		18/08/2021 10:52:...
5		Skill	2 - medium		18/08/2021 10:52:...	1 - low		18/08/2021 10:52:...
6		Technology	2 - medium		18/08/2021 10:52:...	2 - medi...		18/08/2021 10:52:...
7	Collaboration	Cost Competitiveness	3 - high		18/08/2021 10:56:...	3 - high		18/08/2021 10:56:...
8		Investment	1 - low		18/08/2021 10:56:...	1 - low		18/08/2021 10:56:...
9		Outcome	3 - high		18/08/2021 10:56:...	2 - medi...	not matching all of our demands	18/08/2021 10:56:...
10		Process Quality	3 - high		18/08/2021 10:56:...	2 - medi...		18/08/2021 10:56:...
11		Skill	1 - low	can be used without training	18/08/2021 10:56:...	2 - medi...		18/08/2021 10:56:...
12		Technology	3 - high		18/08/2021 10:56:...	2 - medi...		18/08/2021 10:56:...

The report shows a matrix with the following structure:

- **Row headers**: The object to be evaluated is displayed in the first column of the report. The second column lists the indicator type for that the indicators are displayed in the current matrix. The row header columns are frozen, which means that they will remain visible on scrolling horizontally.
- **Column headers**: The first level row headers are listing the aspects relevant for the evaluation. For each aspect, three rows providing information about the value, comment and last update time of the aspect indicator are displayed.

This default layout can be changed via the attributes of the root node of the report assistant.

Attribute	Description
Report Layout	By default, this attribute is set to `Compact` and a static table is displayed with all rows visible. Set this attribute to `Tree` to change the layout to a collapsible dataset with the rows defined by the second row header level being collapsible: 
Indicator Grouping	This attribute defines the grouping of the aspect indicators in the report dataset and in the editor. Select one of the following: • `By Aspect`: Indicator types are used as row headers while aspects are used as column headers. This is the default layout. • `ByIndicatorType`: Aspects are used as row headers while indicator types are used as column headers. 
Object Leading	This attribute defines the order of row headers. By default, it is set to `True` and the first level of row headers displays the objects to be evaluated. Set this attribute to `False` to revers the order of row headers and sort by object in the second level. The first level will then sort by either indicator type or aspect, depending on the setting of the **Indicator Grouping** attribute. 

Attribute	Description
	Please note that this setting also changes the content of the editor. The editor will open for all aspect indicators available for the first level column header for that a row is currently selected in the matrix. In the example above, a user selecting any row beneath `Cost Competitiveness` will be able to set the `Cost Competitiveness` for all combinations of object and aspect. In contrast, the default setting of the report will provide an editor for setting all aspect indicators for a selected object for all combinations of indicator type and aspect.
Comment Visible	Set the attribute to `False` to hide the **Comments** columns from the dataset. If comments are hidden from the dataset, they are nevertheless displayed and editable in the editor.
Last Update Visible	Set the attribute to `False` to hide the **Last Update** columns from the dataset.
Show As Icon	If set to `True`, indicators will be displayed as icons in the report dataset. Please note that indicators must be based on indicator types with an icon definition to show icons in the configured report. For information about configuring indicators to be represented as icons, see *Configuring Indicator Types for an Evaluation Type* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.
Single Object	

Configuring a Report for Multi-Editing of Questionnaire Indicators

The report is configured in the report assistant opening for configured reports based on the template `QuestionnaireEvaluationReport`. For information about creating the report and opening the report assistant, see Creating a New Report for Multi-Editing of Objects.

The content and layout of configured report is defined by setting the attributes in the attribute pane of the elements in the explorer of the **Report Assistant** window.

The following table lists the explorer node elements that can be configured to specify the content of the report as well as provides information about the purpose of each sub-element:

Explorer Node Element	Required to Configure:
Root Node	Definition of the functionality and layout of the report.
Item	Definition of the information displayed in the tabular dataset of the configured report.

Defining the Tabular Dataset of the Questionnaire Evaluation Report

Define the following attributes of the ***Item*** note in the **Report Assistant** to select the data to be displayed in the tabular dataset of the questionnaire evaluation report:

Attribute	Description
Alfabet Query/Native SQL Query / Query as Text	Define either an Alfabet Query or a native SQL query finding objects of the object class `QuestionaryIndicator`. Note the following about the query definition: • Alfabet query: The `FIND` class must be `QuestionaryIndicator`. A definition of show properties is ignored. The return dataset is defined via the **Show Properties** attribute described below. • Native SQL query: The first argument of the `SELECT` statement must return the `REFSTR` of the object class `QuestionaryIndicator`. All other arguments of the `SELECT` statement are not relevant to define the return dataset. The return dataset is defined via the **Show Properties** attribute described below. • The sort order of results returned by the query is not relevant for the sort order of rows in the table of the configured report. Results are by default sorted first by question set, then by question and then by object. The user can change the sort order of the report at runtime to sort by question set, then object, and then question. • WHERE conditions can be defined to define which questionnaire indicators shall be included into the dataset. • Alfabet parameters can be used to define filters, or to refer to the current object or the current user.
Show Properties	Click the **Browse**  button at the right of the attribute field to open the **Select and Reorder Entries** window. Select the checkbox of all object class properties that you would like to add to the tabular dataset of the report. Note the following about the selection: • If you select an object class property establishing a reference to a target object, the object class properties defined in the **Image Properties** attribute in the

Attribute	Description

	class settings for the target object class will be displayed in the tabular dataset of the configured report.
	• The header of the column in the result dataset will be concatenated from object class caption and object class property caption. Some of the data is available two times in the list. For example, selection of `QuestionaryIndicator.Questionary` and `Questionary.Name` will both add a column providing information about the name of the questionnaire the questionnaire indicator belongs to. But the column header will be **Questionnaire Indicator Questionnaire** for the selection of `QuestionaryIndicator.Questionary` or **Questionnaire Name** for the selection of **Questionary.Name**.
	• The selection of the following properties will add a link to the dataset that allows to navigate to the object profile of the selected object:
	  • Questionnary.Name and QuestionaryIndicator.Questionary provide a link to the object profile of the questionnaire.
	  • QuestionaryIndicator.Object provides a link to the object profile of the object the question targets.
	  • QuestionaryIndicator.Question provides a link to the object profile of the question.
	  • QuestionnaireEvaluation.ID and QuestionnaireEvaluation.Name provide a link to the object profile of the questionnaire evaluation. Questionnaire evaluations cannot be performed via button interactions in the configured questionnaire evaluation report. The link is required to provide the ability to evaluate the questionnaire.
	• The sort order of columns in the result dataset from left to right is identical to the sort order of the entries in the selector from top to bottom. The order of entries can be changed with the up and down buttons on the right of the object class property selection field.

Name	Optionally, change the name of the **Item** element in the explorer of the report assistant.

Creating a Configured Report Displaying an Object Profile or Object Cockpit

To define a configured report of the type `ObjectView`:

- In the **Reports** explorer of the **Reports** tab of Alfabet Expand, create a new configured report and define the general attributes for the configured report and the attributes specific for configured reports of the type `ObjectView`. This step is described in this chapter.
- Set the **Report State** of the configured report to `Active`. For information on how to change theattribute of the configured report, see.
- Define access rights for the configured report in the **Reports Administration** functionality of Alfabet. By default, access is possible for all users. For more information, see the chapter *Defining and Managing User Access to Configured Reports* in the reference manual *User and Solution Administration*.

To create a new configured report in Alfabet Expand with a reference to an object profile or an existing object cockpit of an object of a specified object class:

1) In the **Reports** tab, right-click the **Reports** folder or a sub-folder thereof and select **Create New Report**. You will see the new configured report listed below the selected folder.

As an alternative to defining all attributes for a new configured report, you can copy and paste an existing configured report and make modifications, as needed. To do so, right-click the existing configured report that you want to copy and select **Copy**. Right-click the new configured report and select **Paste**. Modify the necessary attributes described below.

2) In the attribute window, select `ObjectView` in the **Type** field.
3) In the attribute window, define the following attributes for the configured report:

- **Name**: Optionally, change the technical name of the configured report. The technical name of the configured report must be unique.

The technical name is used for identification of the configured report in the context of technical processes such as merging the report configuration with a report configuration saved as an XML file. Therefore, the following is required when defining the name of a configured report:

- The name must be unique. The name must be unique regardless of the report folder it is structured in.
- The name may not be changed once a configured report has been created.
- A validation mechanism checks for correct syntax when defining a technical name. Please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not

allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.

- If the technical name of the configured report is changed, the name will be correctly updated in other configuration objects referencing the changed report during design time.

Please note that the name of a changed configuration object will not be updated in guide pages or guide views. If you plan to change the name of a configuration object, the reference in the navigation page should be changed prior to changing the configuration object name. You can use the **Show Usage** functionality in the context menu of the configured report to evaluate whether the configured report has been added to a guide view or guide page.

- **Caption**: Enter a caption for the configured report. The name you define here will be displayed in the **Reports Administration** functionality, the **Configured Reports** functionality or the **Configured Reports** page view in object profiles.
- **Description**: Provide information about the configured report that is useful to users. This comment will be displayed in the **Description** field for the configured report in the **Reports Administration** functionality in the **Administration** module, the **Configured Reports** functionality or the **Configured Reports** page view in object profiles.
- **Report State**: The **Report State** attribute is view only and is set to `Plan` for new configured reports. The configured report can only be edited when the **Report State** attribute is set to `Plan`. After finishing all configuration steps described in the following, the **Report State** attribute must be set to `Active` as described in the section General Guidelines for Creating Configured Reports. The configured report is then visible to users in the Alfabet user interface.
- **Apply to Class**: Click the **Browse** button to open a dialog box and select the object class or object class stereotype for which you want to display the object view. A selector that is automatically added at the top of the configured report allows the user to select an object of the object class or object class stereotype the configured report is applied to and view the object profile or object cockpit for the selected object.

  For configured reports of the type `ObjectView`, the **Apply to Class** attribute is mandatory. However, the attribute does not make the configured report available in the object profile of the selected object class, as is the case for the other types of configured report. The configured report is available to users in the **Configured Reports** functionality of the **Search** module only.

  If the configured report is applied to an object class stereotype, the filter field for selection of the base object will use the caption of the object class stereotype as caption of the filter field and the selector defined in the class settings for the object class stereotype for selection of the base object.

- **View**: Select an object view from the drop-down list. The object profile must display data about the object class specified in the **Apply to Class** field.
- **Object View Presentation Type**: Select `Normal` to display the object profile in the configured report. Select `Dashboard` to display the object cockpit in the configured report. The default object cockpit assigned to the view scheme used by the user profile the user is currently logged in with when opening the configured report is displayed.

- **Help Index**: Specify the location of the external Help file using the following syntax:(For example,:). The external help file will be displayed when the user clicks the context-sensitive Help link in the standardHelp menu.

  Server variables can be used in the Help links. Server variables allow you to define all or part of the URL definition in the server alias configuration of the Alfabet Server instead of directly defining it in the configuration in Alfabet Expand. Use of server variables is useful, For example, in order to change a Web server name defined in multiple URL definitions in Alfabet Expand. Instead of changing all URL definitions, only the variable value in the server alias configuration must be updated. The use of server variables is described in the section Using Server Variables in Web Link and Database Server-Related Specifications in the chapter Configuring Reports.

- **Selector Behavior**: The attribute can be used to exclude configured reports that are defined for special purposes, like For example, triggering of data export via the RESTful API, from being selectable in the **Configured Reports** table. By default, the attribute is set to `Visible` and the configured report is visible in the selector for adding configured reports to the **Configured Reports** table if all other conditions for access to the configured report apply. If the attribute is set to `NotVisible`, the configured report is excluded from the selector for adding configured reports to the **Configured Reports** functionality. Please note however, that this setting is not excluding the configured report from any standard views or customer configured views and selectors, but exclusively from the standard selector for configured reports implemented in Alfabet.

The attribute cannot be edited in the attribute window directly. To change the setting, right-click the configured report and select **Set Report Selector Behavior to 'Not Visible'** or **Set Report Selector Behavior to 'Visible'** respectively.

- **Applicable for AlfaBot**: Specify whether the configured report is accessible via the AlfaBot. By default, the attribute will be set to `True` for new reports. Set the attribute to `False` if the configured report should not be opened outside of a specific context. For example, this may be relevant for reports that are specifically designed to be embedded in an object cockpit.

  This attribute is only relevant if the AlfaBot is implemented. For information about the implementation of the AlfaBot, see Implementing the AlfaBot for Navigation via a Full-Text Command.

- **Business Problem Statement**: Optionally enter a description that will help users to search for the configured report in the AlfaBot. The text is not displayed in the user interface, but it is of special relevance for the search mechanisms implemented for the AlfaBot. If the report caption entered by the user in the AlfaBot for a request to open a configured report does not match the caption of one of the available configured reports, the AlfaBot will split the caption entered by the user into keywords and will perform a keyword search on the **Caption**, **Description** and the **Business Problem Statement** attributes of the configured reports that are accessible via the AlfaBot. In addition to a keyword search, a synonym search is performed on the text provided in the **Business Problem Statement** attribute.

  This attribute is only relevant if the AlfaBot is implemented. For information about the implementation of the AlfaBot, see Implementing the AlfaBot for Navigation via a Full-Text Command.

- **Base Classes**: Define the object classes that must be available to run the configured report. If this attribute is set, the Alfabet Server checks whether the object classes specified as base

classes are excluded from the view scheme used to access the configured report. If the user does not have access permissions to one of the base classes, the configured report will be disabled. To set the attribute, click the **Browse** button to select the object classes that are required to run the configured report and click **OK**.

4) In the toolbar, click the **Save** button to save your changes.

Creating an Alfabet Configured Report That Opens an External Report

To define a configured report of the type `Extern`:

- Generate an external report independent of Alfabet by means of a reporting tool or script and make it available via a URL. If the external report shall display data from the Alfabet database or link to the Alfabet interface, consult the section *Accessing the Alfabet Database with External Applications* in the reference manual *System Administration* for details about the required interfaces with Alfabet.
- In the **Reports** explorer of the **Reports** tab of Alfabet Expand, create a new configured report and define the general attributes for the configured report and the attributes specific for external reports. This step is described below.
- Set the **Report State** of the configured report to `Active`. For information on how to change theattribute of the configured report, see.
- Define access permission for the configured report in the **Reports Administration** functionality of Alfabet. By default, access is possible for all users. For more information, see the chapter *Defining and Managing User Access to Configured Reports* in the reference manual *User and Solution Administration*.

To create a new configured report with a reference to an existing external report that was configured with an external reporting tool:

1) In the **Reports** tab, right-click the **Reports** folder or a sub-folder and select **Create New Report**. You will see the new configured report listed below the selected folder.

   As an alternative to defining all attributes for a new configured report, you can copy and paste an existing configured report and make modifications, as needed. To do so, right-click the existing configured report that you want to copy and select **Copy**. Right-click the new configured report and select **Paste**. Modify the necessary attributes described below.

2) In the attribute window, select `Extern` in the **Type** field.
3) In the attribute window, define the following attributes for the configured report:
   - **Name**: Optionally, change the technical name of the configured report. The technical name of the configured report must be unique.

     The technical name is used for identification of the configured report in the context of technical processes such as merging the report configuration with a report

configuration saved as an XML file. Therefore, the following is required when defining the name of a configured report:

- The name must be unique. The name must be unique regardless of the report folder it is structured in.
- The name may not be changed once a configured report has been created.
- A validation mechanism checks for correct syntax when defining a technical name. Please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.
- If the technical name of the configured report is changed, the name will be correctly updated in other configuration objects referencing the changed report during design time.

Please note that the name of a changed configuration object will not be updated in guide pages or guide views. If you plan to change the name of a configuration object, the reference in the navigation page should be changed prior to changing the configuration object name. You can use the **Show Usage** functionality in the context menu of the configured report to evaluate whether the configured report has been added to a guide view or guide page.

- **Caption**: Enter a caption for the external report that will be accessible via the URL definition. The name you define here will be displayed in the **Reports Administration** functionality and the **Configured Reports** views of the Alfabet interface. When the configured report is assigned to an object profile as a page view, the text is displayed by default as page view caption
- **Description**: Provide information about the external report that is useful to users. This comment will be displayed in the **Description** field for the external report in the **Reports Administration** functionality in the **Administration** application, the **Configured Reports** functionality. When the configured report is assigned to an object profile as a page view, the text is displayed beneath the page view caption as short description.
- **Report State**: The **Report State** attribute is view only and is set to `Plan` for new configured reports. The configured report can only be edited when the **Report State** attribute is set to `Plan`. After finishing all configuration steps described in the following, the **Report State** attribute must be set to `Active` as described in the section General Guidelines for Creating Configured Reports. The configured report is then visible to users in the Alfabet user interface.
- **URL**: Enter the URL targeting the location where the relevant external report is located.

  It is possible to specify part or all of the URL definition in the AlfabetMS.xml configuration file as server variable and reference the appropriate server variable in the URL definition of the external report. For information about how to use server variables see the section Using Server Variables in Web Link and Database Server-Related Specifications.

- **Help Index**: Specify the location of the external Help file using the following syntax:(For example,:). The external help file will be displayed when the user clicks the context-sensitive Help link in the standardHelp menu.

Server variables can be used in the Help links. Server variables allow you to define all or part of the URL definition in the server alias configuration of the Alfabet Server instead of directly defining it in the configuration in Alfabet Expand. Use of server variables is useful, For example, in order to change a Web server name defined in multiple URL definitions in Alfabet Expand. Instead of changing all URL definitions, only the variable value in the server alias configuration must be updated. The use of server variables is described in the section Using Server Variables in Web Link and Database Server-Related Specifications in the chapter Configuring Reports.

- **Selector Behavior**: The attribute can be used to exclude configured reports that are defined for special purposes, like For example, triggering of data export via the RESTful API, from being selectable in the **Configured Reports** table. By default, the attribute is set to `Visible` and the configured report is visible in the selector for adding configured reports to the **Configured Reports** table if all other conditions for access to the configured report apply. If the attribute is set to `NotVisible`, the configured report is excluded from the selector for adding configured reports to the **Configured Reports** functionality. Please note however, that this setting is not excluding the configured report from any standard views or customer configured views and selectors, but exclusively from the standard selector for configured reports implemented in Alfabet.

The attribute cannot be edited in the attribute window directly. To change the setting, right-click the configured report and select **Set Report Selector Behavior to 'Not Visible'** or **Set Report Selector Behavior to 'Visible'** respectively.

- **Applicable for AlfaBot**: Specify whether the configured report is accessible via the AlfaBot. By default, the attribute will be set to `True` for new reports. Set the attribute to `False` if the configured report should not be opened outside of a specific context. For example, this may be relevant for reports that are specifically designed to be embedded in an object cockpit.

  This attribute is only relevant if the AlfaBot is implemented. For information about the implementation of the AlfaBot, see Implementing the AlfaBot for Navigation via a Full-Text Command.

- **Business Problem Statement**: Optionally enter a description that will help users to search for the configured report in the AlfaBot. The text is not displayed in the user interface, but it is of special relevance for the search mechanisms implemented for the AlfaBot. If the report caption entered by the user in the AlfaBot for a request to open a configured report does not match the caption of one of the available configured reports, the AlfaBot will split the caption entered by the user into keywords and will perform a keyword search on the **Caption**, **Description** and the **Business Problem Statement** attributes of the configured reports that are accessible via the AlfaBot. In addition to a keyword search, a synonym search is performed on the text provided in the **Business Problem Statement** attribute.

  This attribute is only relevant if the AlfaBot is implemented. For information about the implementation of the AlfaBot, see Implementing the AlfaBot for Navigation via a Full-Text Command.

- **Base Classes**: Define the object classes that must be available to run the configured report. If this attribute is set, the Alfabet Server checks whether the object classes specified as base classes are excluded from the view scheme used to access the configured report. If the user does not have access permissions to one of the base classes, the configured report will be disabled. To set the attribute, click the **Browse** button to select the object classes that are required to run the configured report and click **OK**.

4) In the toolbar, click the **Save**  button to save your changes.

Creating Configured Reports That Are Containers for Multiple Configured Reports

You can define configured reports that are cockpits for display of multiple configured reports displaying data for the same context. There are two ways to define a configured report on basis of multiple existing configured reports:

- Console Reports

Console reports consist of multiple configured reports and optionally an attribute section. The attribute section is only applicable if the configured report is defined for a single base object.

On top of the configured reports, a filter can be defined that is valid for all configured reports subordinate to the console report.

- Cascading Reports

Cascading reports consist of multiple configured reports that refer to each other. A subordinate configured report can be defined as master of another subordinate configured report. When the user selects an object in the master report, the content of the subordinate configured report displays data about the selected object.

When the configured reports that shall be displayed in a single configured report are all about a defined object, and no further filter is required, you can define an object cockpit for the object profile of the object class the objects belong to instead of defining a console or cascading report. For more information about defining object cockpits, see Configuring Object Cockpits for a Custom Object View.

Creating Console Reports

A console report displays an object cockpit with a filter area. Filter settings done in the filter area of the console report can be applied to all configured reports in the object cockpit.

To generate a console report that displays an object cockpit with a filter:

1) In the **Reports** tab, right-click the **Reports** folder or a sub-folder thereof and select **Create New Report**. You will see the new configured report listed below the selected folder.

As an alternative to defining all attributes for a new configured report, you can copy and paste an existing configured report and make modifications, as needed. To do so, right-click the existing configured report that you want to copy and select **Copy**. Right-click the new configured report and select **Paste**. Modify the necessary attributes described below.

2) In the attribute window, select `Custom` in the **Type** field.

3) The following attribute settings are required to create a valid console report:

- **Base Object Query**: For a console report with a filter that is applied to all subordinate configured reports assigned to the console report, the base object query must find the `UserGlobalData` instance that is assigned to the current user. You can use the following queries.

In native SQL:

```
SELECT REFSTR, USERREF
FROM USERGLOBALDATA
WHERE USERREF = @CURRENT_USER
```

In Alfabet query language:

```
ALFABET_QUERY_500
FIND UserGlobalData
WHERE UserGlobalData.USERREF = @CURRENT_USER
```

- **Template**: This attribute must be left empty.

4) Optionally, you can define the following attributes for the configured report:

- **Name**: Optionally, change the technical name of the configured report. The technical name of the configured report must be unique.

  The technical name is used for identification of the configured report in the context of technical processes such as merging the report configuration with a report configuration saved as an XML file. Therefore, the following is required when defining the name of a configured report:

  - The name must be unique. The name must be unique regardless of the report folder it is structured in.
  - The name may not be changed once a configured report has been created.
  - A validation mechanism checks for correct syntax when defining a technical name. Please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.
  - If the technical name of the configured report is changed, the name will be correctly updated in other configuration objects referencing the changed report during design time.

  Please note that the name of a changed configuration object will not be updated in guide pages or guide views. If you plan to change the name of a configuration object, the reference in the navigation page should be changed prior to changing the configuration object name. You can use the **Show Usage** functionality in the context menu of the configured report to evaluate whether the configured report has been added to a guide view or guide page.

- **Caption**: Enter a caption for the configured report. The caption you define here will be displayed in the **Reports Administration** functionality in the **Administration** module and the **Configured Reports** views of the Alfabet interface. If the configured report is assigned to an object view as a page view, the text will be displayed as the page view caption in the object view. The caption of the configured report may exceed the conventional 64 character limit.

- **Description**: Provide a short information about the configured report that is useful to Alfabet users. This comment will be displayed on the Alfabet user interface in the header of the configured report in a single line under the report caption and the **Description** field for the configured report in the table of all views listing configured reports, like the **Configured Reports** views of the **Search** functionalities. If the configured report is assigned to an object view as a page view, the text will be displayed under the page view caption as short description in the object view.

  In the configured report, the description will be truncated if it is longer than one line on the screen. Therefore, the description should be short to ensure complete display in the configured report header.

- **Usage Guideline**: Provide information that helps users to execute and interpret the configured report. This information will not be displayed directly on the user interface. The user can access the information using the following mechanisms:

  - If the user moves the cursor over the description provided for the configured report with the attribute **Description**, a tooltip opens that includes both the description and the usage guideline.
  - The **Options** button will be displayed on the upper right with an option **Help On Filter Fields** to open a new window displaying the usage guideline in addition to any filter field hints, if configured.

    For views that have filters defined, the **Options** button will only be displayed if the filter panel either allows batch clearing of filter fields or collapsing and expanding of the filter panel. For more information about the configuration of these functionalities, see Configuring the Complete Filter Area to be Collapsible and Allowing the User to Clear the Filter Area.

- **Technical Comment**: Provide information that is useful for solution designers working with Alfabet Expand in order to better understand the configured report from a technical perspective. The **Technical Comment** will not be displayed in the user interface. Nevertheless, you can configure the interface to display the information in, For example, the attribute section of the configured report's object profile in the **Reports Administration** functionality.
- **Report State**: The **Report State** attribute is view only and is set to `Plan` for new configured reports. The configured report can only be edited when the **Report State** attribute is set to `Plan`. After finishing all configuration steps described in the following, the **Report State** attribute must be set to `Active` as described in the section General Guidelines for Creating Configured Reports. The configured report is then visible to users in the Alfabet user interface.
- **Help Index**: Specify the location of the external Help file using the following syntax:(For example,:). The external help file will be displayed when the user clicks the context-sensitive Help link in the standardHelp menu.

  Server variables can be used in the Help links. Server variables allow you to define all or part of the URL definition in the server alias configuration of the Alfabet Server instead of directly defining it in the configuration in Alfabet Expand. Use of server variables is useful, For example, in order to change a Web server name defined in multiple URL definitions in Alfabet Expand. Instead of changing all URL definitions,

only the variable value in the server alias configuration must be updated. The use of server variables is described in the section Using Server Variables in Web Link and Database Server-Related Specifications in the chapter Configuring Reports.

- **Selector Behavior**: The attribute can be used to exclude configured reports that are defined for special purposes, like For example, triggering of data export via the RESTful API, from being selectable in the **Configured Reports** table. By default, the attribute is set to `Visible` and the configured report is visible in the selector for adding configured reports to the **Configured Reports** table if all other conditions for access to the configured report apply. If the attribute is set to `NotVisible`, the configured report is excluded from the selector for adding configured reports to the **Configured Reports** functionality. Please note however, that this setting is not excluding the configured report from any standard views or customer configured views and selectors, but exclusively from the standard selector for configured reports implemented in Alfabet.

The attribute cannot be edited in the attribute window directly. To change the setting, right-click the configured report and select **Set Report Selector Behavior to 'Not Visible'** or **Set Report Selector Behavior to 'Visible'** respectively.

- **Applicable for AlfaBot**: Specify whether the configured report is accessible via the AlfaBot. By default, the attribute will be set to `True` for new reports. Set the attribute to `False` if the configured report should not be opened outside of a specific context. For example, this may be relevant for reports that are specifically designed to be embedded in an object cockpit.

  This attribute is only relevant if the AlfaBot is implemented. For information about the implementation of the AlfaBot, see Implementing the AlfaBot for Navigation via a Full-Text Command.

- **Business Problem Statement**: Optionally enter a description that will help users to search for the configured report in the AlfaBot. The text is not displayed in the user interface, but it is of special relevance for the search mechanisms implemented for the AlfaBot. If the report caption entered by the user in the AlfaBot for a request to open a configured report does not match the caption of one of the available configured reports, the AlfaBot will split the caption entered by the user into keywords and will perform a keyword search on the **Caption**, **Description** and the **Business Problem Statement** attributes of the configured reports that are accessible via the AlfaBot. In addition to a keyword search, a synonym search is performed on the text provided in the **Business Problem Statement** attribute.

  This attribute is only relevant if the AlfaBot is implemented. For information about the implementation of the AlfaBot, see Implementing the AlfaBot for Navigation via a Full-Text Command.

- **Base Classes**: Define the object classes that must be available to run the configured report. If this attribute is set, the Alfabet Server checks whether the object classes specified as base classes are excluded from the view scheme used to access the configured report. If the user does not have access permissions to one of the base classes, the configured report will be disabled. To set the attribute, click the **Browse** button to select the object classes that are required to run the configured report and click **OK**.

- **Can Create Express View**: Select `True` if an express view may be created for the report. Select `False` if an express view may not be created for the report. Please note that for reports that have a custom report view, the setting must be consistent with the setting of the attribute **Can Create Express View** of the custom report view.

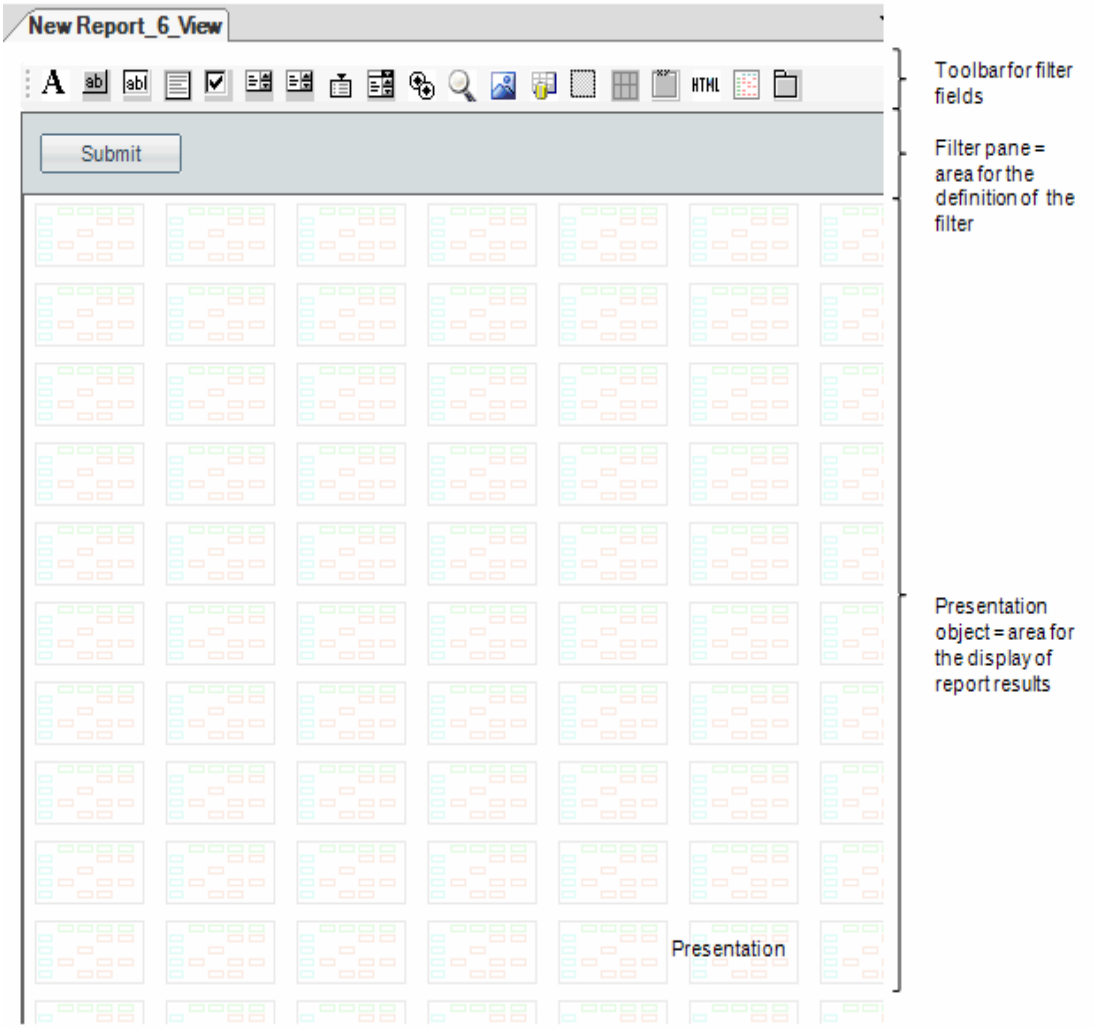
An express view is a link to a specific Alfabet view displaying data that allows individuals inside or outside of the user community to view information in Alfabet. When the express view is created, an email notification is automatically mailed to a specified recipient. The recipient receives a URL that allows him/her to access the current page view in Alfabet. For more information, see the section *Sending and Receiving Express Views* in the reference manual *Getting Started with Alfabet*.

5) In the explorer, right-click the configured report and select **Create Custom Report View**. The view editor opens. In the explorer, the custom report view is displayed as a subordinate object of the configured report. The attributes for the custom report view are displayed in the attribute window. You can optionally edit the following attributes:

- **Name**: Optionally you can change the name for the custom report view. The **Name** must be unique for custom report views.
- **Can Create Bookmark**: Select `True` if a bookmark may be created for the configured report. Select `False` if a bookmark may not be created for the configured report.
- **Can Create Express View**: Select `True` if an express view may be created for the configured report. Select `False` if an express view may not be created for the configured report. Please note that the setting must be consistent with the setting of the attribute **Can Create Express View** of the configured report.

An express view is a link to a specific Alfabet view displaying data that allows individuals inside or outside of the user community to view Alfabet information. When the express view is created, an e-mail notification is automatically mailed to the specified recipient who receives a URL that allows him/her to access the current page view via an Alfabet Web Client. For more information, see *Sending and Receiving Express Views* in the reference manual *Getting Started with Alfabet*.

6) In the view editor, define the filter in the filter panel on top of the panel named ***Presentation***.



The filter must be defined to store data into a custom property of the object class `UserGlobal-Data`. Follow the configuration steps to configure reusable filter settings described in the section Configuring Reusability of Filter Settings Across Reports in the Chapter Defining Queries.

For general information about the definition of the filter panel and the filter fields in the panel, see Defining Filters for Configured Reports and Selectors in the chapter Defining Queries.

7) In the view editor, click inside the presentation object named ***Presentation***.

8) Right-click inside the view and select **Delete** . The ***Presentation*** pane is deleted.

9) In the toolbar of the view editor, click the **Flow Panel** or **Table Layout Panel** button and click in the custom report view pane where you want to place it. The element is added to the pane.

It is recommended to use the **Flow Panel** design which has the advantage that object cockpits are not squeezed, stretched, or truncated when the interface is resized. Group boxes containing attributes and presentation objects (views and reports) will be dynamically positioned according to the current width of the Alfabet Web interface. When resizing the interface, the content displayed in object cockpits will be redistributed in the available space and the content will not be cut-off. The flow panel ensures that a configured view or set of views will be bumped down and placed below the preceding view or

set of views. Business graphics embedded in an object cockpit will be automatically scaled down in size to fit in the available screen space of the device the user is using.

Console reports rendered in static **Table Layout Panel** interface control that does not allow for dynamic positioning of the interface controls based on the width of the software interface. This method is available for reasons of backward compatibility however it is not recommended to use the **Table Layout Panel** interface control because the data in the object cockpit may be squeezed, stretched, or truncated when the interface is resized.

10) Design the object cockpits by adding elements as described in the section Configuring the Object Cockpit By Means of a Flow Panel or Configuring the Object Cockpit By Means of a Table Grid in the chapter Configuring Object Views.

The following special requirements apply to the configuration of object cockpits in a console report:

- Only configured reports and standard Alfabet views can be added to the pane. Presentation objects cannot be added to a console report.
- After having added a configured report to the cockpit, click the presentation object of the configured report in the pane, and in the attribute window, set the **Refresh on Submit** attribute of the presentation object to `True`. This setting is required to update the configured report on submit of the filter settings.
- If **Value Control** elements of the **Sub Type** `Property` are added to the console report, the **Refresh On Submit** attribute of the controls must be set to `True` to update values on submit of the filter settings.

11) In the toolbar, click the **Save**  button to save your changes.

12) In the explorer, right-click the console report and select **Review Report**. The console report opens in a web browser. If the results are not as expected, you can edit the console report until the output of the console report meets your expectations.

The **Review Report** functionality also allows the visibility of toolbar buttons of the configured report to be defined for different view schemes. For more information, see Refining Visibility Issues in the View Scheme.

After having created the configured report, you can define access rights for the configured report in the **Reports Administration** functionality of Alfabet. By default, access is possible for all users. For more information, see the chapter *Defining and Managing User Access to Configured Reports* in the reference manual *User and Solution Administration*.

Creating Cascading Reports

In a cascading report, at least two configured reports are added to the custom report view and one of the configured reports is defined to display data about a single object in the other configured report when a user clicks the object in the configured report.

To define a cascading report with one master report and a second configured report subordinate to the master report, you must define three configured reports, the master report, the subordinate configured report and the cascading report that combines the two configured reports.

1) Define the master report. The master report can be any configured report of the type `Query`, `NativeSQL` or `Custom`.

   For information about the definition of the configured report see the sections Creating a Tabular Configured Report of the Type Query, Creating a Tabular Configured Report of the Type NativeSQL,Creating a Graphic Report.

2) Define the subordinate configured report. The subordinate configured report can be any configured report of the type `Query`, `NativeSQL` or `Custom`. Define the dependency with the master report by using the Alfabet parameter `MASTEROBJECT` in at least one `WHERE` clause of the query that specifies the content of the subordinate configured report. The Alfabet parameter `MASTEROBJECT` returns the `REFSTR` of the object that the user currently selected in the master report during runtime.

   For general information about the definition of the configured report see the sections Creating a Tabular Configured Report of the Type Query, Creating a Tabular Configured Report of the Type NativeSQL, Creating a Graphic Report.

   For general information about the use of Alfabet parameters in queries, see Referring to the Current Alfabet Context in a WHERE Condition and Using Alfabet Parameters in Native SQL Queries in the chapter Defining Queries.

3) Set the **Report State** attribute of both master and subordinate configured report to `Active` before performing the next steps.

4) To define the cascading report that combines the master report with the subordinate configured report, right-click the **Reports** folder or a sub-folder thereof and select **Create New Report**. You will see the new configured report listed below the selected folder.

5) In the attribute window, select `Custom` in the **Type** field.

6) Optionally, you can define the following attributes for the configured report:

   - **Name**: Optionally, change the technical name of the configured report. The technical name of the configured report must be unique.

     The technical name is used for identification of the configured report in the context of technical processes such as merging the report configuration with a report configuration saved as an XML file. Therefore, the following is required when defining the name of a configured report:

     - The name must be unique. The name must be unique regardless of the report folder it is structured in.
     - The name may not be changed once a configured report has been created.
     - A validation mechanism checks for correct syntax when defining a technical name. Please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.

- If the technical name of the configured report is changed, the name will be correctly updated in other configuration objects referencing the changed report during design time.

Please note that the name of a changed configuration object will not be updated in guide pages or guide views. If you plan to change the name of a configuration object, the reference in the navigation page should be changed prior to changing the configuration object name. You can use the **Show Usage** functionality in the context menu of the configured report to evaluate whether the configured report has been added to a guide view or guide page.

- **Caption**: Enter a caption for the configured report. The caption you define here will be displayed in the **Reports Administration** functionality in the **Administration** module and the **Configured Reports** views of the Alfabet interface. If the configured report is assigned to an object view as a page view, the text will be displayed as the page view caption in the object view. The caption of the configured report may exceed the conventional 64 character limit.
- **Description**: Provide a short information about the configured report that is useful to Alfabet users. This comment will be displayed on the Alfabet user interface in the header of the configured report in a single line under the report caption and the **Description** field for the configured report in the table of all views listing configured reports, like the **Configured Reports** views of the **Search** functionalities. If the configured report is assigned to an object view as a page view, the text will be displayed under the page view caption as short description in the object view.

  In the configured report, the description will be truncated if it is longer than one line on the screen. Therefore, the description should be short to ensure complete display in the configured report header.

- **Usage Guideline**: Provide information that helps users to execute and interpret the configured report. This information will not be displayed directly on the user interface. The user can access the information using the following mechanisms:
  - If the user moves the cursor over the description provided for the configured report with the attribute **Description**, a tooltip opens that includes both the description and the usage guideline.
  - The **Options** button will be displayed on the upper right with an option **Help On Filter Fields** to open a new window displaying the usage guideline in addition to any filter field hints, if configured.

    For views that have filters defined, the **Options** button will only be displayed if the filter panel either allows batch clearing of filter fields or collapsing and expanding of the filter panel. For more information about the configuration of these functionalities, see Configuring the Complete Filter Area to be Collapsible and Allowing the User to Clear the Filter Area.

- **Technical Comment**: Provide information that is useful for solution designers working with Alfabet Expand in order to better understand the configured report from a technical perspective. The **Technical Comment** will not be displayed in the user interface. Nevertheless, you can configure the interface to display the information in, For example, the attribute section of the configured report's object profile in the **Reports Administration** functionality.

- **Report State**: The **Report State** attribute is view only and is set to `Plan` for new configured reports. The configured report can only be edited when the **Report State** attribute is set to `Plan`. After finishing all configuration steps described in the following, the **Report State** attribute must be set to `Active` as described in the section General Guidelines for Creating Configured Reports. The configured report is then visible to users in the Alfabet user interface.

- **Apply to Class**: When the configured report is applied to an object class or object class stereotype, a base object can be selected for the configured report. A filter for selection of a base object will automatically be set on top of the configured report. When configuring queries for the configured report, you can use the Alfabet parameter `BASE` to refer to the object selected in the filter and define search conditions dependent on the base object. To apply the configured report to a base class, click the **Browse** button to open a dialog box and select the object class to which you want to apply the configured report and click **OK**

  Note the following:

  - If the configured report is assigned to the object view of an object of the class it is applied to, the configured report will automatically open with the current object selected as base object, even if **Apply to Class** is not set.
  - If the configured report is applied to an object class stereotype, the filter field for selection of the base object will use the caption of the object class stereotype as caption of the filter field and the selector defined in the class settings for the object class stereotype for selection of the base object.
  - Alternatively, the base object of a configured report can be set with the attribute **Base Query**. While the attribute **Apply to Class** enables the user to determine the current base object by setting a filter, the **Base Query** evaluates the base object by execution of a query when opening the configured report. A filter field is not available and the user cannot change the base object.
  - The parameter **Apply to Class** also influences the availability of the configured report on the Alfabet interface. Select a class to make the configured report available in the **Configured Reports** page view on the object view of objects of the selected object class. If you do not set the attribute, the configured report is only available in the **Configured Reports** functionality. The configured report can be assigned to object views and wizards independent from the parameter.

- **Execute on Enter**: Set to `True` to execute the configured report with empty filter settings when the user opens the configured report on the Alfabet interface. Set to `False` to open the configured report without execution of the Alfabet query with empty filter settings. By default, this attribute is set to `True`. It should only be set to `False` when a configured report would result in an exceedingly big dataset when executed with empty filter settings.

  When you set **Execute on Enter** to `False`, you must make sure that the **Submit** button is available in the custom report view of the configured report. If the **Submit** button is deleted from the custom report view, the configured report cannot be displayed. The setting and execution of the **Execute on Enter** attribute is independent from the definition of filters for a configured report. A configured report without

filters must nevertheless have a **Submit** button when **Execute on Enter** is set to `False`.

- **Help Index**: Specify the location of the external Help file using the following syntax:(For example,:). The external help file will be displayed when the user clicks the context-sensitive Help link in the standardHelp menu.

  Server variables can be used in the Help links. Server variables allow you to define all or part of the URL definition in the server alias configuration of the Alfabet Server instead of directly defining it in the configuration in Alfabet Expand. Use of server variables is useful, For example, in order to change a Web server name defined in multiple URL definitions in Alfabet Expand. Instead of changing all URL definitions, only the variable value in the server alias configuration must be updated. The use of server variables is described in the section Using Server Variables in Web Link and Database Server-Related Specifications in the chapter Configuring Reports.

- **Selector Behavior**: The attribute can be used to exclude configured reports that are defined for special purposes, like For example, triggering of data export via the RESTful API, from being selectable in the **Configured Reports** table. By default, the attribute is set to `Visible` and the configured report is visible in the selector for adding configured reports to the **Configured Reports** table if all other conditions for access to the configured report apply. If the attribute is set to `NotVisible`, the configured report is excluded from the selector for adding configured reports to the **Configured Reports** functionality. Please note however, that this setting is not excluding the configured report from any standard views or customer configured views and selectors, but exclusively from the standard selector for configured reports implemented in Alfabet.

The attribute cannot be edited in the attribute window directly. To change the setting, right-click the configured report and select **Set Report Selector Behavior to 'Not Visible'** or **Set Report Selector Behavior to 'Visible'** respectively.

- **Base Classes**: Define the object classes that must be available to run the configured report. If this attribute is set, the Alfabet Server checks whether the object classes specified as base classes are excluded from the view scheme used to access the configured report. If the user does not have access permissions to one of the base classes, the configured report will be disabled. To set the attribute, click the **Browse** button to select the object classes that are required to run the configured report and click **OK**.
- **Can Create Express View**: Select `True` if an express view may be created for the report. Select `False` if an express view may not be created for the report. Please note that for reports that have a custom report view, the setting must be consistent with the setting of the attribute **Can Create Express View** of the custom report view.

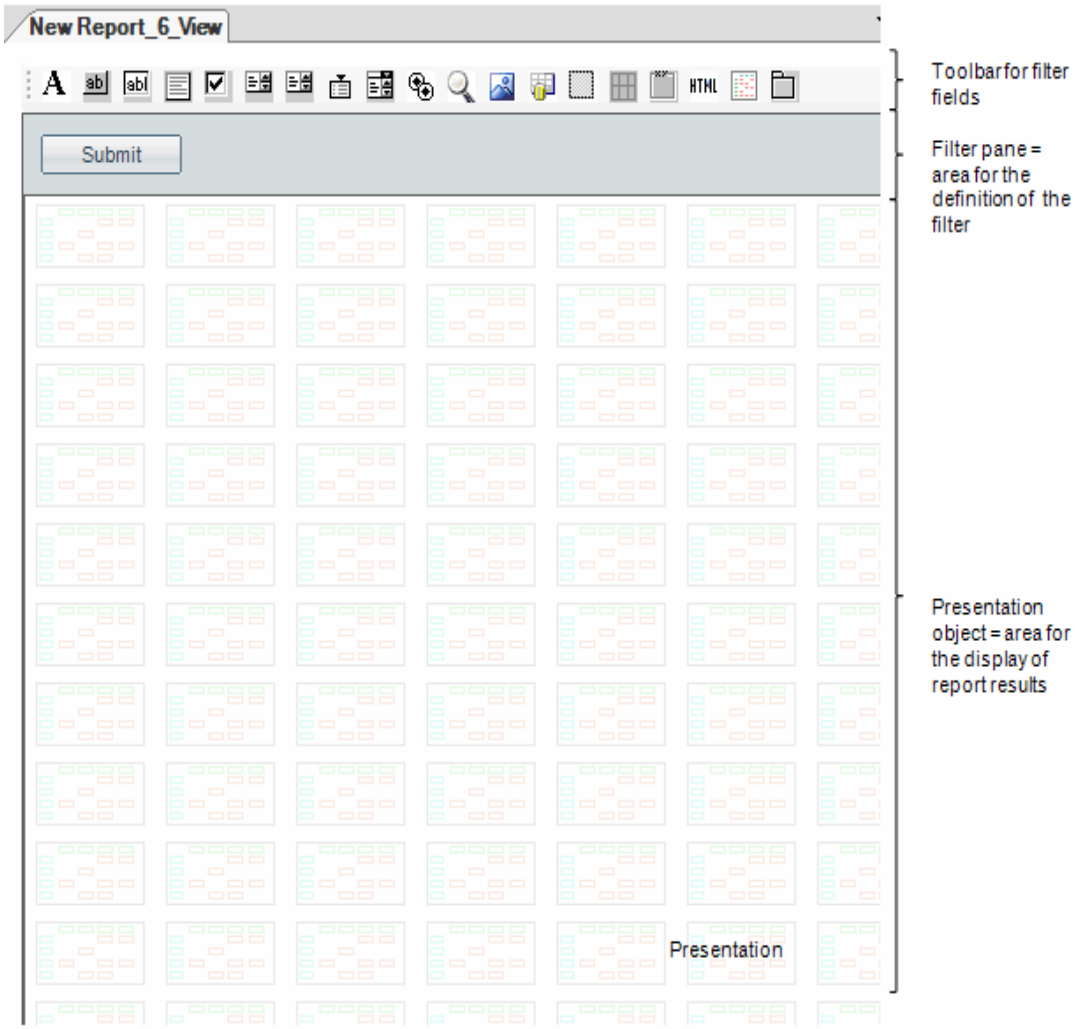
  An express view is a link to a specific Alfabet view displaying data that allows individuals inside or outside of the user community to view information in Alfabet. When the express view is created, an email notification is automatically mailed to a specified recipient. The recipient receives a URL that allows him/her to access the current page view in Alfabet. For more information, see the section *Sending and Receiving Express Views* in the reference manual *Getting Started with Alfabet*.

7) In the explorer, right-click the configured report and select **Create Custom Report View**. The view editor opens. In the explorer, the custom report view is displayed as a subordinate object of the configured report. The attributes for the custom report view are displayed in the attribute window. You can optionally edit the following attributes:

- **Name**: Optionally you can change the name for the custom report view. The **Name** must be unique for custom report views.
- **Can Create Bookmark**: Select `True` if a bookmark may be created for the configured report. Select `False` if a bookmark may not be created for the configured report.
- **Can Create Express View**: Select `True` if an express view may be created for the configured report. Select `False` if an express view may not be created for the configured report. Please note that the setting must be consistent with the setting of the attribute **Can Create Express View** of the configured report.

  An express view is a link to a specific Alfabet view displaying data that allows individuals inside or outside of the user community to view Alfabet information. When the express view is created, an e-mail notification is automatically mailed to the specified recipient who receives a URL that allows him/her to access the current page view via an Alfabet Web Client. For more information, see *Sending and Receiving Express Views* in the reference manual *Getting Started with Alfabet*.

8) In the view editor, either define a filter in the filter panel on top of the panel named ***Presentation*** or delete the filter pane if no filter is required and the attribute **Execute on Enter** of the configured report is not set to `False`.



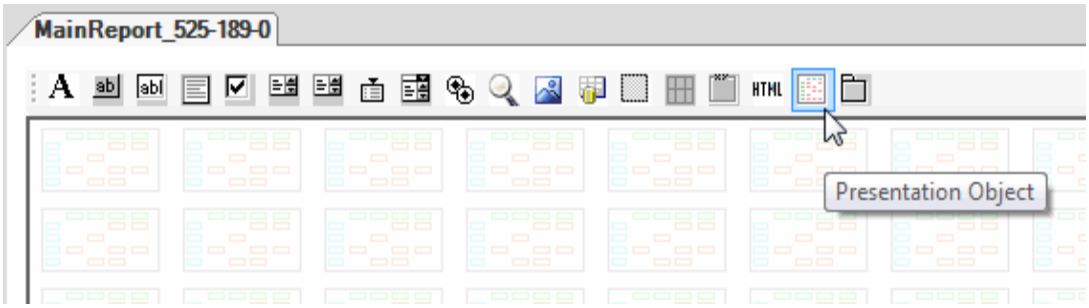
For general information about the definition of the filter panel and the filter fields in the panel, see Defining Filters for Configured Reports and Selectors in the chapter Defining Queries.

9) In the view editor, click inside the presentation object named ***Presentation***.

10) In the attribute window, set the attribute **Dock** of the presentation object to `Top`.

11) Alter the size and position of the presentation object to the size the master report should have in the custom report view. The size and position can be altered using the **Coordinates** attributes in the attribute window or by directly resizing and moving the graphic object in the custom report view.

12) In the toolbar, click the **Presentation Object** button, and then click on the position where the presentation object should be located in the custom report view. The presentation object is added to the custom report view and can be resized and moved to the correct position.

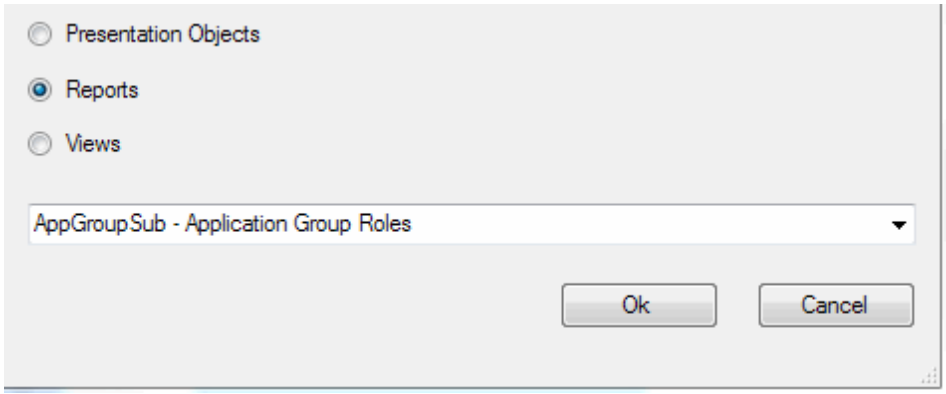


13) In the attribute window, set the attribute **Dock** of the presentation object to `Fill`.

A validation mechanism checks for inconsistencies in the layout and docking definition of interface controls in guide views, configured reports, and other configured views. Please note that an error message will be displayed when the user attempts to open an incorrectly configured view if two or more interface controls have no docking defined in the view. The error message will provide detailed information about the faulty configuration.

14) For both presentation objects, set the following attributes in the attribute window that opens when you click on the presentation object:

- **Name**: The name of the presentation object of the master report is used instead of the configured report name to set the master in the subordinate configured reports. The name of the presentation object should be altered to a string that makes it easy to identify the underlying report configuration. The name is a technical name and shall not contain whitespaces or special characters.
- **Refresh on Submit**: Select `True`.
- **Sub Type**: Select `View`.
- **Source**: Click the button on the right of the field. A selector window opens. Select configured reports and select the configured report that shall be displayed in the drop-down list of available configured reports. Click **OK** to save your selection.



15) For the presentation object of the subordinate configured report, set the following attribute:

- **Master Control**: Click the button on the right of the field. A selector window opens. Select the checkbox of the presentation object containing the master report. Click **OK** to save your selection.

16) In the toolbar, click the **Save** button to save your changes.

17) Right-click the configured report and select **Review Report**. The configured report opens in a Web browser. If the result is not as expected, alter the settings for your configured reports until the configured report is correctly displayed.

The **Review Report** functionality also allows the visibility of toolbar buttons of the configured report to be defined for different view schemes. For more information, see Refining Visibility Issues in the View Scheme.

18) Set the **Report State** attribute of the configured report to `Active`.

After having created the configured report, you can define access rights for the configured report in the **Reports Administration** functionality of Alfabet. By default, access is possible for all users. For more information, see the chapter *Defining and Managing User Access to Configured Reports* in the reference manual *User and Solution Administration*.

Testing Configured Reports

You can test the outcome of your report configuration by opening the configured report directly from Alfabet Expand:

Access of Alfabet Expand to a running Alfabet Web Application must be configured in the server alias of Alfabet Expand to use this feature. For information about the configuration of the server alias, see the section *Installation and Configuration of Alfabet Expand and the Guide Pages Designer* in the reference manual *System Administration*.

1) Prior to using the test functionality, all current changes to the meta-model must be saved to the Alfabet database. If applicable, click the **Save** button to save all changes.

2) In the explorer of the **Reports** tab, right click the configured report that you want to test and select **Review Report** from the context menu.

The configured report opens in a web interface rendered by a running Alfabet Web Application.

Please note the following about the test view for configured reports opened via Alfabet Expand:

- No menu buttons are displayed.
- Navigation from cells or graphic elements in a configured report to views or editors in Alfabet is disabled. These configurations cannot be tested via the **Review Report** function.
- For performance reasons, the web server used to render the report for the **Review Report** function caches part of the meta-model information. If you use the function, change the report and re-use the function within a short time interval, de-synchronization of the cached data and the content stored in the Alfabet database might occur and you might see a message informing you that the design page is not valid any more. If this message is displayed, close the window and re-use the **Review Report** option in the context menu of the configured report to repeat the test. The view then opens with the correct content.

Managing and Structuring Your Configured Reports in Folders

You can organize your configured reports in folders and sub-folders. A report folder can have multiple sub-folders and multiple levels of sub-folders. A configured report can be assigned to any root level folder or sub-folder.

The hierarchy that you define in Alfabet Expand in the **Reports** explorer is also used to present the configured reports to users in the **Configured Reports** functionality and the **Configured Reports** page view in object profiles. Therefore, the names of report folders and the structure should be carefully considered so that they are easily understandable for relevant users.

Creating a New Report Folder

To create a new report folder:

1) In the **Reports** explorer, right-click the root node to create a new folder at the top level of the hierarchy or right-click an existing report folder to add a sub-folder to that report folder.
2) Select **Create New Folder**. The new report folder is added to the explorer tree with a default name.
3) Click the new report folder to open the attribute window. Define the following attributes:
   - **Name**: Enter a unique name for the report folder. This name is used to identify the report folder in technical processes, For example, when merging two configurations.
   - **Caption**: Enter a caption for the report folder. This caption is displayed to the users in the **Configured Reports** view for the report folder.
   - **Description**: Enter a meaningful description that will clarify the purpose of the report folder for the users looking for a configured report.
   - **Technical Comment**: Provide information that is useful for solution designers working with Alfabet Expand. The **Technical Comment** will not be displayed on the user interface. Nevertheless, you can configure the interface to display the information For example, in the attribute section of the report folder's object profile in the **Reports Administration** functionality.

4) Click **OK** to save your report folder or **Cancel** to exit without saving the data.

You will see the new report folder in the table with a + or - symbol next to it. You can now add configured reports to the folder or create additional levels of sub-folders to the reports folder.

Moving Configured Reports and Report Folders in the Hierarchy

You can move an existing report folder or configured report in the hierarchy via drag and drop.

You can only move a report folder or configured report to another report folder. It is not possible to change the order of the report folders within one report folder using drag and drop or copy and paste.

To move a configured report or report folder using drag and drop:

1) In the **Reports** explorer tree of the **Reports** tab, click the configured report or report folder that you want to move and drag it to the new location.

Adding Existing Configured Reports to Report Folders

You can move an existing configured report in the hierarchy using cut and paste or copy an existing configured report to a new location using copy and paste:

1) In the **Reports** explorer tree of the **Reports** tab, right-click the configured report that you want to copy or move and select
   - **Copy** to make a copy of the configured report. The original configured report will remain at the old location.
   - **Cut** to remove the configured report from the old location.
2) Right-click the report folder that you want to add the configured report to and select **Paste**.
3) Right-click the **Reports** root folder and select **Rescan Tree**. If you used cut and paste, the configured report is now removed from the old location and only displayed at the new location.

Deleting a Configured Report or Report Folder

You can delete configured reports that are in the **Report State** `Plan` or `Retired`. It is not possible to delete configured reports in the **Report State** `Active`. If you want to delete a report folder, all configured reports in the folder must have the **Report State** attribute set to `Plan` or `Retired`.

If you delete a configured report or a report folder, the configured report or report folder will be irrevocably deleted from the Alfabet database.

If you delete a report folder that contains configured reports or sub-folders, the configured reports and sub-folders are also deleted irrevocably from the Alfabet database.

To delete an existing configured report or report folder:

1) In the **Reports** explorer in the **Reports** tab, right-click the configured report or report folder that you want to delete and select **Delete**.

2) Confirm the warning by clicking **Yes**, or click **No** to exit without saving your changes.

Alfabet Expand also offers mechanisms to batch delete configured reports on the root level. You can either delete all report folders and their configured reports and sub-folders or delete only configured reports that are created directly at the root level. This mechanism is useful, For example, to manage Ad-Hoc reports that are created to answer one-time requests for information. If all configured reports created for long-term availability on the Alfabet interface are stored in report folders, and Ad-Hoc reports are stored on the root level of the **Reports** tree, the Ad-Hoc reports can be batch deleted without affecting the long-term configured reports.

To batch delete configured reports or report folders on the root level:

1) In the **Reports** explorer in the **Reports** tab, right-click the **Reports** root folder and select one of the following:
   - **Delete All Root Reports** to delete all configured reports that are located directly under the root level folder.
   - **Delete All Folders** to delete all report folders and their sub-folders and configured reports.
2) Confirm the warning by clicking **Yes,** or click **No** to exit without saving your changes.

Changing the Settings for all Configured Reports in a Report Folder on the Report Folder Level

The context menu of the report folder contains options that are also available in the context menu of the configured report. If you select one of these context menu options on the report folder level, the change is performed for all configured reports that are located in the report folder or sub-folders thereof. Please note that changes are applied independent of the setting of the attribute **Report State** of the configured report.

The following option of the configured report can be set for all reports in a report folder:

- Changing the attribute **Report State** of the configured reports subordinate to the report folder.
- Changing the attribute **Selector Behavior** of the configured reports subordinate to the report folder.
- Updating the translation changed for the description and caption of the configured reports subordinate to the configured report folder in the Alfabet database.

Once the **Caption** and **Description** attributes have been specified for a configured report, they will be automatically added to the vocabularies available in Alfabet Expand. Once they are available in the vocabularies, they can be translated like any other string in the **Translation Editor** or exported to a Microsoft Excel file and translated there. For more information, see the section Modifying, Translating and Managing the Vocabularies in the chapter Localization and Multi-Language Support for the Alfabet Interface.

For technical reasons, the translation of report captions and descriptions is coupled with the mechanisms of object data translation. Therefore, the translation provided via the vocabularies is only displayed in the Alfabet interface if data translation is enabled for the object class `ALFA_REPORT` and the option **Update Translation** in the context menu of the configured report was used.

Understanding Private Report Folders

A number of private report folders is available in the **Reports** explorer of any Alfabet installation. The content of these folders is provided by Software AG for storage of preconfigured or automatically generated report required for the following functionalities:

Private report folder	Supported functionality
AlfaBotReports	This folder contains preconfigured reports specifying the configured reports that are included into the faceted search for the Analyze intent of the AlfaBot. For basic information about the `Analyze` intent, see *Using the AlfaBot User Assistance* in the reference manual *Getting Started with Alfabet*. For more information about the configuration required to use the functionality, see Implementing the AlfaBot for Navigation via a Full-Text Command.
Analysis Intent Ad-Hoc Reports	This folder contains reports that are automatically generated to answer questions users are entering into the faceted search view of the `Analyze` intent of the AlfaBot. For basic information about the `Analyze` intent, see *Using the AlfaBot User Assistance* in the reference manual *Getting Started with Alfabet*. For more information about the configuration required to use the functionality, see Implementing the AlfaBot for Navigation via a Full-Text Command. Please note that this folder may contain a high number of reports and shall be cleaned up in regular intervals. The cleanup process is part of the configuration described for implementing the AlfaBot.
DataImport-IntoARIS	This folder contains preconfigured reports for data export from Alfabet to ARIS via the ARIS - Alfabet Interoperability Interface. For more information, see *Configuring Data Export from Alfabet* in the reference manual *ARIS - Alfabet Interoperability*.

You cannot edit the content of private folders.

Integration of Configured Reports in the Alfabet Interface

There are two concepts that allow the integration of configured reports in the Alfabet interface:

- Configured reports can be displayed separate from the standard reports in Alfabet so that they are clearly identifiable as configured reports. They can then be accessed by the user in the standard **Configured Reports** functionality and/or in the **Configured Reports** page view available in the object profile of selected object classes.
- Standard and configured reports can be included in custom object views or object cockpits, wizards and workflows. Hyperlinks to standard and configured reports can be implemented in guide pages. No distinction between a standard or configured report in the guide page, object profile, workflow step or wizard view is visible to the user.

Only configured reports for which the **Report State** attribute is set to `Active` can be executed in the Alfabet interface. When the **Report State** attribute is reset to `Plan` or `Retired`, the user will not be able to access the configured report via a link from a guide page, an object profile, workflow step or wizard view. A respective error message will inform the user that the configured report is currently not available.

Display of Configured Reports in a Separate Functionality

Users can access configured reports in the Alfabet interface either in the **Configured Reports** functionality or in the **Configured Reports** page view available in the relevant object view. Both views provide a table that displays the configured reports structured in folders. These report folders are also created in Alfabet Expand. The user will see the configured reports exactly as structured in the Alfabet Expand explorer.

The table describes the type of configured report and displays the name and the description specified for the configured report. To view the configured report, the user must select a configured report in the table and click the **Go to Report** button in the toolbar. If the configured report is an external report, the specified URL of the report will open in a new window. All other types of configured reports open directly in the Alfabet interface.

User access to a configured report in the **Configured Reports** view will depend on the following:

- **The definition of the Report State attribute of the report.**

Configured reports can have the **Report State** attribute set to `Plan`, `Active` or `Retired`. Configured reports with the **Report State** attribute set to `Plan` are not included in the table. This state is reserved for configured reports that are currently being configured. Only configured reports with the **Report State** attribute set to `Active` will be available to users. Configured reports that are obsolete are set to the state `Retired`. These configured reports are displayed in red in the **Configured Reports** table for the users that have access to the configured report.

- **The definition of the Apply to Class attribute of the configured report.**

Users can access all configured reports via the **Configured Reports** functionality available in the **Search** module. If the **Type** attribute of the configured report is set to `Custom` or `Query` and the **Apply to Class** attribute is defined, the configured report will also be available in the **Configured Reports** page view in the **Configured Reports** workspace of the object view of the specified object class.

Configured reports that are defined for an object class may display results about objects that are completely independent of the object the user is currently working with. For example, an external report defined for the object class Application might show a contact list of all application architects in the enterprise. However, it is also possible to configure the configured report to display results that are specific to the object that the user is currently working with. For example, a configured report might display the projects where the current application is defined as part of the affected architecture.

- **The definition of access permissions configured for the configured report in the Reports Administration functionality of the Administration application in Alfabet .**

Access permissions can be restricted to the configured authorized user and authorized user groups or granted to users logged in with a specified object view, or users that are members of a specified user group.

The administrator can either configure the configured report to be included automatically in the table of reports in the **Configured Reports** view or to be included only when explicitly added to the **Configured Reports** view.

- **The definition of the Selector Behavior attribute of the report.**

The **Selector Behavior** attribute can be used to exclude configured reports that are defined for special purposes (such as triggering data export via the RESTful API) from being selectable in the **Configured Reports** table. By default, the attribute is set to `Visible` and the configured report is visible in the selector used to add configured reports to the **Configured Reports** table if all other conditions for access to the configured report apply. If the attribute is set to `NotVisible`, the configured report will be excluded from the selector used to add configured reports to the **Configured Reports** functionality. The **Selector Behavior** attribute is not set directly in the attribute window of the configured report, but rather by triggering the respective action in the context menu of the configured report. The action can also be triggered for a report folder in order to exclude all reports in a report folder. Please note that this only excludes the configured report from the standard selector for configured reports that are used per default in the **Configured Report** views. If a custom selector is used, the configured report might be selectable if not otherwise defined by the solution designer. The **Selector Behavior** attribute has no impact on the display of the configured report in the Alfabet user interface. The report may even be visible in the **Configured Reports** table if it already has been added to the view or if the configured report has been defined to be added per default to the **Configured Report** table for a user without prior selection by the user.

For detailed information about the configuration of access permissions for configured reports, see the chapter *Defining and Managing User Access to Configured Reports* in the reference manual *User and Solution Administration*.

Integration of Configured Reports as Page Views in Custom Object Views or Object Cockpits

Alfabet is highly customizable. Customization can be performed on the level of explorers, object profiles (object views), editors and page views. Object profiles provide information about the attributes of the current object, allow editing of the object, and provide links to all relevant page views for the object. Page views either allow the user to edit the object or display non-editable data about the objects.

When configuring a custom object view or an object cockpit for a custom object view, the list of relevant views for the object can include standard Alfabet page views as well as configured reports. In the object profile, the configured reports are displayed and accessed as standard page views. The user will not be able to distinguish between standard Alfabet views or configured reports.

For a description of the configuration of object views and object cockpits including the assignment of configured reports to the object profile, see the section Configuring Object Views.

Keep the following in mind when assigning configured reports to custom object views:

- Only configured reports of the type `Query`, `NativeSQL` or `Custom` can be assigned to custom object views or object cockpits.
- Configured reports of the type `Custom` that are based on the template `EvaluationReport` are always ReadOnly if assigned to object views or object cockpits. The editability of this type of configured report has been designed for use in wizards only.

- In the object view, the **Caption** property of the configured report is displayed as the default caption for the configured report and the Description property of the configured report is displayed as the page view description for the configured report. The report caption displayed in the object view can be overwritten by setting a property **Caption** for the respective report icon in the object view.
- When the user opens the configured report from the object view, the caption of the report displays the information about the current object. The **Description** property of the configured report is displayed as a page view description below the caption. If the current object is not the base object of the configured report, the caption is identical to the **Caption** property of the configured report. If the caption has been changed in the object view configuration, this has no effect on the caption displayed in the configured report itself. Therefore, the caption can be different in the object view and the configured report.
- Only configured reports with the **Report State** attribute set to `Active` can be executed in the Alfabet interface. When a configured report is assigned to an object view or object cockpit and the **Report State** attribute is reset to `Plan` or `Retired`, the user will not be able to access the configured report. An error message will inform the user that the configured report is currently not available.
- Configured reports that are assigned to an object view are automatically applied to the object view's class without setting the **Apply To Class** attribute of the configured report. The Alfabet parameter `BASE` can be used in the query of the configured report to specify the object the user is currently working with even if **Apply To Class** is not set. When **Apply To Class** is not set, the user cannot change to another `BASE` object via an automatically generated filter field on top of the configured report but the configured report is limited to results about the current object. Please note that if **Apply To Class** is not set, the configured report can only be used in a context that predefines a `BASE` object, like in an object profile. It cannot longer be used For example, in the **Configured Reports** view of the **Search** functionalities, because this view is not called in the context of working with an object.

Integration of Configured Reports in Wizard Views in Custom Wizards

A wizard is a series of Alfabet editors and/or views that guides users through each step of a multi-step operation. A wizard usually contains multiple Alfabet views that allow you to capture a specific set of data about various aspects of an object. Wizards are typically created in order to standardize the capture of data about an object or for use in the workflow capability.

Configured eports of the type `Query`, `NativeSQL`, or `Custom` can be integrated in a wizard along with the editors and page views assigned to the wizard. For example, a configured report can be useful in a wizard in order to provide additional information that is required about an object or to provide a means to control and review data.

For a description of the configuration of wizards, see the chapter Configuring Wizards.

Integration of Configured Reports in Workflow Steps in Workflows

A workflow is a collaborative process made up of workflow steps that are typically carried out by one or more users. A workflow is based on a configured workflow template that determines a sequence of

workflow steps that are to be performed by specified users. Workflow steps are sequenced in a workflow but they may have specific pre-conditions and post-conditions that determine different branches to take in the workflow depending on the conditions that are or are not met.

A Alfabet view type is defined for each workflow step in order to specify the view that is presented to the user when opening the workflow step. For example, this view can be an editor, a wizard or a standard Alfabet page view. A configured report of the type `Query`, `Native SQL`, or `Custom` can also be added as a workflow step. For example, if objects are assigned to the base object edited in the workflow in one workflow step, the following workflow step might display a configured report that lists all relevant objects that the user can then navigate to.

For a description of the configuration of workflows, see the chapter Configuring Workflows.

Integration of Configured Reports in Custom Explorers

Custom explorers are configurable by the customer in Alfabet Expand. Each node in the explorer except the root node represents an object in the Alfabet database. When you click a node representing an object, the object profile of the object opens. The explorer can optionally be configured to show a graphic view or a configured report when you click the root node.

For a description of the configuration of custom explorers, see the chapter Configuring Standard Business Functions and Custom Explorers.

Integration of Configured Reports in Guide Pages and Guide Views

A guide page is a customized HTML file with hyperlinks that provide users access to the software and support them in their work. A guide view offers enhanced flexibility in designing the view without using a predefined HTML template. A guide page or guide view could serve as the start page for the software and can include, For example, informational text or images about enterprise-specific workflows in the Alfabet solution, links to specific functionalities in Alfabet, URL links to internal documents or the Web, or hyperlinks to other navigation pages.

Guide pages and guide views can provide links to a configured report, including the ability to pre-set the filters to defined values when a user opens the configured report from the guide page or guide view. Configured reports can also be embedded into guide pages and guide views to be displayed directly on the start page.

The configuration of guide pages and guide views is carried out in the tool Guide Pages Designer. For detailed information about how to configure guide pages and guide views for the user profiles in your user community, see the reference manual *Designing Guide Pages for Alfabet*.

In order to display a guide page or guide view as a start page in the Alfabet interface, you must assign it to the relevant user profile. For detailed information about how to assign guide pages to user profiles, see Defining Access to the Functionalities via a Guide Page in the section Configuring User Profiles for the User Community.

Integration of Configured Reports as Report Collection Into Tabular Configured Reports

For each object class and object class stereotype a number of configured reports can be configured that shall be available via tabs in all tabular configured reports finding objects of this object class (stereotype). The report collection available via tabs display information about all objects listed in the tabular configured report they are opened from. They can be used to provide aditional information about the objects listed in the tabular configured report. For example, for a tabular configured report listing a number of application groups, a portfolio can be available via a tab that informs about the criticality and business relevance of all applications in all application groups listed in the table. Via a second tab users can open a gantt chart informing about the lifecycle of the applications in all application groups.

This feature requires the following configuration:

- Defining a Category for Report Collections
- Defining Reports to be Opened in Report Collections
- Making the Report Collection Available for an Object Class or Object Class Stereotype
- Configuring Tabular Reports to Show the Report Collection

Defining a Category for Report Collections

A category must be assigned to configured reports that shall be used in a report collection. Prior to defining the configured reports, you must define the category in the XML object ***UseCaseCategories***:

1) In the **Presentation** tab of Alfabet Expand, expand the explorer node **XML Objects**.
2) Right-click on the XML object ***UseCaseCategories*** in the explorer and select **Edit XML** from the context menu. The XML object opens in the middle pane.
3) In the XML object, add an XML element `UseCaseInfo` as child element of the root XML element `UseCaseCategories` with the following XML attributes and child elements:

   ```
   <UseCaseInfo UseCase="CustomChartViews">
   <ScopeInfo Scope="Report" Categories="CommaSeparatedListOfCategories" />
   </UseCaseInfo>
   ```

   CommaSeparatedListOfCategories must be substituted with at least one category name. This name will be selectable in the attribute **Category** of configured reports to mark them as part of a report collection. If you enter more than one name comma separated, all names will be valid as category name.

4) In the toolbar, click the **Save** button.

Defining Reports to be Opened in Report Collections

You can add any type of configured report to a report collection. Reports for report collection must have the following configuration in addition to the general configuration requirements for the specific type of configured report:

- The **Category** attribute must be set to the category defined in the XML object ***UseCaseCategories*** for the `CustomChartViews` use case.
- The **Apply to Class** attribute must not be set.
- The query or queries the configured report is based on must be defined using the Alfabet query parameter `CVREFS` that returns the list of `REFSTR` values for all objects currently displayed in the tabular dataset the report will be opened from via a tab in the report collection.

  The following parameter shows a `WHERE` clause for a report that shall be added to the report collection of the object class Application in native SQL:

  ```
  WHERE APPLICATION.REFSTR IN (@CVREFS)
  ```

- A report view must be created for the configured report.
- The report must not have any filters. For configured reports of the type `Query` or `NativeSQL` the automatically generated filter field for the `CVREFS` parameter must be deleted from the report view.

Making the Report Collection Available for an Object Class or Object Class Stereotype

The number and order of tabs available as report collection are defined in the class settings of the object class or object class stereotype in Alfabet Expand:

1) Go to the **Presentation** tab and expand the **Class Settings** node.
2) Expand the node of the object class or object class stereotype to which you want to add a report collection and click the relevant class setting.
3) In the attribute window, click the **Browse** button in the **Supplementary Reports** attribute in the **Custom Chart Views** section.
4) In the editor that opens, all configured reports that are assigned to the category of the `CustomChartViews` use case are listed. Click a configured report to select the checkbox. Optionally, you can then move the position of the configured report in the list using the up and down buttons on the right of the report list field. The order of tabs in the report collection on the Alfabet user interface is identical to the order of configured reports in the field.

   You can sort the reports in the list lexicographically by selecting the checkbox **Sort Lexicographically**. Please note that the lexicographic order applies to the names of the reports and not to the report caption. As the text of the tabs is identical to the caption of the configured report, lexicographic sorting via this mechanism might not result in an lexicographic sorting in the user interface. Nevertheless you can use the functionality to find configured reports in a long list of configured reports.

Configuring Tabular Reports to Show the Report Collection

A report collection configured for an object class or object class stereotype is only displayed in configured reports with the following configuration:

- The **Type** attribute of the configured report is set to either `Query` or `NativeSQL`.

- The configured report finds objects of the object class or object class stereotype for that a report collection is defined in the class settings.
- The following attributes of the configured report are set to activate the report collection:
  - **Chart Views Mode**: The attribute is set to `Custom`. By default, the attribute is set to `Standard`, and standard page views providing additional information about the objects in the tabular dataset will be accessible via a button in the toolbar. For more information, see Display of Results in Standard Alfabet Report Formats. The attribute can be set to `None` to provide neither standard chart views nor a report collection.
  - **Custom Chart View Base Class**: This attribute is only visible if the **Chart Views Mode** attribute is set to `Custom`. Select the object class or object class stereotype for which the tabs opening the configured reports for the report collection shall be displayed. Please note that the selected object class must be identical to the object class found in the tabular dataset of the configured report you are currently configuring. If the objects found in the tabular dataset do not match the setting, the report will not show the tabular dataset, but an error message about the incorrect setting in the configuration.

Integration of Quality Widgets in Configured Reports, Object Cockpits and Wizards

Additional information about data in a configured report, object cockpit, or wizard/wizard step can be provided via a quality widget displayed in a pop-up window. The quality widget is either a configured widget report or a configured business chart or Gantt chart report that can be assigned to multiple configured reports, object cockpits, and wizard/wizard steps. When the user opens or loads the configured report, object cockpit, or wizard/wizard step that the quality widget is assigned to, the quality widget will be displayed in a separate pop-up window that opens for a short time. The quality widget pop-up window opens by default in the upper-right corner of the Alfabet user interface below the main menu. This position is configurable. In the case of a wizard/wizard step, the quality widget will be appended to the upper-right edge of the wizard.

Depending on the configuration of the primary configured report, either the complete quality widget will be displayed or only the caption will be displayed in a title bar and the user can click the title to open the quality widget.

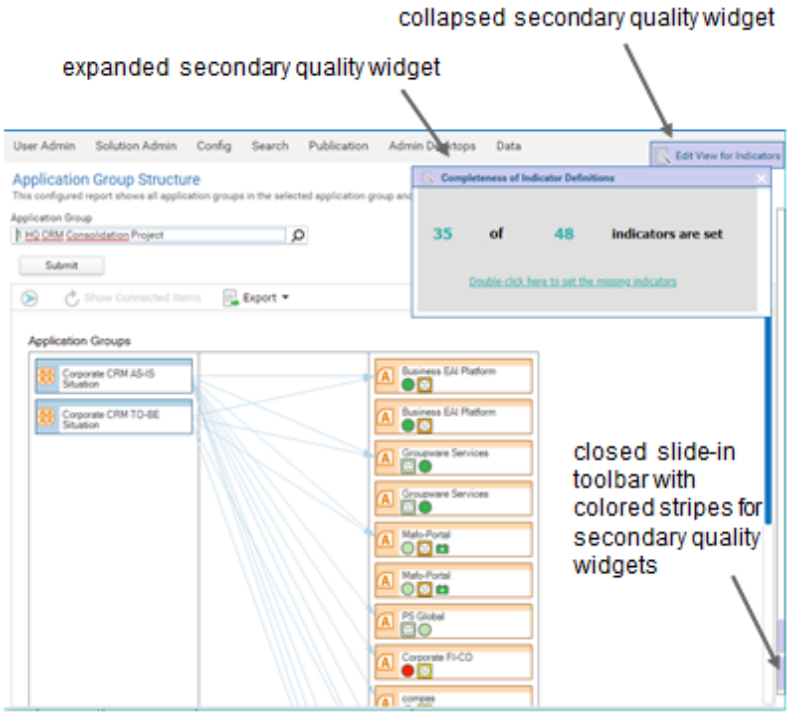


FIGURE: Configured report with a collapsed and an expanded quality widget

If the user does not click the quality widget during the short time that it is visible, the quality widget view will automatically drop into the slide-in toolbar on the right edge of the Alfabet user interface. The slide-in toolbar displays a colored notch for each of the available quality widgets (as well as for other forms of user assistance). The user can access the quality widget anytime by pointing to the notch in the slide-in toolbar and then clicking the quality widget icon to open it. The quality widget will remain open until the user clicks outside of the pop-up window or clicks the close button.

If navigation is configured from the quality widget view to another configured report or graphic view, the navigation target will open in a new browser tab.

A typical use case for quality widgets is a widget report that opens to provide information to the user about the quality of the data in the current configured report, object cockpit, or wizard/wizard step. For example, if objects displayed in a configured report are colored according to an indicator value, the user could be provided with information about the completeness of the indicators specified for the objects in the configured report. A configured link available in the quality widget could then open a view that allows the user to provide the missing data and thus correct the issue. When the user clicks the link available in the quality widget report, the view will open in a new browser tab. The user can then define the relevant indicators and reload the original view with the updated data.

For more information about assigning the configured report as a quality widget to an object cockpit, see the section Adding Data Quality Widgets to Object Cockpits. For more information about assigning the configured report as a quality widget to a wizard or wizard view, see the section Adding Data Quality Widgets to Object Cockpits.

Please note the following for the specification of a configured report that shall be integrated into the Alfabet user interface as a quality widget:

- The configured report must be of the type `Custom` and based on either of the templates `WidgetReport`, `GanttReport` or `BusinessChartReport`.

For more information about defining widget reports, see Defining Widget Reports. For more information about defining business chart reports, see Defining Chart Reports. For more information about defining Gantt chart reports, see Defining Gantt Chart Reports.

- The Alfabet query language parameter `BASE` can be used in the query or queries that the configured quality widget report is based on. The parameter will be substituted with the `REFSTR` of the object that the user is currently working with in the object view. If the quality widget opens for a configured report and the configured report is assigned to a base class with the attribute **Apply To Class**, the parameter `BASE` is substituted with the `REFSTR` of the current object that the user is opening the configured report for.

The configured report used as the quality widget should not be assigned to a base class with the **Apply To Class** attribute. It is automatically assigned to the base class of the view it opens for.

- The configured report must not have filters nor toolbar buttons nor drill-down defined.

The visualization of secondary window skins is configured via the **Quality Widget** attributes for a GUI scheme. For more information, see the section Configuring GUI Scheme Definitions for the Alfabet Interface as well as the section *Attributes Displayed in the Grid Section: Secondary Window Skins* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

To assign a quality widget to an existing configured report or object cockpit:

1) Right-click the configured report or object cockpit and select **Add Secondary View** . A secondary view is added to the explorer and the attributes of the secondary view are displayed in the attribute window.
2) Define the following attributes of the secondary view:
   - **Name**: Enter a unique name for the secondary view. The name is a technical name and shall not contain special characters or whitespaces. The technical name is not displayed in the user interface.
   - **Caption**: Enter a caption for the secondary view that shall be displayed in the title bar of the secondary view in the Alfabet user interface and in the tooltip displayed for the icon of the secondary view in the slide-in toolbar. Please note that the caption should be short. It will not be displayed in the title bar if it is too long.
   - **Role**: Select `QualityWidget`.
   - **View Type**: Select `Report`.
   - **View Name**: Select the configured report that shall be displayed as secondary view. The drop-down list displays the names of all available configured reports of the allowed report types widget report, gantt chart report, and business chart report.
   - **Show Automatically**: Select `True` if you would like the secondary view to be fully displayed for a few seconds when the user accesses the object cockpit or configured report that the secondary view is defined for. Select `False` if only the caption of the secondary view shall be displayed in a title bar for a few seconds when the user accesses the object cockpit or configured report that the secondary view is defined for. The secondary view will then open if the user clicks the title bar.
   - **Position**: Select the position of the open quality widget on the Alfabet user interface. The quality widget window can be placed in the center of the Alfabet user interface or in one of

the corners of the user interface beneath the main toolbar. The quality widget pop-up window opens by default in the upper-right corner of the Alfabet user interface below the main menu.

- **Height**: Define the inner height of the quality widget window in pixel. This is the height available for the display of the configured report opened in the window.
- **Width**: Define the inner width of the quality widget window in pixel. This is the width available for the display of the configured report opened in the window.

3) In the toolbar, click the **Save**  button.

Translating Configured Reports

Alfabet provides a multi-language capability. This allows users to display the Alfabet interface in a secondary languages that are used in your enterprise. For a detailed description of how to implement multi-language support in Alfabet, see the chapter Localization and Multi-Language Support for the Alfabet Interface.

If your enterprises requires the Alfabet interface to be displayed in any of the secondary supported languages, then you must explicitly ensure that the data displayed in configured reports is also translated to the relevant language. The following must be carried out in order to translate the content of a configured report:

- For each report configured in Alfabet Expand, the **Caption** and **Description** attributes will be displayed in the Alfabet interface. These attributes can be translated as described below in the section Translating the Caption and Description of a Configured Report.
- For relevant configured reports configured in Alfabet Expand, the column headers of tabular reports are created based on either the alias definitions in the `Show` properties of the Alfabet query, the `SELECT` clause of the native SQL query, or the instructions defined for the query. The column headers can be translated as described in the section Translating Column Headers or Captions of a Configured Report.
- String values in a graphic report that have been defined directly in an attribute of the report assistant are usually added automatically to the vocabulary files. This includes, For example, the captions for lanes in lane reports, the names of color rules and indicator rules that are used as legend headers, static text definitions in widget reports, or the graphic title, X-axis title, and Y-axis title of business chart reports or portfolio reports. For gauge and map chart reports, this also includes the captions defined for color range members. For general information about vocabulary translation, see Modifying, Translating and Managing the Vocabularies.
- Text defined for the cell tooltip in the `LinkAssignment_Edit` instruction as well as text defined as legend text in the Alfabet query language instructions `ColorAssignment`, `PictureAssignment`, `RowColorAssignment`, `FontStyleAssignment`, `FontStyleColorAssignment` are added automatically to the vocabulary files. For general information about vocabulary translation, see Modifying, Translating and Managing the Vocabularies.
- In the case of configured portfolio, lane, and grid reports, the graphic title, X-axis title, and Y-axis will not be added automatically to the vocabulary files. Texts provided for the calculation of a total value for pie charts will also not be added automatically to vocabulary files. These values can be translated as described in the section Translating Column Headers or Captions of a Configured Report.

Please note that the labels displayed in configured portfolio charts and gauge reports and the labels for enterprise milestones in configured Gantt chart reports cannot be translated.

If configured reports are translated to Arabic, you should be aware that the text is right-aligned instead of left-aligned in user interface elements. For example, for the display of a **Caption** defined for a **Lane** in a lane report, you will see the text above the lanes instead of above the object boxes because the space for the caption spans both object boxes and lanes. To display a caption above the object boxes, you must define the caption in the **Caption** attribute of the **Node** element below the **Lane** element.

If the Alfabet user interface is rendered in Arabic language, the order of columns and display of graphic elements in graphic reports is changed to right to left instead of left to right, as for all other supported languages. Only configured reports based on the template `MatrixMapReport` that have the subtype `Diagram` and Pivot Grids rendered with the embedded DevExpress® component are still displayed left to right.

- For Alfabet objects displayed in a configured report, the translation of relevant object properties is read from the database tables. The following object class properties are translatable and can be displayed in a secondary language:
  - Predefined protected properties such as the object's **Name** and **Description** properties as well as custom properties of the type `String` and `Text`. These properties will be displayed in the target language if the **Support Data Translation** attribute of the relevant culture is set to `True`, and a translation has been made in the relevant object class editor in the Alfabet user interface. The translated values for object data will be displayed automatically in configured reports based on Alfabet query language. For native SQL queries, however, a special code triggering the translation must be added to native SQL queries in order to enable the display of translation values for object data. For more information, see Displaying Translated Object Data In the Current Language of the User Interface in the chapter Defining Queries. For more information about object data translation, see the section Configuring the Translation of Object Data.
  - The caption of the object class and object class stereotype. These captions can be added to a configured report via instructions. If this is the case, the captions of the object class and object class stereotype can be displayed in a target language if they are translated in the vocabularies available in Alfabet Expand. For general information about vocabulary translation, see Configuring the Translation of Object Data.
  - Values defined for object states, release statuses, indicator values, enterprise releases, project milestones, and enumerations. These values can be displayed in a target language if the following applies:
    - The **Enable Data Translation** attribute of the relevant object class property (properties of the type `String` or `Text`) is set to `Manual` or `ManualAndAutomated`.
    - Translations are provided in the vocabularies available in Alfabet Expand. For more information about vocabulary translation, see Configuring the Translation of Object Data.
    - The values are not automatically displayed translated in configured reports. An instruction triggering the translation must be added to queries to enable the display of translation values. For more information, see Translating Enumerations, Object States, Release Statuses, and Indicator Values in a Configured Report.

Status values that are editable in the **Report** editor in the **Reports Administration** functionality available in the Alfabet interface is neither a release status, object state, nor based on an enumeration and Therefore, cannot be translated in the **Report** editor.

Translating the Caption and Description of a Configured Report

The **Caption** and **Description** attributes will be displayed in the Alfabet interface for each report configured in Alfabet Expand. These attributes can be translated to the languages supported by your enterprise.

Once the **Caption** and **Description** attributes have been specified for a configured report, they will be automatically added to the vocabularies available in Alfabet Expand. Once they are available in the vocabularies, they can be translated like any other string in the **Translation Editor** or exported to a Microsoft Excel file and translated there. For more information about this process, see the section Modifying, Translating and Managing the Vocabularies.

For technical reasons, the translation of report captions and descriptions is coupled with the mechanisms of object data translation. Therefore, the translation provided via the vocabularies is only displayed in the Alfabet interface if the following applies:

- The **Support Data Translation** attribute must be set to `True` for the respective culture.
- The translation must be updated in the data translation columns of the database table of the object class `Report`. To write the translated **Caption** and **Description** attributes to the database table of the object class `Report`:
  1) Set the **Report State** attribute for the configured report to `Plan`.
  2) Right-click the configured report and select **Update Translation**.

     To batch update the translation of multiple configured reports, right-click the **Reports** node and select **Update Translation**. The captions and descriptions for all configured reports for which the **Report State** attribute is set to **Plan** will be updated.

  3) Set the **Report State** attribute for the configured report to `Active`.

Translating Enumerations, Object States, Release Statuses, and Indicator Values in a Configured Report

Configured reports require the specification of instructions to display the translated values. Two new Alfabet instructions are available that trigger the display of translated values for indicator ranges, enumerations, object states, and status definition values:

- To display enumeration, object state, and release status values in the translated language, the following Alfabet instruction must be added to the query. The `ColumnName` is the name of the column containing the information that shall be translated:

```
TranslateEnums("ColumnName,ColumnName,..");
```


- To display indicator range values in the translated language, the following Alfabet instruction must be added to the query. The `ColumnName` is the name of the column containing the information that shall be translated:

  ```
  TranslateIndicators("ColumnName,ColumnName,..");
  ```

For detailed information about the definition of the Alfabet instructions see Displaying Translated Values of Enumerations, Object States, Statuses, Milestones and Indicators in Configured Reports in the section Using Instructions to Format the Results of an Alfabet or Native SQL Query.

Translating Column Headers or Captions of a Configured Report

The definition of column header or graphic report elements in a native SQL query or Alfabet query is typically a complex process. This is the case, For example, for a native SQL query containing a `WITH` statement or if instructions are used for renaming and restructuring the results of the query. In such cases, it is not possible to automatically scan the query definitions for strings that need to be translated.

Therefore, if you want to translate column headers, you must write the column header strings in the XML object ***VocXML***. The strings defined in the XML object ***VocXML*** are automatically added to the meta-model section of the vocabularies and can be translated to a defined target language as described in the section Modifying, Translating and Managing the Vocabularies.

Strings that are defined directly in report assistants such as the caption of lane reports or strings that are explicit in the query definition such as column headers defined in `SetColumnShowName` instructions are automatically added to the translation tables. You can check the vocabularies prior to adding strings to the XML object ***VocXML*** to evaluate which strings have already been added automatically to the vocabulary.

To add column headers to the translation table of the Alfabet vocabularies:

1) In the **Presentation** tab of Alfabet Expand, expand the **XML Objects** node and double click the XML object ***VocXML***. The XML object ***VocXML*** opens in the center pane. The editor automatically displays the root XML element `VocXml` of the XML object.
2) For each column header, enter a child element `Entry` to the `VocXml` root element. In the `Entry` element, write the string of the column header in the original language.

   Alfabet vocabularies are string-based. Therefore, if two configured report have the same column header, you must only add the string once to the XML object t ***VocXML***. If the header column string is identical to a string that is used somewhere else in the configuration of the Alfabet meta-model (For example, if the header column is identical to the caption of an object class property), the string will already exist in the vocabulary and does not need to be entered in the XML object t ***VocXML***. If you enter a duplicate string in the XML object ***VocXML*** it will be ignored.

3) In the toolbar, click the **Save**  button to save your changes.

For example, a configured report is based on the following native SQL query:

```
SELECT proj.REFSTR, proj.NAME AS 'Project Name',proj.STEREOTYPE
FROM PROJECT proj
```


The query is combined with the following Alfabet instructions:

```
InsertColumn("1","StereotypeCaption");
SetColumnShowName("StereotypeCaption","Project Stereotype");
SetStereotypeCaption("REFSTR","STEREOTYPE", "StereotypeCaption");
RemoveColumns("STEREOTYPE");
```

The resulting configured report will have two column headers:

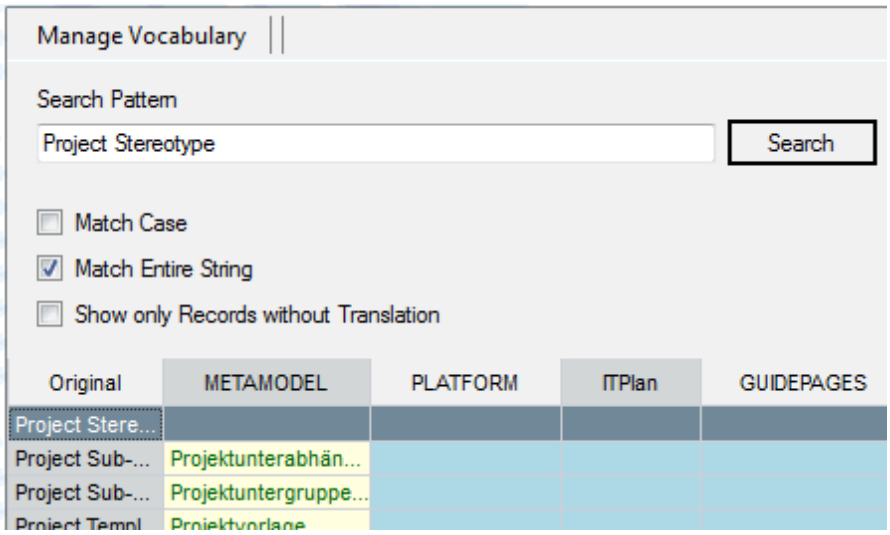
- "Project Name", derived from the alias definition in the `SELECT` clause of the native SQL query
- "Project Stereotype" defined via the `SetColumnShowName` instruction.

	Project Name	Project Stereotype
1	Consolidate Trading Applications	Program
2	Retire GL Applications	Project
3	Upgrade GenLManager	Project
4	Reshape Core Trading Applications	Project
5	Enhance TradeNet	Project
6	Implement Unified Trade Solution	Project
7	UTS phase 1	Project Step
8	UTS phase 2	Project Step
9	UTS phase 3	Project Step
10	Streamline CRM Applications	Program
11	Migrate CRM Opti Retail to CRM CSS	Project
12	Integrate CRM with SAP	Project
13	Retire GL Applications Solution	Project

To translate the headers of the configured report, the two definitions must be written in the XML object ***VocXML***:

```
<VocXml>
   <Entry>Project Name</Entry>
   <Entry>Project Stereotype</Entry>
</VocXml>
```

The terms are then translated to the target language (in this case, German) in the **Translation Editor** of Alfabet Expand:



The translations will be displayed if German is selected as the base culture for th---e culture displayed in the Alfabet interface.

	Projektname	Projekt-Stereotyp
1	Consolidate Trading Applications	Programm
2	Retire GL Applications	Projekt
3	Upgrade GenLManager	Projekt
4	Reshape Core Trading Applications	Projekt
5	Enhance TradeNet	Projekt
6	Implement Unified Trade Solution	Projekt
7	UTS phase 1	Projektschritt
8	UTS phase 2	Projektschritt

Please note: You can see from the example that the content of the configured report is also only partially translated. The project names are displayed in English because data translation has not yet been activated. The stereotype captions are automatically added to the vocabulary and have been translated. Therefore, the translated values are displayed in the configured report. For more information about data translation, see the section Configuring the Translation of Object Data.

Control Report Usage to Remove Unused Configured Reports

Alfabet provides a mechanism to control whether configured reports are still in use. A report can be displayed in Alfabet Expand that informs you about the number of users currently accessing the configured report. When the configured report is not used at all, you can remove the configured report from your configuration. The activity tracking of report usage is not done by default but must be explicitly activated. The usage report only shows data evaluated after activation of activity tracking.

The following workflow is recommended to limit your report configuration to currently active configured reports:

- Activate Usage Tracking to enable count of access to individual configured reports. For more information, see Activating Activity Tracking.

- When you want to decide about retiring configured reports, consult the activity report to evaluate which configured reports are no longer used. For more information, see Reading the Activity Tracking Report.
- Retire the configured reports and remove them from the current configuration before deleting them. For more information, see Removing the Configured Report From the Configuration.

Activity tracking is managed via the Alfabet Server and only configured reports accessed by users working in client-server-mode are included into the tracking report. When the configured report is accessed by a tool with direct access to the Alfabet database, For example, when working with Alfabet Expand in standalone mode, is not taken into account.

Activating Activity Tracking

Report activity tracking is based on the same mechanisms as the usage tracking mechanism that is implemented for metered contracts. Usage tracking must be activated to enable report activity tracking.

For more information about usage tracking, see the section *Usage Tracking* in the *Technical Requirements*.

When usage tracking is activated, activity tracking information is written into *.dat files in the directory defined as **Track Usage Directory** field of the server alias configuration. The information is stored in encrypted form and can only be accessed via the activity tracking report in Alfabet Expand.

Reading the Activity Tracking Report

Do the following to open the Activity Tracking report:

1) Open Alfabet Expand.
2) In Alfabet Expand, go to the **Reports** tab and right-click the **Reports** root node.
3) In the context menu, select **Open Usage Activity Tracking**. The report opens in the middle pane of the user interface of Alfabet Expand.
4) Optionally, you can export the report to Microsoft Excel format to save and view the report outside of Alfabet Expand. Click the **Export** button and select a location to save the report in the explorer that opens to create a Microsoft Excel version of the report.

The report displays all report folders and configured reports available in the **Reports** explorer.

The report is an expandable dataset. Expanded nodes show a + sign and can be collapsed by clicking the + sign. Collapsed nodes are showing a - sign and can be expanded by clicking the - sign.

Report folders and configured reports can be identified in the activity tracking report via their name and caption.

For each configured report, the last usage data is displayed. The number of users that accessed the report is given for four different time periods: The current month, the last 3 month, the last 6 month and the last 12 month.

The report displays the number of users that access the configured report and not the number of times the configured report was used. When a user accesses the configured report multiple times during a defined period, even with different user profiles, this is only raising the number of users displayed for the defined period by one.

The numbers displayed for a report folder are the sum of numbers calculated for the configured reports subordinate to the report folder.

Removing the Configured Report From the Configuration

When a configured report is not used any longer, it can be deactivated by setting the **Report State** to `Retired`. Retired configured reports cannot be accessed any longer on the Alfabet interface. The configured report is displayed struck-through in tables listing available configured reports and when a user tries to access the configured report from For example, an object view, an error message is thrown.

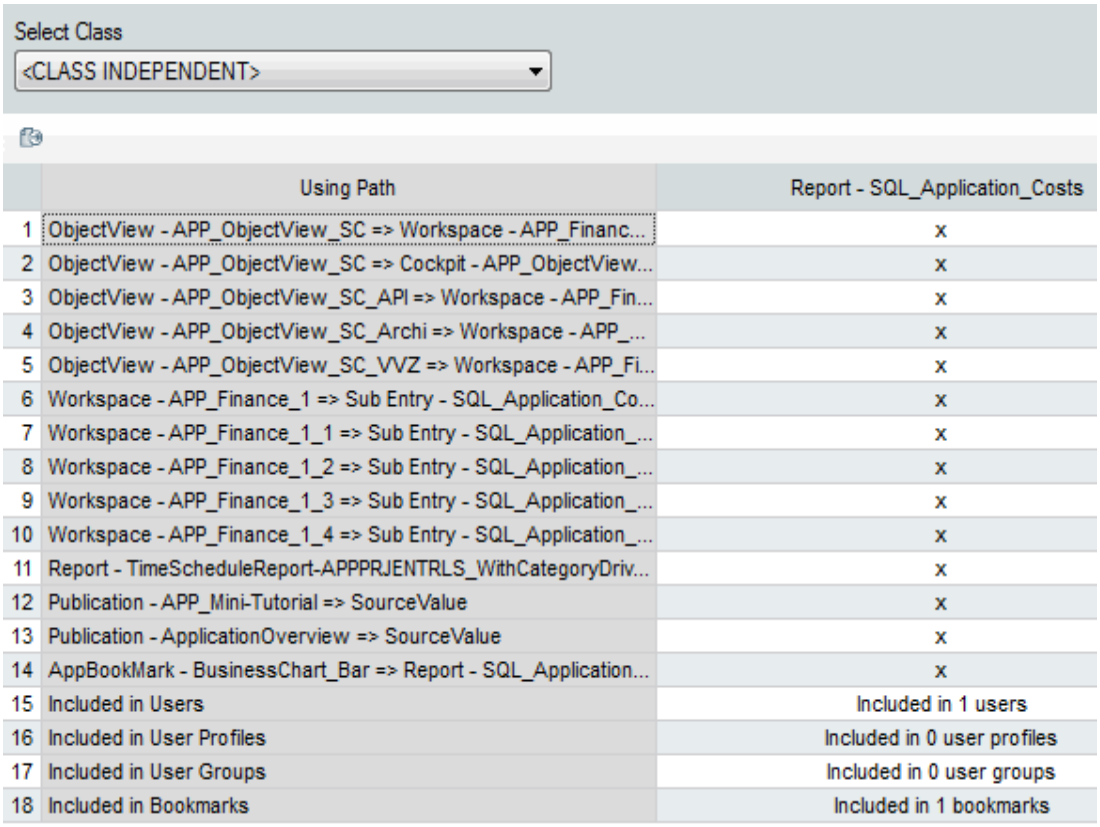
You can decide whether the configured report shall still be kept in the report configuration for future reuse or as template for similar configuration or whether it shall be deleted.

Prior to retiring and optionally deleting a configured report, the configured report must be removed from the configuration. Configured reports can be assigned to object views and object cockpits, wizards, workflows and custom explorers. They can also be assigned directly to a user, user profile or user group in the **Reports Administration** functionality of Alfabet. Guide pages and guide views may include links to the configured report.

A report is available that lists the usage of a configured report in the configuration.

Do the following to delete a configured report from the configuration:

1) In the **Reports** tab of Alfabet Expand, right-click the configured report in the explorer and select **Show Usage** from the context menu. A new window opens displaying the usage of the configured report in the configuration.


Select Class

<CLASS INDEPENDENT>

	Using Path	Report - SQL_Application_Costs
1	ObjectView - APP_ObjectView_SC => Workspace - APP_Financ...	x
2	ObjectView - APP_ObjectView_SC => Cockpit - APP_ObjectView...	x
3	ObjectView - APP_ObjectView_SC_API => Workspace - APP_Fin...	x
4	ObjectView - APP_ObjectView_SC_Archi => Workspace - APP_...	x
5	ObjectView - APP_ObjectView_SC_VVZ => Workspace - APP_Fi...	x
6	Workspace - APP_Finance_1 => Sub Entry - SQL_Application_Co...	x
7	Workspace - APP_Finance_1_1 => Sub Entry - SQL_Application_...	x
8	Workspace - APP_Finance_1_2 => Sub Entry - SQL_Application_...	x
9	Workspace - APP_Finance_1_3 => Sub Entry - SQL_Application_...	x
10	Workspace - APP_Finance_1_4 => Sub Entry - SQL_Application_...	x
11	Report - TimeScheduleReport-APPPRJENTRLS_WithCategoryDriv...	x
12	Publication - APP_Mini-Tutorial => SourceValue	x
13	Publication - ApplicationOverview => SourceValue	x
14	AppBookMark - BusinessChart_Bar => Report - SQL_Application...	x
15	Included in Users	Included in 1 users
16	Included in User Profiles	Included in 0 user profiles
17	Included in User Groups	Included in 0 user groups
18	Included in Bookmarks	Included in 1 bookmarks

The report provides the following information:

- For each configuration object in Alfabet Expand and for each guide page or guide view the configured report is assigned to there is a row in the report that displays the type and name of the configuration object and location of the configured report in the implementation of the configuration object. If the configured report is not assigned to any configuration object, you will not see any configuration object rows in the report.
- The rows **Included By Users**, **Included In Profiles** and **Included In User Groups** are always displayed in the report. They inform about the assignment of configured reports to the **Configured Reports** page view of users in Alfabet. The assignment of configured reports is done in the **Reports Administration** functionality. A configured report may be configured to be available to a user on request only. In that case, the **Show Usage** table shows the number of users that actively added the configured report to their **Configured Reports** page view.
- The row **Included in Bookmarks** is always displayed in the report. It informs about the number of bookmarks created by any user that open the report.
- The **Select Class** filter on top of the **Show Usage** table allows to limit the display to configuration objects that are assigned to the selected class. Configuration objects that are not assigned to any class and the information of usage by person, user profile or user group are always displayed. It is recommended to set the filter to `<CLASS INDEPENDENT>` in this context to make sure that the report is removed from all configuration objects prior to retiring it.

2) In Alfabet Expand, remove the configured report from the object profiles, wizards, workflows and custom explorers the configured report is assigned to.

If a configured report selected in the **Supplementary Reports** attribute of class settings is deleted, it will be removed from the **Supplementary Reports** attribute automatically without any user interaction required.

3) In the **Reports Administration** functionality of Alfabet remove the configured report from all users, user profiles and user groups it is assigned to.
4) In the **Reports** tab of Alfabet Expand, right-click the configured report in the explorer and select **Set Report State to Retired**. The configured report cannot be accessed any longer by users in the Alfabet interface but is still available in the **Reports** explorer of Alfabet Expand.
5) Optionally, right-click the configured report and select **Delete**. The configured report node is irrevocably removed from the explorer.

Chapter 18: Publishing Data In Microsoft® Word Format

The Alfabet Publication Framework (APF) allows you to publish Microsoft® Word documents containing data from the Alfabet database.

APF allows you to select objects via customer-defined queries and define the data that shall be published for an object via Word templates defined directly in Microsoft® Word. Attribute data, data about related objects, and reports available for an object can be added to the published output. The publication can include data about multiple object classes, and a different template can be used to publish data for the different object classes. For example, the publication about the applications in an application group can include the name of the application group and the application portfolio report for the application group. For each application in the application group, the application's name and version and **Evaluations** page view can be added to the publication.

Publication definitions are specified in the configuration tool Alfabet Expand. The definition of the publication includes the following

- The objects for which data will be published. The objects can be selected via a query or a user can select one or multiple objects for publication directly in the Alfabet user interface.
- The Word template used to publish data about the object(s).
- The mapping of the object data to bookmarks defined as placeholders in the Word template
- Which related objects shall also be published.
- The Word template used to publish data about the related objects
- The mapping of the object data of related objects to bookmarks defined as placeholders in the Word template

Publications can be triggered by a batch utility or by the user accessing a report configured for publication purposes in the Alfabet interface. There are two methods for users to trigger a publication:

- The user opens a configured report that allows the objects to be selected, a publication to be triggered, and the document to be published.
- The user clicks a button in the toolbar of an object's profile to open a configured report that allows a publication to be triggered and the document to be published.

The figure below displays a configured report that displays a table showing one or more applications that the user can select for publication. The publication can be executed via the buttons below the table.

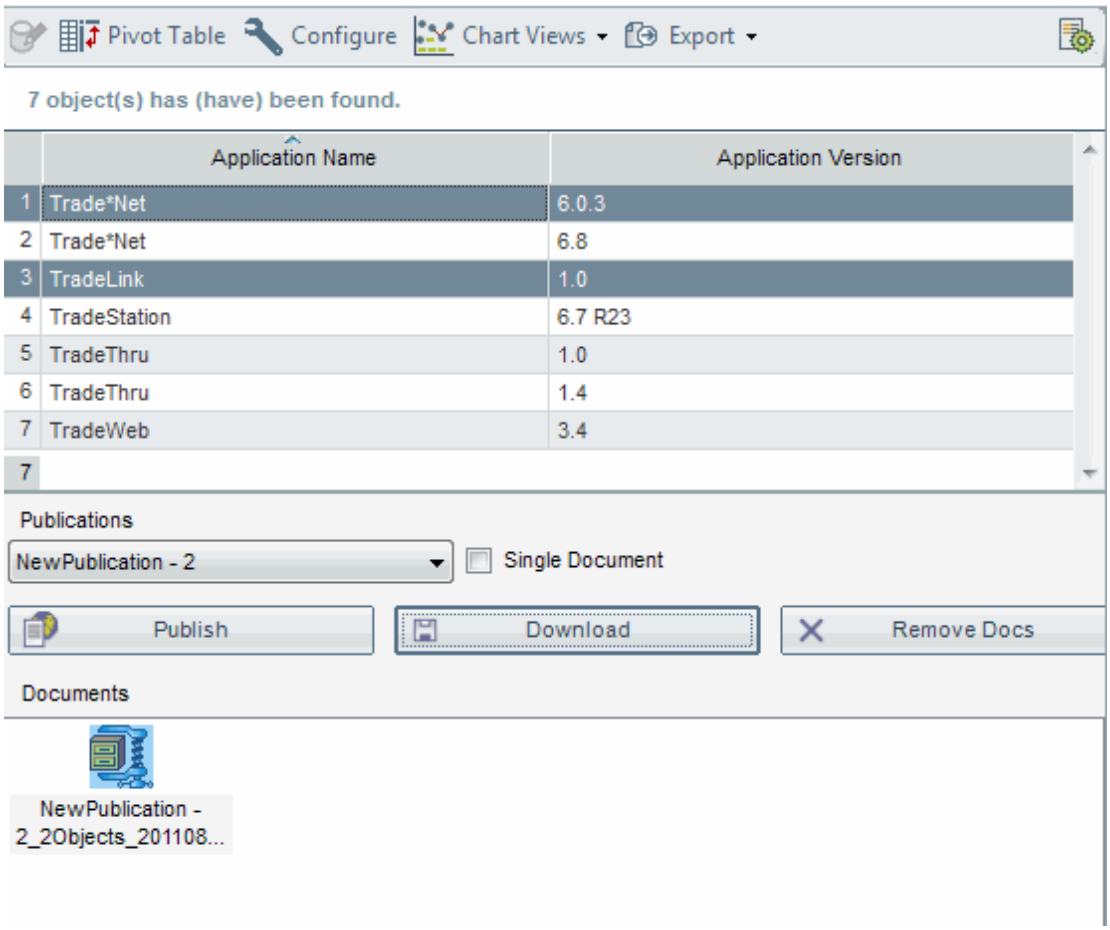


FIGURE: Configured report triggering publications about applications

To trigger a publication, the user must perform the following:

1) Select one or multiple objects in the report.
2) Select a configured publication in the drop-down list in the **Publications** field
3) Optionally, select the checkbox **Single Document** if data about all selected objects shall be published in one document. If the checkbox is not selected, one document will be published per selected base object.

   Publishing data about multiple base objects in a single document may alter the design of the title page of the document. If the Microsoft® Word template is configured to have a title page **without** a header and/or footer while the rest of the document is configured to have a header and footer, the header/footer will be added to all title pages displayed throughout the document.

4) Click the **Publish** button to publish the data. A ZIP file is created with the publication results and the section **Documents** shows an icon for the publication results.

   Alternatively, published documents can be displayed in a table. In the toolbar of the Publications field, select **View** > **Show As List** to display the publications in a table.

5) Click the **Download** button to download the publication results from the Alfabet database to the local file system. Optionally, the documents can be removed from the database after download via the **Remove Docs** button.

In the figure below, a publication is configured to include data about an application group and all applications that are assigned to the application group. The publication definition is mapped to two templates, one for the application group data and one for the applications. To create the document for the single application group, the Word template for application groups will be filled with the data for the application group. The document is then amended with data for each application in the application group. For each application found, the Word template for the applications is filled with the appropriate data for the application and added to the document.

If data is not available for a view that is configured to be added for a bookmark, the publication will display a message stating that no data is available for the report.

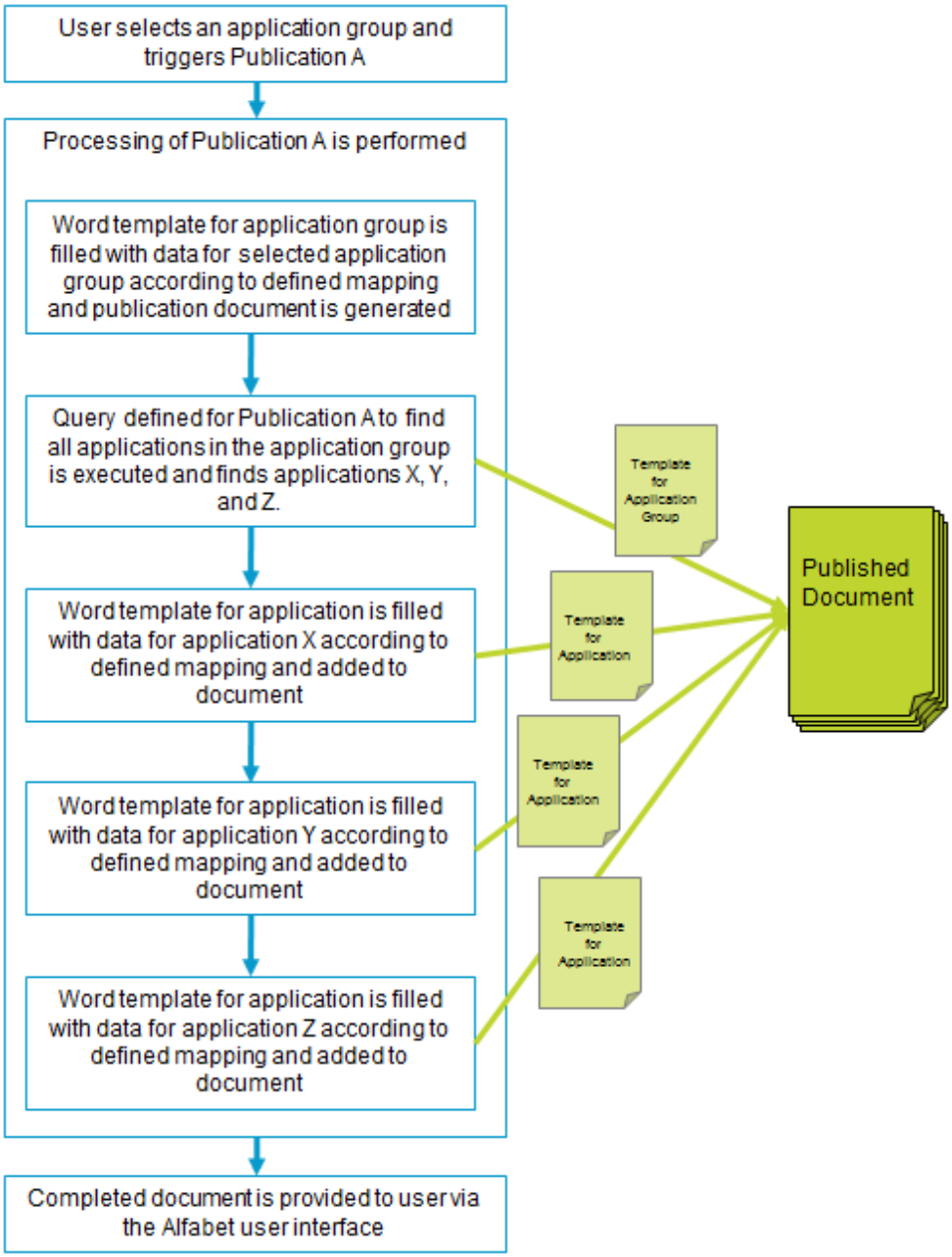


FIGURE: Generation of a Word document with data about an application group

The output is a ZIP file that contains either a document with all relevant data or a set of documents with information about each relevant object.

The name of the published documents and the ZIP file will depend on the number of base objects selected to trigger the report:

- If data about one object is published, the file naming convention will be: <publication name>_<object name>_<timestamp>
- If data about multiple objects is published, the file naming convention will be: <publication name>_<number of objects on first level of publication>_<timestamp>
- If no base objects are selected and data is published on basis of a query definition independent from a current object selection, the file naming convention will be: <publication name>_<timestamp>
- Constraints on the generation of ZIP files may result in a deviation from the naming convention for APF publications:
  - File names may not exceed 256 characters. If the file name is too long, the <object name> will be shortened, or, if applicable, left out. If the file name is still too long, the <publication name> will be shortened.
  - File names must not contain characters other than letters, numbers, underscores, or hyphens. If the object name or publication name contains a special character, the special character will be replaced with an underscore.

The timestamp includes the current date and time in the format: YYYYMMDD_hhmmssmmmm.

The following is required to configure Alfabet to allow users to trigger data publication:

- Design at least one template for the publication in Microsoft® Word. For more information, see the section *Defining Microsoft® Word Templates for Publication*.
- Define the mapping of data in the Alfabet database using the Word templates. This includes the definition of the queries that find the relevant objects for the publication. For more information, see the section *Mapping Alfabet Data to Word Templates*.
- Configure the user interface to allow users to trigger a publication. For more information, see the section *Configuring Access to the Publication Functionality*.

Alternatively, publications can be triggered by a batch job.

Defining Microsoft® Word Templates for Publication

The first step to set up a publication in the APF is the definition of at least one publication template in Microsoft® Word DOT or DOTX format.

Publications can be configured to include information not only about one object or multiple objects of the same object class, but also about related objects. The publication definition can be regarded as a tree. Each node in the tree must be mapped to a Microsoft® Word template. Multiple nodes in the tree can be mapped to the same Word template provided that the same data structure applies to the publication of the data about the objects found for the node.

The mapping of object data to a Word template is defined per node in the publication definition (and not per object class). You can use different Word templates for data about the same object class if the objects of the respective object class are found via different nodes in your publication definition.

The publication templates are defined directly in Microsoft® Word. They are simple Microsoft® Word documents that can be formatted according to your needs using the mechanisms available in Microsoft® Word.

The Word template consists of plain text that will be published for each object found for the node in the publication definition. Microsoft® Word bookmarks can be used in the text as placeholders for data that is to be added about the object.

The following object information can be added to a publication:

- Object class property values
- Graphic views
- Configured reports
- Datasets that are the result of a query defined in the publication definition.

To define a Microsoft® Word template:

1) Open the Microsoft® Word program installed on your computer.
2) Create a document that matches the desired publication in content and style and save it as Word template (*.dot or *.dotx).
3) Add bookmarks to the document wherever you want Alfabet data to be added as follows:
   - Add the name of the bookmark as text to the document.
   - Mark the bookmark name in the text and add a bookmark that is applied to the text with a name identical to the marked text.

Note the following about adding bookmarks to the document:

- The Word template can only be used in the Alfabet Publication Framework (APF) if the text and the name of the bookmarks match identically.
- If you want to add a configured report or standard Alfabet page view including a legend to the document, you must add separate bookmarks for the configured report/page view and for the legend of the configured report/page view.
- If you add a configured report or page view via a bookmark, the report will be inserted as a graphic. The image is resized to fit the document width. If the length of the graphic supersedes the space available on the page, the lower part of the image may be hidden behind the footer defined for the document. This is the standard graphic import behavior of Microsoft Word and cannot be changed by Software AG.
- Other than the table of contents feature, you must not user any other Microsoft Word bookmark field functions in the document.

Using Microsoft Word, you can insert a table of contents in the document by defining the hierarchy of heading levels that are to be included. When creating and updating the table of contents, the information about the document content will be

automatically included in the table of contents. This mechanism can only be used if the formatting of the headings is based on styles.

You can add a table of contents to your template via two methods:

- Use the Microsoft Word mechanisms to add a table of contents anywhere in the Word template. The publication will have a table of contents at the specified position that reflects the current content.
- If you create the Word template without a table of contents, you can configure the publication definition in Alfabet Expand to add a table of contents to the document during publication. The table of contents will be inserted before the Word template. This method is not recommended for templates containing a title page because the table of contents will be located before the title page.

For information about the configuration of the publication definition to include a table of contents in the document, see *Mapping Alfabet Data to Word Templates*.

For either method, the table of contents will not be automatically filled during document creation. When a user opens a publication, he/she will see the text "TOC" where the table of contents has been added. The user must update the table of contents manually to view the contents. To do so, the user must click the word "TOC" and press F9 on his/her keyboard.

Mapping Alfabet Data to Word Templates

The mapping of Alfabet data to bookmarks in Word templates is carried out in the configuration tool Alfabet Expand.

Each publication definition is configured by creating explorer elements as sub-nodes of the **Publications** root node in the **Publications** tab. A publication consists of a publication node with the following child elements:

- A **Templates** folder that contains at least one template element. Each template element corresponds to a Word template. The template assigned to the template element is uploaded to the Alfabet database.
- A publication entry element that assigns the Word template to objects of a defined object class in the Alfabet meta-model. The entry element can contain the following sub-elements:
  - A **Mappings** folder must be available that contains one mapping element for each bookmark
  - Optionally, child publication entry elements can be added to define a hierarchy of publication steps

To configure mapping of data to a Word template:

1) In Alfabet Expand, go to the **Reports** tab.
2) In the explorer, right-click the root node **Publications** and select **Add publication**. A new node is added to the explorer and the attribute window displays the attributes of the new publication.
3) Define the following attributes of the new publication:

- **Name**: Enter a unique name for the publication. The name is a technical name and should not contain whitespaces and special characters.
- **Caption**: Enter a caption for the publication. The caption is displayed in the Alfabet user interface in the drop-down list of the publication selector. The **Name** attribute of the publication will be displayed if no caption is defined.
- **Comments**: Optionally enter a description for the publication.
- **Debug Arguments**: You can define a default for the parameter settings if:
  - publication is to be executed with the command line utility `PublicationConsole.exe`, and
  - the queries defined for the publication contain Alfabet parameters that are to be substituted with values defined for the Alfabet parameters via the command line.

  The required syntax is <parameter name>=<value>. If you want to define multiple parameter values, the entries must be comma-separated. The value must be defined in the exact syntax required to define the Alfabet parameter value in the query. For example, if strings are written in inverted commas in a query, the inverted commas must be part of the Alfabet parameter value. The parameter name is the name of the Alfabet parameter without the prefix @ or:

  For more information about the definition of Alfabet parameter values in the command line of the publication console application, see Defining Publication Output In the Command Line of the Batch Utility.

- **Data Culture**: Select the culture for the publication in the drop-down list. If no culture is selected, the default language English will be used for the publication. The **Data Culture** attribute is only required if object data translation has been activated for the **Support Data Translation** attribute of the relevant culture . If object data translation is activated for a culture, the **Name** and **Description** property of Alfabet may be translated to the language associated with the currently selected culture. Translated strings are stored in the database table of the object class in a separate column. For more information about the specification of cultures and translation, see the chapter Localization and Multi-Language Support for the Alfabet Interface.
- **Document Format**: If you select `Word`, the result of the publication will be a Microsoft Word document. If you select `PDF`, the result of the publication will be a PDF document.
- **Group**: You can optionally structure and organize publications in publication group folders. To create a publication group, enter a name for the publication group in the **Group** field and press ENTER. The publication group appears as a folder ascendant to the publication. If other publications are already structured in publication groups, existing publication groups can be selected from a drop-down list in the **Group** attribute. If an existing publication group is selected, the publication will be added to the selected publication group folder.
- **Insert Table of Contents**: Optionally, you can trigger the generation of a table of contents for the document by setting this attribute to `True`.

  In Microsoft Word you can insert a table of contents in the document by defining a hierarchy of the heading levels that are to be included. When creating and updating the table of contents, the information about the document content will be automatically included in the table of contents. This mechanism can only be used if the formatting of the headings is based on styles.

You can add a table of contents to your template via two methods:

- Use the Microsoft Word mechanisms to add a table of contents anywhere in the Word template. The publication will have a table of contents at the specified position that reflects the current content.
- If the Word template has been created without a table of contents, set the attribute **Insert Table Of Contents** of the publication definition to `True` to add a table of contents to the document during publication. The table of contents will be inserted before the Word template. This method is not recommended for Word templates containing a title page because the table of contents will be located before the title page.

With both methods, the table of contents will not be automatically filled during document creation. When a user opens a publication, he/she will see the text "TOC" where the table of contents has been added. The user must update the table of contents manually to view the contents. To update the table of contents, the user must click the word "TOC" and press F9 on the keyboard.

If you specify `True` for the **Insert Table of Contents** attribute, you should also expand the section **TOC Info** and edit the following attributes:

- **Start Level**: Define the first heading level that shall be included in the table of contents.
- **End Level**: Define the last heading level that shall be included in the table of contents.
- **Heading**: Enter the text that shall be displayed as heading text above the table of contents.
- **Heading Style Name**: Enter the style name that shall be used to format the heading of the table of contents. The style must exist in the style configuration of your Word template.
- **Separate Page**: Select `True` if a page break shall be inserted after the table of contents.

4) Expand the new publication node in the explorer and right-click the **Templates** folder.
5) In the context menu, select **Add template**.
6) In the explorer that opens, select the Word template that you want to use for the publication of data. The Word template is added as a new sub-node to the **Templates** folder.

The template is uploaded to the database when you select it in the explorer. Once the template is uploaded to the database, the following is possible by right-clicking the template node:

- **Show Publication Template**: Select this option to open a preview of the template in Alfabet Expand. Please note that only one template can be displayed in the middle pane. If you use the option on a publication template while another publication template is aready displayed, the preview of previously displayed template is substituted with the preview of the selected publication template.
- **Reload Publication Template**: Select to update changes made to the template stored in the local file system. Any changes performed after the template has been assigned to the publication configuration must be reloaded in order to be

updated to the solution configuration. If you select **Reload Template**, an explorer will open that allows you to select the changed document in the local file system. After upload, the mapping nodes of an existing configuration will be automatically adapted to the new template without any loss to the existing configuration. Renaming a bookmark requires re-configuration of the bookmark mapping because the old bookmark will be regarded as deleted and the new name will be interpreted as a new bookmark.

- **Save Publication Template to Disk**: Select this option to export a copy of the uploaded template to the local file system.
- **Delete**: Select this option to delete the uploaded template from the database. If you delete a template, the mapping for the template will also be deleted. All mapping nodes and the configuration in the nodes will be removed from the publication node.

Information about the creation and last update performed on the Word template node are available in the attribute window of the template node. Expand the **Tech Info** attribute to view the information.

7) In the attribute window, define a unique name for the Word template in the **Name** attribute The name is a technical name and should not contain whitespaces and special characters.
8) Right-click the publication node and select **Add Publication Entry**. The publication entry is added as a new sub-node to the publication node and the attributes are displayed in the attribute window.
9) In the attribute window, define the following attributes:
   - **Name**: Enter a unique name for the publication entry.
   - **Parameter Name**: Enter a name that can be used as parameter name in queries. The parameter name can be used for any publication entries in the publication configuration to refer to the `REFSTR` of the current object found for this publication entry.

     For more information about how to define Alfabet parameters in Alfabet queries, see the section Referring to the Current Alfabet Context in a WHERE Condition in the chapter Defining Queries.

     For more information about how to define Alfabet parameters in native SQL queries, see the section Using Alfabet Parameters in the section Defining Native SQL Queries of the chapter Defining Queries.

   - **Source Type**: Select one of the following:
     - `None`: Select this option if you want to publish data about the base object of the publication. The base object is the object that the user selects to trigger the publication for in a configured report or the object the user is currently working with if he/she triggers the publication via a configured button in the object profile.
     - `Query`: Select this option if you want to publish data about the objects found by the Alfabet query. If this option is selected, the Alfabet query must be configured to find the base objects for the publication in either the **Alfabet Query** attribute or the **Alfabet Query as Text** attribute.

- `NativeSql`: Select this option if you want to publish data about the objects found by a native SQL query. If this option is selected, the native SQL query must be configured to find the base objects for the publication in the **Native Query** attribute.

  If the **Source Type** attribute is set to `None`, the publications can neither be tested in Alfabet Expand nor published with the publication console application. If you want to test a report that should be based on a base object selected by the user (**Source Type** attribute = `None`), you must simulate the user selection during the configuration: To do so, select `Query` or `NativeSql` in the **Source Type** attribute and define a query that finds one single object of the base object class. After successful testing, you can change the **Source Type** attribute to `None`. If you want a publication to be executed with the publication console application with the selection of a different base object for each publication, you can select the option `Query` or `NativeSql` in the **Source Type** attribute and define an Alfabet parameter in the query. Current Alfabet parameter values can be selected in the command line of the publication console application. For more information, see the section Defining Publication Output In the Command Line of the Batch Utility.

- **Native Query** / **Alfabet Query** / **Alfabet Query as Text**: If the **Source Type** attribute is set to `Query` or `NativeSql`, you must define a query that finds the object(s) for which data should be published. The parameter `BASE` can be used in the query to refer to the `REFSTR` of the object that the user has currently selected for publication.

  For more information about how to define Alfabet queries or native SQL queries in the context of Alfabet configurations, see the chapter Defining Queries.

- **Template**: Select the Word template to use for the publication of data. The drop-down list displays all uploaded Word templates.

- **Use separate page**: Select `True` if you want a line break to be added to the document when starting publication of a new object of this level.

10) After having selected a Word template for the **Template** attribute of the publication entry, a **Mappings** folder with one mapping node for each bookmark found in the Word template will be added automatically to the publication entry node.

    To check the position of the bookmark you are currently mapping to the Word template, right-click the mapping node and select **Show bookmark in template**. A document preview opens in the center pane displaying the page that contains the currently selected bookmark.

    In the **Source Type** attribute of the mapping node, map the bookmark to the Alfabet data:

- `Property`: Select to display the current value of a defined object class property.

- `GraphicView`: Select to display a graphic view with the current object as base object. If the graphic view should have a legend in the publication, the legend must be added with a separate bookmark mapped to a graphic view legend.

- `GraphicViewLegend`: Select to display the legend of a standard graphic view.

- `CustomReport`: Select to display a configured report with content for the current object as base object. Configured reports must be applied to an object class to be displayed in a publication. If the report should have a legend in the publication, the legend must be added with a separate bookmark mapped to a configured report legend.

Console Reports, that means reports that display a number of subordinate reports in a table layout, cannot be added to publications. When a console report is added to a publication, the first subordinate report in the console report is displayed in the publication and all other reports are ignored.

- `CustomReportLegend`: Select to display the legend of a configured report.
- `Query`: Select to display the dataset returned by an Alfabet query defined in the **Alfabet Query** attribute or the **Alfabet Query as Text** attribute.
- `NativeQuery`: Select to display the dataset returned by an SQL query defined in the **Native Query** attribute.

11) Define the attributes required for the value selected in the **Source Type** attribute for the mapping node. Only the relevant values for the selected **Source Type** attribute will be displayed in the **Source** section of the attribute window:

- **Property Path**: Define the object class property for which the current value should be published. The object class property to be displayed can be an object class property of the current object or an object class property of a related object. Each object class must be defined by the **Parameter Name** specified in the publication entry. The object class and object class property must be separated by a period.

  To display an object class property of the object class defined in the current publication entry, define < **Parameter Name** value>.<object class property name> (For example, the start date of the current application with a **Parameter Name** definition of `App` in the current publication entry):

  ```
  App.StartDate
  ```

  To display a property of a superordinate object in the hierarchy (For example, the name of the application group that the application is assigned to with a parent publication entry that defines the **Parameter Name** of the application group as `AppGroup`):

  ```
  AppGroup.Name
  ```

- **View**: Select the Alfabet page view that you want to display. The drop-down list includes all views available for Alfabet. You must ensure that the selected view is relevant for the current object.

  To evaluate which page views are relevant for an object class, you can consult the object view definition of the object class. The standard object views for an object class that are visible in the **Object View** explorer node in the **Presentation** tab in Alfabet Expand contain the relevant page views for the respective object class.

  Note the following about the display of views in the publication:

  - If a view has filters, the view will be executed with empty filter fields. If a default is available for a filter setting, the default will be used. The filter settings currently stored for the user triggering the publications will be ignored. If a report shall be executed with defined filter settings, you can either use a configured report that returns a dataset that would result from the filter settings on the standard view or define a query to return the required result via either the **Alfabet Query** attribute, **Alfabet Query as Text**

attribute, or **Native Query** attribute. If you define the return dataset via a query, you can use Alfabet parameters to refer to variables that are specified either in the command line of the batch job triggering the publication or in the **Debug Arguments** attribute of the report. This allows filters to be simulated for a report when triggering the publication via the command line.

- Grouped views allow the user to expand or collapse sections of the report to view or hide subordinate information. These are displayed with all collapsible levels expanded to display all available information in the publication. This applies to Gantt charts and expandable tabular reports, For example,.
- The scaling of graphic reports is identical to the scaling used in the Alfabet user interface if the resulting image size fits on the document page. If the image is larger then the width of the document page, the image is resized to fit the page.

- **Apply to Class**: If you want to map the bookmark to a configured report applied to an object class via the **Apply to Class** attribute of the configured report, you must first select the object class that the configured report is applied to.
- **Custom Report**: Select a configured report to map the bookmark to a configured report. If you select an object class in the **Apply to Class** attribute, the drop-down list will display all configured reports that apply to the defined class. If the attribute **Apply to Class** is not specified, the drop-down list will display all configured reports that are not applied to the class Reports.

  Note the following when adding a configured report to a publication:

  - Reports that are configured to trigger publications cannot be selected.
  - The rules that apply to the display of standard Alfabet views (listed above) also apply to the display of configured reports.

- **Alfabet Query/Alfabet Query as Text**: Define an Alfabet query either in the **Alfabet Query** attribute using the **Alfabet Query Builder** or in the **Alfabet Query as Text** attribute using a text editor. The dataset returned by the Alfabet query is displayed in the publication. The attributes available in the **Source** section allow you to format the display of the dataset. The **Parameter Name** values of this or any other publication entry in the publication definition can be used in the query as a parameter in order to return the `REFSTR` of the current object of the respective publication level.

  For more information about how to define Alfabet queries, see the chapter Defining Queries.

- **Native Query**: Define a native SQL query. The dataset returned by the native SQL query is displayed in the publication. The attributes available in the **Source** section allow you to format the display of the dataset. The **Parameter Name** values of this or any other publication entry in the publication definition can be used in the query as a parameter in order to return the `REFSTR` of the current object of the respective publication level.

  For more information about how to define native SQL queries in the context of Alfabet configurations, see the section Defining Native SQL Queries in the chapter Defining Queries.

- **String Formatting Type**: Select one of the following types of formatting for the dataset returned by the defined query:
  - `DataSet`: Select to display a tabular output of the query results.
  - `DelimitedString`: Select to display search results in a string with a defined delimiter between each column value of the returned dataset. Whether data is displayed in a new row for each dataset or without line breaks between the returned datasets will depend on the setting of the **Use New Line For Each Row** attribute
  - `FormattedString`: Select to display search results as variable part of a static string defined with the attribute **Format String**.
- **Format String**: Define the string that shall be displayed for the search results of the defined query. To display current query results in the string, add a variable in the format {dataset column name}.

  The name and caption of a dataset column are not necessarily identical. Which value must be taken over depends on how the Alfabet query or a native SQL query is defined. For more information about how to identify the correct technical name of a dataset column, see the section Defining Column Names and Captions in the chapter Defining Queries.

  For example, data about an application is to be published. For the current bookmark, the name of the ICT object that the application is assigned to shall be published as part of a sentence. The ICT object is found by a native SQL query definition:

  ```
  SELECT icto.REFSTR, icto.NAME as "ICTObject"
  FROM ICTOBJECT icto, APPLICATION app
  WHERE app.REFSTR = @BASE
  AND app.ICTOBJECT = icto.REFSTR
  ```

  The text string is defined as follows to include the name of the ICT object:

  ```
  The application is assigned to the ICT object {ICTObject}.
  ```

- **Delimiter**: Define the delimiter between values within a dataset. By default, values are delimited with a space.
- **Use New Line for Each Row**: Select `True` to display each dataset returned by the query in a new line. Select `False` to display all results without line breaks between datasets.

12) You can optionally add additional entries to the publication either in parallel to the first publication entry or as sub-elements of the already configured publication entry. Note the following when configuring publications with multiple entries:

- Only publication entries of the first level can have the **Source Type** attribute set to `None`.
- Publication entries that are child elements of parent publication entries must have the **Source Type** attribute set to `Query` or `NativeSql`. An Alfabet query or native SQL query must be defined by means of the respective **Alfabet Query (as Text)** attribute or **Native Query** attribute. The query must relate the objects found for the current level with the objects found for the parent level. The Alfabet query language parameters (For example, the parameter `BASE`) that refers to the current object of the parent level can be used in the query.

In addition, the query can refer to any ancestor object in the hierarchy of publication entries using the value defined in the **Parameter Name** attribute as the parameter to refer to the `REFSTR` of the ancestor object.

13) In the toolbar of Alfabet Expand, click the **Save** button to save your changes.

14) In the explorer, right-click the publication node and select **Test Publication**. If the publication configuration is incorrect, an error message will be displayed that provides information about the location of the error. If the publication configuration is correct, a process information will be displayed. When you confirm the process protocol, the word document with the current data from the Alfabet database will open in Microsoft® Word. If the results are not as expected, you can edit the publication configuration until the output of the publication meets your expectations.

Please note that the publishing protocol dialog is displayed in the language set in the **Data Culture** attribute of the publication independent from the language settings selected for the Alfabet Expand user interface.

15) Click the publication node and select **Set Publication State to 'Active'**. Only publications with a **Publication State** attribute set to **Active** can be displayed and accessed in the Alfabet user interface.

Publications with the **State** attribute set to `Active` cannot be edited in Alfabet Expand. If you want to change a publication configuration that is currently `Active` and used on the Alfabet user interface, re-set the **Publication State** attribute to `Plan` by right-clicking the publication node. The publication configuration can then be edited. As long as the **Publication State** attribute is set to `Plan`, the publication will not be available in the Alfabet user interface. It will not be displayed in drop-down lists displaying available configurations. Functionalities that directly execute the publication will throw an error message informing the user that the publication is currently inactive. Publications in with the **Publication State** attribute set to `Retired` can neither be executed in the Alfabet user interface nor edited in Alfabet Expand.

The options to change the state of publications are also available on publication group folders. If you use the options on the publication group folder, the status change will be applied to all publications in the publication group folder.

Publishing Documents in Different Languages

Alfabet can be configured to translate strings displayed on the user interface and the **Name** and **Description** properties of objects defined by the Alfabet user community. When the user logs in to the Alfabet user interface, the user interface will display the language associated with the default culture defined either in the user's personal settings or defined in the server alias configuration used for login to the Alfabet Web Application. The user can switch to a different culture via the **Language** button in the upper-right corner of the Alfabet interface.

If multi-language support is available for your Alfabet user community, then the publications should also be available in the various languages that are supported. A separate publication must be configured for each language for which you want to provide a translated publication.

The following is recommended to generate a language version for a publication configuration:

1) Create a copy of the Word template(s) assigned to the original publication definition and translate all static content to the target language.

2) Open Alfabet Expand and go to the **Reports** tab.

3) Right-click the original publication and select **Copy**.

4) Right-click the new **Publications** root node and select **Paste**. A copy of the original publication is added to the explorer tree with the name **<source publication name> - 1**.

5) If the **State** attribute of the new publication is not set to `Plan`, right-click the new publication and select **Set State to Plan**.

6) Define the following attributes of the new publication:

   - **Name**: Enter a unique name for the publication. The name is a technical name and should not contain whitespaces and special characters. The name should reflect that the publication is a language version of the original publication (For example, by defining <source publication name>-<language code>).

   - **Caption**: Enter a caption for the publication. The caption is displayed in the Alfabet user interface in the drop-down list available for the **Publication** field. The **Name** attribute will be displayed if no caption is defined. It is recommended that you define a name that allows the publication to be identified as a language version of the source publication (For example, "Publication X (German)"). The caption can be defined in the base language of the meta-model (which is English). It can later be translated to other languages via the **Translation Editor** available in Alfabet Expand. For more information about the translation capability available in Alfabet Expand, see the chapter Localization and Multi-Language Support for the Alfabet Interface.

   - **Comments**: Optionally enter a description for the publication. The description can be defined in the base language of the meta-model. It will later be translated to other languages via the translation editor of Alfabet Expand.

   - **Data Culture**: Select the culture for the publication in the drop-down list. If no culture is selected, the default language English will be used for the publication. The **Data Culture** attribute is only required if instance translation has been activated for the **Support Data Translation** attribute of the relevant culture. If instance translation is activated for a culture, the **Name** and **Description** property of Alfabet may be translated to the language associated with the currently selected culture. Translated strings are stored in the database table of the object class in a separate column. For more information about the specification of cultures, see the chapter Localization and Multi-Language Support for the Alfabet Interface.

   - **Group**: You can optionally structure and organize publications in publication group folders. To create a publication group, enter a name for the publication group in the **Group** field and press ENTER. The publication group appears as a folder ascendant to the publication. If other publications are already structured in publication groups, existing publication groups can be selected from a drop-down list in the **Group** attribute. If an existing publication group is selected, the publication is added to the selected publication group folder.

7) Expand the new publication node in the explorer and expand the **Templates** node.

8) Right-click a Word template and select **Reload template**.

9) In the explorer that opens, select the translated version of the Word template. The Word template is substituted with the language version.

The template is uploaded to the database when you select it in the explorer. Once the template is uploaded to the database, the following is possible by right-clicking the template node:

- **Show Publication Template**: Select this option to open a preview of the template in Alfabet Expand. Please note that only one template can be displayed in the middle pane. If you use the option on a publication template while another publication template is aready displayed, the preview of previously displayed template is substituted with the preview of the selected publication template.
- **Reload Publication Template**: Select to update changes made to the template stored in the local file system. Any changes performed after the template has been assigned to the publication configuration must be reloaded in order to be updated to the solution configuration. If you select **Reload Template**, an explorer will open that allows you to select the changed document in the local file system. After upload, the mapping nodes of an existing configuration will be automatically adapted to the new template without any loss to the existing configuration. Renaming a bookmark requires re-configuration of the bookmark mapping because the old bookmark will be regarded as deleted and the new name will be interpreted as a new bookmark.
- **Save Publication Template to Disk**: Select this option to export a copy of the uploaded template to the local file system.
- **Delete**: Select this option to delete the uploaded template from the database. If you delete a template, the mapping for the template will also be deleted. All mapping nodes and the configuration in the nodes will be removed from the publication node.

Information about the creation and last update performed on the Word template node are available in the attribute window of the template node. Expand the **Tech Info** attribute to view the information.

10) If the publication shall be triggered via a configured report, go to the **Reports** section of the explorer and select the configured report that triggers the publication in the user interface.
11) Right-click the **Publications** sub-folder of the report node and select **Add Publication**. Select the newly created language version of the publication and click **OK**. The language version of the publication is now available for selection in the report.
12) In the **Translation Editor** available in Alfabet Expand, **translate the caption and description of the original and the language version of the publication to the language of the translated publication.** For more information about the translation of strings available in the Alfabet user interface, see the section Modifying, Translating and Managing the Vocabularies in the chapter Localization and Multi-Language Support for the Alfabet Interface..

Configuring Access to the Publication Functionality

After the publication definition based on Word templates has been configured, you must define a configured report that allows the user to trigger the publication. The publication functionality can be included directly in a configured report that displays the report results or the report can be configured to trigger publications only.

Publication can be performed in the Alfabet interface using one of the following methods:

- A configured report of the type `Query`, `NativeSql`, or `Custom` that displays a dataset that is returned by the query. The user can select one or multiple objects in the dataset and then trigger a

publication. The publication functionality is displayed below the dataset and includes the necessary buttons to trigger the publication.

The objects selected in the dataset are the base objects for the publication. For first level publication nodes with a **Source Type** attribute set to `None`, the data about the selected object will be published. For first level publication entries with a **Source Type** attribute set to `Query` or `NativeQuery`, the selected objects can be referenced by means of the parameter `BASE` in the query defined for the publication entry.

- A configured report that displays only the publication functionality. The user can trigger the publication for base objects that are selected by one of the following methods:
  - The first level publication entries of the publication has the **Source Type** attribute set to `Query` or `NativeSql` and the query defines the objects to be selected without a reference to a base object (in other words, without using the parameter `BASE`).
  - The configured report is assigned to an object class via the **Apply to Class** attribute of the configured report. A selector is then displayed in the publication interface that allows an object to be selected as the base object for the configured report. The first level publication entries of the publication can either have the **Source Type** attribute set to `None` and configured to publish data about the current object, or have the **Source Type** attribute set to `Query` or `NativeQuery`. The query references the selected base object using the parameter `BASE`.
  - A custom button is configured for an object profile that opens the configured report triggering the publication and the **Operation Apply To** attribute of the button is set to `BaseInstance`. The object the user is currently working with is the base object of the report. The first level publication entries of the publication can either have the **Source Type** attribute set to `None` and configured to publish data about the current object, or have the **Source Type** attribute set to `Query` or `NativeSql`. The query can reference the selected base object using the parameter `BASE`.

For information about the creation of custom buttons to open a configured report, see the ssection Configuring Custom Buttons for the Toolbar of a Custom Object View in the chapter Configuring Object Views.

The following information is available:

- *Defining a Configured Report for Base Object Selection and Publication*
- *Defining a Configured Report to Trigger Publications Only*

Defining a Configured Report for Base Object Selection and Publication

To define a configured report to provide both a report for selection of objects and an interface to trigger publications:

1) In the **Reports** tab of Alfabet Expand, either:
   - Create a new configured report that allows the data to be selected for which you would like to trigger publication, or
   - Go to an existing report that matches the required configuration and make sure that the **State** attribute is set to `Plan`.

For a detailed description of how to configure a configured report, see the chapter Configuring Reports.

2) Right-click the configured report and select **Add Publication**.

3) In the selector that opens, select one or multiple publication configurations that should be triggered from the configured report. A folder **Publications** is added to the configured report with the publication configurations added as sub-nodes to the folder.

To add additional publication configurations to the configured report, right-click the **Publications** folder and select **Add Publication**.

To remove a publication configuration from the configured report, right-click the publication configuration node in the **Publications** folder and select **Detach Publication**.

4) Right-click the configured report and select **Add Publication Control**. A new window opens.

5) In the **Add Publication Control** window, select the position of the publication interface in the configured report with the **Dock** attribute.

6) In the toolbar of Alfabet Expand, click the **Save** button to save your changes.

7) Optionally, right-click the report view node and select **Design View**. The report view opens in the center pane and shows both the presentation object of the configured report and the presentation object for the publication interface. You can resize the presentation objects, if necessary.

8) In the toolbar, click the **Save** button to save your changes.

9) Right-click the configured report and select **Set State to Active**.

Defining a Configured Report to Trigger Publications Only

To define a configured report that only provides an interface to trigger publications:

1) In the **Reports** tab, right-click the **Reports** folder or a sub-folder and select **Create New Report**. You will see the new configured report listed below the selected folder.

As an alternative to defining all attributes for a new configured report, you can copy and paste an existing configured report and make modifications, as needed. To do so, right-click the existing configured report that you want to copy and select **Copy**. Right-click the new configured report and select **Paste**. Modify the necessary attributes described below.

2) In the property window, define the following attributes for the configured report:

- **Name**: Optionally, change the technical name of the configured report. The technical name of the configured report must be unique.

The technical name is used for identification of the configured report in the context of technical processes such as merging the report configuration with a report configuration saved as an XML file. Therefore, the following is required when defining the name of a configured report:

- The name must be unique. The name must be unique regardless of the report folder it is structured in.
- The name may not be changed once a configured report has been created.
- A validation mechanism checks for correct syntax when defining a technical name. Please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.
- If the technical name of the configured report is changed, the name will be correctly updated in other configuration objects referencing the changed report during design time.

Please note that the name of a changed configuration object will not be updated in guide pages or guide views. If you plan to change the name of a configuration object, the reference in the navigation page should be changed prior to changing the configuration object name. You can use the **Show Usage** functionality in the context menu of the configured report to evaluate whether the configured report has been added to a guide view or guide page.

- **Caption**: Enter a caption for the configured report. The caption you define here will be displayed in the **Reports Administration** functionality in the **Administration** module and the **Configured Reports** views of the Alfabet interface. If the configured report is assigned to an object view as a page view, the text will be displayed as the page view caption in the object view. The caption of the configured report may exceed the conventional 64 character limit.
- **Description**: Provide a short information about the configured report that is useful to Alfabet users. This comment will be displayed on the Alfabet user interface in the header of the configured report in a single line under the report caption and the **Description** field for the configured report in the table of all views listing configured reports, like the **Configured Reports** views of the **Search** functionalities. If the configured report is assigned to an object view as a page view, the text will be displayed under the page view caption as short description in the object view.

  In the configured report, the description will be truncated if it is longer than one line on the screen. Therefore, the description should be short to ensure complete display in the configured report header.

- **Usage Guideline**: Provide information that helps users to execute and interpret the configured report. This information will not be displayed directly on the user interface. The user can access the information using the following mechanisms:
  - If the user moves the cursor over the description provided for the configured report with the attribute **Description**, a tooltip opens that includes both the description and the usage guideline.
  - The **Options** button will be displayed on the upper right with an option **Help On Filter Fields** to open a new window displaying the usage guideline in addition to any filter field hints, if configured.

    For views that have filters defined, the **Options** button will only be displayed if the filter panel either allows batch clearing of filter fields or collapsing

and expanding of the filter panel. For more information about the configuration of these functionalities, see Configuring the Complete Filter Area to be Collapsible and Allowing the User to Clear the Filter Area.

- **Technical Comment**: Provide information that is useful for solution designers working with Alfabet Expand in order to better understand the configured report from a technical perspective. The **Technical Comment** will not be displayed in the user interface. Nevertheless, you can configure the interface to display the information in, For example, the attribute section of the configured report's object profile in the **Reports Administration** functionality.
- **Type**: Select `Query`. This is the default for new configured reports.
- **Report State**: The **Report State** attribute is view only and is set to `Plan` for new configured reports. The configured report can only be edited when the **Report State** attribute is set to `Plan`. After finishing all configuration steps described in the following, the **Report State** attribute must be set to `Active` as described in the section General Guidelines for Creating Configured Reports. The configured report is then visible to users in the Alfabet user interface.
- **Help Index**: Specify the location of the external Help file using the following syntax:(For example,:). The external help file will be displayed when the user clicks the context-sensitive Help link in the standardHelp menu.

  Server variables can be used in the Help links. Server variables allow you to define all or part of the URL definition in the server alias configuration of the Alfabet Server instead of directly defining it in the configuration in Alfabet Expand. Use of server variables is useful, For example, in order to change a Web server name defined in multiple URL definitions in Alfabet Expand. Instead of changing all URL definitions, only the variable value in the server alias configuration must be updated. The use of server variables is described in the section Using Server Variables in Web Link and Database Server-Related Specifications in the chapter Configuring Reports.

- **Apply to Class**: To apply the configured report to one specified object class, click the **Browse** button to open a dialog box and select the object class to which you want to apply the configured report and click **OK**. If you set the attribute, a selector will be displayed at the top of the configured report. The user can select a base object for the report in the selector.
- **Base Classes**: Define the object classes that must be available to run the configured report. If this attribute is set, the Alfabet Server checks whether the object classes specified as base classes are excluded from the view scheme used to access the configured report. If the user does not have access permissions to one of the base classes, the configured report will be disabled. To set the attribute, click the **Browse** button to select the object classes that are required to run the configured report and click **OK**.

3) Right-click the configured report and select **Add Publication**.
4) In the selector that opens select one or multiple publication configurations that should be triggered via the configured report. A **Publications** folder is added to the configured report. The publication configurations are added as sub-nodes to the folder.

   To add additional publication configurations to the configured report, right-click the **Publications** folder and select **Add Publication**.

To remove a publication configuration from the configured report, right-click the publication configuration node in the **Publications** folder and select **Detach Publication**.

5) Right-click the configured report and select **Create Custom Report View**. The configured report view is added to the configured report and opens in the center pane.
6) Delete all controls and elements from the report view.
7) In the toolbar of Alfabet Expand, click the **Save** button to save your changes.
8) Right-click the configured report and select **Add Publication Control**. A new window opens.
9) In the **Add Publication Control** window, set **Dock** to `Fill`. All other fields must be left empty.
10) Click **OK** to apply your changes.

The new control is only visible in the report view displayed in the center pane if you close and reopen the report view in the middle pane.

11) In the toolbar of Alfabet Expand, click the **Save** button to save your changes.

Triggering Publication Via a Batch Utility

APF allows you to directly publish data to the local file system using a batch utility.

Software AG provides a Windows® command line tool `PublicationConsole.exe` that allows a publication to be triggered via a command line or by means of a Windows® batch job.

A publication must be completely configured and the **State** attribute set to `Active` before starting a publication job. The publication must be defined in the Alfabet database that the Alfabet Server connects to and must be defined to be independent of the selection of a base object. If you want to publish data about different objects with the same publication via the batch job, you can define Alfabet parameters in the queries of the publication definition that are substituted with values defined in the command line when starting the publication console.

For information about running a publication job with `PublicationConsole.exe`, see *Batch Processes Relevant for the Alfabet Publication Framework* in the reference manual *System Administration*.

Defining Publication Output In the Command Line of the Batch Utility

If a publication is to be triggered via the `PublicationConsole.exe`, the selection of the base objects of the publication must be specified in a query in the publication definition. This results in a publication that returns results for the same set of base objects each time it is executed.

If you want to select different base objects for each publication with `PublicationConsole.exe`, you can use Alfabet parameters in the queries selecting the base object as well as in the queries defining the data to be published about the objects. The value of the Alfabet parameter is then defined in the command line of the publication console application when triggering a publication.

For more information about how to define Alfabet parameters in Alfabet queries, see Referring to the Current Alfabet Context in a WHERE Condition in the chapter Defining Queries.

For more information about how to define Alfabet parameters in native SQL queries, see Using Alfabet Parameters in the section Defining Native SQL Queries of the chapter Defining Queries.

The following configuration is required to use Alfabet parameters in the command line:

- When defining the publication, define queries that contain Alfabet parameters in `WHERE` conditions.
- Optionally, define a default value for each Alfabet parameter in the **Debug Arguments** attribute of the publication. The definition must be defined in the following syntax:
  - Each default must be defined as <parameter name>=<value>.
  - When multiple default values are added, they must be comma-separated.
  - The <parameter name> does not include the prefix @ or: of the Alfabet parameter.
  - The <value> must be written in the attribute as if defined in a query. For example, if a string is written in inverted commas in the query, the inverted commas must be part of the value specification.
- When starting the command line application, the value substituting the Alfabet parameter must be defined in the command line as -<parameter name> <value>. The parameter name is the Alfabet parameter without the prefix @ or: and the value must be defined as required by the query syntax. In other words, a string written in inverted commas in the query must be defined with the inverted commas.

A publication triggers the publication of data about a number of applications found via their name. The query finding the application is defined in the publication entry as:

```
ALFABET_QUERY_500

FIND Application

WHERE Application.Name LIKE:AppName
```

The **Debug Arguments** attribute of the publication definition specifies a default value for the application name. If no parameter value is defined in the command line, the publication will be created for all applications with a name that starts with `CMS`:

```
AppName='CMS%'
```

If you were to start the `PublicationConsole.exe` in order to create the publication about an application named "`Central Server`", the command line must look like this:

```
PublicationConsole.exe -msalias Production -alfaLoginName
SystemAdmin -alfaLoginPassword AdminPassword -publication
ApplicationData -outputfile CentralServer.zip -AppName 'Central
Server'
```


Chapter 19: Defining Queries

Queries allow for objects to be efficiently searched for and retrieved in Alfabet based on object attributes and relationships. Queries can be implemented in a wide variety of contexts, thus allowing the solution to be customized for many of the Alfabet capabilities. Queries can be written in either the Alfabet query language or in native SQL. The table below provides an overview of when native SQL and the Alfabet query language can be used.

Query Context	Description	Native SQL	Alfabet Query Language
Custom Reports	The reports functionality allows you to configure customized reports about objects in the Alfabet database. Queries in configured reports are also used to allow users to quickly and efficiently search for objects via custom filters. For detailed information about configuring reports, see Configuring Reports.	Yes	Yes
Custom Selectors	Queries in custom selectors are used to allow users to quickly and efficiently search for objects via custom filters. For detailed information about configuring custom selectors, see the section Configuring a Custom Selector for Search Functionalities.	Yes	Yes
Custom Editors	Queries in custom editors are used to specify the content in various filter options such as combo-boxes, checked list boxes, edit searches, and radio group buttons. For detailed information about configuring custom selectors, see the section Configuring Custom Editors.	Yes	Yes
Workflows	The workflow functionality may require queries in order to find objects that are the target of a workflow, users responsible to perform a workflow step, pre-conditions or post-conditions for a workflow step, or objects to be updated in the context of the workflow. For detailed information about configuring workflows, see the chapter Configuring Workflows.	Yes **NOTE**: For specific information about the use of native SQL queries in the context of workflows, see the section Special Rules for the Specification of Native SQL in the Context of Alfabet Configurations.	Yes
Wizards	The wizard functionality may require queries in order to change the target object of the wizard, pre-conditions or post-conditions for a wizard step, or objects to be updated in the context of the workflow. For detailed information about	Yes **NOTE**: For specific information about the use of native SQL queries in the	Yes

Query Context	Description	Native SQL	Alfabet Query Language
	configuring wizards, see the section Configuring Wizards.	context of wizards, see the section Special Rules for the Specification of Native SQL in the Context of Alfabet Configurations.	
Object Views	Queries can be specified to find a property to display in the **Attributes** section that is not a standard. For detailed information about the configuration of object views and their object cockpits, see the section Configuring Object Views.	No	Yes
Object Cockpits	Queries can be specified to fetch a property or validate complete data via a check entry in the object cockpit. For detailed information about the configuration of object views and their object cockpits, see the section Configuring Object Cockpits for a Custom Object View in the chapter Configuring Object Views.	Yes	Yes
Custom Explorer	Queries are specified to determine the content of explorer nodes in custom explorers. For information about configuring custom explorers, see the section Configuring Standard Business Functions and Custom Explorers.	Yes. **NOTE**: For specific information about the use of native SQL queries in the context of custom explorers, see the section Special Rules for the Specification of Native SQL in the Context of Alfabet Configurations.	Yes
Rule-Based Access Permissions	Read/Write access to objects can be granted by means of permission rules based on a query. Permission rules are defined in the XML object ***AlfaRightsManager*** in Alfabet Expand. For more information, see the section Configuring Permission Rules for Access to Objects.	Yes **NOTE**: For specific information about the use of native SQL queries in the context of rule-based access permissions, see the section Special Rules for the Specification of Native SQL in the Context of Alfabet Configurations.	Yes

Query Context	Description	Native SQL	Alfabet Query Language
Full-Text Search	Queries are required in order to find the objects indexed for the full-text search. For more information, see the section Configuring the Full-Text Search Capability.	Yes	Yes
Compliance Policy	Compliance policies determine the rules to find the objects to be targeted in a compliance project and the users responsible to provide information about the objects. Alfabet queries can be defined in order to search for the objects and persons relevant for the compliance project. For more information about the configuration of a compliance policy, see the section Configuring Queries for Compliance Policies in the chapter Configuring Alfabet Functionalities Implemented in the Solution Environment.	Yes	Yes
Computation Rules	A computation rule may be based on a query that allows for the traversal across object relationships to find content to be evaluated. For detailed information about defining computation rules, see the section *Specifying Computation Rules for Indicator Types* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.	Yes	Yes
Color Rules	A color rule consists of one or more Alfabet query-based rules that highlight a found set of objects. Color rules can be configured to be applied to objects in several matrix reports available for the object class Map View as well as to objects displayed in standard Alfabet diagrams in which a diagram view is selected. For more information about the configuration and implementation of color rules, see the *Configuring Color Rules for Map Views and Diagram Views* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.	Yes	Yes
Text Templates	Queries can be added to text templates in order to add content to the text such as a document link, list of relevant objects, or contact information about the sender or recipient of the email. For more information about the configuration of text templates, see the section Configuring Text Templates for Email Notifications in	Yes	Yes

Query Context	Description	Native SQL	Alfabet Query Language
	the chapter Configuring Alfabet Functionalities Implemented in the Solution Environment..		
Notification Monitors	A notification monitor alerts users about objects. Queries define the properties of an object class that are monitored as well as the users who are the recipients of the notification sent when the monitor is triggered. For more information about the configuration of notification monitors, see the section *Defining Notification Monitors* in the reference manual *User and Solution Administration*.	Yes	Yes
Consistency Monitors	A consistency monitor is based on a query that defines the object classes targeted by the query as well as the inconsistent properties to be detected. For more information about the configuration of consistency monitors, see the section *Defining Consistency Monitors* in the reference manual *User and Solution Administration*.	Yes	Yes
Technical Environment Items	A technical environment describes a specific area of the technology required to support the development, maintenance, operation, or testing of an object. The query searches for a set of objects that may be defined as elements in the technical architecture definition. For more information about the configuration of notification monitors, see the section *Configuring Technical Environment Definitions* in the reference manual *User and Solution Administration*.	No	Yes
Alfabet Publication Framework (APF)	Queries are used in APF to select objects and define the data to be published in Microsoft® Word®.	Yes	Yes
Alfabet Data Integration Framework (ADIF)	Queries are used in ADIF for the batch import, export, and manipulation of data in the Alfabet database.	Yes	No

The following information is available:

- Defining Alfabet Queries

Defining Alfabet Queries

By means of a class-oriented query language, Alfabet users can efficiently search for objects by defining Alfabet queries. This query language uses the CODBI (Class Oriented Database Interface) standard, which allows direct access to the model structure. Knowledge of the SQL language is not necessary to create or use Alfabet queries. Once an Alfabet query is defined, the Alfabet platform will translate the Alfabet query into native SQL and send it to the database. The Alfabet query searches for the relevant objects and generates a result list comprised of the objects and relevant properties.

The method to build Alfabet queries is similar in all contexts mentioned above. This chapter provides a general instruction about the Alfabet query language/syntax as well as how to operate the tool **Alfabet Query Builder** that supports the definition of valid Alfabet queries.

In Alfabet Expand, Alfabet queries are generally defined in the attribute window of a configuration object. The following example displays the property window of a report:

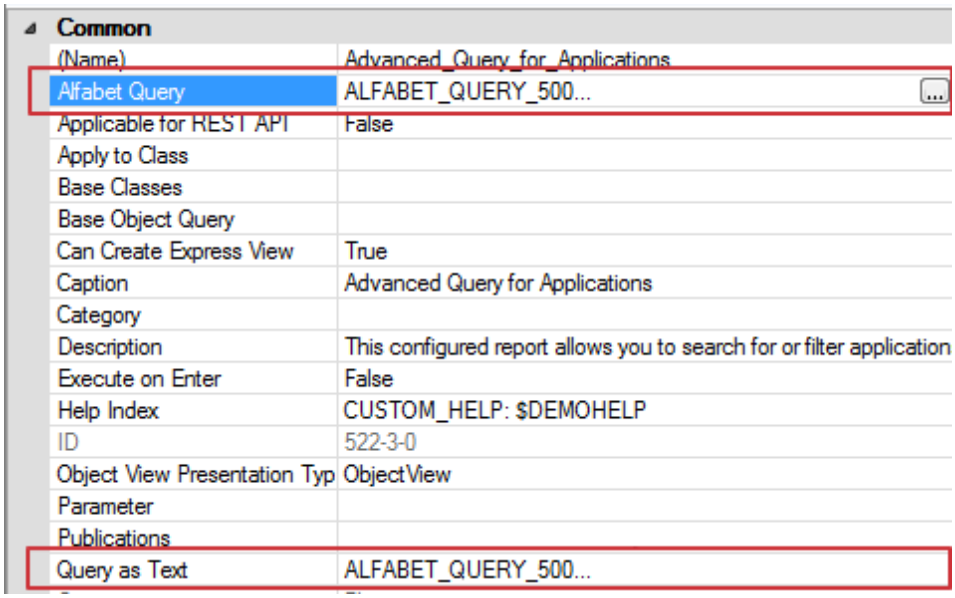


FIGURE: Property grid supporting query definition

For reports, For example, you can configure the Alfabet query using either the **Alfabet Query Builder** or the text editor. You can switch between the text editor and the **Alfabet Query Builder**, as needed:

- **Defining Alfabet queries in the Alfabet Query Builder**: Click the **Browse** button in the **Alfabet Query** field to open the **Alfabet Query Builder**.
- **Defining Alfabet queries in a text editor**: Click the **Browse** button in the **Alfabet Query as Text** field to open a text editor. You can write the Alfabet query based on the Alfabet query language directly in the editor.

Please note that the query that you define with the Alfabet Query Builder is only saved if you close the Alfabet Query Builder by clicking on the button **OK**. If you close the Alfabet Query Builder by clicking the **X** on the upper right corner of the window will close the Alfabet Query Builder without saving your changes. A query defined in the context of a configuration must additionally be saved to the Alfabet database by clicking the Save button of the Alfabet Expand Web menu after having closed the Alfabet Query Builder.

The following section provides an overview of the basic structure of the Alfabet query and the Alfabet query language and its display in the **Alfabet Query Builder**. Thereafter follows a section that will guide you through the steps required to configure a simple Alfabet query. Detailed information will follow that provides specific information about the individual elements of the Alfabet query language.

- Understanding Alfabet Queries
- *Building an Alfabet Query*
- Specifying WHERE Clauses in Alfabet Queries
- Specifying JOINs in Alfabet Queries
- Specifying Show and Sort Properties in Alfabet Queries
- Defining Filters for Configured Reports and Selectors
- Using Instructions to Format the Results of an Alfabet or Native SQL Query

- Defining Queries in XML Elements
- Testing Queries for Compliance with the Current Release
- *Creating an Index to Improve Performance for Query Searches in Database Tables*

Understanding Alfabet Queries

If you open an Alfabet query with the **Alfabet Query Builder**, you will see an interface with three main workspaces that represent the main structure of an Alfabet query written in Alfabet query language.

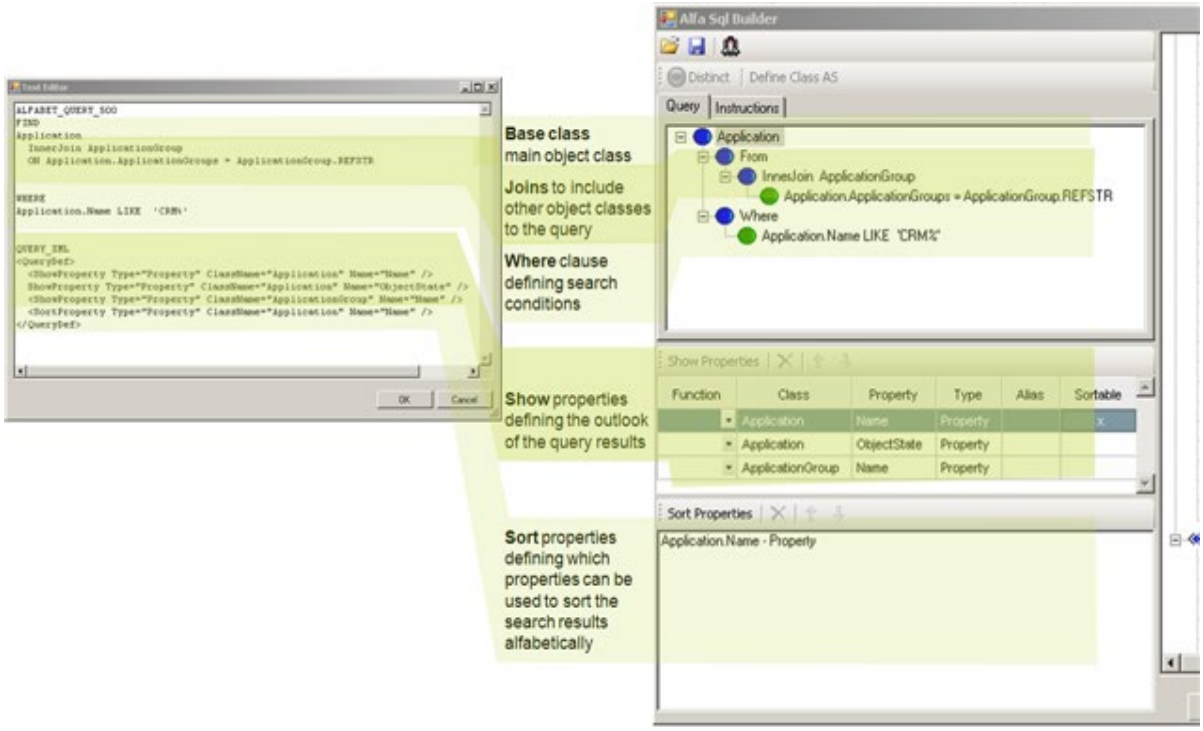


FIGURE: Alfabet query displayed in the text editor (left) and the same query displayed in the Alfabet Query Builder (right)

- The first section of the Alfabet query defines the objects to be searched for in the Alfabet database.
  - The Alfabet query starts with the base class, which is the class that is being searched for. In a report, the user can navigate to the object profile of this object class by clicking on an object displayed in the search results. In the example above, the base class is `Application`.
  - A `JOIN` allows you to include other object classes in the Alfabet query. The **Alfabet Query Builder** specifies the `JOIN`s in a `FROM` clause. If your search shall include data from object classes other than the base object class, you must define the relation between that object class and the base object class with a join. For example, to create a report about the applications that are assigned to a defined application group, you must define the relation between applications and application groups in a `JOIN`.

    Please note that whereas the **Alfabet Query Builder** specifies `JOIN`s via a `FROM` clause, the Alfabet query language does not use the expression `FROM` for a `JOIN`.

- A `WHERE` clause allows you to define the conditions that objects must meet to be included in the search condition. For example, you can search for applications that are assigned to a specific ICT object or that are members of a specific application group.

- In the **Show Properties** window, you can define which properties of the objects found by your Alfabet query shall be displayed in the resulting report or delivered as output to the processing functionality. If the output of your Alfabet query is a report table, you must define the columns of the report in **Show Properties** window. A column is added to the report for each show property defined.

- In the **Sort Properties** window, you can define the sort order of your Alfabet query results. You can sort the results in alpha-numerical order for any property included in the **Show Properties** section

A typical tabular report based on an Alfabet query is shown below:

Application Name	Assignment Name	Assignment Status	Assignment AssignmentType	Assignment TargetDate	Person Name
Mortgage Register Ma	Check if this application	Accepted	Optional	8/8/2008	Administrator
alfabet SITM	Applications ending late	Created	Mandatory	10/14/2008	Ngombe
Mortgage Register Ma	Please check if the life-	Created	Optional	7/30/2008	Administrator
SAP International	Outdated Compliance C	Created	Mandatory	11/24/2008	Picard
TradeWeb	Outdated Compliance C	Created	Mandatory	11/24/2008	Customer
alfabet SITM	Applications starting ear	Created	Mandatory	10/14/2008	Ngombe
Customer Manager	Please check if this app	In Progress	Mandatory	7/10/2008	Administrator
Credits&Loans BO	Please check the busin	Re-Assigned	Optional	6/30/2008	Gossarah

FIGURE: Report results based on an Alfabet query

Show properties define the number and content of the columns in the report. The **Sort properties** define the columns that can be sorted. These columns are underlined. The user can click the column header to sort the results as needed.

Each result found by the Alfabet query is displayed in its own row in the report. The user can double-click a row in the results table to navigate to the base class object. In the example above, the base class is the class Application. Double-clicking the second row will open the object profile of the application `alfabet SITM`.

Tables in tabular reports are displayed with a default height of one line of text. If the search result includes a line break, you can see only the first line of the results. To see the following lines, click the lower border of the table cell and drag it to increase the size of the table.

Building an Alfabet Query

The following steps are required to build an Alfabet query:

- *Identifying the Requirements for the Alfabet Query Output*
- *Defining the Base Class*

- *Defining Show Properties* to test the output of the Alfabet query.
- Defining the complete show properties for the data output.
- *Defining WHERE Conditions to Filter Search Results*
- *Defining Sort Conditions to Structure the Output*.

In the EBNF code writing standard, any variable that is exchanged with a specific value when writing the code is defined in angle brackets. In the code examples for Alfabet queries in this chapter, any term that is variable and must be exchanged with a specific value is written in italics rather than angle brackets This is done to distinguish variables from XML elements that are also used in Alfabet queries.

When a code is specified as:

```
ALFABET_QUERY_500
FIND
NameOfObjectClass
```

`NameOfObjectClass` must be substituted with the name of the object class when writing the following example query:

```
ALFABET_QUERY_500
FIND
Application
```

Identifying the Requirements for the Alfabet Query Output

Before you start defining an Alfabet query, it is recommended that you make a rough sketch of the requirements informing the query. Changing an existing Alfabet query because you discover that the results do not fulfill your expectations may be time-consuming and complicated.

The following questions may influence the way you need to define the Alfabet query:

- What is the expected result of my report, custom explorer, rule-based access permission, etc.?
- Which object classes are required in the report, custom explorer, rule-based access permission, etc. ?
- Which of these object classes is the base class?
- Starting with the base class, what are the relationships between the required classes?
- Which conditions apply to searching for objects in the defined object classes in order to get the expected result?

In the following, an example will accompany the description of the definition of an Alfabet query. In the example, a tabular report will be generated to allow users to check which applications in an application group have the status Draft.

The report will display:

- The name of the application.

- The version of the application.
- The name of the assigned application group(s)

The user shall be able to navigate to the application to edit relevant data. Therefore, the object class `Application` is the base class.

The report must show the following properties of the following classes:

- Class `Application`: properties `Name`, `Version`
- Class `ApplicationGroup`: property `Name`

Defining the Base Class

The base class defines the primary object class of the Alfabet query. The search for objects in the database starts with the database table of the base class. All objects that are to be found by the Alfabet query must either belong to the base class or must have a relation to the base class that is specified in the Alfabet query.

If the output of the Alfabet query is a report, the user can only navigate to the base class objects from the report. For example, when the search results display data about an application including its assignments and responsible users, and a user double-clicks a row in the results, the object view of the application displayed in that row will open even if the cursor is in a field displaying other referenced data.

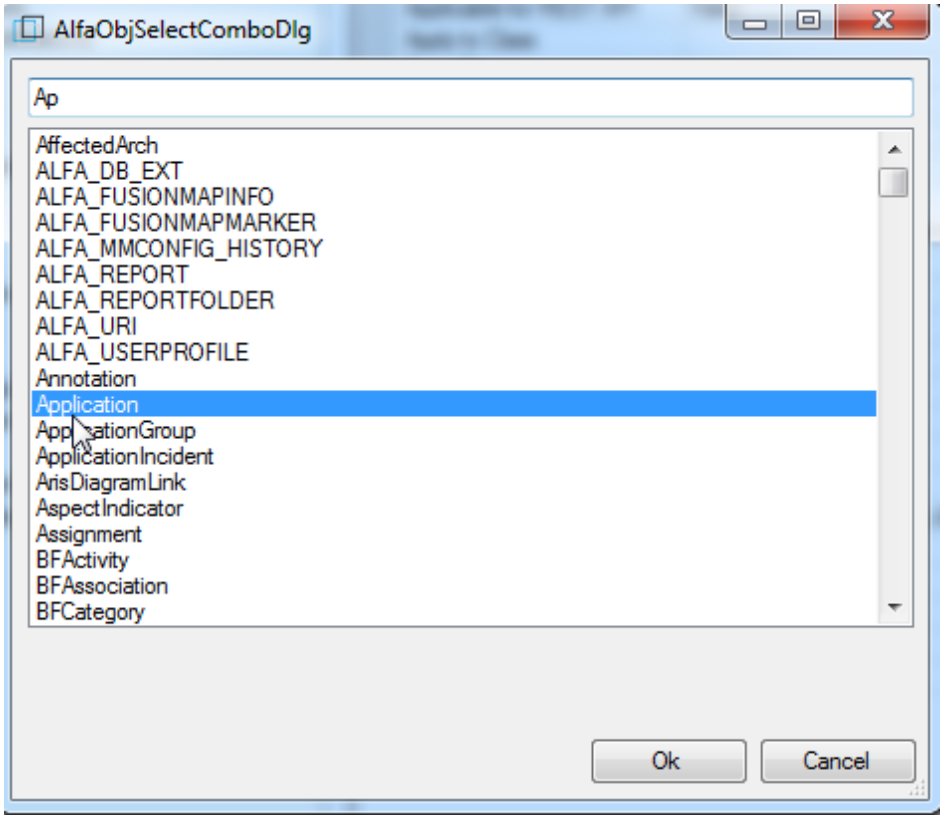
Navigation to the base class object view from a report is limited to object classes that represent artifacts in the meta-model.

When you open the **Alfabet Query Builder** to build a new Alfabet query, you will first see a dialog box listing all available object classes that can be specified as the base class of an Alfabet query.

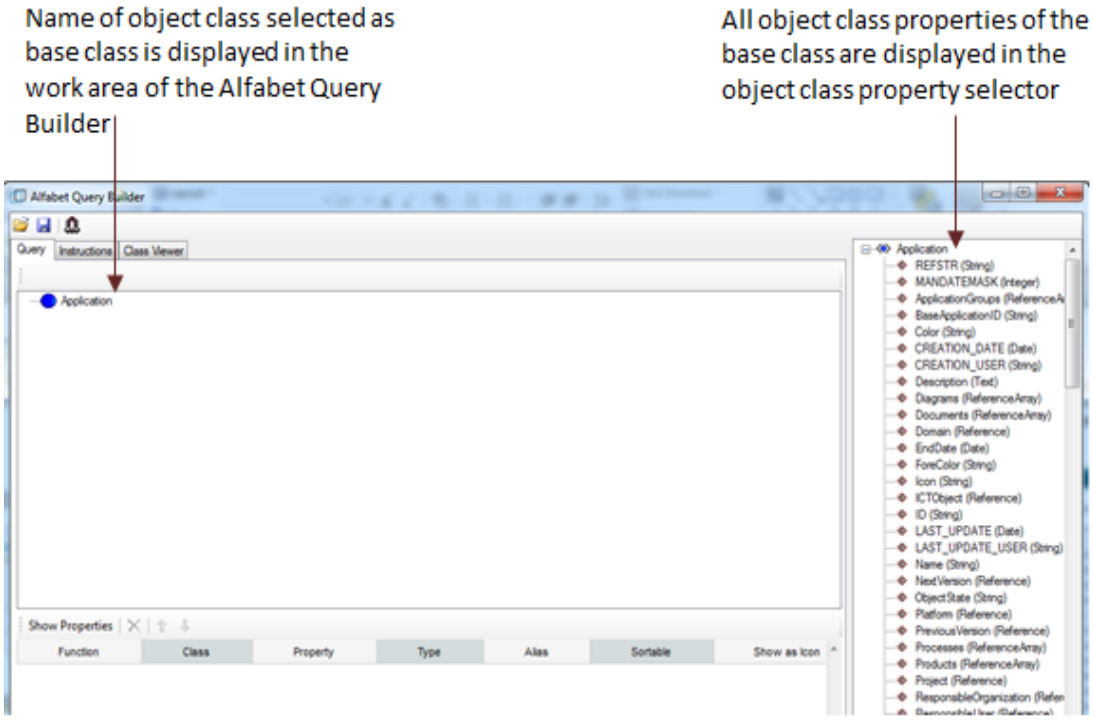
Select the base object class for your Alfabet query:

1) Click the name of the object class in the list or write the name of the object class in the upper field.

   If you start writing a name into the upper field, the cursor will move to the first search result matching your text in the list of classes.



2) Click **OK**. The **Alfabet Query Builder** opens with the base class displayed as the root node of the Alfabet query. On the right, all object class properties of the base class are listed in a separate pane, which is called the object class property selector.



To define a new Alfabet query and its base class in a text editor, enter the following lines of code:

```
ALFABET_QUERY_500
FIND
NameOfBaseClass
```

The Alfabet query must start with the header line `ALFABET_QUERY_500` followed by a `FIND` clause defining the base class. It is not possible to define more than one `FIND` clause in an Alfabet query.

Defining Show Properties

The Alfabet query finds objects in the Alfabet database that match specific criteria. To display the output in a report, the object class properties that shall be shown for each object found by the Alfabet query must be defined.

All object class properties that can be selected as Show properties are listed in the object class property selector in the right pane of the **Alfabet Query Builder**. The object class property selector lists all object class properties of all object classes that are included in the Alfabet query as a base class or in a `JOIN`. You can select the following:

Object class properties of the object class. When you select an object class property, the value of the object class property is displayed in the results.

Role type assigned to the object class in the **Configuration** module in Alfabet. When you select a role type, the name of the user to which the role is assigned for the object is displayed in the results. The name of the user is displayed as Name followed by First Name.

Indicator types assigned via evaluation types to the object class in the **Configuration** module in Alfabet. When you select an indicator type, the value for the indicator is displayed in the results.

The **Show Properties** section of the **Alfabet Query Builder** displays all object class properties defined to be displayed in the output of the Alfabet query in a table that has the following columns:

- **Function**: Select a function that shall be applied to the values of the show property prior to display. For example, the function `SUM` returns the sum of a set of numeric values. For more information, see the section *Defining Show Properties*.
- **Class**: Displays the name of the object class.
- **Property**: Displays the name of the object class property.
- **Type**: Displays the type of show property to display. The type can be:
  - `Property`: An object class property of the object class in the class model.
  - `Indicator`: An indicator of the object class.
  - `Role`: A role specified for the object class.

- **Alias**: Text written into a cell in this column is displayed as column caption in the tabular output of the Alfabet query.
- **Sortable**: Click a cell in this column to make the property represented by the row sortable for the user opening a tabular report..

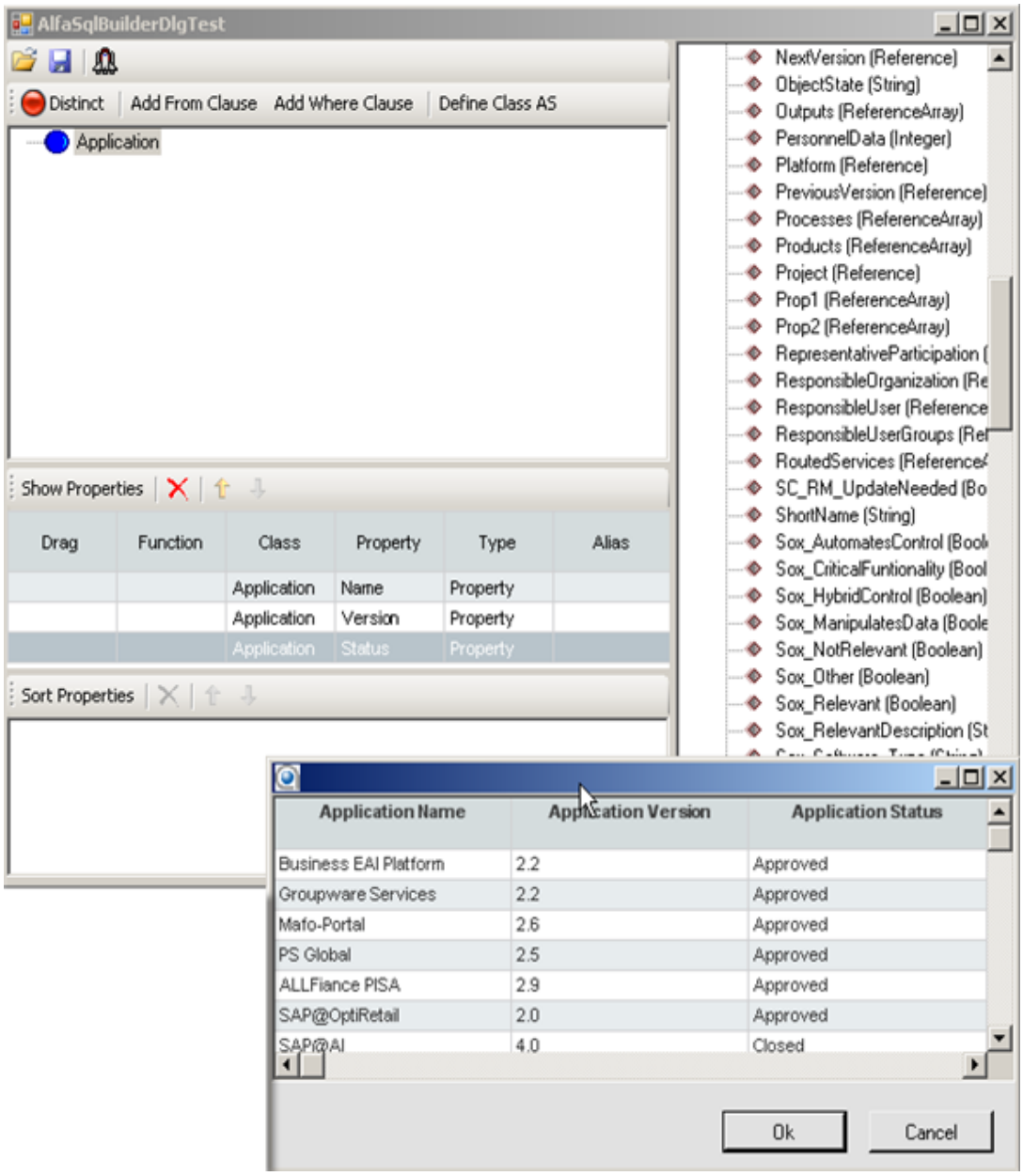
To define the Show properties in the report:

1) Click an object class property of the base class in the object class property selector and drag it to the **Show Properties** section of the **Alfabet Query Builder**. The columns **Class**, **Property** and **Type** are automatically filled.

   Object class properties that specify references to objects of other object classes (of the data type `Reference` or `ReferenceArray`) cannot be added to the Show properties. A `JOIN` is required to add references to query results.

2) Repeat step 1) for all properties of the base class that you want to display in the report.
3) Optionally, enter a column header for the property in the resulting report in the **Alias** column. By default, the column headers display the class name followed by the property name.
4) Optionally, sort the columns in the report. The columns are displayed in the order they are defined in the **Show Properties** section. To change the position of a column in the result data set, click the respective row in the **Show Properties** section and use the **Up** and **Down** buttons in the toolbar of the **Show Properties** section to change the position in the list.
5) Click the **Test Query** button. The list of all available objects of the base class is listed in a separate table.

For the example, the base class properties `Name` and `Version` are selected as well as the property `Status` in order to control the `WHERE` condition for the Alfabet query defined in one of the following steps. The resulting report shows all applications independent of the Release Status they are in:



To define show properties for an Alfabet query in a text editor, you can define the show properties either in an XML element or with a `SHOW` statement. The automatic code generated by the **Alfabet Query Builder** generates the XML definition of the show properties. Nevertheless, you might also require alternative syntax (For example, to specify an Alfabet query within an XML definition or when defining treemap reports).

For more information about the definition of show properties in the Alfabet query language, see the section Specifying Show and Sort Properties in Alfabet Queries which includes the following information:

- Defining Show and Sort Properties in an XML Element
- Displaying Users/Organizations Defined for a Role in Alfabet Query Results
- Displaying Indicator Values in Alfabet Query Results

- Calculating Values in Alfabet Query Results

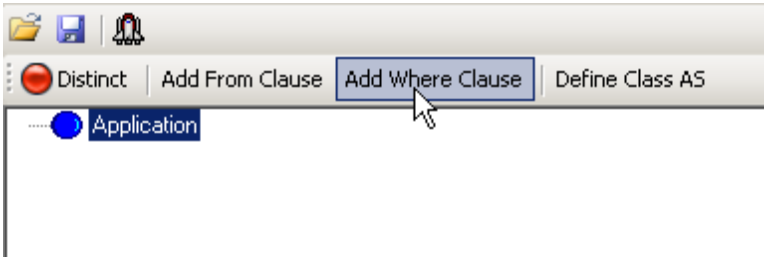
Defining WHERE Conditions to Filter Search Results

`WHERE` clauses limit the search output to a subset of the available objects in the database table of an object class that match a condition specified in the `WHERE` clause.

In the example, the result of the report shall display only objects of the class Application that are in the release status Draft.

To add a `WHERE` clause to the Alfabet query:

1) In the **Alfabet Query Builder**, click the base class in the main area.



2) In the toolbar, select **Add Where Clause**. In the **Alfabet Query Builder**, the parameter ***Where*** is written below the base class and the node ***And*** is written below the parameter ***Where***.
3) In the Alfabet query, click ***And***. The buttons in the toolbar change to the available types of `WHERE` clauses.
4) In the toolbar, select **Property**. The **Properties Browser** opens
5) Drag an obect class property to the lower pane of the **Properties Browser** and click **OK**. The object class property is displayed in the Alfabet query nested below the `WHERE` clause.
6) In the Alfabet query, click the property in the `WHERE` clause. The buttons in the toolbar change to the available operators for the selected object class property type.

   For information about the meaning of each operator, see the Operators for WHERE Clauses Dependent on Property Data Type.

7) In the toolbar, select an operator. A dialog box will open for all operators except the operators `IS NULL` and `IS NOT NULL`.
8) In the dialog box, enter the value that the selected object class property should have and click **OK** . The condition is displayed in the `WHERE` clause in the Alfabet query.
9) Click the **Test Query**  button. The list of all found objects of the base class is displayed in a separate table.

You can change the conditions for the selected property later. Click the condition and carry out one of the following procedures: To change the operator, select the new operator in the toolbar. If the value of the condition is an object property, the operator will be changed and the toolbar will be cleared. To see the toolbar buttons again, you must click the condition again. If the

condition is a string or parameter, the dialog box for the condition value will open after selecting the new operator and you can change the value.

- To change the condition value, you must proceed differently for comparisons with object properties vs. string values or parameters.
  - If you want to change an object property, drag the new property from the property selector window to the condition.
  - If you want to change a property value that is a string, select the current operator in the toolbar. The dialog box for the value specification opens and allows to edit the value.

To define a `WHERE` clause for an Alfabet query in the text editor, write the following lines of code under the `FIND` statement:

```
WHERE
ObjectClassName.PropertyName Operator OperatorValue
```

The `WHERE` clause must start with `WHERE` followed by the specification of an object class property of an object class and the condition that has to apply to the object class property. When the condition is a string, it must be written in single quotes. For example,:

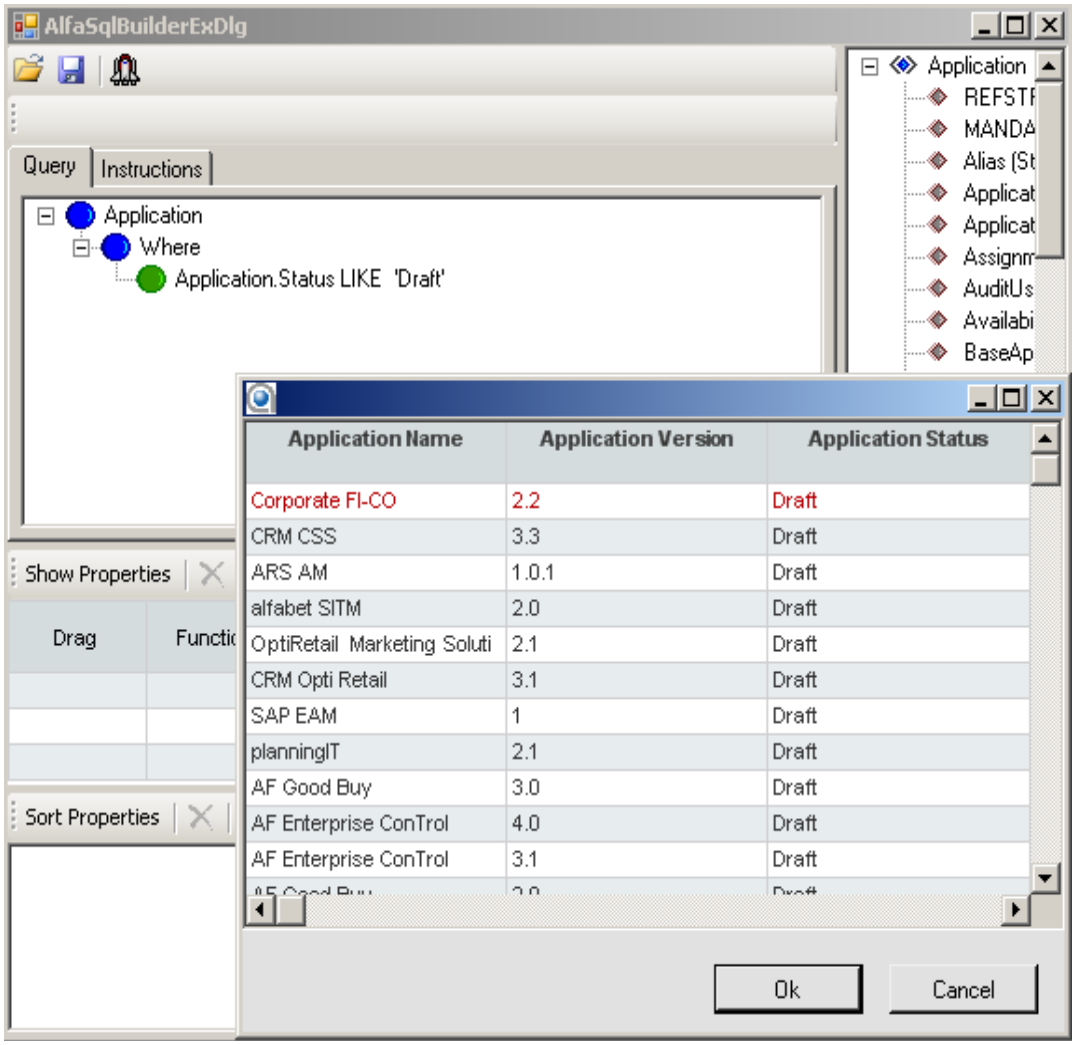
```
WHERE
Application.Status LIKE 'DRAFT'
```

The operators that can be used depend on the data type of the object class property. For detailed information about the available `WHERE` operators, see Operators for WHERE Clauses Dependent on Property Data Type in the section Specifying WHERE Clauses in Alfabet Queries.

For example, the `WHERE` condition specifies that the object class property `Status` (Release Status) of the object class `Application` is `Draft`.

```
WHERE
Application.Status LIKE 'DRAFT'
```

The resulting report now shows only applications that are in the release status `Draft`:



You can add multiple `WHERE` conditions to the `WHERE` clause of an Alfabet query. For more information see the section Adding Multiple WHERE Conditions to an Alfabet Query. If you define ony one `WHERE` clause and you close and reopen the Alfabet Query Builder, the And node is removed from the query definition.

For detailed information about the definition of `WHERE` clauses in Alfabet queries, see the following:

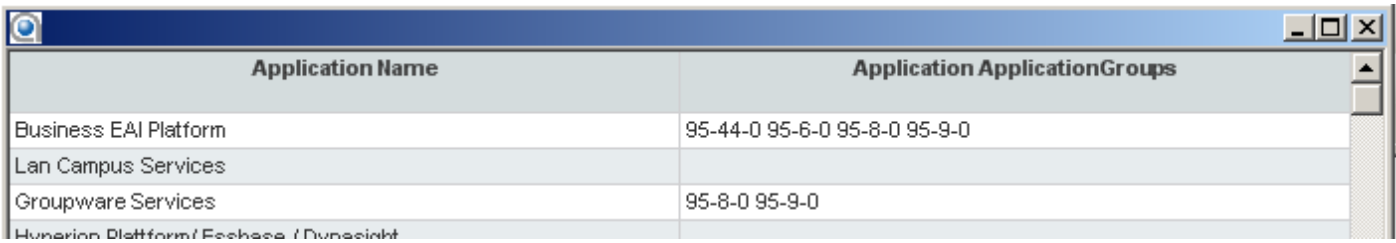
- Specifying WHERE Clauses in Alfabet Queries
  - Operators for WHERE Clauses Dependent on Property Data Type
  - Booleans in WHERE Clauses
  - Specification of Dates
  - Using Wildcards in WHERE Conditions
  - Using Special Characters in WHERE Conditions
  - Defining Case-Sensitivity for WHERE Clauses
  - Comparing Property Values with Other Property Values
  - Referring to the Current Alfabet Context in a WHERE Condition

- Adding Multiple WHERE Conditions to an Alfabet Query
- Creating a Filter Field with a WHERE Clause

Defining JOINs

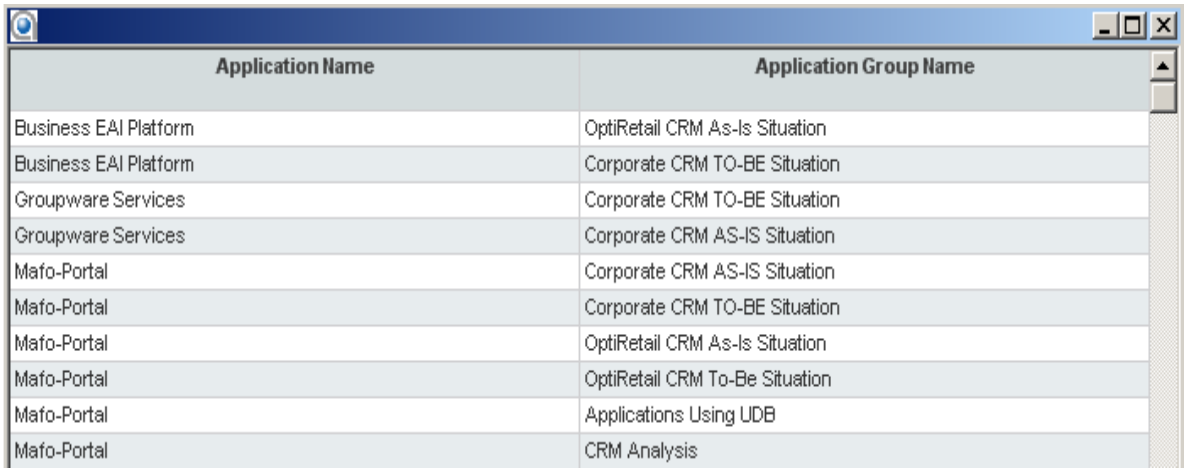
By defining one or more `JOIN`s, you can add objects from other object classes that are related to the base class through relationships defined in the meta-model. If you do not join other object classes to your Alfabet query, you will only find and display objects in the base object class in the Alfabet query results.

Relationships to other object classes are specified in object class properties of the type `Reference` or `ReferenceArray`. If a report should display all application groups that the application is assigned to, you cannot simply add the property `ApplicationGroup` to the show properties of the Alfabet query because you would then get a list of reference strings for internal database identification of the reference:



Application Name	Application ApplicationGroups
Business EAI Platform	95-44-0 95-6-0 95-8-0 95-9-0
Lan Campus Services	
Groupware Services	95-8-0 95-9-0
Hyperion Plattform/ Essbase / Dynasight	

To display the name of the referenced application groups, you must add a `JOIN` to the Alfabet query. The join specifies that application groups are added to a query result if they are referenced in the object class property `ApplicationGroups` (Application Groups). If multiple application groups are referenced by the application, a separate line is displayed in the results for each relation found:



Application Name	Application Group Name
Business EAI Platform	OptiRetail CRM As-Is Situation
Business EAI Platform	Corporate CRM TO-BE Situation
Groupware Services	Corporate CRM TO-BE Situation
Groupware Services	Corporate CRM AS-IS Situation
Mafo-Portal	Corporate CRM AS-IS Situation
Mafo-Portal	Corporate CRM TO-BE Situation
Mafo-Portal	OptiRetail CRM As-Is Situation
Mafo-Portal	OptiRetail CRM To-Be Situation
Mafo-Portal	Applications Using UDB
Mafo-Portal	CRM Analysis

The properties of the object class `ApplicationGroup` (Application Group) are then displayed in the property selector of the **Alfabet Query Builder** and you can add the property `Name` of the class `ApplicationGroup` to the show properties.

Deciding About the Type of JOIN to Use

If an object class A references the object class B, a relation exists between objects of both classes. But there are normally also objects of object class A that do not reference object of the object class B and objects of the object class B that are not referenced by object class A. In the example above, you will find applications assigned to application groups, but also applications that are not assigned to an application group. There are also application groups in the Alfabet database that only include other application groups and have no applications directly assigned.

If you join an object class to your base object class in the Alfabet query, you can use four different kinds of `JOINS` to select a different subset of objects:

InnerJoin	FullJoin	LeftJoin	RightJoin
Selects all instances from both classes that are related to each other	Selects all instances from both classes	Selects all instances from both classes that are related to each other and all instances from A that do not have a reference to B.	Selects all instances from both classes that are related to each other and all instances from B that are not referenced by A.

Application	Application Group
A 1	AG 1
A 2	
A 3	AG 2
A 4	
	AG 3

Application	Application Group
A 1	AG 1
A 2	
A 3	AG 2
A 4	
	AG 3

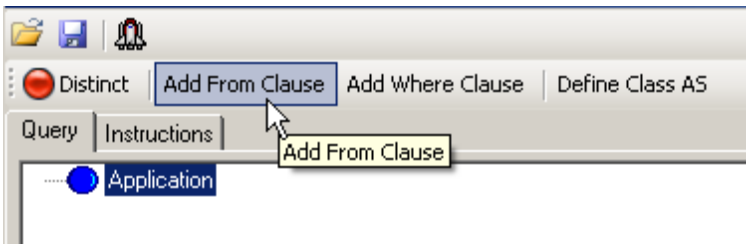
Application	Application Group
A 1	AG 1
A 2	
A 3	AG 2
A 4	
	AG 3

Application	Application Group
A 1	AG 1
A 2	
A 3	AG 2
A 4	
	AG 3

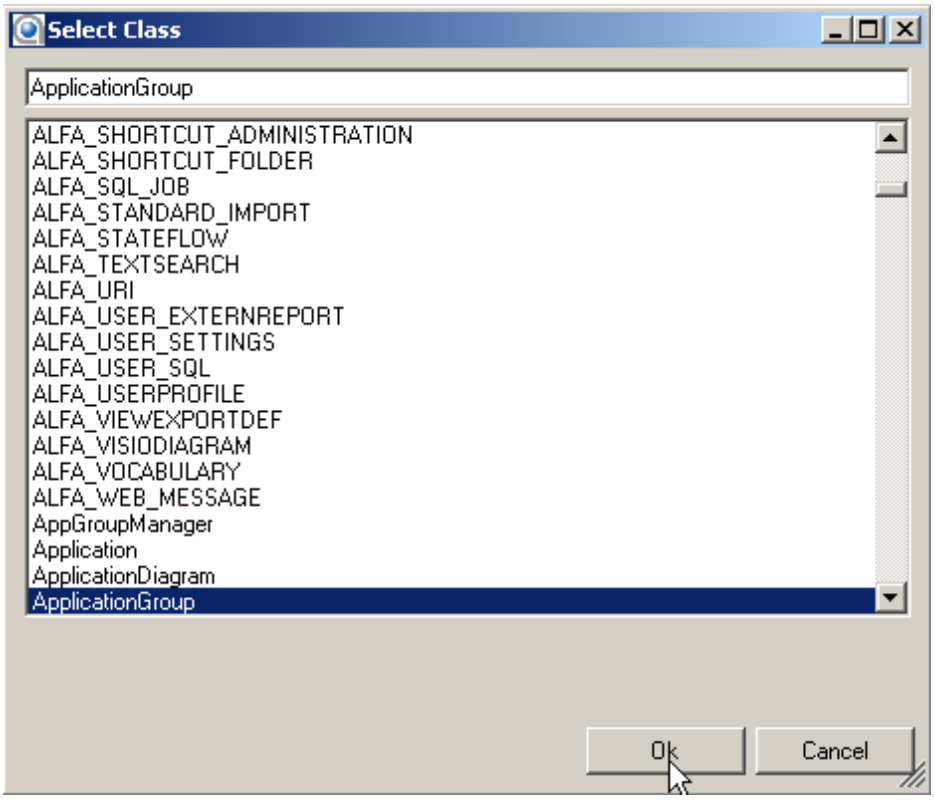
Defining JOINS in the Alfabet Query Builder

In the **Alfabet Query builder**, a `JOIN` is a sub-clause of a `FROM` clause. An Alfabet query can have only one `FROM` clause. The `FROM` clause must include all defined `JOINS`.

1) In the **Alfabet Query Builder**, click the base class of the Alfabet query.
2) In the toolbar, select `Add From Clause`.

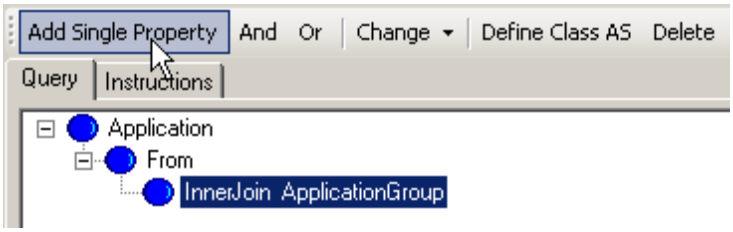


The **Select Class** dialog box opens.

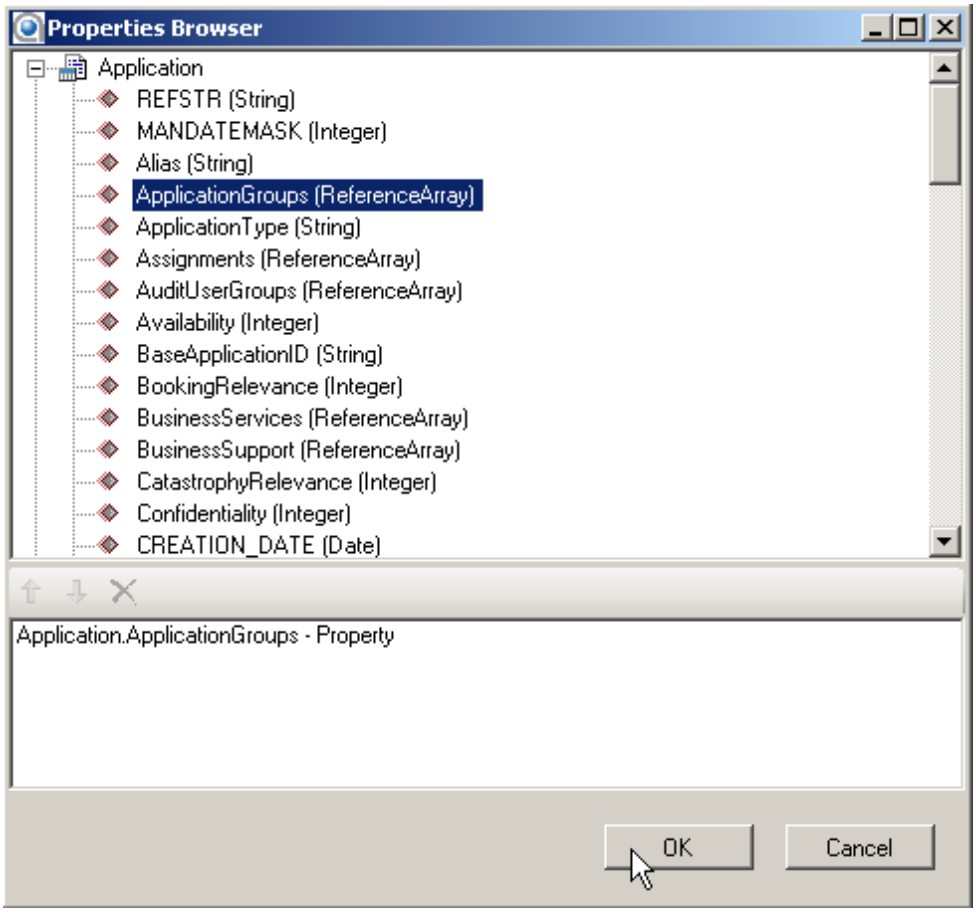


3) In the dialog box, select the class that you want to add and click **OK** . An `InnerJoin` referencing the selected object class is added to the Alfabet query

4) If you do not want to specify an `InnerJoin`, click the **Change** button in the toolbar and select the required type of `JOIN` in the drop-down list.



5) In the toolbar, select **Add Single Property**. The **Properties Browser** opens



6) In the **Properties Browser**, expand the object class properties of the object class with the object class property referencing the other object class. In the example, this is the object class `Application`. Select the object class property specifying the reference and drag it to the lower pane of the **Properties Browser**.

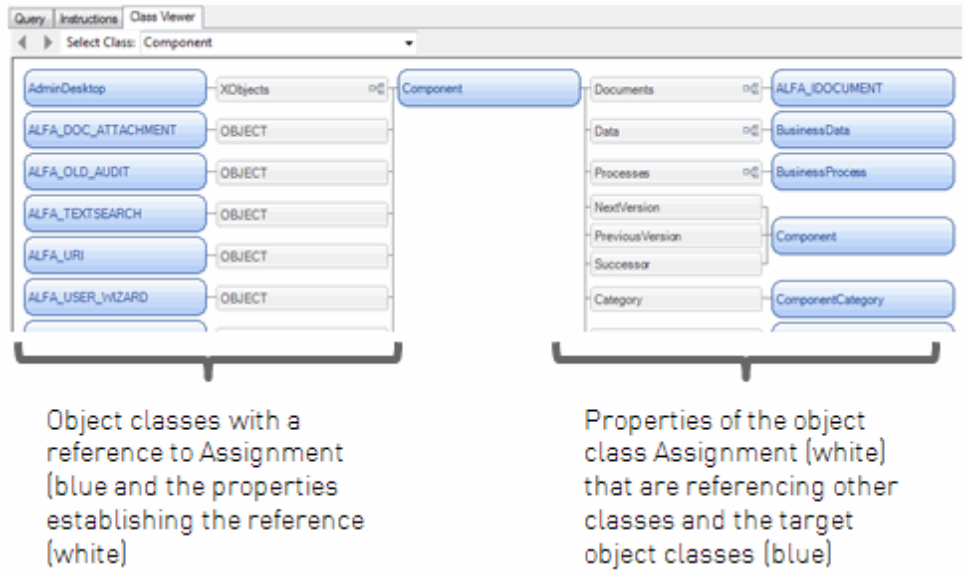
The object class property establishing the reference is not necessarily an object class property of the base class or an object class already joined to the Alfabet query. It can also be a property of the object class that you want to join referencing an already joined class.

For the example given, it would also be possible to select the object class property `Applications` of the class `ApplicationGroup` that you currently want to join and specify that it has to contain the `REFSTR` of the application.

It is also possible to specify `JOIN`s by comparing object class properties of two object classes in `JOIN`s without one object class property referencing the other object class. For more information, see Operators for the Definition of JOIN Conditions.

If you do not know which object class property references which object class, go to the **Class Viewer** tab in the **Alfabet Query Builder**.

The base class of your Alfabet query is displayed in the top center of the graphic. On the left, all object classes referencing the selected base class via one or multiple of their object class properties are listed in blue boxes. The object class property that establishes the reference is displayed in a white box between the object classes. On the right, all object class properties of the selected object class that store references to other object classes are listed. The target object classes are listed in the blue boxes on the right.



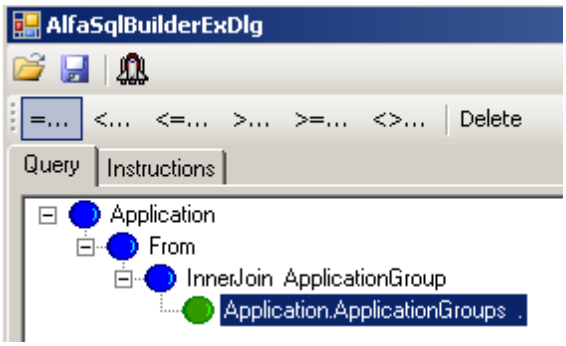
When you double-click a referenced object class in the graphic, a new tab opens that displays the references of this object class to other object classes.

You can select another object class whose references you want to see in the **Select Class** field.

The arrow  buttons next to the **Select Class** field allow navigation to previously shown object class relations.

7) Click **OK** to save your selection. The object class property is displayed in the **Query** tab in the **Alfabet Query Builder**.

8) Click the object class property name. The toolbar displays all available operators for the object class property's data type.



9) From the toolbar, select an operator. The **Properties Browser** opens.

10) In the **Properties Browser**, expand the properties of the object class that is referenced by the object class property displayed in the `JOIN`. In the example, this is `ApplicationGroup`. Select the referenced object class property and drag it to the lower pane of the **Properties Browser** . In the example, this is the `REFSTR` property of the class `ApplicationGroup`.

11) Click **OK** to save your selection. The object class property is displayed in the **Query** tab of the **Alfabet Query Builder**.

12) If you want to display object class properties of the joined class in the query results, add the desired object class properties to your show properties. For information on how to add a show property, see *Defining Show Properties*.

13) Click the **Test Query**  button. You will see the selected object class properties of your joined object class in the report. Test whether the results are as expected. If this is not the case, you should check whether you have chosen the wrong type of `JOIN` or whether results must be filtered by another `WHERE` condition that specifies a condition for an object class property of the joined object class.

When you want to join a second class to the Alfabet query, you can click ***From*** in the Alfabet query displayed in the **Alfabet Query Builder** and directly select the type of `JOIN`. All `JOIN`s are written to the same `FROM` clause.

You can change the conditions for the selected property later. Click the condition and carry out one of the following procedures: To change the operator, select the new operator in the toolbar. If the value of the condition is an object property, the operator will be changed, and the toolbar will be cleared. To see the toolbar buttons again, you must click the condition again. If the condition is a string or parameter, the dialog box for the condition value will open after selecting the new operator and you can change the value.

- To change the condition value, you must proceed differently for comparisons with object properties vs. string values or parameters.
  - If you want to change an object property, drag the new property from the property selector window to the condition.
  - If you want to change a property value that is a string, select the current operator in the toolbar. The dialog box for the value specification opens and allows to edit the value.

It is not possible to delete an intermediate `JOIN` in a query with two or more `JOIN`s. For example, if a query joins class 1 to the base class, class 2 to class 1 and class 3 to class 2, the `JOIN` of class 2 to class 1 cannot be deleted.

To add a `JOIN` to an object class for an Alfabet query defined in a text editor, you must specify the `JOIN` under the base class definition with the following line of code:

```
JoinTypeNameOfClassToJoinON
ObjectClassName.PropertyNameOperatorObjectClassName.PropertyName
```

In case the `JOIN` is specified to link an object class that is referenced by an object class property of the type `ReferenceArray` or `Reference`, the `JOIN` is defined as:

```
JoinTypeNameOfClassToJoinONNameOfReferencingClass.NameofPropertythatReferencesToTheOtherClass=NameofReferencedClass.REFSTR
```

The type of `JOIN` can be `InnerJoin, OuterJoin, LeftJoin`, and `RightJoin`. For more information about `JOIN` types, see *Deciding About the Type of JOIN to Use*.

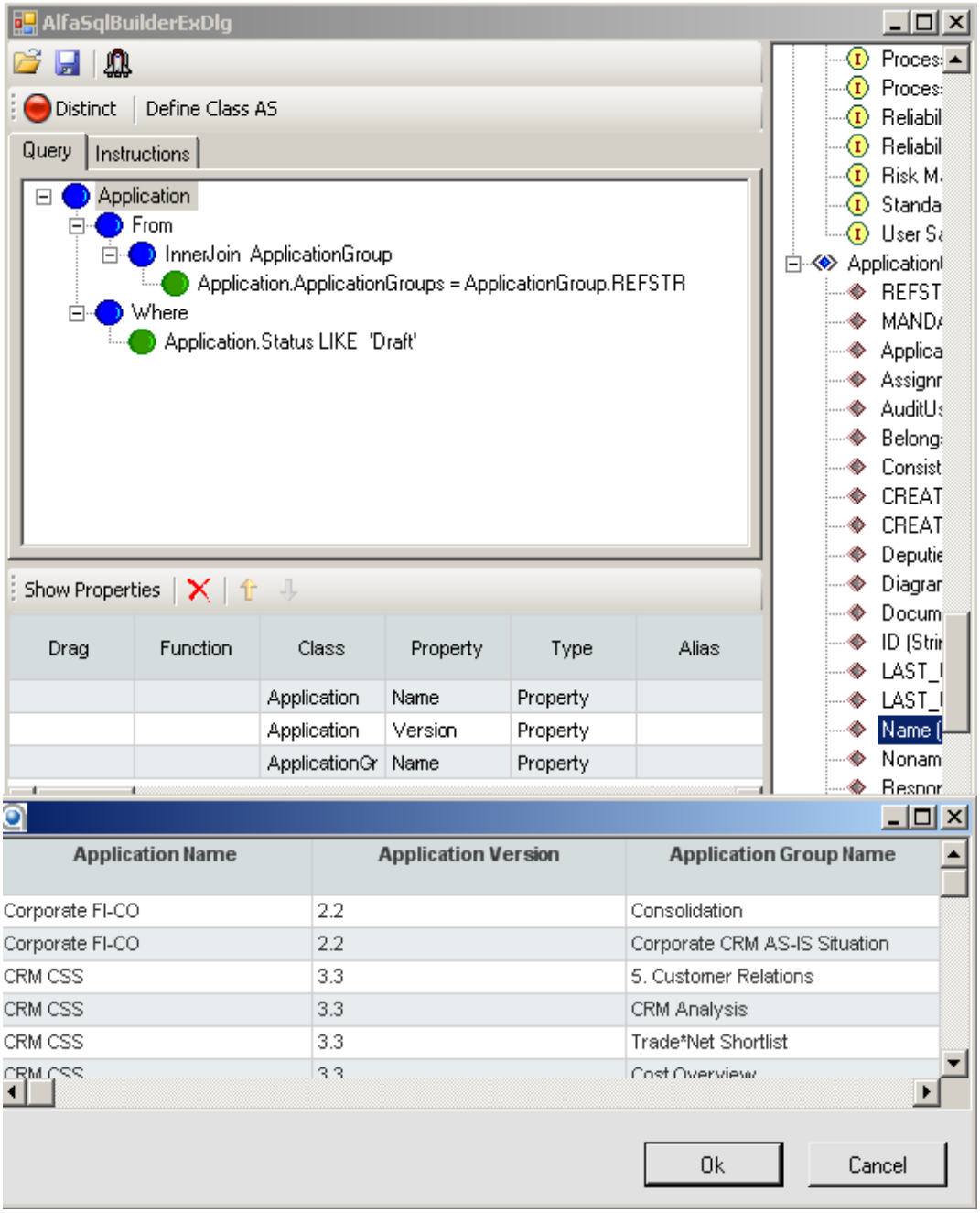
In the example, the object class `ApplicationGroup` is added via an `InnerJoin` to the base class `Application` because only applications that are assigned to application groups are relevant for the report and vice versa.

```
ALFABET_QUERY_500

FIND

Application
```



```
InnerJoin ApplicationGroup ON Application.ApplicationGroups =
ApplicationGroup.REFSTR
```

In the example, the object class property `Status` of the object class `Application` is deleted from the Show properties section because it is no longer required and the object class property `Name` of the class `ApplicationGroup` is added:



For detailed information about the definition of `JOIN`s in Alfabet queries, see the section Specifying JOINs in Alfabet Queries. The following information is addressed:

- Operators for the Definition of JOIN Conditions
- Specifying Multiple JOIN Conditions for one JOIN

- Joining Classes Already Specified in the Alfabet Query
- Defining JOINs for Roles
- Defining JOINs for Indicators
- Defining JOINs Between Value Nodes, Projects, or Demands and the Affected Architecture
- Defining JOINs to Object Class Stereotypes

Defining Sort Conditions to Structure the Output

The results of an Alfabet query are listed in the order that they are found in the database. However, you may want the results to be sorted alphanumerically based on the values of selected columns in the result table.

Sort conditions in Alfabet queries allow you to define the columns in the report that are to be sorted in ascending alphanumeric order during execution of the Alfabet query. All rows with an equal value are then sorted according to the second criteria, and so on. The user can change the sort order of the rows in the report by clicking any column header of the report. The report is then sorted in ascending alphanumeric order based on the values in the selected column. If he/she clicks the column header of a column that is already used for sorting, the sorting order is reverted from ascending to descending alphanumeric order and vice versa.

You can define multiple Sort properties in one Alfabet query. Sort properties cannot be defined for roles and indicators directly added to the Show properties via the object class property selector.

To define Sort conditions in the **Alfabet Query Builder**:

1) In the **Show Properties** section, click the cell in the column **Sortable** of the object class property that you want to select as `SORT` property. An x is displayed in the cell and the object class property is added to the **Sort Properties** section.

To define Sort properties for an Alfabet query in a text editor, you can define the Sort properties either in an XML element or with a `SORT` statement. The automatic code generated by the **Alfabet Query Builder** generates the XML definition of the Sort properties. Nevertheless, you might also require alternative syntax (For example, to specify an Alfabet query within an XML definition or when defining treemap reports).

The following information about the definition of Sort properties is available:, see the section Specifying Show and Sort Properties in Alfabet Queries.

- Specifying Show and Sort Properties in Alfabet Queries
  - Defining Show and Sort Properties in an XML Element
  - Displaying Users/Organizations Defined for a Role in Alfabet Query Results
  - Displaying Indicator Values in Alfabet Query Results
  - Calculating Values in Alfabet Query Results

Specifying WHERE Clauses in Alfabet Queries

A WHERE clause can be added to Alfabet queries to limit the search results to a subset of the objects defined for the base class or the object classes added to an Alfabet query by means of a JOIN. An object class property of an object class is compared to a given value in a WHERE clause. For example, you can limit the search output to objects with a name starting with "A" or with a start date before the current date.

A WHERE clause consists of the object class property, the operator defining the compare mode, and the value with which the object class property is compared:

```
WHERE
ObjectClassName.PropertyNameOperatorValue
```

```
WHERE
Application.Name LIKE 'A%'
```

Information about how to add a WHERE clause to an Alfabet query in the **Alfabet Query Builder** is provided in *Defining WHERE Conditions to Filter Search Results* in the section *Building an Alfabet Query*.

This chapter provides information about the different kind of WHERE clauses that can be specified. The following information is available:

- Operators for WHERE Clauses Dependent on Property Data Type
- Booleans in WHERE Clauses
- Specification of Dates
- Using Wildcards in WHERE Conditions
- Defining Case-Sensitivity for WHERE Clauses
- Comparing Property Values with Other Property Values
- Referring to the Current Alfabet Context in a WHERE Condition
- Adding Multiple WHERE Conditions to an Alfabet Query
- Creating a Filter Field with a WHERE Clause

Operators for WHERE Clauses Dependent on Property Data Type

The following list provides an overview of the operators that are allowed in Alfabet queries for different data types.

Note the following about the use of operators in Alfabet queries:

- If you define the Alfabet query with the **Alfabet Query Builder**, only the allowed operators will be displayed in the menu when you click the property defined in a WHERE clause. For some special properties (For example, the REFSTR or the MANDATEMASK property), you can use only a subset of the operators specified in the table below. The table lists the default for the use of the operators per data type.

- The range of allowed operators for a data type includes combinations that are only relevant for special contexts. Not all allowed combinations of data type or operator may be meaningful for the context you are working with.
- `WHERE` clauses can only be built on properties of the data type `ReferenceArray` if the object class property's attribute **Reference Support** is set to `True` in the Alfabet meta-model. For the rare case that **Reference Support** is set to `False`, the property cannot be used in Alfabet queries.

Operator	Description	For Data Type	Example
`LIKE`	Similar to a defined string.	`String` `StringArray`	`Application.Name LIKE 'A%'` (finds all applications whose names begin with "A")
`NOT LIKE`	Different to a defined string.	`String`	`Application.Name LIKE 'A%'` (finds all applications whose names do not begin with "A")

`CONTAINS`	Similar to.
	NOTE: `CONTAINS` differs from `LIKE` for the generation of filter fields in reports based on Alfabet queries. For more information about the types of filter fields, see Defining Filters for Configured Reports and Selectors.
		`String`
		`StringArray`
		`Reference`
		`ReferenceArray` (with the **Reference Support** attribute set to `True`)
			`Application.Name CONTAINS 'A%'`
			(finds all applications whose names begin with "A")
			NOTE: For properties of the type `StringArray`, only entries exactly matching the defined string will be found. Therefore, if you want to find entries containing only one specific selection, enter 'Selection'. To find entries containing one specific selection and optionally other selections, enter '`%`Selection`%`'. If you want to find entries all with at least two defined selections, enter '`%`Selection1`%`Selection2`%`'. The selections must be listed in the order they are listed in the enumeration.

`CONTAINSOR`	The object class property value is to be found in the values of a list. The list is enclosed in inverted commas and the individual values are separated by a comma.
	NOTE: `CONTAINSOR` can be used to create multi-select
		`String`
		`StringArray`
		`Reference`
		`ReferenceArray` (with **Reference Support** set to `True`)
			`Person.Name CONTAINSOR 'Miller,Young,Singer'`
			(finds all users with the name Miller, Young or Singer)
			NOTE: In the **Alfabet Query Builder**, the comma-separated list of values must be defined in the dialog box for entering values without the inverted commas.

Operator	Description	For Data Type	Example
	combo-box filters. For more information about the types of filter fields, see Defining Filters for Configured Reports and Selectors.		
IN ()	The object class property value is to be found in the values of a list. The list is enclosed in a parenthesis (). The individual values are comma-separated and each written into inverted commas.	Integer Real Reference **NOTE**: The operator IN can also be used to compare the REFSTR property of an object class with a list of values.	Project.ReportProgress IN ('10','20',30') (finds all projects with a report progress of 10, 20, or 30) **NOTE**: In the **Alfabet Query Builder**, the list of values must be defined in the dialog box for entering values without the parenthesis, separating commas and inverted comma. Instead, values must be separated with a whitespace. The list of integers in the example above must be written into the field as: 10 20 30
NOT IN ()	The object class property value is not to be found in the values of a list. The list is enclosed in a parenthesis (). The individual values are comma-separated and each written into inverted commas	Integer Real Reference **NOTE**: The operator IN can also be used to compare the REFSTR property of an object class with a list of values.	Project.ReportProgress NOT IN ('10','20','30') (finds all projects with a report progress of 10, 20, or 30) **NOTE**: In the **Alfabet Query Builder**, the list of values must be defined in the dialog box for entering values without the parenthesis, separating commas and inverted commas. Instead, values must be separated with a whitespace. The list of integers in the example above must be written into the field as: 10 20 30
BETWEEN	Selection of a value range. The value is specified in the format: *LowerLimit*AND*UpperLimit*	Date Integer Real	ProjectGroup.SpendLimit BETWEEN 0 AND 10000 (finds projects with a spend limit higher than 0 and lower than 10000) **NOTE**: If the condition is written in the Alfabet query language as text and the values are dates, they must be written in inverted commas:

Operator	Description	For Data Type	Example
			`Appliation.StartDate BETWEEN '25/03/2008' AND '28/03/2008'` (finds applications with a start date between March 25th, 2008 and March 28th, 2008) **NOTE**: In the **Alfabet Query Builder**, the upper and lower limit of the range must be defined in the dialog box for entering values without the `AND`, but only separated with a whitespace. Dates are not written in inverted commas in this dialog box.

`NOT BETWEEN`	The object class property value is not to be found in a selection of a value range. The value is specified in the format:
	
	LowerLimit`AND`*UpperLimit*
		`Date`
		
		`Integer`
		
		`Real`
			`ProjectGroup.SpendLimit NOT BETWEEN 0 AND 10000`
			
			(finds projects with a spend limit higher than 10000)
			
			NOTE: If the condition is written in the Alfabet query language as text and the values are dates, they must be written in inverted commas:
			
			`Application.StartDate NOT BETWEEN '25/03/2008' AND '28/03/2008'`
			
			(finds applications with a start date before March 25th, 2008 and after March 28th, 2008)
			
			NOTE: in the **Alfabet Query Builder**, the upper and lower limit of the range must be defined in the dialog box for entering values without the `AND`, but only separated with a whitespace. Dates are not written in inverted commas in this dialog box.

=	Equal to.
	
	If the object class property value is of the type `String`, `Date` or `Boolean`, it must be written in inverted commas in the Alfabet query.
		`String`
		
		`Date`
		
		`Integer`
		
		`Real`
		
		`Boolean`
		
		`Reference`
		
		`ReferenceArray` (with **Reference Support** set to `True`)
			`Application.StartDate = '28/03/2008'`
			
			(finds applications with a start date of March 28th, 2008)
			
			NOTE: Wildcards are not allowed in value specifications for this operator.
			
			NOTE: In the **Alfabet Query Builder**, the value of any data type is written without inverted commas into the dialog box for entering values.

<	Lesser than.
	
	If the object class property value is
		`String`
		
		`Date`
			`Application.StartDate <= '28/03/2008'`

Operator	Description	For Data Type	Example
	of the type `String` or `Date`, it must be written in inverted commas in the Alfabet query.	`Integer` `Real`	(finds applications with a start date before March 28th, 2008) **NOTE**: In the **Alfabet Query Builder**, the value of any data type is written without inverted commas into the dialog box for entering values.

<=	Lesser than or equal to.
	If the object class property value is of the type `String` or `Date`, it must be written in inverted commas in the Alfabet query.
		`String`
		`Date`
		`Integer`
		`Real`
			`Application.StartDate <= '28/03/2008'`
			(finds applications with a start date of March 28th, 2008 and before)
			NOTE: In the **Alfabet Query Builder**, the value of any data type is written without inverted commas into the dialog box for entering values.

>	Greater than.
	If the object class property value is of the type `String` or `Date`, it must be written in inverted commas in the Alfabet query.
		`String`
		`Date`
		`Integer`
		`Real`
			`Application.StartDate > '28/03/2008'`
			(finds applications with a start date after March 28th, 2008)
			NOTE: In the **Alfabet Query Builder**, the value of any data type is written without inverted commas into the dialog box for entering values.

>=	Greater than or equal to.
	If the object class property value is of the type `String` or `Date`, it must be written in inverted commas in the Alfabet query.
		`String`
		`Date`
		`Integer`
		`Real`
			`Application.StartDate >= '28/03/2008'`
			(finds applications with a start date of March 28th, 2008 and higher)
			NOTE: In the **Alfabet Query Builder**, the value of any data type is written without inverted commas into the dialog box for entering values.

<>	Not equal to.
	If the object class property value is of the type `String` or `Date`, it must be written in inverted
		`String`
		`Date`
		`Integer`
		`Real`
			`Application.StartDate <> '28/03/2008'`
			(finds applications with a start date other than March 28th, 2008)

Operator	Description	For Data Type	Example
	commas in the Alfabet query.		**NOTE**: In the **Alfabet Query Builder**, the value of any data type is written without inverted commas into the dialog box for entering values.

`IS NULL`	The object class property is undefined. It has either not yet been assigned a value or a previous value has been deleted by the user.
	The operator has no value specification.
		`String`
		`Date`
		`Integer`
		`Real`
		`Reference`
		`ReferenceArray`
		`Boolean`
			`Application.ApplicationGroups IS NULL`
			(finds all applications that are not assigned to an application group)

`IS NOT NULL`	A value has been assigned to the object class property.
	The operator has no value specification.
		`String`
		`Date`
		`Integer`
		`Real`
		`Reference`
		`ReferenceArray`
		`Boolean`
			`Application.ApplicationGroups IS NOT NULL`
			(finds all applications that are assigned to at least one application group)

Booleans in WHERE Clauses

Boolean values are stored as `0` for false or `1` for true. To limit search results to objects for which a specific property of the data type `Boolean` is either false or true, you can use any of the following `WHERE` conditions:

For property = true	For property = false

`WHERE ClassName.PropertyName = 1`
`WHERE ClassName.PropertyName = true`
`WHERE ClassName.PropertyName = True`
	`WHERE ClassName.PropertyName = 0`
	`WHERE ClassName.PropertyName = false`
	`WHERE ClassName.PropertyName = False`

Please note that properties of the data type boolean can have three values:

- `0` = the value is set to false
- `1` = the value is set to true
- `Null` = the value is not set

In the Alfabet query language, a property that is `Null`, that means not set, is regarded as set to false and included into the results accordingly.

Specification of Dates

The format for dates is a string written as follows: DD/MM/YYYY. For example, a valid Alfabet query condition would be `Application.Startdate` = '27/09/2005'. The majority of date properties can be written like this, but there are a few system properties that save not only the date, but an actual timestamp that records the date and the time in milliseconds.

In the example `Application.CREATION_DATE` = '27/09/2005', it is likely that nothing will be returned even though it is a valid condition. But because every creation date is stored as a timestamp, it will only return applications that were created on the 27th of September in the year 2005 at exactly 12:00:00.000 pm (which is highly unlikely). However, you could theoretically write a condition like `Application.CREATION_DATE` = '27/09/2005 09:13:04.568 am', if you know the exact timestamp when the application you seek was created. Alternatively, you can find applications created on a specific date with a `WHERE` condition using a `BETWEEN` operator, For example, `Application.CREATION_DATE BETWEEN` '26/09/2005' `AND` '28/09/2005'.

Using Wildcards in WHERE Conditions

You can use the `%` symbol as a wildcard in the specification of `WHERE` clauses. For example,:

```
FIND
Application
WHERE Application.Name LIKE 'A%'
```

finds all application with a name starting with "A".

```
FIND
Application
WHERE Application.Name LIKE '%a%'
```

finds all application with a name containing an "a" at any position in the name.

You should take case-sensitivity into account when specifying `WHERE` clauses to search for a string or part of a string. For more information, see Defining Case-Sensitivity for WHERE Clauses.

Using Special Characters in WHERE Conditions

When the value specified for your `WHERE` clause contains special characters, the results might not be as expected because the characters are treated as operators rather than characters in a string. For example, the symbol `%` is used as a wildcard in the Alfabet query language. Therefore, the `WHERE` clause

```
WHERE ObjectClassName.PropertyName LIKE '10%'
```

will not limit search results to all values that equal "10%" but all values containing "10" followed by any characters.

You cannot search for the following special characters in Alfabet queries:

- %
- _
- *

Defining Case-Sensitivity for WHERE Clauses

By default, the specification of strings as values is case sensitive in `WHERE` conditions. However, case sensitivity also depends on the database server used and the specific setup. The definition of

```
WHERE Application.Name LIKE '%m%'
```

and

```
WHERE Application.Name LIKE '%M%'
```

will typically result in different output. The first definition finds all applications that have a lower-case m in their names and the second definition finds all applications that have an upper-case M in their names.

Per default, properties of the data type text (For example, descriptions) are read by the Alfabet query as all upper-case letters.

If you want to make the specification case-insensitive, you can define the Alfabet query to read all strings in the Alfabet database as written in upper-case letters only or in lower-case letters only and compare the results with the specified value.

To specify the comparison of the specified value to object class property values read in upper-case or lower-case letters for an existing `WHERE` condition in the **Alfabet Query Builder**:

1) In the **Query** tab of the **Alfabet Query Builder**, select the `WHERE` condition that you want to make case-insensitive.
2) In the toolbar, select either:
   - **To Upper**: To read all property values in upper-case letters.
   - **To Lower**: To read all property values in lower-case letters.

You can undo the choice by selecting **No Conversion** in the toolbar.

In the Alfabet query language, the conversion of property values to upper-case or lower-case letters is specified as:

```
WHERE (UPPERObjectClassName.PropertyName)OperatorValue
```

or

```
WHERE (LOWERObjectClassName.PropertyName)OperatorValue
```

With the `WHERE` clause:

```
WHERE (LOWER Application.Name) LIKE '%m%'
```

the Alfabet finds all applications with a lower-case "m" or an upper-case "M" in their name.

Reading all letters in a string as lower-case or upper-case letters is not case-insensitive. You must take this into account when specifying the value for your `WHERE` condition.

When you specify

```
WHERE (UPPER Application.Name) LIKE '%m%'
```

you will get no results because you are looking for an application with a lowercase "m" in the name while at the same time reading all letters in all application names as upper-case letters.

Comparing Property Values with Other Property Values

You can define a `WHERE` clause that checks whether the value of an object class property is identical to the value of another object class property. In the **Alfabet Query Builder**, the object class property you want to define as a value for the operator in the `WHERE` clause cannot be specified in the dialog box used to specify values for `WHERE` conditions.

To compare two properties in the **Alfabet Query Builder**:

1) In the **Query** tab of the **Alfabet Query Builder**, define a `WHERE` clause with an object class property in the usual way:

2) Select the object class property that you want to compare the defined object class property with in the object class property selector pane of the **Alfabet Query Builder** and drag it to the object class property name specification in the `WHERE` condition. The two properties are compared with the "=" operator.

3) To change the operator, click the `WHERE` condition and select the operator from the toolbar. The dialog box to enter a value opens.

4) Leave the fields in the dialog box empty and click **OK** . The operator in the `WHERE` condition has changed.

To compare two properties in the Alfabet query language as text, the value of the WHERE clause must be specified as ObjectClassName.PropertyName. The value is not written in inverted commas.

The following WHERE condition defines that only applications with an end date after the end date of the assigned ICT object are displayed in the report:

```
WHERE
Application.EndDate > ICTObject.EndDate
```

Referring to the Current Alfabet Context in a WHERE Condition

The Alfabet query language allows the specification of Alfabet parameters that refer to the current context in the Alfabet interface (For example, the object the user is currently working with, the user executing the report, the current workflow being executed).

In the Alfabet query language, Alfabet parameters are written as follows:

```
:ParameterName
```

or

```
@ParameterName
```

For example,:

```
FIND
ApplicationGroup
WHERE ApplicationGroup.BelongsTo CONTAINS:BASE
```

finds all application groups subordinate to the application group the user is currently working with.

During execution of the Alfabet query, the Alfabet parameter is substituted with the current return value. If no value is returned, the Alfabet query is executed without the WHERE statement.

The **Test Query**  button of the **Alfabet Query Builder** cannot be used to test queries with Alfabet parameters. If you define a configured report, you can test the query using the **Review Report** option of the context menu of the report.

The following Alfabet parameters can be used for Alfabet queries:

Parameter Name	Description
CURRENT_PROFILE	Returns the value of the REFSTR property of the current user profile
CURRENT_USER	Returns the value of the REFSTR property of the current user.

Parameter Name	Description

CURRENT_MANDATE	Returns the current mandate of the user. The Alfabet parameter can be used For example, to limit search results to objects that are assigned to the mandate the user is currently logged in with, For example,:
	`ALFABET_QUERY_500`
	`FIND ICTObject`
	`WHERE ICTObject.MANDATEMASK CONTAINS:CURRENT_MANDATE`
	NOTE: The parameter CURRENT_MANDATE (and the whole statement it is used in) is ignored if the user has no mandate assigned or is working as mandate master.

CURRENT_CULTURE	Returns the Microsoft® Windows® Locale identifier (LCID) of the current culture. Please note that the culture that is selected as default culture is always returned as 127, which means default, independent from the LCID of the culture.
TODAY	Returns the current date.
BASE	Returns the value of the REFSTR property of the current object.
ALFA_SERVER_URL	Returns the value of the attribute **Web Server** of the server alias configuration of the Alfabet component processing the query. The attribute Web Server is editable in the **Overview** tab of the server alias configuration editor.
BASE<number>	For queries defined in the context of layered diagrams and grid reports only! If the report is configured to allow cascading queries, and more than one level of related objects us defined, BASE0 returns the value of the REFSTR property of the parent object in the directly superordinate level of the diagram. BASE1 returns the ancestor object in the level superordinate to the parent object level etc.
WIZARDBASE	For queries defined in the context of wizard configuration only! Return the value of the REFSTR property of the base object of the wizard. The base object of the wizard can be different from the base object of a wizard step. This parameter allows to refer to the base object of the wizard For example, in pre-conditions of steps that have a different base object than the overall wizard.
WORKFLOW	For queries defined in the context of workflow configuration only! Returns the value of the REFSTR property of the workflow. Each workflow corresponds to an object Workflow in the Alfabet database. The parameter is useful when defining queries that refer to properties of the class Workflow, like For example, the workflow owner or the workflow initiator. For more information, see Defining the Users Responsible for a Workflow Step.

Parameter Name	Description
WORKFLOWBASE	For queries defined in the context of workflow configuration only! Returns the value of the REFSTR property of the base object of the workflow. The base object of the workflow can be different from the base object of a workflow step. This parameter allows to refer to the base object of the workflow For example, in pre-conditions of steps that have a different base object than the overall workflow.
LAST_EXECUTION	For queries defined in the context of monitors only. Returns the date of the last execution of the monitor.

Adding Multiple WHERE Conditions to an Alfabet Query

You can add multiple WHERE conditions to the WHERE clause of an Alfabet query. When you do so, you must specify in the WHERE clause whether the WHERE conditions are AND or OR related:

AND: Returns True when all expressions are true.

OR: Returns True when at least one expression is true.

More than two WHERE conditions can be related with each other in an AND or OR relationship.

You can combine multiple AND and OR conditions hierarchically. For example, you can define that condition A and either condition B or condition C should apply.

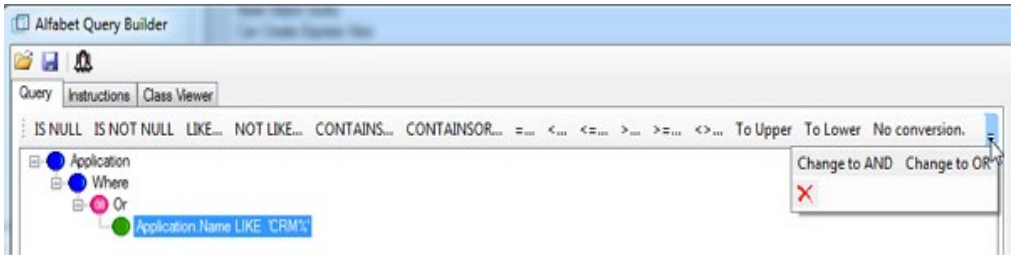
All WHERE conditions of an Alfabet query are written in the same WHERE clause. It is not allowed to specify multiple WHERE clauses in an Alfabet query.

In the **Alfabet Query Builder**, the relation ***And*** is automatically added to the WHERE clause upon definition of the first WHERE condition.

When you close and re-open the **Alfabet Query Builder** while a single condition is defined, the node And is removed from the Alfabet query. If you want to add another condition, you can re-establish the And node or create an Or node:

1) In the **Query** tab of the **Alfabet Query Builder**, click the condition in the WHERE clause.
2) In the toolbar, select **Change to AND** to add an ***And*** relation or **Change to OR** to add an ***Or*** relation.

If a high number of operators is available for the condition, you might not see the options **Change to AND** and **Change to OR** in the toolbar because there is not sufficient space to display all options. Click on the small field on the right of the toolbar to open a drop-down menu with the menu options that were hidden.



You can do one of the following to build a hierarchy of related conditions:

- To add a condition to an ***And*** or ***Or*** node, click the node and select **Property** from the toolbar.
- To add a subordinate condition to an ***And*** or ***Or*** node, click the node and select **AND** or **OR** from the toolbar.
- To change an ***And*** node to an ***Or*** node, click the node and select **Change to OR** from the toolbar.
- To change an ***Or*** node to an ***And*** node, click the node and select **Change to AND** from the toolbar.
- To delete an ***And*** node, an ***Or*** node, a condition or the complete `WHERE` clause from the Alfabet query, click the respective node and select **Delete** from the toolbar.

In the Alfabet query language, multiple `WHERE` conditions in an Alfabet query are defined as:

```
WHERE
    (AND|OR
        Condition1
        Condition2
    )
```

The expression:

```
WHERE
    (AND
        Application.ResponsibleUser IS NULL
        Application.Status = 'Approved'
    )
```

finds all applications with the status `Approved` and no responsible user assigned.

In the Alfabet query language as text, the combination of `AND` or `OR` relations results in a code like the following:

```
WHERE
    (AND|OR
        Condition1
        (AND|OR
            Condition2
            Condition3
```



```
        )
    )
```

Creating a Filter Field with a WHERE Clause

You can allow users to select the output of a configured report at runtime by means of filters that you configure. The definition of filters is described in detail in the section Defining Filters for Configured Reports and Selectors.

Specifying JOINs in Alfabet Queries

Basic information about `JOIN`s is provided in the section *Building an Alfabet Query*:

- *Defining JOINs*
- *Deciding About the Type of JOIN to Use*

This section provides additional information on how to add special classes to Alfabet queries or how to add `JOIN`s for specific circumstances. The following information is available:

- Operators for the Definition of JOIN Conditions
- Specifying Multiple JOIN Conditions for one JOIN
- Joining Classes Already Specified in the Alfabet Query
- Using DISTINCT to Avoid Rows with Identical Content in Results
- Defining JOINs for Roles
- Defining JOINs for Indicators
- Defining JOINs Between Value Nodes, Projects, or Demands and the Affected Architecture
- Defining JOINs to Object Class Stereotypes

Operators for the Definition of JOIN Conditions

The general construction of a `JOIN` is:

```
JoinTypeReferencedObjectClassONObjectClassName.PropertyNameRelationalOperatorObjectClassName.PropertyName
```

The most straight-forward relationship between two object classes is one in which an object class property with the data type `Reference` or `ReferenceArray` references another object class. To specify that relation, you must define the `JOIN` as follows:

```
JoinTypeReferencedObjectClassONObjectClassName.ReferencingProperty=ReferencedObjectClassName.RESTR
```

or:

*JoinTypeReferencedObjectClass*ON*ReferencedObjectClassName*.*ReferencingProperty*=*ObjectClassName*.RESTR

For example, the JOIN

```
LeftJoin Application ON BusinessProcess.Applications =
Application.REFSTR
```

adds all applications assigned to the business processes with the property Applications to the search result of the Alfabet query.

The ability to build a JOIN based on properties of the type ReferenceArray depends on the attribute **Reference Support** of the property:

- If **Reference Support** is set to True, the relational operator "=" is used in the expression. It is used in the JOIN as "includes" or "is included in" rather than "is equal to" operator.
- If **Reference Support** is set to False, you cannot build a JOIN on the property.

JOINs can also specify other relations between two objects either instead of or in addition to a direct reference by a property of the type Reference or ReferenceArray. Any property of the joined object class can have a relation with any other property of the base class or an already joined class in the Alfabet query: .

For example, the JOIN

```
FIND Application

InnerJoin ApplicationGroup ON ApplicationGroup.ResponsibleUser =
Application.ResponsibleUser
```

finds all combinations of application groups and applications for that the same user is defined as responsible user.

In most cases, this type of comparison is useful in combination with a JOIN condition based on a direct reference. In the example, it is probable that the user wants to make sure that the person that is responsible for an application group is a supervisor that is not identical with the person responsible for any individual applications in the application group. The information about the responsible user, that has nothing to do with the relation between application group and applications itself, also combines application groups with applications that are not assigned to that application group. A second JOIN condition must be defined that limits the search results to applications that are assigned to the application group. For information about how to join a class with multiple JOIN conditions, see Specifying Multiple JOIN Conditions for one JOIN.

JOIN conditions allow the use of the following operators:

Operator	Description
=	Equal to
<	Lesser than

Operator	Description
<=	Lesser than or equal to
>	Greater than
>=	Greater than or equal to
<>	Not equal to

Specifying Multiple JOIN Conditions for one JOIN

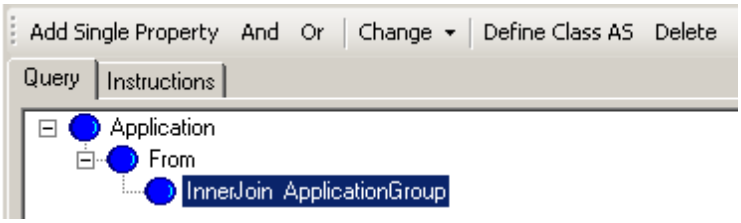
In some cases, you may need to define that more than one property from the first class must match a property of the second class. To do so, you must define all `JOIN` conditions in one `JOIN` and specify in the `JOIN` whether the `JOIN` conditions are `AND` or `OR` related:

`AND`: Returns `True` when all expressions are true.

`OR`: Returns `True` when at least one expression is true.

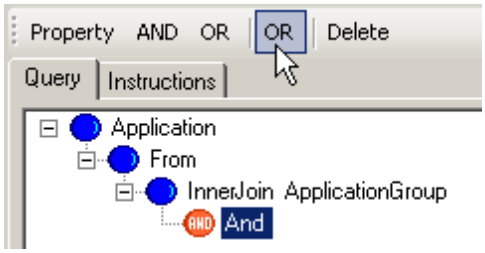
In the **Alfabet Query Builder**, you can decide to use multiple `JOIN` conditions directly after defining the `JOIN`. If a `JOIN` already has a condition, you must first delete that condition to be able to specify multiple conditions.

1) In the **Query** tab of the **Alfabet Query Builder**, click the `JOIN` that has no condition specified.



2) In the toolbar, select either **And** or **Or** to define whether the conditions are `AND` or `OR` related. The selected condition is added to the `JOIN`.
3) In the toolbar, select **Property** and specify the `JOIN` condition as usual.
4) Repeat step 3) to add the second condition.

You can later decide to change the condition from `AND` to `OR` and vice versa. To do so, click the condition in the Alfabet query. You will see the buttons **Property**, **AND**, and **OR** in the toolbar and behind that another button for the condition that you might want to change to. Click that button to change the condition.



In the Alfabet query language, multiple `JOIN` conditions in an Alfabet query are defined as follows:

```
JoinTypeObjectClassNameON (And|OrCondition1Condition2)
```

The expression:

```
InnerJoin ApplicationGroup ON (And Application.ApplicationGroups=
ApplicationGroup.RESTR Application.MANDATEMASK
<>ApplicationGroup.MANDATEMASK)
```

finds all application groups that the application is assigned to and that have a different mandate specification as the application.

More than two `JOIN` conditions can be related with each other in an `AND` or `OR` relationship.

You can combine multiple `AND` and `OR` conditions hierarchically. For example, you can define that condition A and either condition B or condition C should apply.

In the **Alfabet Query Builder**, you can select a new `AND` or `OR` relation from the toolbar when you click an existing `AND` or `OR` relation in the Alfabet query.

In the **Alfabet Query Builder**, the combination of `AND` or `OR` relations results in a code similar to the following:

```
JoinTypeObjectClassNameON (And|OrCondition1 (And|OrCondition2Condition3))
```

The expression:

```
InnerJoin ApplicationGroup ON (And
Application.ApplicationGroups=ApplicationGroup.RESTR (Or
Application.MANDATEMASK<>ApplicationGroup.MANDATEMASK
Application.ResponsibleUser<>ApplicationGroup.ResponsibleUser))
```

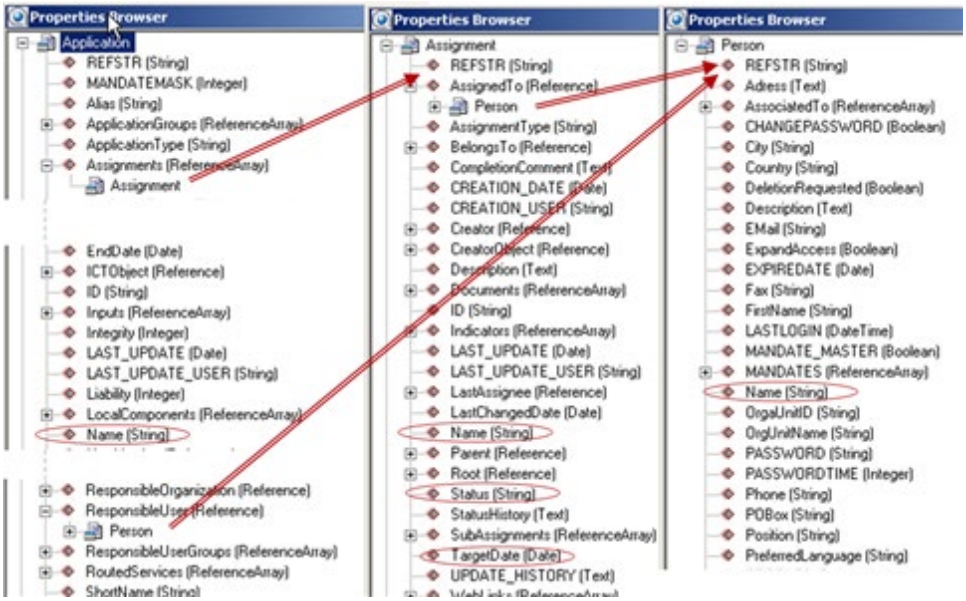
finds all application groups that the application is assigned to and that have either a different mandate specification or a different responsible user as the application.

Joining Classes Already Specified in the Alfabet Query

For some queries, it is necessary to join the same object class two times with different `JOIN` conditions.

For example, an Alfabet query is defined for a report about assignments for applications. The name and responsible user of the application, the name of the assignment, and the status, target date and user responsible for the assignment shall be displayed in the report.

JOINs to the classes Assignment and Person must be added to the Alfabet query in order to find the relevant assignments of the query and the users that are the originators and the responsible users/organizations for the assignment.



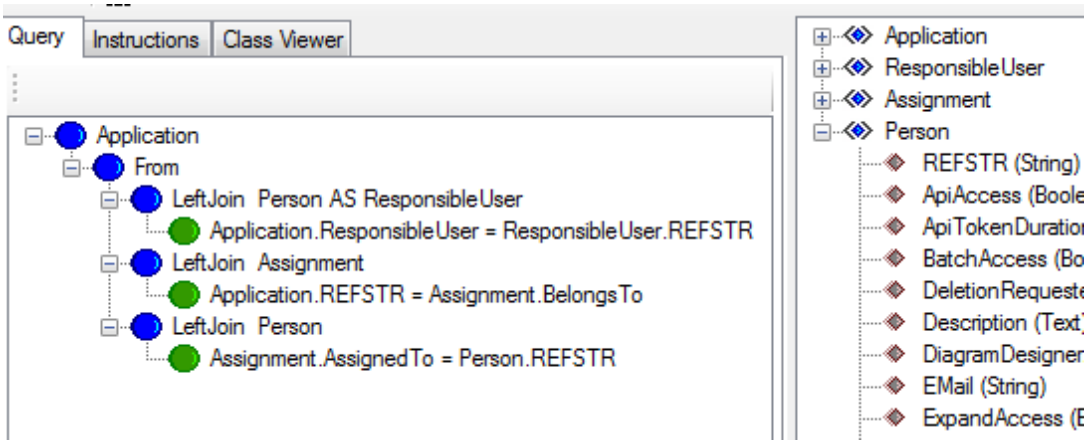
The Assignments property of the class Application references the class Assignment. The attribute ResponsibleUser of the class Application and the attribute AssignedTo of the class Assignment reference the class Person. That means that the class Person is referenced twice in the Alfabet query, each JOIN finding a different subset of objects of the class Person. The name of the person shall be displayed twice in the report: in a column for responsible user and in a column for the assignment's responsible person/organization.

To specify two different JOINs to the same object class, you must define an alias for the object class in the first JOIN before adding the second JOIN. The alias can contain letters and numbers, but no special characters or whitespaces. The alias will be used as the object class name for the object classes matching the specified JOIN conditions. You must use the alias instead of the original object class name in the specification of the JOIN condition, WHERE clauses, and Show and Sort properties.

To define the alias for a joined class in the **Alfabet Query Builder**:

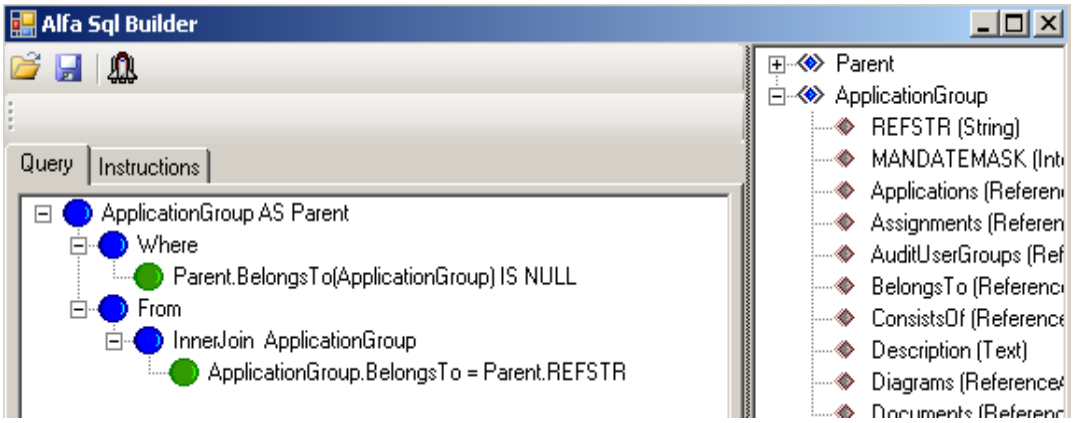
1) In the **Query** tab of the **Alfabet Query Builder**, add a JOIN for the object class.
2) Click the JOIN and select **Define Class As** in the toolbar. A dialog box opens.
3) Define the alias name in the dialog box and click **OK** . You can see the alias class name in the property selector and, if a condition for the JOIN was already defined, in the JOIN conditions.

In the following example, the class Person is renamed to ResponsibleUser in the specification of the user that is the assigned responsible user for the base class application. The class is not renamed for the JOIN that specifies the relation between the classes Person and Assignment. In the property selector, ResponsibleUser is listed as well as Person. Both sections contain all properties, roles and indicators for the object class Person.



Specification of an alias is also required to add a `JOIN` to the base object class of the Alfabet query. The alias is then required on the base class. In the **Alfabet Query Builder**, an alias on the base class can be specified by clicking the base class instead of the `JOIN` and following the procedure described above.

In the following example, application groups that are assigned to top-level application groups shall be displayed in a report. The base class is `ApplicationGroup`, specified with the alias `Parent`. A `WHERE` clause defines that only application groups with no superordinate application groups are found as parent. Subordinate application groups are defined with a `JOIN` to all objects of the class `ApplicationGroup` that belong to one of the parent application groups.



In the Alfabet query language, the code for the definition of an alias is as follows:

```
ObjectClassNameASAliasName
```

To define an alias for the base class:

```
FIND

ObjectClassNameASAliasName
```

To define an alias for a joined class:

```
JoinTypeReferencedObjectClassASAliasNameONObjectClassName.PropertyNameRelati
onalOperatorAliasName.PropertyName
```

In the XML definition of Show and Sort properties, the original object class name must be defined in the attribute `ClassName` and an additional attribute `ClassAlias` must be added that specifies the alias name.

For example, the following Alfabet query:

```
FIND

Application

LeftJoin Person AS ResponsibleUser ON Application.ResponsibleUser =
ResponsibleUser.REFSTR

LeftJoin Assignment ON Application.REFSTR = Assignment.BelongsTo

LeftJoin Person AS Assignee ON Assignment.AssignedTo =
Assignee.REFSTR

QUERY_XML

<QueryDef>

   <ShowProperty Type="Property" ClassName="Application" Name="Name"
   />

   <ShowProperty Type="Property" ClassName="Person"
   ClassAlias="ResponsibleUser" Name="Name"/>

   <ShowProperty Type="Property" ClassName="Assignment" Name="Name"
   />

   <ShowProperty Type="Property" ClassName="Person"
   ClassAlias="Assignee" Name="Name" />

</QueryDef>
```

finds all applications, their assignments and all users that are defined as the responsible user for an application or assignee of an assignment. In the Show properties, the `Name` property of the object class person is displayed two times. Because of the specification of an alias, the distinction can be made between the responsible user and the assignee.

Using DISTINCT to Avoid Rows with Identical Content in Results

If an object class is joined to the base class by means of multiple joins, and the Show properties do not include data about all of the object classes in the `JOIN`, the results may display multiple rows with identical data. In some cases, the results can be limited to rows with differing content by using a `DISTINCT` clause in the specification of the base class.

For example,:

The Alfabet query finds all business functions that are related to ICT objects. However, business functions are not directly related to ICT objects. Business functions use business services that are provided by applications. Therefore, business functions that use business services by applications that are assigned to an ICT object are the relevant business functions that are related to the ICT object.

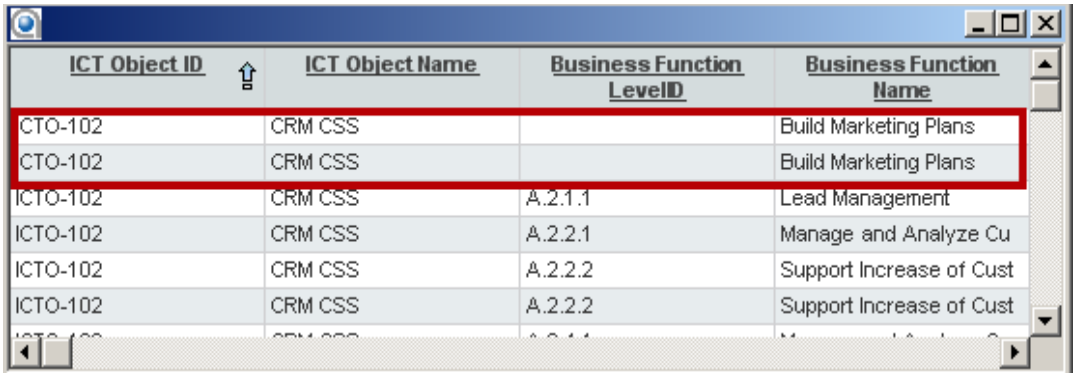
The Alfabet query must Therefore, join three object classes: `Application, BusinessService` and `BusinessFunction`. In the example above, ICT objects, applications assigned to the ICT object, and the business functions that use business services of the applications are displayed in the results.

Some business functions use business services from more than one application assigned to an ICT object. As a result, the report contains rows that only differ in the data about the application:


```
ALFABET_QUERY_500

FIND

ICTObject

InnerJoin Application ON Application.ICTObject = ICTObject.REFSTR

InnerJoin BusinessService ON Application.REFSTR =
BusinessService.Object

InnerJoin BusinessFunction ON BusinessService.Function =
BusinessFunction.REFSTR

QUERY_XML

<QueryDef>

   <ShowProperty Type="Property" ClassName="ICTObject" Name="ID" />

   <ShowProperty Type="Property" ClassName="ICTObject" Name="Name"
   />

   <ShowProperty Type="Property" ClassName="Application" Name="ID"
   />

   <ShowProperty Type="Property" ClassName="Application" Name="Name"
   />

   <ShowProperty Type="Property" ClassName="BusinessFunction"
   Name="LevelID" />

   <ShowProperty Type="Property" ClassName="BusinessFunction"
   Name="Name" />

   <SortProperty Type="Property" ClassName="ICTObject" Name="ID" />

   <SortProperty Type="Property" ClassName="ICTObject" Name="Name"
   />

   <SortProperty Type="Property" ClassName="BusinessFunction"
   Name="LevelID" />

   <SortProperty Type="Property" ClassName="BusinessFunction"
   Name="Name" />

</QueryDef>
```

ICT Object ID	ICT Object Name	Application ID	Application Name	Business Function Level	Business Function Name
ICTO-102	CRM CSS	APP-2694	CRM CSS		Build Marketing
ICTO-102	CRM CSS	APP-2695	CRM CSS		Build Marketing
ICTO-102	CRM CSS	APP-2695	CRM CSS	A.2.1.1	Lead Manageme
ICTO-102	CRM CSS	APP-2695	CRM CSS	A.2.2.1	Manage and Ana
ICTO-102	CRM CSS	APP-2695	CRM CSS	A.2.2.2	Support Increase
ICTO-102	CRM CSS	APP-2694	CRM CSS	A.2.2.2	Support Increase
ICTO-102	CRM CSS	APP-2694	CRM CSS	A.2.4.1	Manage and Ana

If the report displays only data about the ICT object and the business function, you will see a report with rows displaying identical data. The difference in the data set (in this case, the application providing the business service used by the business support) is not visible:



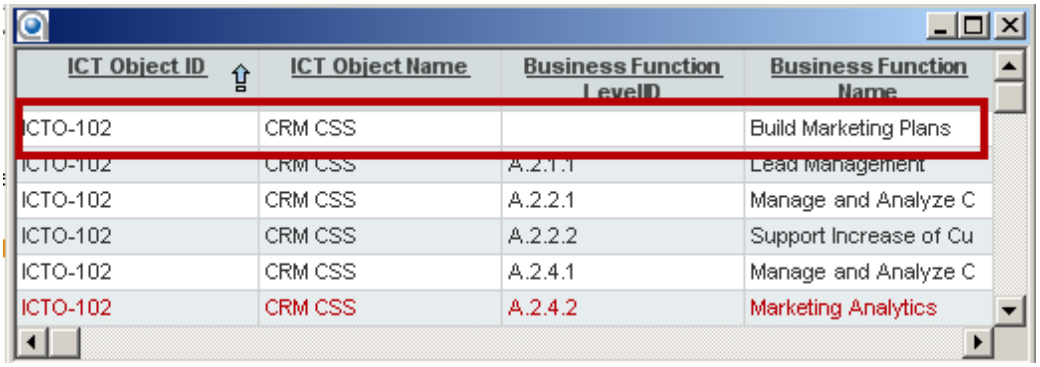
ICT Object ID	ICT Object Name	Business Function LevelID	Business Function Name
CTO-102	CRM CSS		Build Marketing Plans
CTO-102	CRM CSS		Build Marketing Plans
ICTO-102	CRM CSS	A.2.1.1	Lead Management
ICTO-102	CRM CSS	A.2.2.1	Manage and Analyze Cu
ICTO-102	CRM CSS	A.2.2.2	Support Increase of Cust
ICTO-102	CRM CSS	A.2.2.2	Support Increase of Cust

To solve the problem of the duplicate display of data in a scenario as described above, the Alfabet query must have a `DISTINCT` option in the specification of the base class. `DISTINCT` specifies that the resulting report should contain only rows displaying distinct objects. .

For example, if the Alfabet query specified in the example above contains a `DISTINCT` option and no Show properties for applications:

```
ALFABET_QUERY_500

FIND

DISTINCT ICTObject

InnerJoin Application ON Application.ICTObject = ICTObject.REFSTR

InnerJoin BusinessService ON Application.BusinessServices =
BusinessService.REFSTR

InnerJoin BusinessFunction ON BusinessService.Function =
BusinessFunction.REFSTR

QUERY_XML

<QueryDef>

   <ShowProperty Type="Property" ClassName="ICTObject" Name="ID" />

   <ShowProperty Type="Property" ClassName="ICTObject" Name="Name"
   />

   <ShowProperty Type="Property" ClassName="BusinessFunction"
   Name="LevelID" />

   <ShowProperty Type="Property" ClassName="BusinessFunction"
   Name="Name" />

   <SortProperty Type="Property" ClassName="ICTObject" Name="ID" />

   <SortProperty Type="Property" ClassName="ICTObject" Name="Name"
   />

   <SortProperty Type="Property" ClassName="BusinessFunction"
   Name="LevelID" />

   <SortProperty Type="Property" ClassName="BusinessFunction"
   Name="Name" />

</QueryDef>
```

The resulting report shows only lines with different content:



ICT Object ID	ICT Object Name	Business Function LevelID	Business Function Name
ICTO-102	CRM CSS		Build Marketing Plans
ICTO-102	CRM CSS	A.2.1.1	Lead Management
ICTO-102	CRM CSS	A.2.2.1	Manage and Analyze C
ICTO-102	CRM CSS	A.2.2.2	Support Increase of Cu
ICTO-102	CRM CSS	A.2.4.1	Manage and Analyze C
ICTO-102	CRM CSS	A.2.4.2	Marketing Analytics

Note the following when using `DISTINCT`:

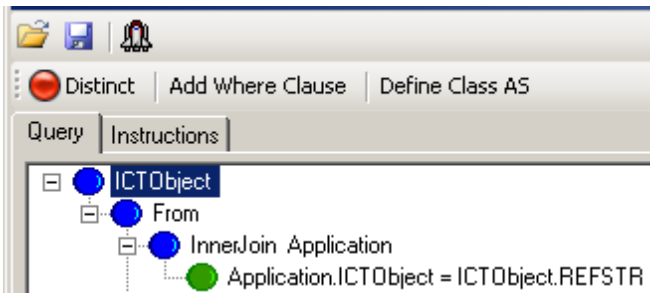
- The parameter `DISTINCT` cannot be used in Alfabet queries that contain object class properties of the **Data Type** `StringArray` or `Text` if the Alfabet database is located on an Oracle® 10g database server.
- In the Alfabet query language, `DISTINCT` forces the display of different objects in each row. This behavior is different from the use of `DISTINCT` in SQL, where `DISTINCT` refers to different object properties.

  For example, if you find applications with the `DISTINCT` option and you only display the name of the application in the report, you may see rows that look identical regardless of the `DISTINCT` setting, because the `Name` property is not unique for applications. Two applications can have the same name as long as they have different versions.

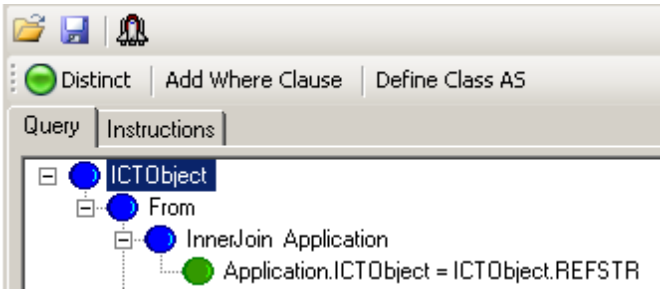
- Also, `DISTINCT` is only applied to the base class of the Alfabet query. If your Alfabet query does not contain any Show properties for the base class, `DISTINCT` will not have any effect.

To add `DISTINCT` to the base class definition in the **Alfabet Query Builder**:

1) In the **Query** tab of the **Alfabet Query Builder**, click the base class. In the toolbar, the **Distinct** button is displayed with a red light, which means that `DISTINCT` is not selected:



2) In the toolbar, click the **Distinct** button. The button now displays a green light which means that `DISTINCT` is selected.



To add `DISTINCT` clause to the base class definition in the Alfabet query language, write `DISTINCT` between `FIND` and the name of the base class:

```
ALFABET_QUERY_500
FIND DISTINCT Application
```

Defining JOINs for Roles

An object of the object class `Role` is created each time a responsibility is defined for an object based on a preconfigured role type.

Role types define the type of responsibility a user or organization has for an object in the Alfabet database. Role types are configured in the **Reference Data** functionality of the **Configuration** module of Alfabet. They are then assigned to object classes in the **Class Configuration** functionality.

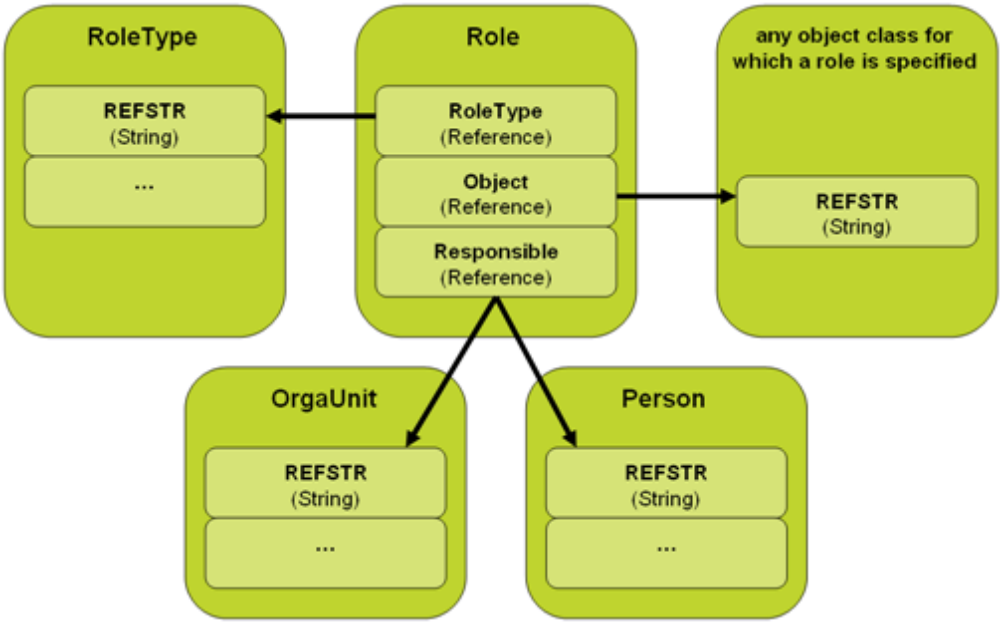
In the **Responsibilities** page view of an object, the user can now create a role by defining a responsibility for either a user or an organization. In the editor for the role definition, the responsible user or organization and the role type that defines the type of responsibility for the object is defined.

The Alfabet query language offers an easy way to include information about responsible users or organizations defined for roles in the results of an Alfabet query. Roles can be specified in the Show properties in an XML definition. The role is automatically interpreted in the XML definition and the name of the responsible user or organization is returned in a separate column per role type. For information about the definition of Show properties for roles, see the section Displaying Users/Organizations Defined for a Role in Alfabet Query Results.

Nevertheless it might be necessary to use `JOIN`s to display the responsible users and organizations defined for roles in reports:

- If the output of the report shall not include the name of the responsible users/organizations, in a separate column per role type. For example, you might want to display users and organizations in different columns, or you might want to add a column showing the role type and a second column showing the responsible for the role type for the object.
- If the output of the report is to be sorted by the user/organization that is responsible for the role. It is not possible to include the role output in the Sort properties without `JOIN`s.

In the Alfabet class model, the relation between the role type, the object, and the responsible person or organization is only specified in the database table for the class `Role`:



To include information about roles in the Alfabet query results, you must first join the class `Role` to the Alfabet query. You can then join all other involved object classes with a relation to the class `Role`.

The property `Responsible` of the class `Role` can reference two object classes: `OrgaUnit` and `Person`. Therefore, you must join both object classes in the Alfabet query to find all roles defined for an object.

For example, to display information about the roles that are defined for objects of a defined object class:

1) Create a `JOIN` to the class `Role` that finds all roles with a reference to objects of the base class in the property `Object`. Create an inner `JOIN` to include only objects with defined roles in the report or a left `JOIN` to find all objects of the base class:

2) Create a `JOIN` to the class `RoleType` that finds all role types that are referenced in the `RoleType` attribute of the roles found in your Alfabet query. You can use a left `JOIN` or an inner `JOIN`. Both results are identical because each role must have a role type defined:

   You can now add the `Name` property of the role type to your Show properties.

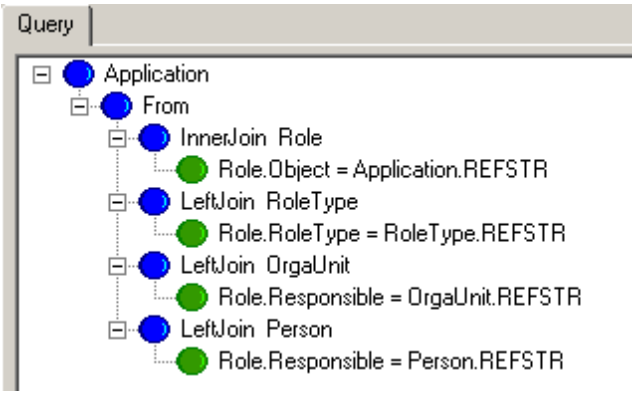
3) Create a left `JOIN` to the class `Organization` that finds all organizations that are referenced in the `Responsible` attribute of the roles found in your Alfabet query. You must define a left `JOIN` to also display applications with responsibilities defined for users instead of organizations. If you are interested in only organizational responsibilities, you can define an inner `JOIN`, but then applications without any roles defined will not be included in the Alfabet query results.

   You can now add the `Name` property or any other properties of the responsible organization to your Show properties.

4) Create a left `JOIN` to the class `Person` that finds all users that are referenced in the `Responsible` attribute of the roles found in your Alfabet query. You must define a left `JOIN` to also display applications with responsibilities defined for organizations instead of users. If you are interested in only user responsibilities, you can define an inner `JOIN`, but then applications without any roles defined will not be included in the Alfabet query results.

You can now add the `Name` property or any other properties of the user to your Show properties.

The resulting Alfabet query will appear as follows:



Defining JOINs for Indicators

In the **Evaluations and Portfolios** functionality in the **Configuration** module in Alfabet, you can define evaluation types that bundle a set of indicator types in order to evaluate objects in Alfabet. The evaluation types are assigned to object classes in the **Class Configuration** functionality.

The indicator types defined for the evaluation types that are assigned to the object class are displayed on the *Evaluation Page View* of each object in the object class. The user can define indicators for the object by defining the value of the indicator type in the *Evaluation Page View*.

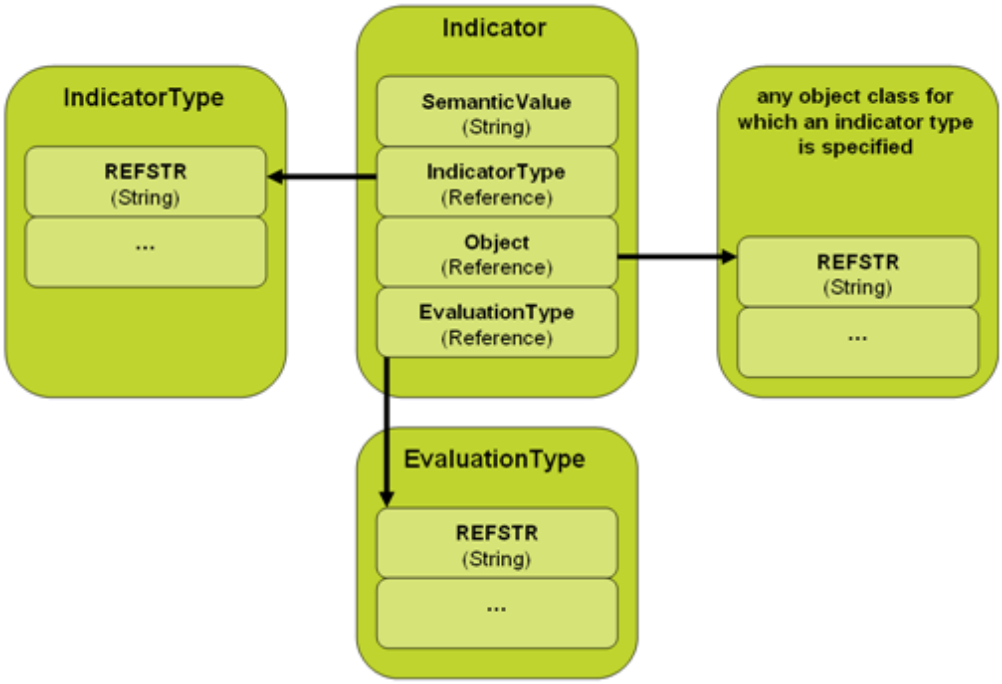
An object of the object class Indicator is created each time a value is entered for an indicator type on the **Evaluations** page view of an object.

The Alfabet query language offers an easy way to include information about indicators in the results of an Alfabet query. Indicators can be specified in the Show properties in XML definition. The indicator is automatically interpreted in the XML definition and the name of the values are returned in a separate column per indicator type. For information about the definition of Show properties for indicator types, see Displaying Users/Organizations Defined for a Role in Alfabet Query Results.

Nevertheless it might be necessary to use `JOIN`s to display indicators in reports:

- If the output of the report shall not display a separate column per indicator type. For example, you might want to display one column for the indicator type and one for the value defined for the indicator type.
- If the output of the report shall be sorted by indicator value. It is not possible to include the indicator output in the Sort properties without `JOIN`s.

In the Alfabet class model, the value defined for an object's indicator type is only specified in the database table for the class `Indicator`. The class `Indicator` specifies the object and the indicator type that each value was defined for and the evaluation type that the indicator type is assigned to:

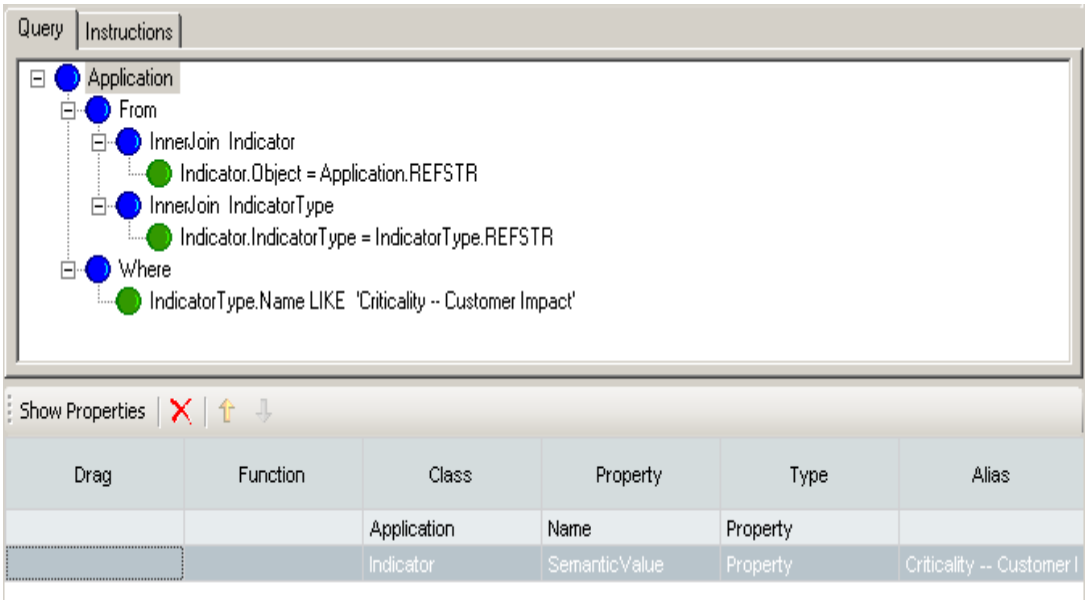


To include information about indicators in the Alfabet query results, you must first join the class `Indicator` to the Alfabet query. You can then join all other relevant object classes via a relation to the class `Indicator`.

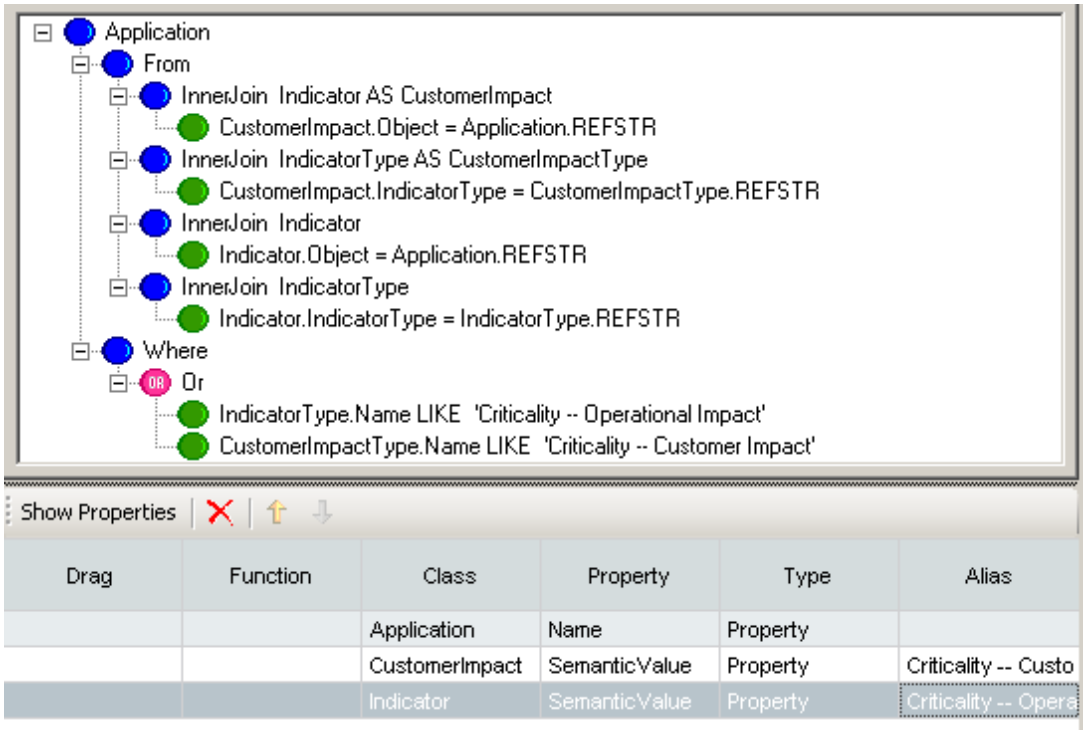
For example, to display information about the values of a specific indicator type that is defined for objects of a defined object class:

1) Create a `JOIN` to the class `Indicator` that finds all Indicators with a reference to objects of the base class in the property `Object`. Create an inner `JOIN` to include only objects with defined indicators in the report or a left `JOIN` to find all objects of the base class:
2) Create a `JOIN` to the class `IndicatorType` that finds all Indicator types that are referenced in the `IndicatorType` property of the indicators found in your Alfabet query. You can use a left `JOIN` or an inner `JOIN`. Both results are identical because each indicator must have an indicator type defined:
3) Add a `WHERE` condition to your Alfabet query that defines that the `Name` property of the indicator type must correspond to the name of the indicator for which you want to display values in the report.
4) Define the property `Name` of the object and the property `SemanticValue` of the indicator as Show properties and change the column caption to the name of the indicator.

The resulting Alfabet query will appear as follows:



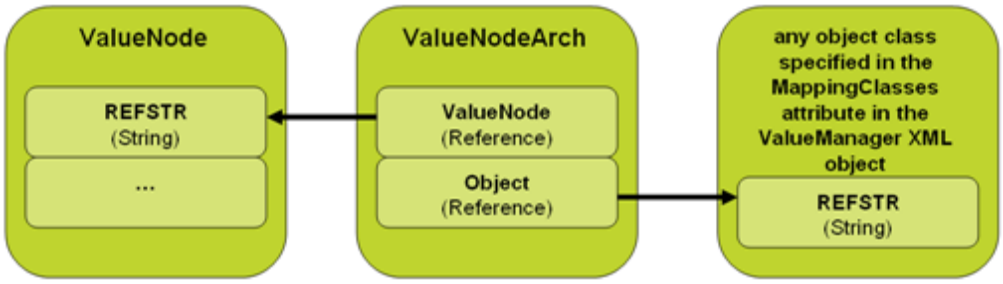
If you want to show the values of more than one indicator type and add one column to your report for each indicator type, you must join the class Indicator two times to your report and specify an alias name for one of the indicators. The same applies to the indicator type `JOIN`. For information on how to join one object class multiple times to an Alfabet query, see the section Joining Classes Already Specified in the Alfabet Query.



Defining JOINs Between Value Nodes, Projects, or Demands and the Affected Architecture

For value nodes, projects and demands, the affected architecture is not defined as a property of the value node, project and demand, but with a special object class `ValueNodeArch`, `DemandArch` and `ProjectArch` respectively. The class `ValueNode` is used to describe the relation to the affected architecture in the following example. The relation between projects or demands and their affected architecture follows the same rules.

An object of the object class `ValueNodeArch` (value node arc) is created each time a user adds an architecture element to the affected architecture of a value node stereotype either on the *Affected Architecture Page View* of the value node stereotype or on the *Impacting Value Nodes Page View* for the architecture object. The relation between the value node and the architecture object is neither stored in the database table of the value node nor in the database table of the architecture object. The relation between the a value node and its affected architecture is only specified in the database table for the class `ValueNodeArch`:



To include information about the affected architecture of a value node or the impacting value nodes assigned to an object in the Alfabet query results, you must first join the class `ValueNodeArch` to the Alfabet query. You can then join all other involved object classes with a relation to the class `ValueNodeArch`:

- Value nodes are based on value node stereotypes. To limit the search results to value nodes of a certain value node stereotype only, you must include a `WHERE` statement as described in the section Defining JOINs to Object Class Stereotypes.
- The property `Object` of the class `ValueNodeArch` can reference objects of any object class that are defined in the `MappingClasses` attribute for a value node stereotype in the XML object ***Value Manager***. To create a `JOIN` to the affected architecture, you must add a `JOIN` to all involved object classes to the Alfabet query. For information about the XML specification for value nodes, see Configuring the Strategy Deduction Capability in the chapter Configuring Alfabet Functionalities Implemented in the Solution Environment.

For example, an Alfabet query shall find all objects of the affected architecture for a value node of the stereotype `Architecture Requirements` that has the following stereotype definition:

```
<ValueStereotype
   Name="Architecture Requirements"
   Level="5"
   MappingClasses="Domain,BusinessProcess,OrgaUnit,MarketProduct"
   Picture="ArchitectureRequirement"/>
```

In the Alfabet query, you must define the following:

1) Define `ValueNode` as the base class.

You can now add the name of the value node to your Show properties.

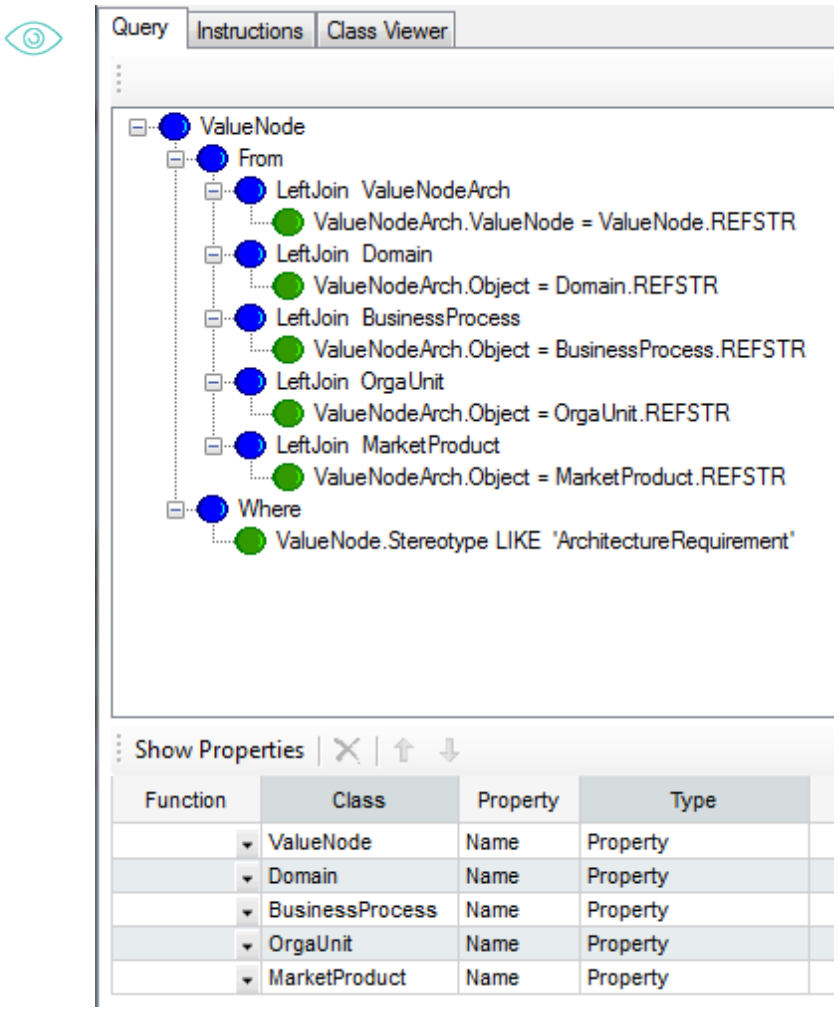
2) Add a `WHERE` clause to the Alfabet query that defines that the `Stereotype` property of the value nodes must have the value "`Architecture Requirements`".

3) Create a `JOIN` to the class `ValueNodeArch` that finds all value node arcs. Create an inner `JOIN` to include only value nodes with defined affected architecture in the report or a left `JOIN` to find all value nodes.

4) Create a `JOIN` to each of the object classes defined as `MappingClasses` in the stereotype definition of the value node. The `JOIN` condition must define that the object class is referenced by the object class property `Object` of the object class `ValueNodeArch`. You can use a left `JOIN` or an inner `JOIN`. Both results are identical because each value node arch must have an object defined.

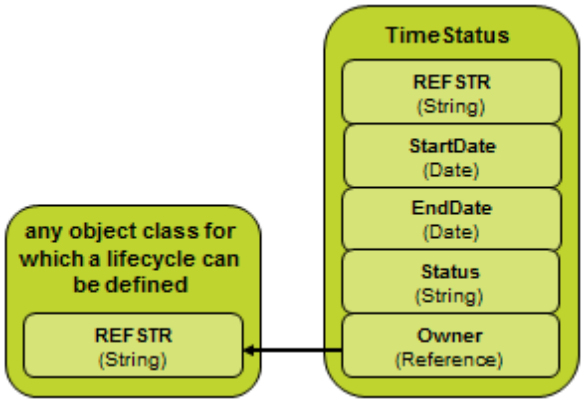
   You can now add the names of the object classes to your Show properties.

The resulting Alfabet query will appear as follows:



Referring to the Lifecycle of an Object in the Alfabet Query

In the Alfabet database, the lifecycle definition and lifecycle phases for an object are stored as objects of the object class `TimeStatus`. Each `TimeStatus` object provides information about the start date, the end date and the lifecycle status of the lifecycle phase stored in the object as well as a reference to the object that the lifecycle phase is defined for in the property `Owner`. The lifecycle status can be any of the names of the lifecycle phases defined for the object class in the XML object ***ObjectLifeCycleManager*** in Alfabet Expand.



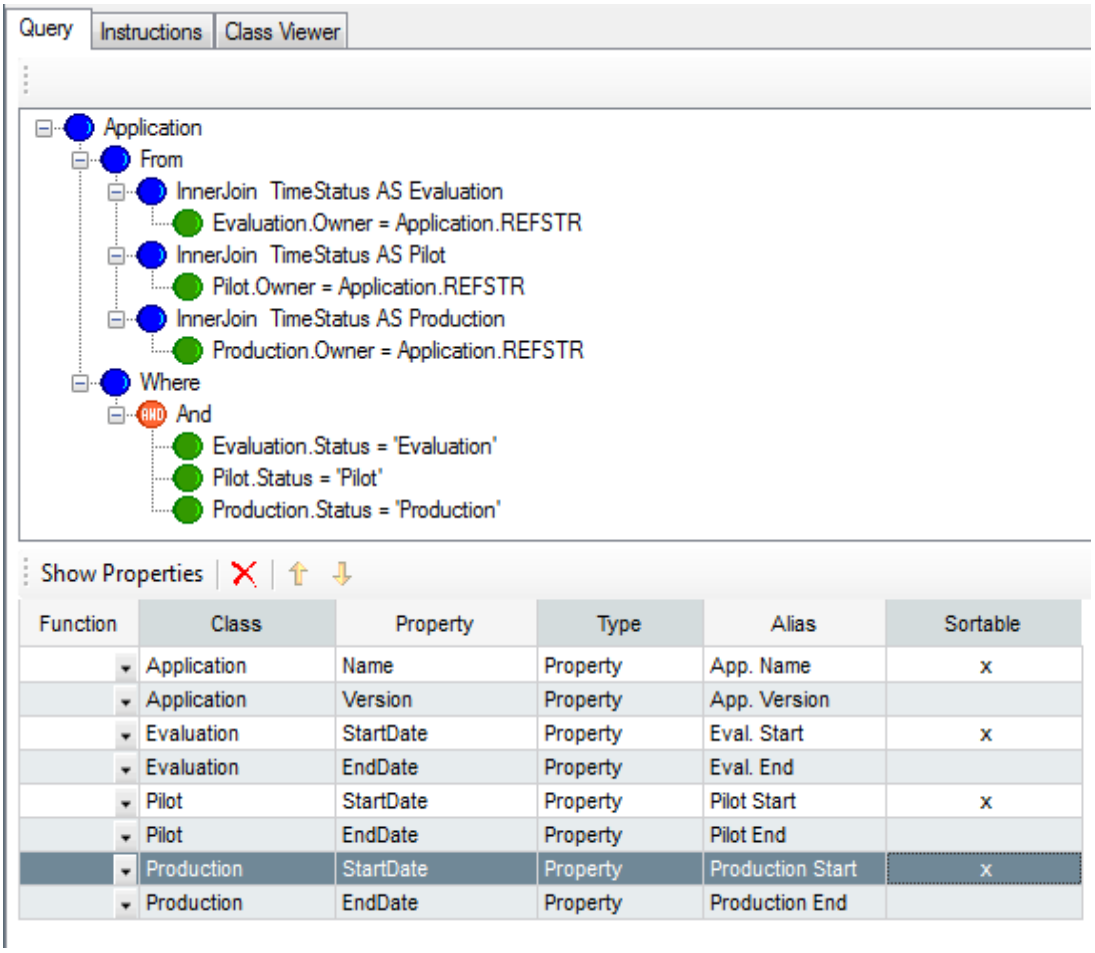
To include information about the lifecycle of an object in the Alfabet query results, you must first join the class `TimeStatus` to the Alfabet query.

For example, a report shall display the start and end dates for the `evaluation`, `pilot` and `production` lifecycle phases for applications:

App. Name	App. Version	Eval. Start	Eval. End	Pilot Start	Pilot End	Production Start	Production End
ACCOUNT	1	30/09/2002	17/06/2003	17/06/2003	20/11/2004	20/11/2004	07/08/2010
ACCOUNT	1.2	28/09/2008	20/05/2009	20/05/2009	02/09/2010	02/09/2010	23/10/2015
Administrative	1.0	29/02/2008	15/11/2008	15/11/2008	21/04/2010	21/04/2010	06/01/2016
AF Enterprise	4.0	30/03/2005	15/12/2005	15/12/2005	21/05/2007	21/05/2007	03/02/2013
AF Enterprise	3.1	15/12/2005	31/03/2006	31/03/2006	29/10/2006	29/10/2006	25/02/2009
AF Good Buy	2.0	08/06/2006	12/09/2006	12/09/2006	25/03/2007	25/03/2007	08/05/2009
AF Good Buy	3.0	31/03/2011	13/07/2011	13/07/2011	06/02/2012	06/02/2012	19/05/2014
AF HR Online	3.0	24/02/2005	14/11/2005	14/11/2005	24/04/2007	24/04/2007	27/01/2013

Each lifecycle phase (`production`, `pilot` and `evaluation`) corresponds to one object of the class `TimeStatus`. Therefore, each row displays information about three different objects of the same class. Therefore, the Alfabet query must include three `JOIN`s to the same object class, using an alias for the object classes as described in the section Defining JOINs for Indicators. `WHERE` clauses must be added to define which value of the object class property `Status` a `TimeStatus` object must have to be displayed in the respective row.

The resulting Alfabet query will appear as follows in the **Alfabet Query Builder**:



Defining JOINs to Object Class Stereotypes

The concept of object class stereotype has been implemented for many object classes. Object class stereotypes differ in name and the range of functionalities available for them. The object class stereotype definition also includes the structure of the stereotypes in the hierarchy.

For an overview of the object classes supporting object class stereotype configuration, see the section Configuring Object Class Stereotypes for Object Classes.

On the user interface, the caption of the object class stereotype is used instead of the caption of the object class. For example, the caption of the object view displays the object class stereotype caption and not the object class name. But in the Alfabet database, all objects of an object class are stored in the same database table regardless of their object class stereotype.

When a user creates an object class stereotype, the information about the configuration applied to the object is stored in the object class property `Stereotype`. The value of the property `Stereotype` is the value of the attribute ***Name*** of the corresponding stereotype configured in the relevant XML object.

To find objects of a specific object class stereotype, you must add the object class for which the object class stereotype is defined to the Alfabet query either as the base class or in a `JOIN`, and you must restrict the search results to the object class stereotype in a `WHERE` clause.

For example, to find projects of the project stereotype `StatementOfWork`:

```
FIND
Project
WHERE Project.Stereotype LIKE 'StatementOfWork'
```

Specifying Show and Sort Properties in Alfabet Queries

Show properties define the object class properties of objects found by the Alfabet query that will be displayed in the results of the Alfabet query.

Sort properties define the subset of Show properties that are relevant for alphanumerical sorting of the results in the result table during execution of the Alfabet query. The results are first sorted in ascending order based on the first Sort property defintion. All rows having an equal value are then sorted according to the second criteria, and so on. The user can change the sort order of the rows in the report by clicking any column header of the report. The report is then sorted in ascending alphanumeric order of the values in the selected column. If he/she clicks the column header of a column that is already used for sorting, the sorting order is reverted from ascending to descending alphanumeric order and vice versa.

Show and Sort properties are only required if the Alfabet query result is displayed on the interface (For example, for Alfabet queries in reports). No Show and Sort property definition is required for Alfabet queries that are used to find objects for further processing by a Alfabet functionality, (For example, Alfabet queries for wizard check entries).

Please note that Show properties cannot be defined for an Alfabet query used in a computation rule.

Information about how to define Show and Sort properties in the **Alfabet Query Builder** is explained in *Defining Show Properties* in the section *Building an Alfabet Query*.

This chapter provides information about the definition of Show properties for object class properties, indicators and roles and Sort properties for object class properties in the Alfabet query language as an XML element. The following information is available:

- Defining Show and Sort Properties in an XML Element
- Displaying Users/Organizations Defined for a Role in Alfabet Query Results
- Displaying Indicator Values in Alfabet Query Results
- Calculating Values in Alfabet Query Results

Defining Show and Sort Properties in an XML Element

The Alfabet query language allows you to specify the Show and Sort properties in an XML element ***QueryDef*** that is written in a ***QUERY_XML*** clause. For each Show property, an XML element ***ShowProperty*** is written into the ***QueryDef*** element. For each Sort property, an XML element ***SortProperty*** is written into the ***QueryDef*** XML element. This is the preferred syntax.

The ***ShowProperty*** XML element has different XML attributes for the definition of object class property values, indicators and names of the users/organizations defined for a role. This section describes the XML attributes that are required to show and sort object class properties in the report results. Show properties for indicators and roles are described in separate sections below.

The general syntax to add a Show property for object class properties to an Alfabet query is:

```
QUERY_XML
<QueryDef>
   <ShowProperty Type="Property" ClassName="ObjectClassName"
   Name="PropertyName" ShowName="ColumnHeaderString" />
</QueryDef>
```

You can define a subset of the selected Show properties as Sort properties by adding an XML element ***SortProperty*** to the ***QueryDef*** XML element.

```
QUERY_XML
<QueryDef>
   <SortProperty Type="Property" ClassName="ObjectClassName"
   Name="PropertyName" />
</QueryDef>
```

The ***SortProperty*** XML element has the same XML attributes as the ***ShowProperty*** element. The XML attributes of the ***SortProperty*** XML element must have identical values to an existing ***ShowProperty*** XML element:

```
QUERY_XML
<QueryDef>
   <ShowProperty Type="Property" ClassName="Application" Name="Name"
   />
   <SortProperty Type="Property" ClassName="Application" Name="Name"
   />
   <ShowProperty Type="Property" ClassName="Domain" Name="Name"
   ShowName="In Domain"/>
   <SortProperty Type="Property" ClassName="Domain" Name="Name"
   ShowName="In Domain"/>
</QueryDef>
```

The table below lists all available XML attributes in the ***ShowProperty*** and ***SortProperty*** XML elements of the type `Property`:

Attribute	Allowed Values	Default	Description
`Type`	`Property` `Role`	**MANDATORY**	Defines the type of Show property. The XML attributes described in this table are for the type `Property` only.

Attribute	Allowed Values	Default	Description
	Indicator		Show properties of the type Property define the display of object class properties in the Alfabet query results. See Displaying Users/Organizations Defined for a Role in Alfabet Query Results and Displaying Indicator Values in Alfabet Query Results for a description of the XML attributes for roles (Type="Role") and indicators (Type="Indicator").
ClassName	Name of object class	**MANDATORY**	The name of the object class that the object class property specified with the attribute Name belongs to.
ClassAlias	Alias name of object class		When multiple JOINs to the same object class are defined in an Alfabet query, the object class is assigned a different alias name in each JOIN in order to distinguish between the objects of that class found by one or the other JOIN. If the role is specified for an object class that has an alias name in the Alfabet query, the alias name must be specified with this XML attribute.
Name	Name of property	**MANDATORY**	The name of the object class property that is displayed in the table results.
ShowName	String	ObjectClassName PropertyName	Optionally the name of the column header can be defined with the attribute ShowName.
Function	AVG SUM MIN MAX COUNT		If set, the specified function will be assigned to the Show property. For information about assigning functions to Show properties, see Calculating Values in Alfabet Query Results

Displaying Users/Organizations Defined for a Role in Alfabet Query Results

An object of the object class Role is created each time a responsibility is defined for an object based on a preconfigured role type.

Information about which user/organization is responsible for a specific role for an object can be added to an Alfabet query directly in the XML definition of Show properties without using JOINs to the involved object classes.

The XML definition of the Show property for a role generates one column in the query output for each role type. The search returns the name and first name of the responsible user or the name of the responsible organization for the role.

The XML definition of Show properties for roles is not always applicable:

- If you specify an Alfabet query in an XML attribute of an XML element, you cannot use the XML definition of Show properties. This is required, For example, for the definition of treemap and layered diagram reports, wizards, workflows and computation rules.
- If you want the role responsibilities in the Alfabet query results to be displayed in another form than the one predefined for XML definitions of Show properties of the type `Role`. For example, if the users/organizations defined for all roles shall be listed in one column and the role type of the role shall be listed in another column.
- If you want to sort the results by responsible. The definition of Sort properties is not possible for XML definitions of Show properties for roles.
- If you want to define an Alfabet query instruction that colors the cell dependent on the availability of a role on the object. For more information about the definition of cell coloring depending on search results, see Defining Coloring for Cells or Rows Containing Specific Data.

If you want to add information about a role to the Alfabet query results without using the XML definition, you must join the relevant object classes and define their object class properties via Show properties. Detailed information about joining roles is provided in the section Defining JOINs for Roles.

The following information is available:

- Defining Roles as Show Properties in the Alfabet Query Builder
- Defining Roles as Show Properties in the XML Element

Defining Roles as Show Properties in the Alfabet Query Builder

To add a role to the Show properties in the **Alfabet Query Builder**:

1) Click a role in the object class property selector and drag it to the **Show Properties** section of the **Alfabet Query Builder**. The columns **Class**, **Property** and **Type** are automatically filled.

   If the object class supports object class stereotype definition, the object class property selector displays all role types assigned to the object class and to any of the defined object class stereotypes. If you are defining a query that finds objects of a given object class stereotype, you may see role types in the list that are not applicable for the stereotype that you are specifying in the Alfabet query.

2) If the role is not defined for the base class of the Alfabet query, but for an object class added with a `JOIN` to the Alfabet query:
   - Drag the `REFSTR` property of the selected object class from the object class property selector to the **Show Properties** section.
   - Select the **Instructions** tab of the **Alfabet Query Builder** and enter:

     ```
     REMOVECOLUMNS("ObjectClassName.REFSTR")
     ```

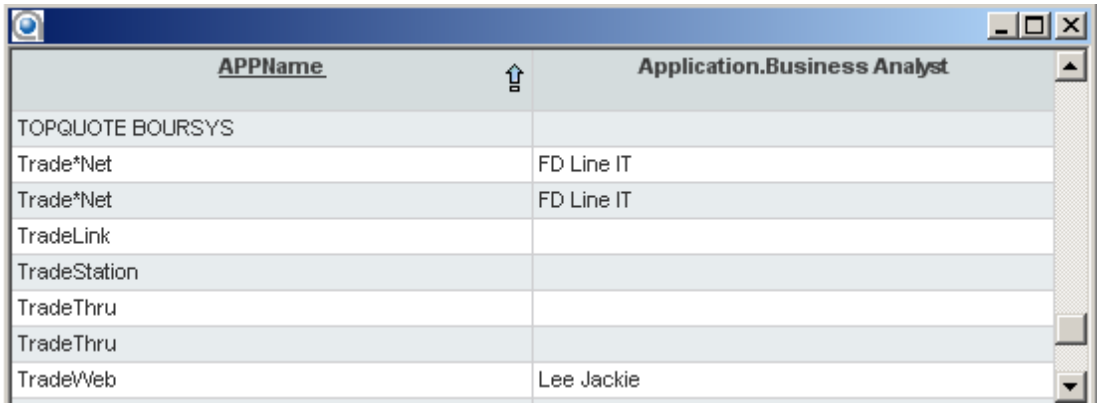

The REFSTR property defined in the Show properties is not displayed in the results table.

If you want to hide REFSTR columns for multiple object classes, the columns can be added to the command in comma-separated format:

```
REMOVECOLUMNS("ObjectClassName.REFSTR","ObjectClassName.REFSTR")
```

Whitespaces are not allowed in the command.

Dragging a role property from the property selector to the **Show Properties** section of the **Alfabet Query Builder** results in the following output:



APPName	Application.Business Analyst
TOPQUOTE BOURSYS	
Trade*Net	FD Line IT
Trade*Net	FD Line IT
TradeLink	
TradeStation	
TradeThru	
TradeThru	
TradeWeb	Lee Jackie

- One column is added to the report showing the responsible users/organizations for the role per object.
- If multiple users/organizations defined for a role have the same role for an object, one row is added to the report for each responsible user/organization.
- Base class objects with no role defined are displayed in the report and the report column is left empty. This corresponds to a left JOIN of the object class Role to the base class.
- Responsible organizations and responsible users are displayed in the same column.
- For responsible organizations, the name of the organization is displayed. For responsible users, the name and first name of the user is displayed.

If you want to change the information that is displayed about the responsible users/organizations in the rows of the report, you can edit the XML show property definition in a text editor. All other changes require the definition of JOINs.

Defining Roles as Show Properties in the XML Element

In the XML definition of Show properties described in the section Defining Show and Sort Properties in an XML Element above, roles are defined in an XML element ***ShowProperty*** of the type Role with the following XML attributes:

```
<ShowProperty Type="Role" ClassName="ObjectClassName" Name="RoleTypeName"
ShowName="ColumnCaption" ImageProps1="PropertiesDisplayedForUser"
ImageProps2="PropertiesDisplayedForOrganization" />
```


Attribute	Allowed Values	Default	Description
Type	Property Role Indicator	**MANDATORY**	Defines the type of show property. The attributes described in this table are for the type Role only. Show properties of the type Role define the display of users/organizations defined for a role for a defined role type without adding JOINs to the Alfabet query.
ClassName	Name of object class	**MANDATORY**	The name of the object class for which the responsible user/organization defined for the role shall be displayed.
ClassAlias	Alias name of object class		When multiple JOINs to the same object class are required in an Alfabet query, the object class is assigned a different alias name in each JOIN to distinguish between the objects of that object class found by one or the other JOIN. If the role is specified for an object class that has an alias name in the Alfabet query, the alias name must be specified with this XML attribute.
Name	RoleTypeName	MANDATORY	The name of the role type for which the responsible users/organizations shall be displayed. This XML attribute is used to identify the role type and also defines the display of the role type in the column header by default. The column header can be overwritten with the XML attribute ShowName.
ImageProps1	PropertyName[, PropertyName]	Name, FirstName	Defines the display of responsible users in the Alfabet query output. One object class property or a comma-separated list of object class properties of the object class Person (User) can be specified. The values of the listed object class properties are displayed in the specified order and separated with whitespace in the results.
ImageProps2	PropertyName[, PropertyName]	Name	Defines the display of responsible organizations in the Alfabet query output. One object class property or a comma-separated list of object class properties of the object class OrgaUnit (Organization) can be specified. The values of the listed object class properties are displayed in the specified order and separated with whitespace in the results.
ShowName	String	ObjectClassName PropertyName	Optionally the caption of the column header can be defined with the XML attribute ShowName. If ShowName is not defined, the name of the role type as specified with the XML attribute Name is displayed in the column header.

If the object class that the role is assigned to is not the base class, the following specifications are additionally required in the Alfabet query:

- The specification of the `REFSTR` property of the class the role is assigned to as show property.
- The specification of an instruction that hides the `REFSTR` property from the report. Instructions are written into a section `Instructions` in the Alfabet query text. If you want to hide multiple columns, you can write the instructions into the same section.

Thus, the complete specification of show properties is:

```
Instructions
REMOVECOLUMNS("ObjectClassName.REFSTR")
EndOfInstructions
QUERY_XML
<QueryDef>
   <ShowProperty Type="Property" ClassName="ObjectClassName" Name="REFSTR"
   />
   <ShowProperty Type="Role" ClassName="ObjectClassName" Name="RoleTypeName"
   ShowName="ColumnCaption" Reference="ReferenceString"
   ImageProps1="PropertiesDisplayedForUser"
   ImageProps2="PropertiesDisplayedForOrganization" />
</QueryDef>
```

If you want to hide `REFSTR` columns for multiple object classes, the columns can be added to the command in comma-separated format:

```
REMOVECOLUMNS("ObjectClassName.REFSTR","ObjectClassName.REFSTR")
```

Whitespaces are not allowed in the command.

Displaying Indicator Values in Alfabet Query Results

In the **Evaluations and Portfolios** functionality in the **Configuration** module in Alfabet, you can define evaluation types that bundle a range of indicator types to evaluate objects in Alfabet. The evaluation types are assigned to object classes in the **Class Configuration** functionality.

The indicator types of the evaluation types assigned to the object class are displayed in the **Evaluations** page view of each object of the class. The user can define indicators for the object by defining the value of the indicator type in the **Evaluations** page view.

Information about the setting of indicators for objects of object class can be added to an Alfabet query directly in the XML definition of Show properties without using `JOIN`s to the involved object classes.

The XML definition generates one column in the output for each indicator type. The search returns the indicator, that means the value of the indicator type for the current object depending on the configuration of the value in the indicator type definition. If an icon gallery is assigned in the **Visualization** tab of the indicator's editor, the icons are displayed. Otherwise the display depends on the **Range** and **Hide Number** settings of the indicator:

- If a **Range** definition exists in the indicator type definition, the display depends on the setting of the property **Hide Number** of the indicator type. If **Hide Number** is activated, only the string defined after the hyphen is displayed, while the complete definition including the numeral and the hyphen is displayed if **Hide Number** is deactivated.

  The semantic value following the hyphen in the **Range** definition of an indicator type is translatable and will be displayed in the language of the cultures currently selected by the user if a translation is provided. The **Range** definitions are included into the vocabularies for interface string translation of Alfabet without the numeral part and the hyphen.

- If a **Range** definition does not exist, the value of the indicator is displayed as follows:
  - if the value contains only numerals, it will be assumed that the value is a number. The numeric value will be displayed in a form appropriate to the specified culture.
  - if the value contains no numerals or a mixture of non numerical characters and numerals, the value is displayed as a string.

The assignment of indicator types to user profiles is not taken into consideration by the XML definition of indicators in Alfabet queries.

The XML definition of Show properties for indicators is not always applicable:

- If you want the indicators in the Alfabet query results to be displayed in another form than the one predefined for XML definitions of Show properties of the type `Indicator`. For example, if the indicators of all indicator types shall be listed in one column and their corresponding indicator types shall be listed in another column.
- If you want to sort the results by indicator. The definition of Sort properties is not possible for XML definitions of Show properties for indicators.
- If you want to restrict the display of indicators to indicator types assigned to the user profile that the user is currently logged in with. The assignment of indicator types to user profiles is not taken into consideration by the XML definition of Show properties for indicators.
- If you want to define an Alfabet query instruction that colors the cell depending on the value of the indicator. For more information about the definition of cell coloring, see Defining Coloring for Cells or Rows Containing Specific Data.
- If you define a query for a configured report displaying a chart.

For this configurations you must add the indicator to the query via `JOIN`s and define the indicator value as a normal Show property.

The following information is available about the definition of Show properties for indicators:

- Defining Indicators as Show Properties in the Alfabet Query Builder
- Defining Indicators as Show properties in the XML Element ShowProperty

Defining Indicators as Show Properties in the Alfabet Query Builder

To add an indicator to the Show properties in the **Alfabet Query Builder**:

1) Click an indicator  in the object class property selector and drag it to the **Show Properties** section of the **Alfabet Query Builder**. The columns **Class**, **Property** and **Type** are automatically filled and an **X** is automatically defined in the column **Show As Icon**.

   If the object class supports stereotype definition, the object class property selector displays all indicators assigned to the object class and to any of the defined stereotypes. If you are defining a query that finds objects of a given stereotype, you may see indicators in the list that are not applicable for the stereotype for which you define the Alfabet query.

2) By default the indicator is represented by icons an icon gallery is assigned in the **Visualization** tab of the indicator's editor. If you want to display semantic values instead, click in the **Show As Icon** cell to remove the **X**.

3) If the indicator is not defined for the base class of the Alfabet query, but for an object class added with a `JOIN` to the Alfabet query:

   - Drag the `REFSTR` property of the selected object class from the object class property selector to the **Show Properties** section.
   - Select the **Instructions** tab of the **Alfabet Query Builder** and enter:

     ```
     REMOVECOLUMNS("ObjectClassName.REFSTR")
     ```

The `REFSTR` property defined in the show properties is not displayed in the results table.

If you want to hide `REFSTR` columns for multiple object classes, the columns can be added to the command in comma-separated format:

```
REMOVECOLUMNS("ObjectClassName.REFSTR","ObjectClassName.REFSTR")
```

Whitespaces are not allowed in the command.

Defining Indicators as Show properties in the XML Element ShowProperty

In the XML definition of Show properties described in the section Defining Show and Sort Properties in an XML Element above, roles are defined in a `ShowProperty` XML element of the type `Indicator` with the following XML attributes:

```
<ShowProperty Type="Indicator" ClassName="ObjectClassName"
Name="IndicatorTypeName" ShowName="ColumnCaption" />
```

Attribute	Allowed Values	Default	Description

`Type`	`Property`
	`Role`
	`Indicator`
		MANDATORY	Defines the type of Show property. The XML attributes described in this table are for the type `Indicator` only.
			Show properties of the type `Indicator` define the display of indicators without adding `JOIN`s to the Alfabet query.

Attribute	Allowed Values	Default	Description
`ClassName`	Name of object class	**MANDATORY**	The name of the object class for which the indicator shall be displayed.
`ClassAlias`	Alias name of object class		When multiple `JOIN`s to the same object class are required in an Alfabet query, the object class is assigned a different alias name in the `JOIN` to distinguish between the objects of that object class found by one or the other `JOIN`. If the indicator is specified for an object class that has an alias name in the Alfabet query, the alias name must be specified with this attribute.
`Name`	Evaluation Type/Indicator Type		The name of the evaluation type and the name of the indicator type of the indicator separated with a slash. The value of the `Name` XML attribute is used to identify the indicator and as part of the column header of the output report if no XML attribute `ShowName` is specified.
`ShowName`	String	Value of the XML attribute `Name`	Optionally the caption of the column header can be defined with the XML attribute `ShowName`. If `ShowName` is not specified, the value defined for the `Name` XML attribute is displayed in the column header.
`ShowAsIcon`	`true` `false`	`true`	Select `false` to display the semantic value of the indicator in the Alfabet query output. Select `true` to display the indicator values as an icon in the Alfabet query output. **NOTE**: The icon gallery must be assigned to the relevant indicator type to display the indicators as icons. If the icon gallery is not assigned to the indicator types (in the **Evaluations and Portfolios** functionality of Alfabet), the semantic value for the indicator is displayed even if `ShowAsIcon` is set to `True`.

If the object class that the indicator is assigned to is not the base class, the following additional specifications are required in the Alfabet query:

- The specification of the `REFSTR` property of the object class the indicator is assigned to as Show property.
- The specification of an instruction that hides the `REFSTR` property from the report. Instructions are written in the section `Instructions` in the Alfabet query text. If you want to hide multiple columns, you can write the instructions into the same section.

Thus, the complete specification of Show conditions is:

```
Instructions
REMOVECOLUMNS("ObjectClassName.REFSTR")
EndOfInstructions
```



```
QUERY_XML<QueryDef>
    <ShowProperty Type="Property" ClassName="ObjectClassName" Name="REFSTR"
    />
    <ShowProperty Type="Indicator" ClassName="ObjectClassName"
    Name="EvaluationType/IndicatorType" ShowName="ColumnCaption"
    Reference="ReferenceString" />
</QueryDef>
```

If you want to hide `REFSTR` columns for multiple object classes, the columns can be added to the command in comma-separated format:

```
REMOVECOLUMNS("ObjectClassName.REFSTR","ObjectClassName.REFSTR")
```

Whitespaces are not allowed in the command.

Calculating Values in Alfabet Query Results

You can assign aggregation functions to Show properties of the type `Property`. Aggregation functions **cannot** be assigned to Show properties of the type `Role` or `Indicator`.

You can use the following functions in the Show properties of Alfabet queries:

Function	Description
`MIN`	Return the minimum of a set of values (Only works with numeric values - `Real` or `Integer`)
`MAX`	Return the maximum of a set of values (Only works with numeric values - `Real` or `Integer`)
`SUM`	Return the sum of a set of values (Only works with numeric values - `Real` or `Integer`)
`AVG`	Return the average of a set of values (Only works with numeric values -`Real` or `Integer`)
`COUNT`	Return the number of references

The result will be grouped by all properties not aggregated.

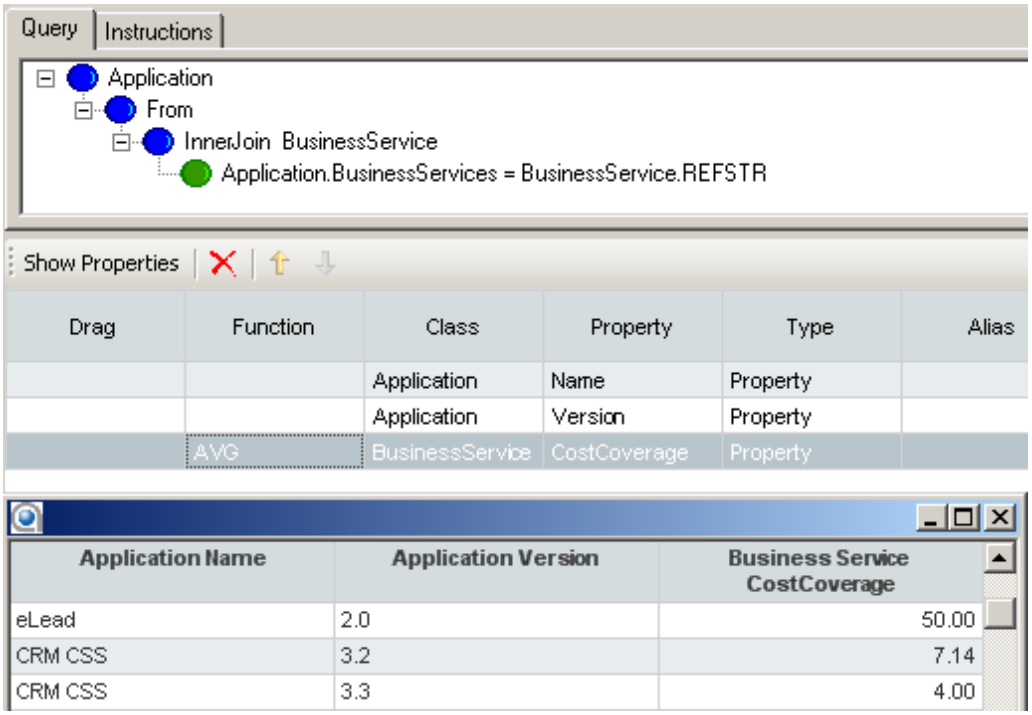
For example, an Alfabet query shall show the average of cost coverage values of all business services assigned to an application. If you define the Show properties for the Alfabet query to show the name and version of the application and the cost coverage values of the assigned business services, you will get the following report (the Alfabet query and the report are displayed in the following figures):

Show Properties

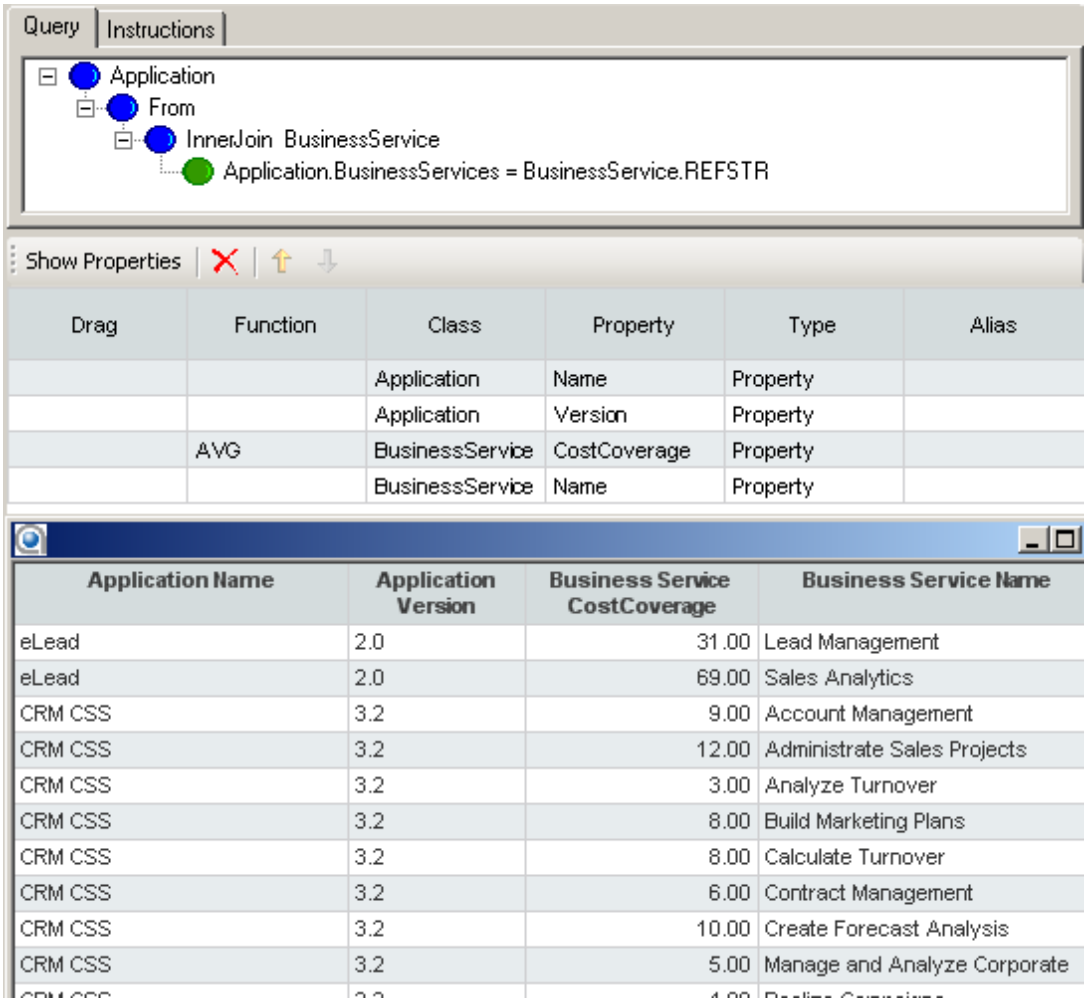
Drag	Function	Class	Property	Type
		Application	Name	Property
		Application	Version	Property
		BusinessService	CostCoverage	Property

Application Name	Application Version	Business Service CostCoverage
eLead	2.0	31.00
eLead	2.0	69.00
CRM CSS	3.2	6.00
CRM CSS	3.2	9.00
CRM CSS	3.2	12.00
CRM CSS	3.2	3.00
CRM CSS	3.2	8.00
CRM CSS	3.2	8.00
CRM CSS	3.2	10.00
CRM CSS	3.2	5.00
CRM CSS	3.2	4.00

To display the average cost coverage value per application, you must define in the Show properties that the function `AVG` is applied to the Show property `BusinessService.CostCoverage`. The report now displays one row for each application that displays the name and version of the application and the average cost coverage of the business services:



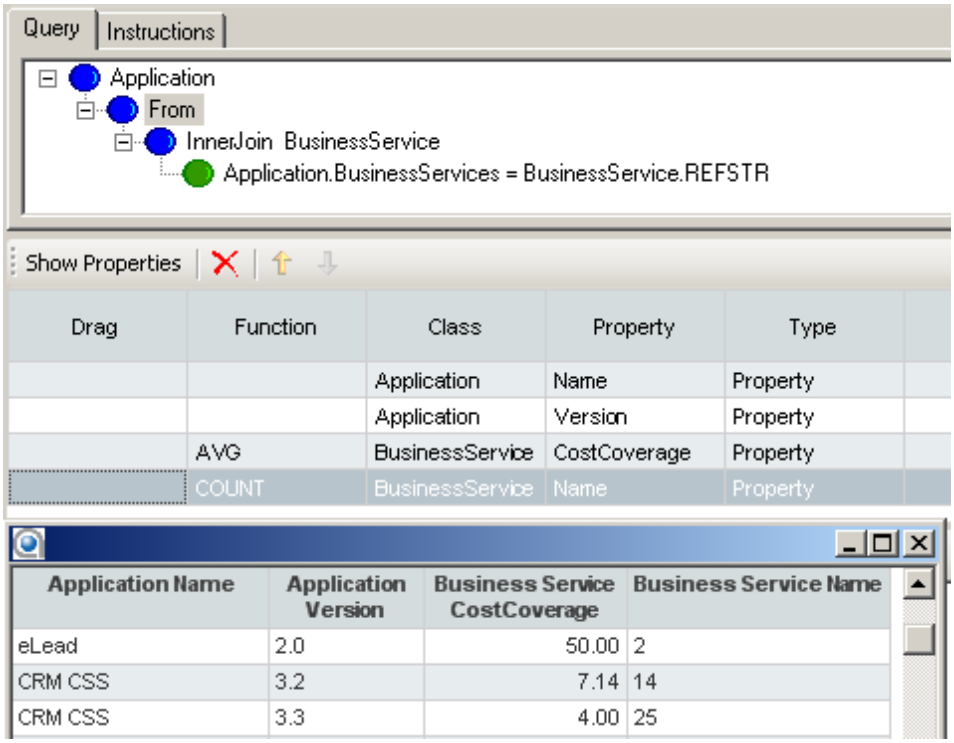
To see the average of the defined cost coverage values, you must make sure that no other properties of the business services are defined in the Show properties. If, For example, the report shows the name of the business service as well as the cost coverage values, a separate result row is displayed for each business service of the application to include the business service name in the report. Although the `AVG` function was assigned on the business service cost coverage, the function cannot be executed:



Drag	Function	Class	Property	Type	Alias
		Application	Name	Property	
		Application	Version	Property	
	AVG	BusinessService	CostCoverage	Property	
		BusinessService	Name	Property	

Application Name	Application Version	Business Service CostCoverage	Business Service Name
eLead	2.0	31.00	Lead Management
eLead	2.0	69.00	Sales Analytics
CRM CSS	3.2	9.00	Account Management
CRM CSS	3.2	12.00	Administrate Sales Projects
CRM CSS	3.2	3.00	Analyze Turnover
CRM CSS	3.2	8.00	Build Marketing Plans
CRM CSS	3.2	8.00	Calculate Turnover
CRM CSS	3.2	6.00	Contract Management
CRM CSS	3.2	10.00	Create Forecast Analysis
CRM CSS	3.2	5.00	Manage and Analyze Corporate

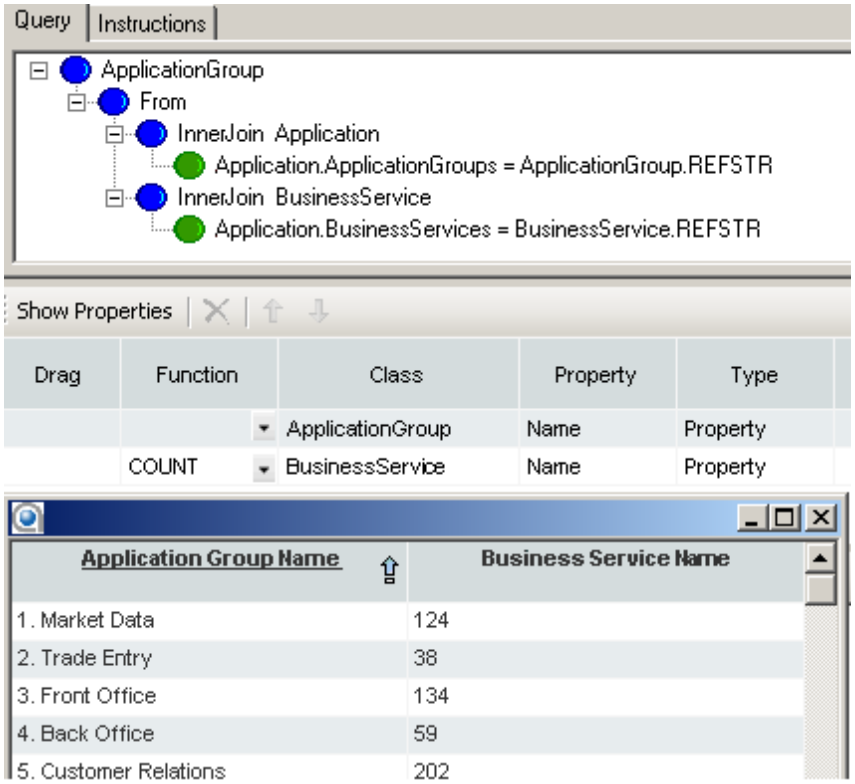
To see a report that not only shows the average cost coverage, but also the number of business services for which the average was calculated, you must include the business service name as Show property, but assign the `COUNT` function to it:



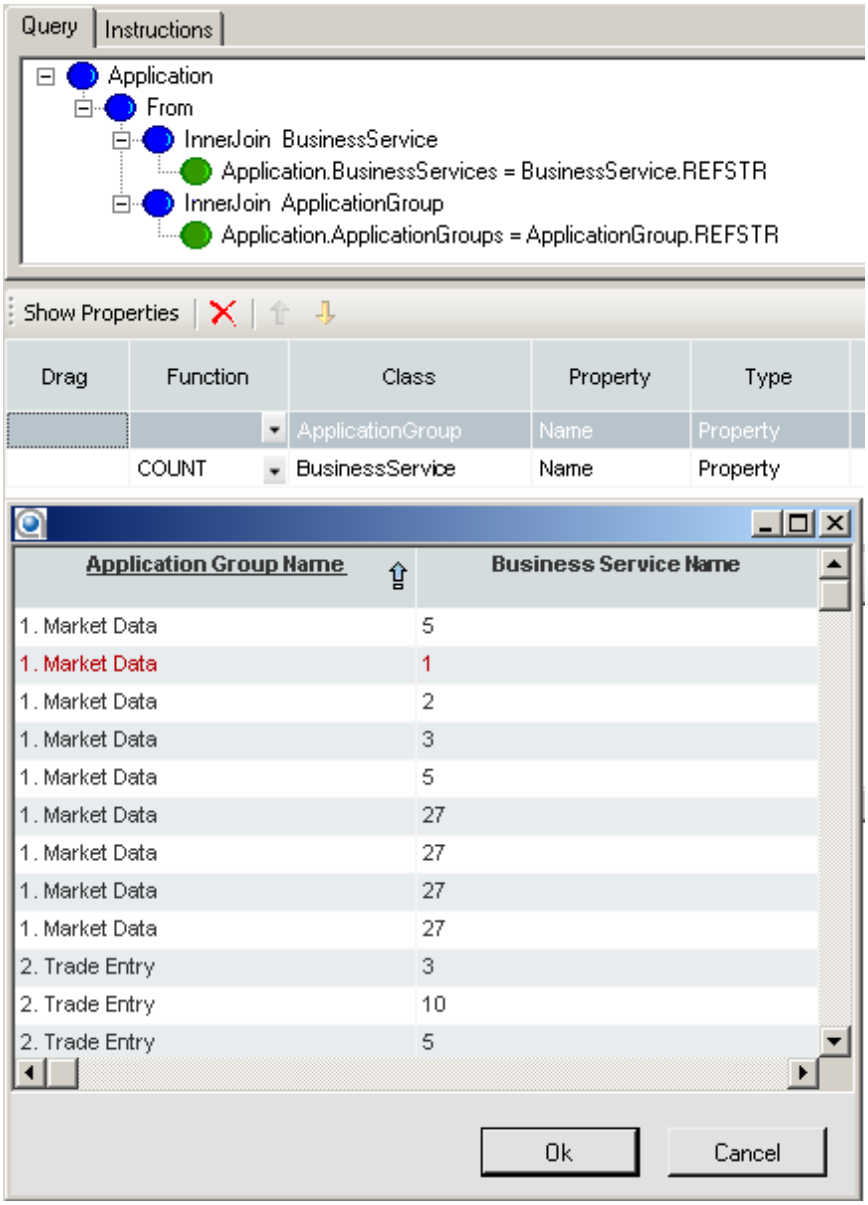
You have to take the following into account when using aggregation functions in queries with multiple `JOIN`s:

- Grouping is always performed on the `FIND` class. There is at least one row in the report for each object of the `FIND` class.
- The show properties must be limited to properties of the base class and the aggregated property.

To display the number of business services provided by all applications in an application group, the class `ApplicationGroup` must be specified as the `FIND` class of the report because values are to be aggregated per application group. One `JOIN` is defined to specify the relation between the application groups and the applications, and one `JOIN` is defined to specify the relation between the applications and the provided services. Show properties are limited to the application group name and a `SUM` over the business service name. The report now displays one row for each application group that displays the name of the application group and the number of business services provided by the applications in the application group:



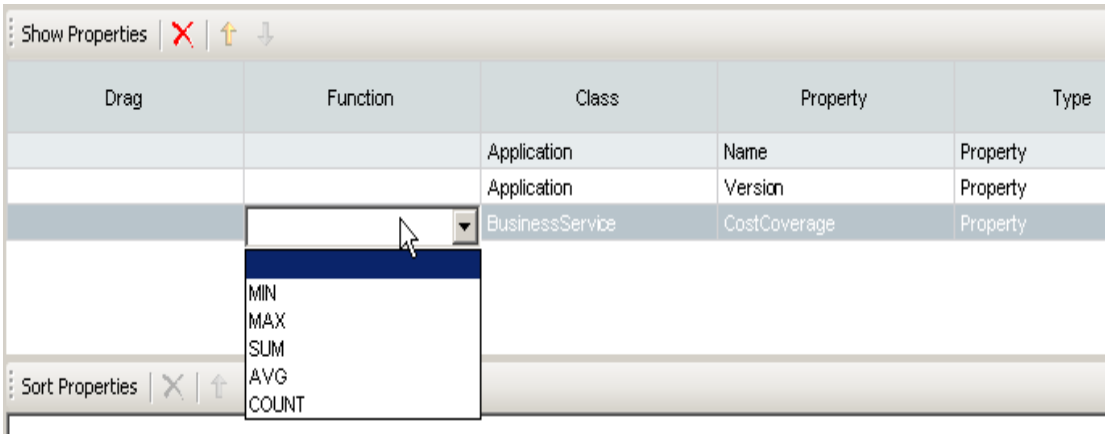
If you specify `Application` as `FIND` class and `JOIN` both application groups and business services to the `FIND` class, the results will not be as expected. Although only the name of the application group and the number of business services are displayed, the report contains one row for each application in an application group and the aggregated sum of business services in each row is the number of business services per application. In the example, the report is sorted by application group name to highlight this effect.



The same output will result from an Alfabet query for that `ApplicationGroup` is correctly defined as the `FIND` class, but object class properties of the object class `Application` are included in the Show properties. Aggregation is only performed on rows that are identical in all properties displayed in addition to the aggregated property. The values for application name and version are different for each application in an application group. An aggregation is performed on the level of applications instead of application groups.

To assign a function to a Show property in the **Alfabet Query Builder**:

1) In the **Show Properties** section, click the **Function** cell of the Show property to which you want to assign a function and select the function in the drop-down menu.



To assign a function to a Show property in the Alfabet query language, add the XML attribute `Function=" FunctionName "` to the `ShowProperty` XML element of the Show property that you want to assign the function to:

```
<ShowProperty ClassName="BusinessFunction" Name="CostCoverage"
Function="AVG"/>
```

In the alternative specification of Show properties in `SHOW` clauses, the function is specified in the `SHOW` clause with the prefix `AQL_`:

General syntax:

```
SHOW AQL_FUNCTION(ObjectClassName.PropertyName)
```

Example:

```
SHOW Application.Name Application.Version
AQL_AVG(BusinessService.CostCoverage)
```

The query syntax `AQL_<PROPERTY>` is reserved for Alfabet queries and may not be used as a name for custom properties.

Defining Native SQL Queries

Some queries defined to configure Alfabet can be defined in native SQL instead of using the Alfabet query language. For more information about the context in which native SQL can be used, see the table in the section Defining Queries.

You should have a general knowledge of native SQL. For general information about how to built a native SQL query consult the relevant publications on SQL. This section is limited to specific information that is only relevant for the use of native SQL in configurations in Alfabet:

- O/R Mapping Information Relevant for SQL-Based Access
- General Rules for the Specification of Native SQL for Alfabet
  - Allowed Query Types
  - Definition of the SELECT Clause

- Definition of Conditions
- Using Alfabet Parameters
  - Using the Alfabet Parameter CURRENT_MANDATE in Native SQL
- Using Native SQL in Combination with Alfabet Instructions
- Displaying Translated Object Data In the Current Language of the User Interface
- Defining Native SQL in XML Elements

- Special Rules for the Specification of Native SQL in the Context of Alfabet Configurations

O/R Mapping Information Relevant for SQL-Based Access

The name of the database table is specified by the **Tech Name** attribute of the object class. The names of the columns in the table are specified by the **Tech Name** attributes of the object class properties of the respective object class. For most standard Alfabet object classes and properties, the **Tech Name** is identical to the **Name** attribute of the object class or property. All **Tech Name** attributes are stored in upper case letters in Alfabet (<TECHNAME>).

If the technical name derived from the object class name conflicts with keywords reserved for the relational database management system (`RDBMS`) of the Oracle or Microsoft SQL Server, the **Tech Name** will be written with a prefix "T_" for tables of object classes and a prefix "A_" for columns representing attributes.

Reserved keywords are different for Oracle® and Microsoft® SQL® relational database management systems. Therefore, the migration of a Alfabet database between these environments may fail due to conflicts between the **Tech Name** attribute and the reserved keywords of the new relational database management system. Before the migration process is executed, the **Tech Name** specifications must be checked for keywords reserved for the new relational database management system and altered, if necessary.

For custom attributes defined by your company with the tool Alfabet Expand, the **Tech Name** attribute must be explicitly defined. Otherwise database migration procedures will not work. For more information about defining the **Tech Name** attribute, see the section Configuring Custom Properties for Protected or Public Object Classes in the reference manual *Configuring Alfabet with Alfabet Expand*.

When the **Support Data Translation** attribute is set to `True` for a language definition in Alfabet, the `Name` and `Description` properties of the objects in the database can be translated to the language culture. When a property is translatable, the translation is written to the database table column "<property tech name>_<language code>"

	REFSTR	INSTGUID	ID	NAME	NAME_1031	DES
▶	76-2518-0	117F79F6CC694...	APP-2518	Business EAI Pla...	*NULL*	*NULL*
	76-2525-0	C88B1BFE80A24...	APP-2525	Groupware Servi...	*NULL*	*NULL*
	76-2538-0	9447219E743F4...	APP-2538	Mafo-Portal	*NULL*	*NULL*

FIGURE: Section of a database table for applications with column headers specified with the TECHNAME property

The `REFSTR` property is a unique internal object ID that explicitly identifies each object in the database and is used for references between objects in the database.

The REFSTR property shall not be used in queries to identify single objects. Instead, it is recommended that you use unique key attributes such as an object's name and version. If you archive a database and restore the archive in another database, the REFSTR property of single objects may change. This is not affecting references between objects. References are automatically adapted to the new REFSTR scheme.

Data for the Alfabet property types Text (For example, object descriptions, attributes of the type StringArray) and Date are stored differently in different database servers:

Database	Data type: Text	Data type: Date
Oracle 9/10/11	NCLOB	TIMESTAMP
Microsoft SQL 2005/2008	nvarchar(max)	datetime

These data types may require special handling or possibly may not be queried from a certain reporting tool.

Object class properties of the type ReferenceArray are handled in two different ways dependent of the setting of the attribute **Reference Support** of the object class property:

- If **Reference Support** is set to True, the relations specified by the property are stored in the RELATIONS table. No database column is available for the property in the table of the object itself. The RELATIONS table specifies the relation in the following database columns:
  - FROMREF: The REFSTR value of the object that is referencing another object.
  - PROPERTY: The property of the object specified with FROMREF that defines the relation between the objects.
  - TOREF: The REFSTR value of the referenced object
- If **Reference Support** is set to False, the relations are stored directly in the database table of the object as a string. The REFSTR of all objects referenced by the property are listed whitespace separated.

General Rules for the Specification of Native SQL for Alfabet

For each configuration in Alfabet based on a query you can define a native SQL query instead of an Alfabet query in the respective text editor or input field. All queries starting with ALFABET_QUERY_500 are processed as Alfabet query, all other queries are processed as native SQL queries. There is no header definition required for native SQL.

```
SELECT app.NAME FROM APPLICATION app WHERE app.NAME LIKE 'CRM%'
```

The text editors in Alfabet Expand that allow the definition of a native SQL query have a button **Check**. You can use this button to check the correctness of Alfabet parameter definitions in the native SQL query. The **Check** functionality does not check the general syntax of the native SQL

query. The usage of Alfabet parameters in native SQL is described in the section Using Alfabet Parameters.

The following rules for the definition of native SQL are only relevant when defining the native SQL query in the context of Alfabet:

- Allowed Query Types
- Definition of the SELECT Clause
- Definition of Conditions
- Using Alfabet Parameters
  - Using the Alfabet Parameter CURRENT_MANDATE in Native SQL
- Using Native SQL in Combination with Alfabet Instructions
- Displaying Translated Object Data In the Current Language of the User Interface
- Defining Native SQL in XML Elements

Allowed Query Types

A native SQL query must follow the rules listed below when used in Alfabet configurations:

- Native SQL queries must be defined as ANSI-SQL-92 and no specific driver commands should be used.

  The embedded SQL parser is run in database-specific modes in order to take database-specific SQL syntax into account. The SQL parser will be automatically run in Oracle mode if the Alfabet database is installed on an Oracle® database server and in SQL Server mode if the Alfabet database is installed on a Microsoft® SQL Server®.

- XML extensions provided by the database server are not supported.
- Native SQL queries can be defined for either the selection of data from the Alfabet database for display on the Alfabet interface or for the selection of objects for via a valid configuration. Therefore, only SQL queries with a `SELECT` statement are allowed in the context of Alfabet configurations.

SQL statements that alter the data in the Alfabet database, For example, `DELETE`, `INSERT` or `UPDATE`, are not allowed in the context of Alfabet configurations with the exception of the SQL statements in ADIF schemes. In addition, `UPDATE` statements are allowed in workflow step actions and and wizard step actions.

- SQL queries must start with a `SELECT` or a `WITH` statement to be considered in Alfabet configurations.

  If you use `WITH` statements to built a temporary table within native SQL statements, you should make sure that the table name and all column names of the temporary table are written in lowercase letters only. Uppercase letters can lead to errors with some database server types or configurations.

It is possible to add a comment as first line of a query. All lines that are empty or that are starting with /* or with - are ignored by the check for correct definition of the native SQL query.

- Only one `SELECT` statement can be defined in a native SQL query in the context of Alfabet configurations.
- To use the function `ROW_NUMBER()` in SQL queries defined in the context of Alfabet configurations you must use `CAST` to convert the values to the type `INTEGER`.

  For example, instead of

  ```
  ROW_NUMBER() OVER (ORDER BY value DESC)
  ```

  you have to use

  ```
  CAST(ROW_NUMBER() OVER (ORDER BY value DESC) AS INTEGER)
  ```

- Native SQL queries might not be parsed successfully if a `HAVING` statement is used. If the `HAVING` statement cannot be parsed correctly, it is recommended that you use sub-queries instead of a `HAVING` statement to restrict the results.

  The following query is not correctly parsed:

  ```
  SELECT ou.REFSTR, ou.NAME, ou.STEREOTYPE,
  (SELECT '(' + CAST(COUNT(*) AS VARCHAR(5)) + ')' FROM
  ICTOBJECT icto WHERE icto.OWNER = ou.REFSTR HAVING COUNT(*) >
  0)
  FROM ORGAUNIT ou
  ORDER BY ou.NAME
  ```

  The following query is correctly parsed. It returns the same dataset without using the `HAVING` statement in the query:

  ```
  SELECT ou.REFSTR, ou.NAME, ou.STEREOTYPE,
  (SELECT '(' + CAST(value AS VARCHAR(5)) + ')'
  FROM (SELECT COUNT(*) AS value FROM ICTOBJECT icto WHERE
  icto.OWNER = ou.REFSTR) x
  WHERE x.value > 0)
  FROM ORGAUNIT ou
  ORDER BY ou.NAME
  ```

- `FOR XML PATH` cannot be used in SQL queries in the context of Alfabet configurations.
- Values for object class properties of the data type `REAL` are stored as database data type `FLOAT` in the Alfabet database. Independent from the database storage type, real values can be added for display in configured reports based on native SQL queries using the data types `real`, `float(53)` and `float(24)`.

Definition of the SELECT Clause

The SELECT clauses are used to build a result table that is then used as a basis for the following technical processes. Each property defined in the SELECT clause is one column in the resulting report table. The Alfabet software reads the result table to evaluate the data required to execute the process for which the query was defined. Therefore, the caption and order of the columns in the report is crucial for the execution of the functionality in which the query is involved. The required order and caption of report columns is described in the context of the documentation of the relevant functionality.

In Alfabet, the evaluation of SELECT clauses differs from the interpretation by SQL parsers. When defining the native SQL query, you must take into account that your result table is subject to the following changes:

- The first column of the result set is not displayed in the query results. It must specify the REFSTR of the object class selected for further processing by Alfabet components.

  If you configure a customer configured report of the type NativeSql and you do not want users to navigate to objects from a report, you can specify NULL as first column instead of the REFSTR of an object class. The first column is then ignored and navigation to objects from the report is disabled.

  NOTE: All but the basic functionality to display the query results in a table is disabled when the REFSTR is not specified in the first column of the SELECT clause. In other words, features like expandable report tables and the display of results in standard Alfabet report formats cannot be used.

- Names of database columns and database tables in the Alfabet database must be written in upper case letters in native SQL queries while names of temporary tables resulting from WITH statements must be written in lower case letters. The database table and column names are defined by the attribute **Tech Name** of the object class or object class property. The value of attribute is not necessarily identical with the value of the **Name** attribute of the object class or object class property.

- When no alias is specified for a column header in the SELECT clause, the property name is used as a column header. If two properties have the same name, the column headers in native SQL have the same name too. For example, if you display the NAME of applications and the NAME of local components, both column headers are named "NAME". When specifying native SQL in the context of Alfabet, two column headers with the same name are not allowed. It is recommended that you specify an alias name for the columns. If you do not specify an alias, the first column is given the name of the property and the following column headers are named <property name><number>, which will result in NAME1 for the second column header in the example.

- If the Alfabet database is located on an Oracle database server, the following Oracle specific requirements apply:

  - Numeric values are treated as integers regardless of the data type of the data in the database if no precision (the total number of digits both left and right of the decimal point) and scale (the number of digits on the right of the decimal point) are defined via a CAST statement. For example, to return a decimal number with a precision of 10 and a scale of two, the following CAST statement is required:

    ```
    CAST(ClassTechName.PropertyTechNameAS DECIMAL(10,2)) AS A_VALUE
    ```

- Column headers of the query result set are all converted to all upper-case letters if not written into inverted commas. In Alfabet, column headers of the query result set are often used for further processing and must meet specific output formats. Although in most cases the Alfabet components

adapt to this database condition, it is recommended that you either write alias names in inverted commas or only use upper-case letters in your alias specifications.

Definition of Conditions

Please note the following about the definition of `WHERE` conditions in native SQL in the context of Alfabet configurations:

- It is recommended to use the prefix `N` in `WHERE` conditions performing a string comparison. The `N` prefix ensures that the string is not converted into the character set as specified through the collation of the database. This is of special relevance for databases with a collation defining a character set not corresponding to Latin-1.

  For example,

  ```
  WHERE APPLICATION.STEREOTYPE=N'Technical Application'
  ```

- Please note that properties of the data type boolean can have three values:
  - `0` = the value is set to false
  - `1` = the value is set to true
  - `Null` = the value is not set

For boolean properties that are not configured to be mandatory, `WHERE` conditions must be formulated to take all three settings into account.

- Alfabet parameters can be used in `WHERE` conditions to base a comparison on a return value from a filter field or from the current environment, For example, the object the user is currently working with. For more information see Using Alfabet Parameters.

Using Alfabet Parameters

The Alfabet query language allows the specification of parameters for the following use cases:

- A set of specified parameters is available that refer to the current conditions in the Alfabet interface (For example, the object the user is currently working with or the user executing the configured report). Specified environment parameters can be used in `JOIN`s and `WHERE` clauses and `SELECT` statements for building temporary tables. For information about Alfabet parameters that refer to the current working environment, see Referring to the Current Alfabet Context in a WHERE Condition.

  The parameter `@BASE` corresponds to the `REFSTR` property of the current object and the parameter `@CURRENT_USER` refers to the `REFSTR` property of the current user. Therefore, parameters cannot be used to find single properties of the current object or user. Constructs like `@BASE.NAME`, to get the name of the current object, are not valid. On the other hand, the parameter can be used as a target for a reference without specifying the object class of the current object in the Alfabet query. For example, when a query shall display all applications in the domain the user is currently working with, the native SQL query is defined without adding the class `Domain` as:

  ```
  SELECT app.REFSTR, app.NAME
  ```



```
FROM APPLICATION app
WHERE app.DOMAIN = @BASE
```

The parameter is substituted at runtime with the `REFSTR` of the current application group. For example,:

```
SELECT app.REFSTR, app.NAME
FROM APPLICATION app
WHERE app.DOMAIN = '95-07-0'
```

- If the query output is a configured report, you can allow the user to specify the conditions for the report output by defining `WHERE` conditions with Alfabet parameters. If the condition value of the `WHERE` condition is an Alfabet parameter, that is any string starting with a @, a filter is created for the report that allows the user to define the value for the `WHERE` condition. The technical name of the filter is the string defined as Alfabet parameter. The kind of filters resulting from combinations of data types and operators differs between native SQL and the Alfabet query language. An specific overview of how to generate filter fields with native SQL is given in the section Defining Filters for Configured Reports and Selectors.

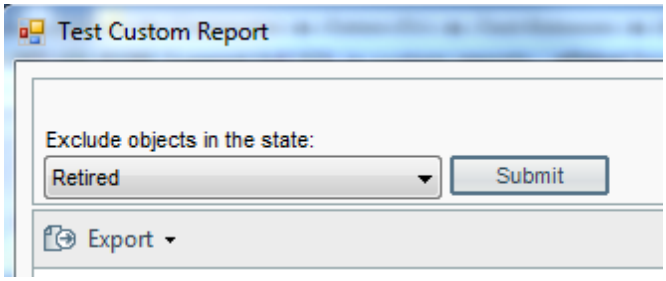
  After filter fields have been created, you should check that the automatically evaluated **Value Type** setting derived from the query data is matching the required setting of the filter field specified in the section Defining Filters for Configured Reports and Selectors. If the automatic evaluation failed, the filter field will not work.

- If the query output is a configured report with different layers referencing each other, the Alfabet parameter `@BASE` is used to refer to the object specified by the query in the parent layer. This is the case for most graphic representations in configured reports of the type `Custom`. For example, a layered diagram may show domains in one layer and applications assigned to the domains in the second layer. The parameter `@BASE` corresponds to the `REFSTR` of an object. Therefore, the native SQL query defining the parent layer must be defined with the `REFSTR` property of the object as the first column. As the first column is not displayed in the results, the definition of the `REFSTR` as first statement in the `SELECT` clause has no effect on the graphic output.

- For configured reports based on native SQL queries, the setting for **Show Retired Objects** in the users personal settings is ignored. This setting allows the user to decide whether objects that are in the object state `Retired` are displayed in report results. You can allow the user to decide about the display of retired objects per native SQL based report by creating a filter that allows to display all objects that are not in a defined object state. For example,:

  ```
  SELECT app.REFSTR, app.NAME, app.VERSION, app.STARTDATE
  FROM APPLICATION app
  WHERE app.OBJECTSTATE != @ExcludeState
  ```

This query generates a combo-box that allow the user to select an object state. Selection of an object state limits the display of objects in the report to objects that are not in the selected state. The caption of the filter field be changed to reflect that fact:



You can use the Alfabet parameters in the definition of `JOIN`s and `WHERE` clauses and `SELECT` statements for building temporary tables when defining a native SQL query for Alfabet. On execution of the native SQL query, an internal SQL parser will resolve the Alfabet parameter and generate a pure native SQL query resulting in the desired output.

Alfabet parameters are written as follows:

```
@ParameterName
```

This differs from the syntax in Alfabet query language, where parameters can also be written with a colon followed by the Alfabet parameter name.

For example,:

```
SELECT ag.REFSTR, ag.NAME
FROM APPLICATIONGROUP ag
WHERE ag.BELONGSTO = @BASE
```

finds all application groups subordinate to the application group the user is currently working with.

```
SELECT app.REFSTR, app.NAME, app.VERSION, app.STARTDATE
FROM APPLICATION app
WHERE app.NAME = @Name
```

The query creates a report table with a filter field that has the technical name **Name** and allows the user to specify search conditions for the name of the applications that shall be displayed in the report.

When resolving the Alfabet parameter on execution of the query, the SQL parser checks whether the value type of the return value is correct. This check restricts the definition of the native SQL query. For example, you may not use any function that leads to a change of the Alfabet parameter value type or want to compare the Alfabet parameter return value with a value from a temporary table in your native SQL query. You can disable the check by starting the parameter name with `NSQL_`:

```
@NSQL_ParameterName
```

For example,:

```
SELECT app.REFSTR, app.NAME
FROM APPLICATION app
WHERE EXTRACT(YEAR FROM STARTDATE) = @NSQL_year
```


finds all applications with a start date laying within the year that is defined in a filter field. Without the NSQL prefix, the SQL parser would throw an error because a data value type DateTime is expected as a compare value for STARTDATE.

Please note the following:

- When a filter field is generated based on an Alfabet parameter setting in a query of a configured report or a custom selector, the value type check is required to generate the correct type of filter field. If the NSQL_ prefix is used, no check of value type is performed which may result in a wrong type of filter field to be generated. The filter field must then be edited manually to correct the type.
- If Alfabet query language parameters are used in the SELECT statements for building temporary tables, the data type cannot be checked. A CAST statement is required at least for parameters of type DateTime to define the data type.

If no value is returned for a parameter, the following applies:

- In WHERE clauses, the WHERE condition containing the parameter is removed from the query.
- In EXIST statements and IsNULL or IN conditions, the removal of the part containing the parameter can lead to unexpected results. It is Therefore, recommended to add an alternative condition for each WHERE statement containing a parameter that will provide the correct result if the parameter has been removed.

For example, the following query returns true in case a 1=1 condition is added or false if a 1=2 condition is added:

```
SELECT dem.REFSTR, dem.ID, dem.NAME, dem.STATUS,
dem.CLASSIFICATION

FROM DEMAND dem

WHERE EXISTS(SELECT *

    FROM SCF_SplitString(dem.CLASSIFICATION,
    CHAR(13)+CHAR(10)) part

    WHERE part.value IN (@NSQL_ClassFilterOR) AND 1=1)

OR (SELECT COUNT(*)


       FROM SCF_Enum('DemandClassification') part

       WHERE part.VALUE IN (@NSQL_ClassFilterAND) OR 1=2)

    =

    (SELECT COUNT(DISTINCT value)

       FROM SCF_SplitString(dem.CLASSIFICATION,
       CHAR(13)+CHAR(10)) part

       WHERE part.value IN (@NSQL_ClassFilterAND) OR 1=2)
```

The text editors in Alfabet Expand that allow native SQL queries to be defined have a button **Check**. You can use this button to check the correctness of Alfabet parameter definitions in the

native SQL query. The **Check** functionality does not check the general syntax of the native SQL query.

The following information is available:

- Using the Alfabet Parameter CURRENT_MANDATE in Native SQL

Using the Alfabet Parameter CURRENT_MANDATE in Native SQL

The Alfabet parameter `CURRENT_MANDATE` returns the mandate mask resulting from the current mandate settings the user is logged in with. When using the Alfabet parameter in native SQL query language, you must take into account the way mandate masks are defined and compared in Alfabet.

The assignment of objects to mandates is stored in the database column `MANDATEMASK` in the database tables for all object classes that can have mandates in the Alfabet database. The mandate mask is an integer that represents a bit string calculated based on the number and kind of mandates assigned to an object.

When a company specifies a new mandate, an object of the class `ALFA_MANDATE` is created. Each mandate has an `ID` property. The `ID` is automatically assigned to the new mandate. The `ID` is a running number starting with 1 that defines the position of the mandate in the bit string of the mandate mask.

The mandate mask consists of one bit for each mandate. When the object is assigned to a mandate, the bit on the position of the mandate in the mask is set to 1. If the object is not assigned to a mandate, the bit on the position of the mandate is set to 0. The first bit in the bit string (the position 0) is set to 1 if the object is not assigned to any mandate. For objects assigned to a mandate, the first bit in the bit string is set to 0.

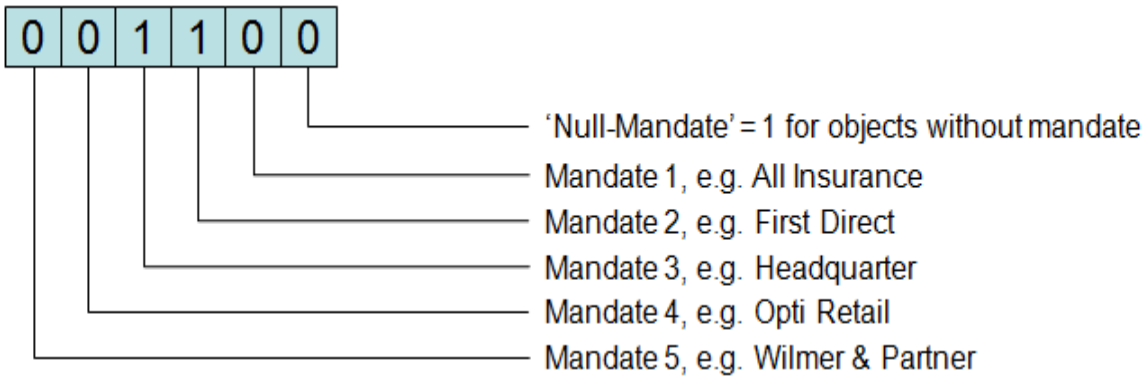


FIGURE: Example of a mandate mask in a configuration with 5 mandates. The object is assigned to mandate 2 and 3.

To evaluate whether an object is visible to a user, the mandate mask of the object is compared bit-wise with a mandate mask generated from the mandate settings of the current user.

When the mandate-related access permissions for an object are evaluated, the mandate mask of the current object is compared with the mandate mask of the current mandate that the user is logged in with. This mandate mask consists of one bit for each mandate. The bit on the position of the current mandate and the first bit in the bit string is set to 1.

The mandate mask generated to evaluate the current access permissions of the user is not identical to the mandate mask of the user stored in the property `MANDATEMASK` of the object class `Person`. The mandate mask stored with the property `MANDATEMASK` defines the access permissions for other users to the data about the user stored in the Alfabet database. The current user

mandate mask generated to evaluate access permissions is not stored in the database but generated during evaluation of access permissions as a parameter describing the current environment.

During evaluation of access permissions, the current user mandate mask and the object mandate mask are compared bitwise. If the bit on a position is 1 for both object and user, the resulting bit on the position is 1. If either the user or the object or both has a bit value of 0, the bit on the position is set to 0.

The result is a bit string representing an integer. If the bit string contains at least one bit set to 1, the resulting integer is higher than 0 and the object is visible to the user.

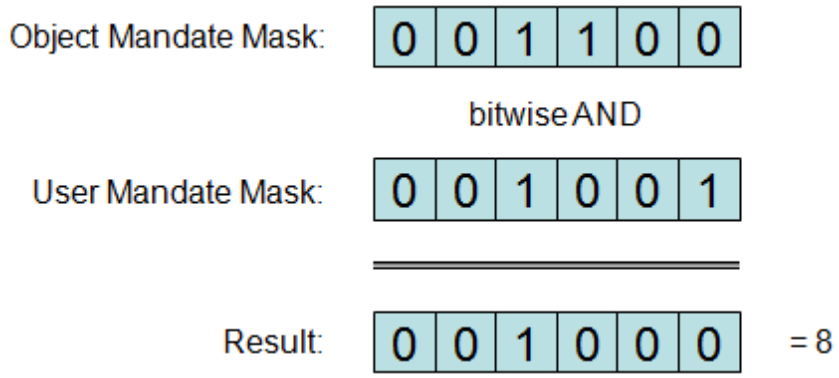


FIGURE: Calculation of the permission to an object based on the mandate masks of the current user mandate and the object

The Alfabet parameter `CURRENT_MANDATE` returns the current user mandate mask. You can use the parameter in native SQL to compare the bit strings of the current user mandate mask with the `MANDATEMASK` of the current object. For example, to show a list of all applications the current user can access with his/her current mandate settings, the query can be defined as:

```
SELECT app.REFSTR, app.NAME + ' ' + app.VERSION AS 'With your current
mandate, you have access to these application objects:'

FROM APPLICATION app, PERSON user

WHERE usr.REFSTR = @CURRENT_USER

AND (bitand(app.MANDATEMASK,@CURRENT_MANDATE)<>0)
```

To list all applications a user can access with any of the mandates assigned to him/her, the Alfabet parameter `CURRENT_MANDATE` is not suitable. Instead, the native SQL query must compare the `MANDATEMASK` property of the objects with the mandate settings of the user that are defined by a reference array with the property `MANDATES` of the object class `PERSON`. To show a list of all applications that the current user can access with any of the mandates assigned to him/her, the native SQL query can be defined as:

```
SELECT app.REFSTR, app.NAME + ' ' + app.VERSION AS 'According to all
your mandates combined, you have access to these application
objects:'

FROM APPLICATION app, PERSON user

WHERE usr.REFSTR = @CURRENT_USER

AND 0 <>app.MANDATEMASK &

    (
```



```
SELECT ISNULL(SUM(POWER(2,m.ID)),0)+1
FROM ALFA_MANDATE m
WHERE (usr.MANDATE_MASTER=1
    OR EXISTS(SELECT *
    FROM RELATIONS
    WHERE FROMREF=usr.REFSTR
    AND PROPERTY = 'MANDATES'
    AND TOREF=m.REFSTR))
)
```

Using Native SQL in Combination with Alfabet Instructions

This feature is only relevant for configured reports of the type `Custom` that are based on a template or configured reports of the type `NativeSQL`. These configured reports allow the definition of Alfabet instructions in combination with native SQL queries for the following purposes:

- To build a configured report with an expandable table. The grouping of results and the display of results as expandable table parts is based on instructions for column handling. For more information about building expandable table reports, see the section Grouping Query Results in Expandable Reports in the chapter Defining Queries.
- To specify the output format, For example, for date and time information.
- To specify the coloring of a table cell.
- To change the name or header caption of a table column.

These Alfabet instructions must be defined in Alfabet query language even if the query itself is written in native SQL. Therefore, Alfabet provides a separate tab for the definition of instructions in the text editor of the native SQL report and a separate field in the row and column definition tabs of the Alfabet report assistance of template-based matrix reports.

Alfabet instructions defined within an Alfabet query are built on basis of the Show properties of the Alfabet query. If the query is written in native SQL, the instructions refer to the definitions in the `SELECT` clause.

When defining native SQL queries in the context of the configuration of Alfabet, the first column of a result set is not displayed in the query results. (For more information, see the section Definition of the SELECT Clause). This applies to the result set after execution of the Alfabet instructions. The following order of execution of a native SQL query must be taken into account when defining the native SQL query:

- Alfabet -specific parameters in the Alfabet query are translated to corresponding native SQL code and a result set is created. (For information on parameters, see the section Using Alfabet Parameters.)
- Alfabet instructions are executed on the result set. This may influence the availability and display of columns in the result table.
- The first column in the result table after execution of instructions is not displayed.

For information about how to write instructions and the effect of instructions on the result table of configured reports, see the section Using Instructions to Format the Results of an Alfabet or Native SQL Query.

Displaying Translated Object Data In the Current Language of the User Interface

Some predefined protected properties and custom properties of the type `String` and `Text` can be translated to the secondary languages for which data translation is enabled in the cultures.

For more information about object data translation, see the section Configuring the Translation of Object Data in the chapter Localization and Multi-Language Support for the Alfabet Interface.

For configured reports of the type `Query` the translated object data is automatically displayed in the report in the language currently displayed in the user interface. For native SQL queries the change to the current language must be explicitly configured within the native SQL query. This applies to configured reports and all other configurations of the display of values on the Alfabet user interface based on native SQL, like For example, value controls in object cockpits.

The following configuration is required:

- The code `/*CULTURE_CODE*/` must be added after the object class property specification in the native SQL query.
- In `SELECT` statements, an alias must be defined for the column name of the dataset column containing the translated values.
- The object class property value is displayed automatically in the language currently selected for display of the Alfabet user interface.

If no data translation is configured for the current language for rendering the interface, the culture code setting will be ignored and the original language is used for display of results.

If the current languare for rendering the interface is configured to provide data translation, but no translation is provided, the respective cell in the result dataset is empty. To fall back to the original language if no translation is provided, the query should be defined with a condition that checks whether the translation is empty and displays the original language if applicable. For example,: `ISNULL(APPLICATION.NAME/*CULTURE_CODE*/, APPLICATION.NAME) AS NAME` as column definition in the `SELECT` statement. This is also relevant for specification of conditions refering to filter fields that may return values in either original or current language.

The following query defines a dataset that displays the name and description of applications in the current language with a fallback to the original language. A `WHERE` condition refering to a filter field is also added:

```
SELECT app.REFSTR, ISNULL(app.NAME/*CULTURE_CODE*/, app.NAME) AS
NAME, ISNULL(app.DESCRIPTION/*CULTURE_CODE*/, app.DESCRIPTION) AS
DESCRIPTION

FROM APPLICATION app

WHERE ISNULL(app.NAME/*CULTURE_CODE*/, app.NAME) LIKE @QUERY
```

Defining Native SQL in XML Elements

The configuration of some functionalities for Alfabet requires the definition of queries in XML objects. For example, this applies to the definition of treemap and layered diagram reports, the full-text search index and the rule-based access permissions.

The native SQL is then defined as value of an attribute of the relevant XML element.

```
<Query
   Class="Application"
   Query="SELECT app.NAME FROM APPLICATION app WHERE app.NAME LIKE
   'CRM%'"/>
```

You have to take the following into account to specify a native SQL query in an XML element.

- When the Alfabet query contains special characters (For example, a greater then (>) or lesser then (<) symbol) in the `WHERE` clause, you have to replace it with the HTML code for angle brackets:
  - > for >
  - < for <
  - " for "
  - [for [
  -] for]

Special Rules for the Specification of Native SQL in the Context of Alfabet Configurations

Depending on the configuration, you must take the following into account when specifying a native SQL query:

Feature	Query Defined In	Special Requirements
Configured reports of the type `NativeSQL`	text editor available via the attribute **Query as Text**	Alfabet instructions and parameters for the creation of filter fields can be used with the query. Alfabet parameters for referring to the current working environment can be used in the query. `WITH` statements are allowed to build complex queries. For information about the definition of configured reports of the type `NativeSQL`, see Creating a Tabular Configured Report of the Type NativeSQL.
Template-based configured reports of the type `Custom`	text input fields of the report assistant for query definition. Instructions must be defined in a separate text input field.	Alfabet instructions and parameters for the creation of filter fields can be used with the query. Alfabet parameters for referring to the current working environment can be used in the query.

Feature	Query Defined In	Special Requirements
		`WITH` statements may be used to build complex queries. For information about the definition of template based configured reports of the type `Custom`, see Creating a Graphic Report and Creating Configured Reports With Editing Capabilities.
Workflow configuration	text editor opening with the respective attribute **Query as Text**	A `SELECT` query with the `REFSTR` of the base object class defined as first `SELECT` argument must be defined. Alfabet parameters for referring to the current working environment can be used in the query. For information about the configuration of workflows, see Configuring Workflows.
Wizard Configuration	text editor opening with the respective attribute **Query as Text**	A `SELECT` query with the `REFSTR` of the base object class defined as first `SELECT` argument must be defined. Alfabet parameters for referring to the current working environment can be used in the query. For information about the configuration of wizards, see Configuring Wizards.
Access permission rules	XML object ***RightsManager***, attribute `Query` of the element ***RightsRule***	A `SELECT` query with the `REFSTR` of the base object class defined as first `SELECT` argument must be defined. Alfabet parameters for referring to the current working environment can be used in the query. The special requirements for Defining Native SQL in XML Elements apply. For more information about access permission rules, see Configuring Permission Rules for Access to Objects.
Computation rules	**Queries** tab of the editor **Computation Rule** via the user interface.	A `SELECT` query with one argument only that defines a property of the value type double or integer. The result of the query must correspond to a configured report with one column and one row. Native SQL queries must start with a `SELECT` statement to be executed in computation rules.

Feature	Query Defined In	Special Requirements
		Alfabet parameters for referring to the current working environment can be used in the query. For information about computation rules, see the chapter *Specifying Computation Rules for Indicator Types* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.
Color Rules	**Queries** tab of the editor **Color Rule** via the user interface. In the editor, queries must be defined within an XML object.	A `SELECT` query with the `REFSTR` of the base object class defined as first `SELECT` argument must be defined. The special requirements for Defining Native SQL in XML Elements apply. For information about color rules, see the chapter *Configuring Color Rules for Map Views and Diagram Views* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.
Consistency Monitors	**Query** tab of the editor **Consistency Monitor** via the user interface.	A `SELECT` query with the `REFSTR` of the base object class defined as first `SELECT` argument must be defined. The query must find all objects with the inconsistent attributes. The `SELECT` clause defines the object data that is displayed on the **Inconsistent Objects** page view.

Defining Filters for Configured Reports and Selectors

You can allow users to select the output of a report or search selector at runtime by means of filters that you configure. The process for the definition of filters is similar for reports and selectors. The following information focuses on the definition of filters for reports, but the same procedure can be applied to selectors. For more information about the configuration of reports, see the chapter Configuring Reports and for more information about the configuration of search selectors, see the section Configuring a Custom Selector for Search Functionalities.

The definition of filters for configured reports of the type `Query` differ from the general method described in this section and is described separately in the context of the documentation on how to create the report in the section Defining Filters for an Alfabet Query Based Configured Report.

The following steps are required to define filters for a configured report or selector:

- Defining a WHERE Clause that Causes the Generation of a Filter Field in the Report
- Defining Filter Fields in the **Custom Report View** of the configured report or the header panel of the page in the custom selector.
- Test the functionality of the filter fields in the configured report or custom selector.

The following information is available:

Defining a WHERE Clause that Causes the Generation of a Filter Field in the Report

To set a filter for a report or selector, you must configure a `WHERE` clause for the property you want the user to define in the filter field. The `WHERE` clause must be defined as follows:

- When defining the value for the condition, enter @ followed by a technical name of the filter field. The value is not used to search in the property, but rather only to assign a technical name to the filter field.

  An example `WHERE` clause for the generation of a filter field in a native SQL query:

  ```
  WHERE APPLICATION.NAME LIKE @ApplicationName
  ```

  and in Alfabet query language:

  ```
  WHERE Application.Name LIKE @ApplicationName
  ```

- Select a condition operator that allows the specification of a value. For example, `IS NULL` cannot be used for a filter because it does not require the specification of a value whereas, For example, the condition `LIKE` allow the specification of filter fields.

Depending on the data type of the property and condition that you have configured, different type of filter fields may be defined. The filter field types are described in detail in the section Defining Filter Fields.

Note the following when defining filter field name parameters in `WHERE` clauses:

- Queries with filter field name parameters can be defined for all queries in a report, including color rules and indicator rules.
- A filter field value can be used in multiple `WHERE` clauses by defining the same parameter name.
- When a filter field is not filled, the query processing mechanisms of the Alfabet components delete the condition with the filter parameter from the `WHERE` statement before executing the query. For example, the following `WHERE` statement of a native SQL query

```
WHERE APPLICATION.NAME CONTAINS @PARAM
AND APPLICATION.STATUS = 'Plan'
```

is shortened to the following statement before executing the query if the filter field with the name `@PARAM` is not filled:

```
WHERE APPLICATION.STATUS = 'Plan'
```

- In standard Alfabet views and configured reports based on Alfabet queries entries in filter fields are interpreted case insensitive. Filter values passed on to a native SQL query are interpreted case sensitive. To enhance usability it is recommended that the `WHERE` condition of the native SQL query that contains the parameter substituted by the filter field value is designed to enable a case insensitive comparison. This can be achieved by using the UPPER() or LOWER() function:

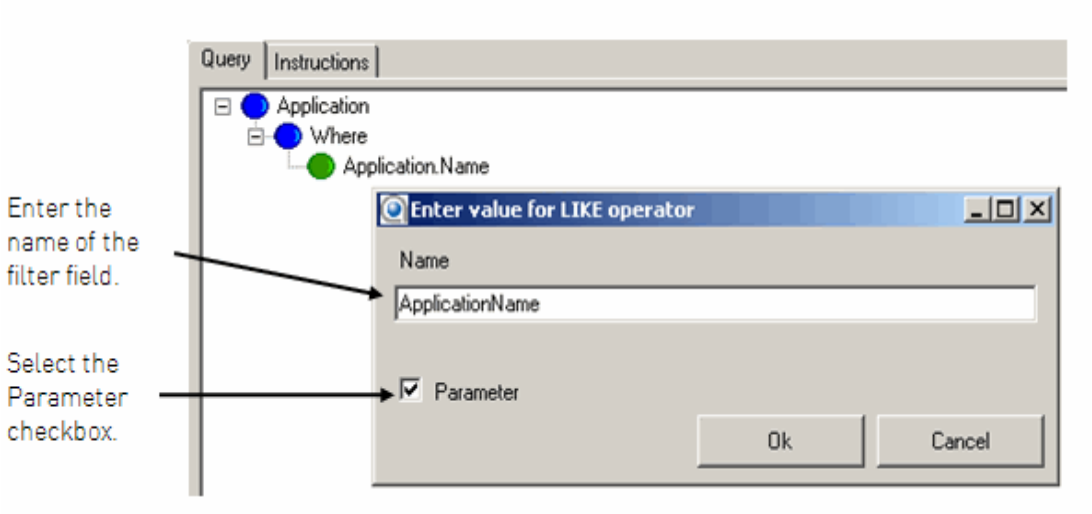
```
WHERE UPPER[APPLICATION.NAME LIKE UPPER(@ApplicationName)
```

- The search behaviour in edit fields is controlled by the setting of the ***AutoWildCard*** attribute of the XML object ***SearchManager***. When ***AutowildCard*** is set to `true`, a wildcard is added automatically to search strings entered by the user. Therefore, the = condition cannot be used for the definition of edit fields because the condition does not allow wildcards in the argument. It is also recommended to inform users with a static text in the filter section whether wildcards are automatically added to strings.

For more information about the ***AutoWildCard*** setting in the XML object ***SearchManager***, see the section Configuring the Wildcard for Standard and Custom Search Functionalities.

Note the following for the definition of the `WHERE` clause in Alfabet query language:

- When defining the value for the condition in the **Alfabet Query Builder**, select the checkbox **Parameter** in the dialog box, shown below. In the **Name** field, enter a technical name of the filter field. The value in the **Name** field is not used to search in the property, but rather only to assign a technical name to the filter field.



After clicking **OK**, you will see the `Where` clause in the **Alfabet Query Builder** with a colon written before the value.

- When defining the value for the condition in the Alfabet query syntax, you can enter a colon instead of an @ followed by the name of the resulting filter field:

```
WHERE
    Application.Name LIKE:ApplicationName
```

Defining Filter Fields

When the query that a custom selector or a configured report is based on includes one or multiple `WHERE` clauses with a parameter defining a filter name, a filter field must be added for each `WHERE` clause in the **Custom Report View** of the configured report or the header panel of the page in the custom selector.

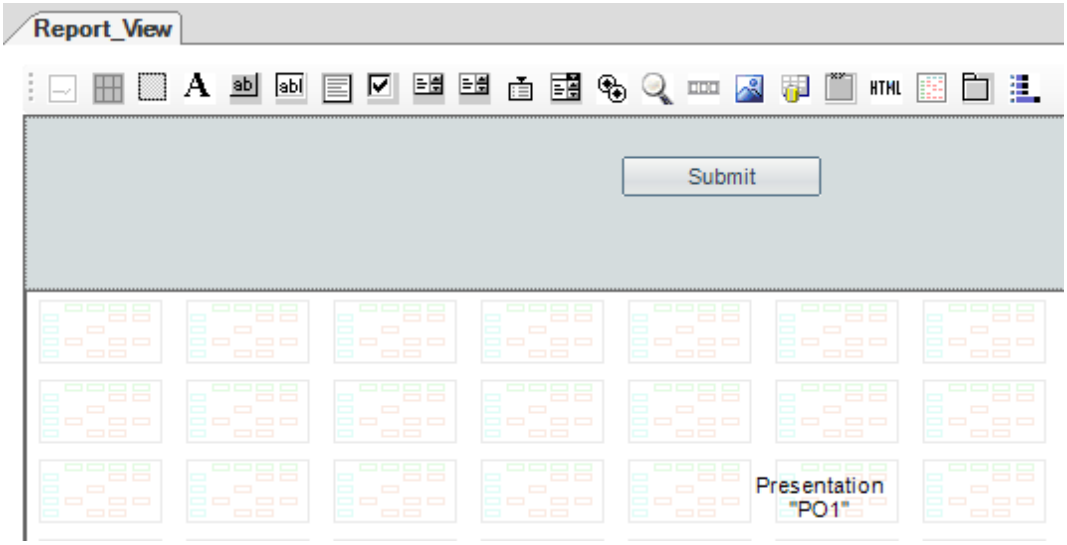
There are two mechanisms for adding filter fields:

- In general, filter fields are added manually to the filter panel. This method can be applied to both configured reports and custom selectors. The filter fields can be defined before or after writing the query for the configured report or custom selector page.
- For configured reports of the type `NativeSQL`, filter fields are generated automatically if the **Custom Report View** of the configured report is created after the query for the report has been defined. This special case is described separately in the section Configuring a Native SQL Query for a Configured Report in the chapter Configuring Reports.

Do the following to manually add a filter field to the report:

1) In the query of the configured report or custom selector, add a condition with a parameter as described in Defining a WHERE Clause that Causes the Generation of a Filter Field in the Report.

2) In the explorer, double-click the **Custom Report View** of the configured report or the presentation object of the class entry of the custom selector. The view editor opens with a toolbar for interface control creation on top. The interface controls are representing different types of filter fields. Beneath the toolbar, the view is displayed with a filter panel on top and a presentation object beneath the filter panel:



Please note that the filter panel can be moved from top to the left or right of the presentation object. the required configuration is described in the section Changing the Position of the Filter Panel.

3) In the toolbar, click the icon of the interface control type that you want to add. When you move the cursor over an icon, a tooltip gives information about the interface control type represented by the icon.

4) Drag the mouse on the filter panel in the report editor to create a text box in the location and width of your choice.

5) In the attribute window of the new filter field, change the name of the filter field to the parameter specified in the query of the report.

For the filter of the following native SQL query:

```
SELECT app.REFSTR, app.NAME, app.VERSION
FROM APPLICATION app
WHERE app.NAME LIKE @ApplicationName
```

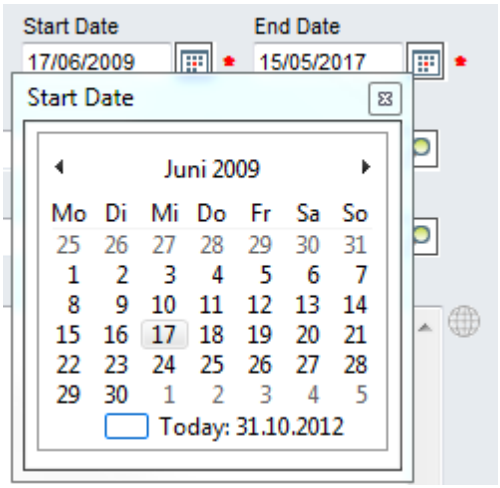
the technical name of the filter field must be

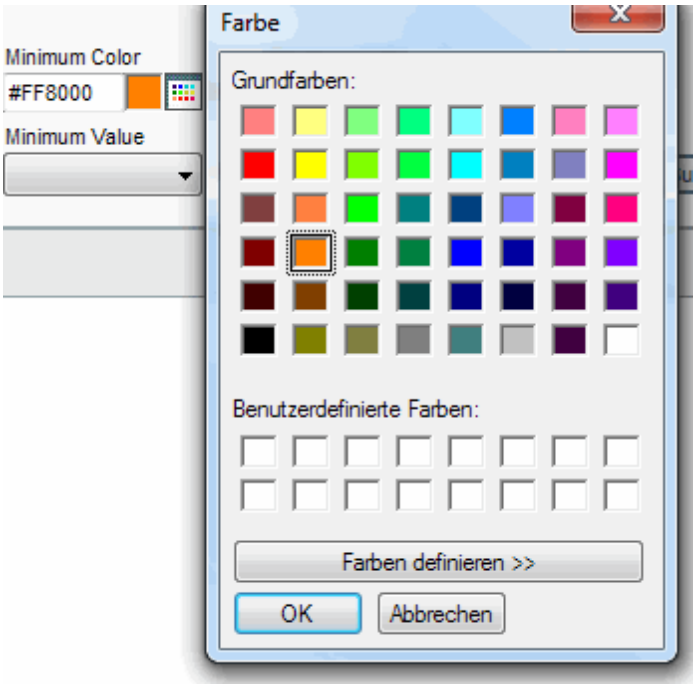
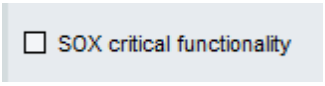
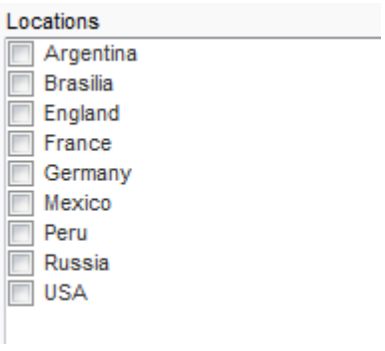
```
@ApplicationName
```

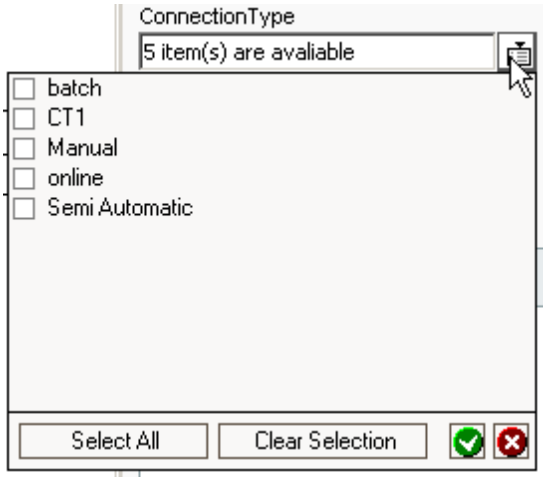
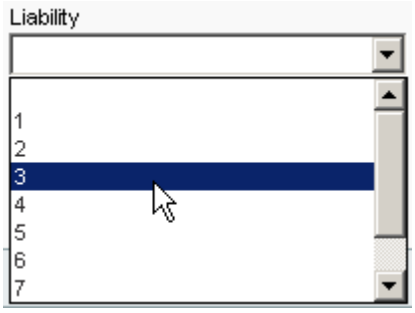
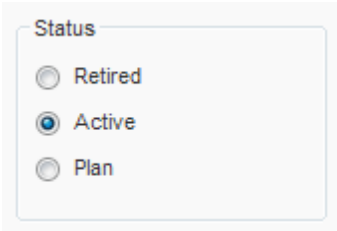
6) If applicable, specify all other attributes that are required for the type of control. The required settings for the addition of each control type are described below.

7) In the toolbar, click the **Save** button to save your changes.

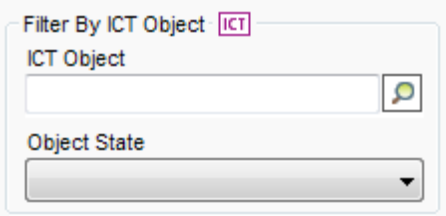
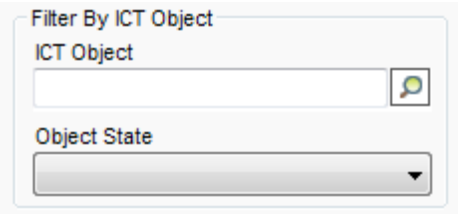
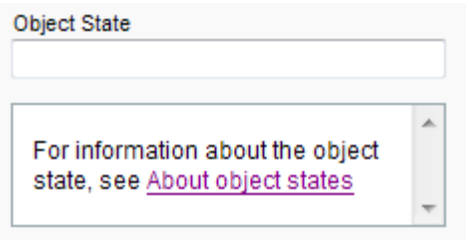
The toolbar includes all interface controls that can be used in configuration objects of Alfabet Expand. This also includes interface controls that are not suitable for use in filters. Only interface control types included in the following list can be used in filter panels:

Menu Option	Generated Element	For more information see:
Filter fields for the definition of values returned to the underlying query		
Edit	The Edit field allows to search for data matching a string entered by the user.  If an Edit field is defined for a property of the data type `Date`, it includes a date picker:  If an edit field is defined for a property of the data type `Color`, it includes a color selector:	Configuring Edit Fields Configuring Date Pickers Configuring Color Pickers

Menu Option	Generated Element	For more information see:
		
Edit Search	The Edit Search field allows to search for an object via a selector or by entering the name or part of the name of the object. 	Configuring Edit Search Fields
Check Box	Check Boxes allow to select whether a boolean value is true. 	Configuring Check Boxes
Checked List Box	Checked List Boxes allow multiple values to be selected from a set of values. 	Configuring Radio Button Groups, Combo-Boxes, Checked Combo-Boxes, List Boxes or Checked List Boxes

Menu Option	Generated Element	For more information see:
Checked Combo Box	Checked Combo Boxes allow multiple values to be selected from a set of values. 	Configuring Radio Button Groups, Combo-Boxes, Checked Combo-Boxes, List Boxes or Checked List Boxes
Combo Box	Combo Boxes allow one value to be selected from a set of values. Combo Boxes are by default not set and allow deselection of values by selection of an empty value. 	Configuring Radio Button Groups, Combo-Boxes, Checked Combo-Boxes, List Boxes or Checked List Boxes
Radio Button Group	Radio Button Groups allow one value to be selected from a set of values. In radio button groups a value must be selected and when a user first opens the report, one of the values is preselected by default. 	Configuring Radio Button Groups, Combo-Boxes, Checked Combo-Boxes, List Boxes or Checked List Boxes

Design elements to enhance usability. These controls are not related to the underlying query

Menu Option	Generated Element	For more information see:
Static Text	A static text displayed on the filter, For example, to add an explanation on the filter settings.	Adding Static Information to the Filter Panel
Button	A button triggering an action. Both filter panels for configured reports and custom selectors have a preconfigured button **Submit** to apply the filter settings to the search result. On a filter, no other buttons are allowed. The button control can be used to re-create the **Submit** button if it has been deleted by mistake.	Configuring the Submit Button
Icon	An icon from the icon gallery displayed on the filter for design reasons or as an image with underlying link to another Alfabet view. 	Adding an Icon With or Without a Link to Another Alfabet View to the Filter Panel
Group Box	A Group Box groups filter fields optically to enhance usability. 	Changing the Sequence of the Focus on Filter Fields
HTML Control	Used to add HTML text, URL, or a link that opends a document located in the **Internal Document Selector**. 	Adding Static Information to the Filter Panel

Menu Option	Generated Element	For more information see:
Slider Control	Slider Controls allow multiple numeric values to be selected independently from each other or summing up to a defined value. Each value is selected by moving a slider on a scale. 	Configuring Slider Controls

The following sections describe the basic configuration settings of the filter field controls related to the query:

- Configuring Edit Fields
- Configuring Date Pickers
- Configuring Color Pickers
- Configuring Edit Search Fields
- Configuring Check Boxes
- Configuring Radio Button Groups, Combo-Boxes, Checked Combo-Boxes, List Boxes or Checked List Boxes
  - Attribute Settings for Selection of Objects Found via a Query
  - Attribute Settings for Selection of a String Value
  - Attribute Settings for Selection of a Value Defined in a Configuration Object
- Configuring Slider Controls

Functionality available for all filter fields can additionally be configured by setting the respective attributes of the filter fields. These options are described separately in the sections:

- Configuring Filter Layout
- Defining Mandatory Filter Fields
- Excluding Filters From Storage in Bookmarks
- Automatically Adding Wildcards to Search Strings
- Configuring Cascading Filters
- Configuring Reusability of Filter Settings Across Reports

Configuring Edit Fields

Available for Data Types: String, `Text`, `Date`, `DateTime`, `Integer`, `Real`

Dependent on the operator used in the `WHERE` condition in the query, the edit field allows search with or without wild cards. For example, using the =, != or <> operators results in a text field that requires the specification of the whole property value without wild cards.

To define an edit field for a filter:

1) In the explorer, double-click the **Custom Report View** of the configured report or the presentation object of the class entry of the custom selector. The view editor opens with a toolbar for interface control creation on top.

2) In the toolbar, click the **Edit**  icon.

3) Drag the mouse on the filter panel in the report editor to create a text box in the location and width of your choice.

4) In the attribute window of the new filter field, set the following attributes:

   - **Name**: Change the name of the filter field to the parameter specified in the query of the report.
   - **Caption**: Enter a caption for the filter field. The caption is displayed on top of the field.
   - **Hint**: Optionally define a tooltip that is displayed for the filter field when the user moves the cursor over the field caption.
   - **Value Type**: Select the data type of the property that you compare with the filter value.
   - **Allow Wildcards**: The setting of wildcards in filter fields depends on the configuration of the `AutoWildCard` element in the XML object ***SearchManager***. The search function can be configured to automatically add a wildcard before and after the search string entered in an edit field and Therefore, search for any string containing the search criteria. Alternatively, the search function can be configured to use the search string as explicitly entered in the field. In other words, the user entering the string in the edit field must add a wildcard to the string in the field if he/she wants to search for a string containing but not equal to the defined search string. If wild cards are automatically set and the edit field requires that the whole string entered into the filed is equal to a property value, the attribute **Allow Wildcards** must be set to `False`. Otherwise the automatically set wild cards will be added to the string before comparing it to the property value and inhibit search for the string.
   - **Enable Look-Ahead Typing**: If the **Value Type** attribute of the filter field is set to `String`, you can set this attribute to `True` to enable an auto-fill capability for the filter field. When the user types in the filter field, the look-ahead functionality will search for a matching string that has previously been used to search in the current configured report or custom selector. A list of matching objects will be offered in a drop-down menu and the user can select the relevant option. The list of matching objects will be updated with each additional letter typed in to the search field. The search can be triggered to find only matching search strings previously entered by the current user or by all users in the user community.

     The XML attribute `SearchOnlyCurrentUserStrings` in the XML object ***Solution-Options*** should be set to "`true`" if the stored search strings of the current user shall be used for the search context or set to "`false`" if the stored search strings of all users in the user community shall be used for the search context. The default value is set to "`true`".

5) In the toolbar, click the **Save**  button to save your changes.

Configuring Date Pickers

Available for Data Types: `Date`

To define a date picker for a filter:

1) In the explorer, double-click the **Custom Report View** of the configured report or the presentation object of the class entry of the custom selector. The view editor opens with a toolbar for interface control creation on top.
2) In the toolbar, click the **Edit**  icon.
3) Drag the mouse on the filter panel in the report editor to create a text box in the location and width of your choice.
4) In the attribute window of the new filter field, set the following attributes:
   - **Name**: Change the Name of the filter field to the parameter specified in the query of the report.
   - **Caption**: Enter a caption for the filter field. The caption is displayed on top of the field.
   - **Hint**: Optionally define a tooltip that is displayed for the filter field when the user moves the cursor over the field caption.
   - **Value Type**: Select `Date`.
5) In the toolbar, click the **Save**  button to save your changes.

Date pickers can also be configured for filters that are compared to properties of the data type `DateTime` if the operator is <, >, >= or <=. The interface control must then be configured to return the data type `Date`. If you compare the date time property with a date, the date is amended with a time of 00:00:00, which means that setting a filter that defines that the value must be higher than 24/11/2010, the value 24/11/2010 15:51:98 is regarded as higher as the selected filter setting.

Configuring Color Pickers

Available for Data Types: `String` representing a color.

Color pickers returns the color code name or the hexadecimal notation for the selected color.

A color selector may For example, be used in a graphic report with a color rule specification to allow the user to select which color shall be used to display objects matching a defined condition. In that case, the filter field is not used to limit the search results to a subset of the available objects but to alter the appearance of the graphic display of the report.

To define a color picker for a filter:

1) In the explorer, double-click the **Custom Report View** of the configured report or the presentation object of the class entry of the custom selector. The view editor opens with a toolbar for interface control creation on top.
2) In the toolbar, click the **Edit**  icon.

3) Drag the mouse on the filter panel in the report editor to create a text box in the location and width of your choice.

4) In the attribute window of the new filter field, set the following attributes:

   - **Name**: Change the Name of the filter field to the parameter specified in the query of the report.
   - **Caption**: Enter a caption for the filter field. The caption is displayed on top of the field.
   - **Hint**: Optionally define a tooltip that is displayed for the filter field when the user moves the cursor over the field caption.
   - **Value Type**: Select `String`.
   - **Sub Type**: Select `Color`.

5) In the toolbar, click the **Save** button to save your changes.

Configuring Edit Search Fields

Available for Data Types: `Reference`, `ReferenceArray`

To define an edit search field for a filter:

1) In the explorer, double-click the **Custom Report View** of the configured report or the presentation object of the class entry of the custom selector. The view editor opens with a toolbar for filter field creation on top.

2) In the toolbar, click the **Edit Search** icon.

3) Drag the mouse on the filter panel in the report editor to create a text box in the location and width of your choice.

4) In the attribute window of the new filter field, set the following attributes:

   - **Name**: Change the **Name** of the filter field to the parameter specified in the query of the report.
   - **Caption**: Enter a caption for the filter field. The caption is displayed on top of the field.
   - **Hint**: Optionally define a tooltip that is displayed for the filter field when the user moves the cursor over the field caption.
   - **Value Type**: Select the data type of the property that you compare with the filter value.
   - **Parameters**: Enter

     `ClassName``|`*`PropertyName`*`|OPERATOR`

where `ClassName` and `PropertyName` define the name of the object class and object class property the filter value is compared to.

For example, to define an **Edit Search** field that allows to select a person that shall be defined as responsible user for an application, the **Parameters** attribute must be set to:

`Application|ResponsibleUser|OPERATOR`

5) In the toolbar, click the **Save** button to save your changes.

Configuring Check Boxes

Available for Data Types: `Boolean`

In the Alfabet database, Boolean values can be `True`, `False` or `NULL`, where `NULL` is the value stored for a boolean property that is not set to `True` by the user and for which no default value is defined. If you specify a check box to filter for a boolean value, selecting the checkbox will find all objects for that the property is set to `True`. Deselecting the checkbox will find all objects for which the property is set to `False`. Objects for which the property is not defined (set to `NULL`), will never be displayed in the report. To avoid such misleading search results in filters, a default value shall be defined for each Boolean property in Alfabet.

To define an edit search field for a filter:

1) In the explorer, double-click the **Custom Report View** of the configured report or the presentation object of the class entry of the custom selector. The view editor opens with a toolbar for interface control creation on top.
2) In the toolbar, click the **Check Box** icon.
3) Drag the mouse on the filter panel in the report editor to create a box in the location and width of your choice.
4) In the attribute window of the new filter field, set the following attributes:
   - **Name**: Change the **Name** of the filter field to the parameter specified in the query of the report.
   - **Caption**: Enter a caption for the filter field. The caption is displayed on the right side of the check box.
   - **Hint**: Optionally define a tooltip that is displayed for the filter field when the user moves the cursor over the field caption.
   - **Value Type**: Select `Boolean`.
5) In the toolbar, click the **Save** button to save your changes.

Configuring Radio Button Groups, Combo-Boxes, Checked Combo-Boxes, List Boxes or Checked List Boxes

Available for Data Types: `Reference`, `ReferenceArray`, `String`, `StringArray`

In Alfabet multiple different types of filter fields are available that allow to select one or multiple values from listed pre-defined options:

- In combo boxes and radio button groups, a single value can be selected from the listed values. When a user opens the report for the first time, no value is pre-selected.
- In checked list boxes and checked combo boxes, multiple values can be selected from the listed values. When a user opens the report for the first time, no values are pre-selected.

The values available for selection by the user can be defined by one of the following mechanisms:

- A query can be defined in the filter field that finds objects to be selected. The selector displays the Show properties defined in the Alfabet query or the properties defined in the `SELECT` statement of the native SQL query and returns the `REFSTR` of the object represented by each selection.
- A list of strings can be provided that are selectable by the user. The selector field returns the selected string value.
- The selector can be configured to return predefined values from an enumeration or values defined in an XML object or mandates defined for your company.

The selectable single values must not contain commas. If a property shall be selectable via a multi select filter field and the property shall contain a comma separated list of values, the property should be of the type StringArray or the commas must be replaced with other separator when defining the filter field content.

To define an edit search field for a filter:

1) In the explorer, double-click the **Custom Report View** of the configured report or the presentation object of the class entry of the custom selector. The view editor opens with a toolbar for interface control creation on top.
2) In the toolbar, click the icon of the box that you want to define:
   -  for a radio button group
   -  for a checked list box
   -  for a combo box
   -  for a checked combo box
3) Drag the mouse on the filter panel in the report editor to create a box in the location and width of your choice.
4) In the attribute window of the new filter field, set the following attributes:
   - **Name**: Change the **Name** of the filter field to the parameter specified in the query of the report.
   - **Caption**: Enter a caption for the filter field. The caption is displayed on the right side of the check box.
   - **Hint**: Optionally define a tooltip that is displayed for the filter field when the user moves the cursor over the field caption.
5) In the attribute window, define the content of the combo box. The content depends on the type of data that shall be found in the selection:
   - To find the target of a property of the type `Reference` or `ReferenceArray`, a query can be defined in the filter field that finds objects to be selected. The selector displays the Show

properties defined in the Alfabet query or the properties defined in the `SELECT` statement of the native SQL query and returns the `REFSTR` of the object represented by each selection. For more information, see Attribute Settings for Selection of Objects Found via a Query.

- A list of strings can be provided that are selectable by the user. The selector field returns the selected string value. For more information, see Attribute Settings for Selection of a String Value.
- The selector can be configured to return predefined values from an enumeration or values defined in an XML object or mandates defined for your company. This configuration applies when the selection shall return one of the following:
  - The object's object state.
  - The object's release status.
  - The mandate assignment of the object.
  - A lifecycle status of the object.
  - The object's stereotype.
  - A value selectable from an enumeration.

For more information see Attribute Settings for Selection of a Value Defined in a Configuration Object.

6) In the toolbar, click the **Save** button to save your changes.

Attribute Settings for Selection of Objects Found via a Query

The following settings are required in the attribute section of the filter field to define the content on basis of a query:

- **Value Type**: For a combo box or radio button group, select `Reference`. For a checked combo box, select `ReferenceArray`.
- **Sub Type**: Select `SqlEnum`
- **Range**: Define an Alfabet query or a native SQL query to specify the content of the combo-box.

  The query should not return more than twenty values to ensure usability of the filter field.

For example, a filter shall allow to limit display of information flows to information flows with a defined connection type. Connection types for information flows are user defined objects in the Alfabet database that the information flow links to via the property Connection Type that is of the data type `Reference`.

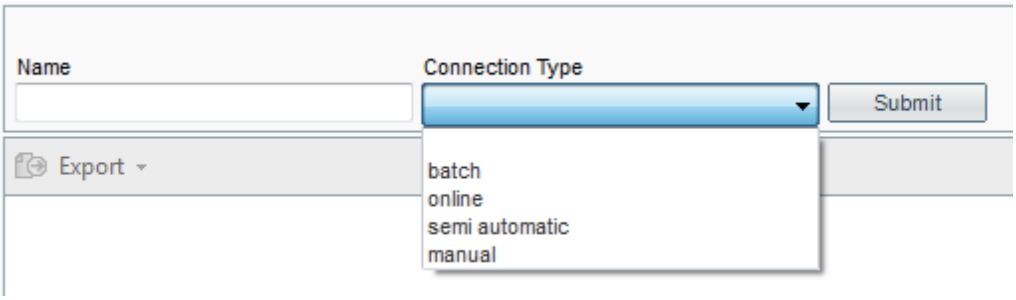
In the first configuration example, the filter shall include a combo box that allows one connection type to be selected.

To display the available connection types in the combo box, you must specify a query for the **Connection Type** combo box in the **Range** attribute of the filter field. The native SQL query

must find connection types and defines the property `NAME` of the connection type in the `SELECT` statement:

```
SELECT REFSTR, NAME
FROM CONNECTIONTYPE
```

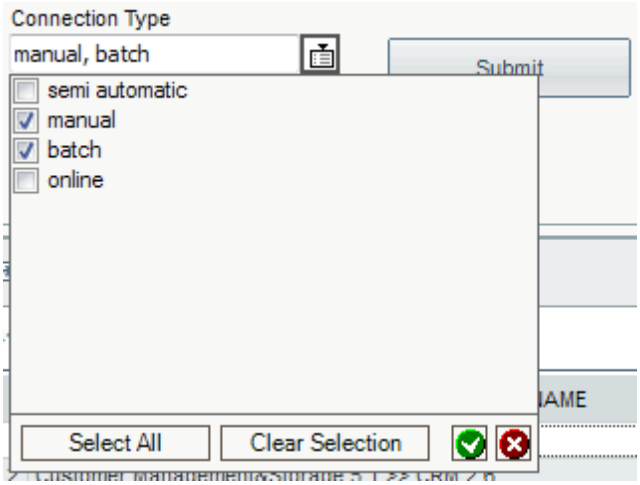
The checked combo box will then allow the user to select between the available connection types:



The combo box will return the `REFSTR` of the selected connection type. Therefore, the native SQL query the report is based on includes a `WHERE` clause that defines that the `REFSTR` stored in the property connection type of the information flow must be identical to the returned filter value:

```
SELECT inf.REFSTR, inf.NAME, con.NAME
FROM INFORMATIONFLOW inf, CONNECTIONTYPE con
WHERE inf.CONNECTIONTYPE=con.REFSTR
   AND inf.CONNECTIONTYPE = @type
```

In the second configuration example, the filter shall include a checked combo-box that allows multiple connection types to be selected:



The query used for the combo box above is also used for the checked combo box to fill the checked combo box with the names of the selectable connection types. Please note that although the property connection type of the information flow is of the data type Reference, the **Value Type** attribute of the checked combo box must be set to `ReferenceArray`. The **Value Type** attribute of the checked combo box specifies, which data type is returned by the filter field. When the user selects multiple values from the box, the returned value is a Reference Array.

In the query the report is based on, this has to be taken into account. The `WHERE` clause must be defined to find the connection type of the current information flow in the set of connection types returned by the filter field:

```
SELECT inf.REFSTR, inf.NAME, con.NAME
FROM INFORMATIONFLOW inf, CONNECTIONTYPE con
WHERE inf.CONNECTIONTYPE=con.REFSTR
   AND inf.CONNECTIONTYPE IN (@type)
```

Attribute Settings for Selection of a String Value

The following settings are required in the attribute section of the filter field to define a selectable list of strings:

- **Value Type**: Select `String`.
- **Sub Type**: Select `StringEnum`
- Range: Define each value separated by a line break, that means each value in a separate line:

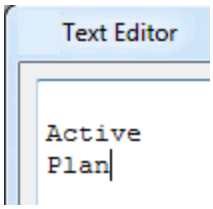
```
String1
String2
```

Limit the definition to no more than twenty values to ensure usability of the filter field.

If the user shall be allowed to deselect the filter setting, an empty line can be included.

Empty selections are not allowed for radio button groups and if defined for a radio button group, they are ignored.

For example, a configured report about applications shall include a filter field that allows the user to limit display to applications in a defined object state. The applications in the object state Retired are not relevant for the user and shall Therefore, not be selectable in the filter. In the filter a checked list box is defined with a **Value Type** `String` and a **Sub Type** `StringEnum`. In the attribute **Range**, the string values of the selectable object states are defined as well as an empty selection that allows the user to see all applications regardless of the object state:



In the checked list box, the user can select between "not set", "Active" and "Plan":

In the underlying query the value for the object state for each application is compared to the value returned by the filter field. The checked list box allows the selection of multiple values and the `WHERE` condition in the query checks whether the object state value of the application is within the range of values returned from the filter setting:

```
SELECT REFSTR, NAME, VERSION, OBJECTSTATE
FROM APPLICATION
WHERE OBJECTSTATE IN (@state)
```

Attribute Settings for Selection of a Value Defined in a Configuration Object

Operators are available to automatically fetch the values that are defined in XML objects or enumerations.

The following settings are required in the attribute section of the filter field to define the content on basis of content defined in an enumeration or an XML object.

1) In the attribute window, set the following attributes:

- **Value Type**: Select `String`
- **Parameter**: Enter the call of the operator that you want to use:
  - For enumerations: <name of object class>|<name of property based on enum>|`OPERATOR`
  - For lifecycle statuses: `TimeStatus|Status|OPERATOR`
  - For mandates: `Mandate|OPERATOR`
  - For stereotypes: <name of object class>|`Stereotype|OPERATOR`
  - For release statuses: <name of object class>|`Status|OPERATOR`
  - For object states:<name of object class>|`ObjectState|OPERATOR`
- **Custom Attribute**:
  - For enumerations: This attribute is not evaluated. No entry is required.
  - For lifecycle statuses: enter the name of the object class for that the lifecycle definitions shall be displayed.
  - For mandates: This attribute is not evaluated. No entry is required.
  - For stereotypes: This attribute is not evaluated. No entry is required.
  - For release statuses: To display the available release status definitions for a project stereotype in a combo box or a drop-down list, enter the name of the project

stereotype in the attribute field **Custom Attribute**. The release status for projects is configured individually for each project stereotype. For other release status definitions, this attribute is not evaluated. No entry is required.

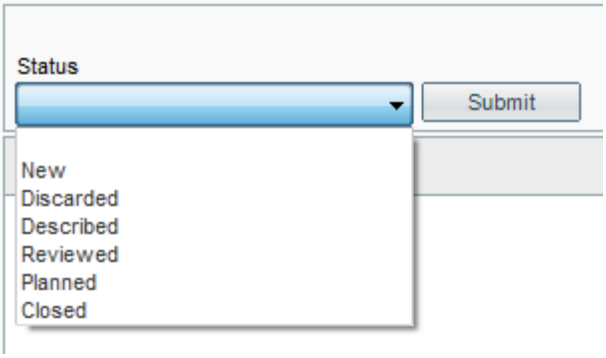
- For object states: This attribute is not evaluated. No entry is required.

A report shall allow users to view information about projects of the project stereotype `Program` with a specific release status. The query for the report defines the search condition for the stereotype as a fixed value and the search condition for the release status as a filter:

```
SELECT proj.REFSTR, proj.NAME, proj.STATUS
FROM PROJECT proj
WHERE proj.STEREOTYPE = 'Program'
   AND proj.STATUS LIKE @Status
```

To display release statuses in the combo-box, you must specify which release status definition you want displayed. In the example, the report lists projects of the project stereotype `Program`. Therefore, Program must be entered as **Custom Attribute** property.

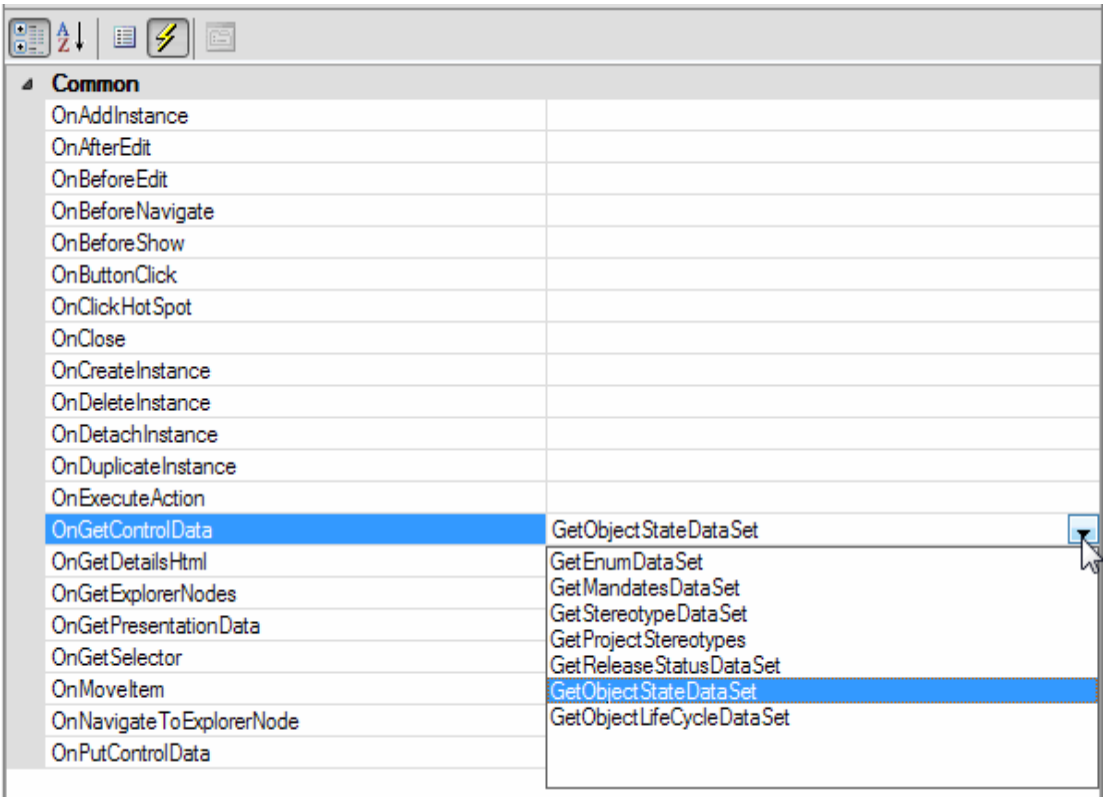
The checked combo box will then allow the user to select between the available release statuses of the project stereotype `Program`:



2) In the toolbar on top of the attribute window, click the  button. The content of the attribute window changes to display available operations.

To return to the attribute view, click the  button in the toolbar.

3) In the ***OnGetControlData*** field, select an operator from the drop-down list:



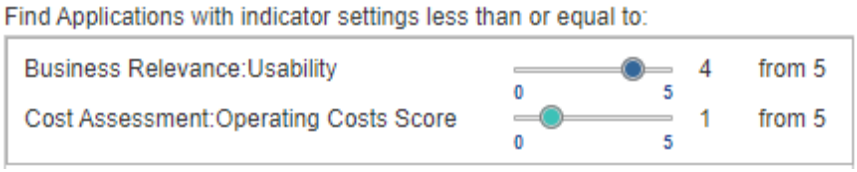
- For enumerations: `GetEnumDataSet`
- For lifecycle statuses: `GetObjectLifeCycleDataSet`
- For mandates: `GetMandatesDataSet`
- For stereotypes: `GetStereotypeDataSet` or `GetProjectStereotypes` if the object class is Project
- For release statuses: `GetReleaseStatusDataSet`
- For object states:`GetObjectStateDataSet`

4) In the toolbar of Alfabet Expand, click the **Save** button to save your changes.

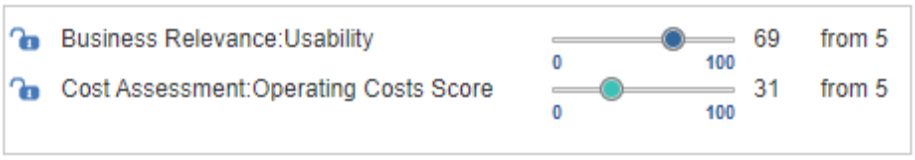
Configuring Slider Controls

Available for Data Types: `Integer`, `Real`

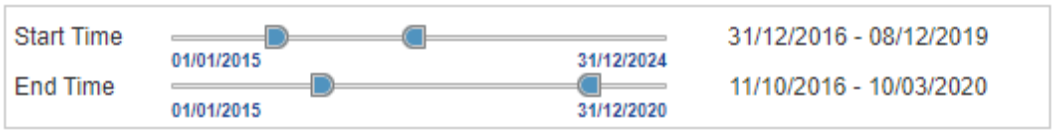
Slider controls combine the selection of multiple numeric or date values in one field. For each selectable value, a slider line is displayed with a slider that can be set to the currently selected value. A caption informing about the kind of value to be selected is displayed in front of each slider line and optionally the unit for the slider line values can be displayed at the end of the slider line. Information about the start and end values of the slider bar are automatically added as text beneath the start and end point of the slider. Optionally, the slider handlers can be colored differently in each bar:



Slider bars can be set independent from each other or the slider control can be configured to force the settings in all slider bars to sum up to a specified value or to be less than or equal to a specified value. If the slider bar settings must be less or equal to a specific value and the maximum value is reached, the user can only move a slider to a higher value if he/she has reduced the value for another slider bar first. If the slider bar settings must match a defined sum, moving one slider will automatically move all other sliders to matching values. In that case, a lock is displayed on the left of the slider caption:



Sliders can be configured to allow a range of values to be selected. In each slider line, two sliders are displayed, one for the selection of the start value of the range and one for the selection of the end value of the range.



In the query of the configured report, the values set for each slider bar can be used as parameters in `WHERE` clauses. The parameter must be identical to the name of the slider control followed by two underscores and the number of the slider bar, starting with zero for the first slider bar, For example, `@SliderControl__0`.

For sliders that allow a range of values to be selected, the parameter name must have an f for the "from value" and a t for the "to value" added in front of the slider bar number, For example, `@SliderControl__f0` and `@SliderControl__t0` for the from and to value in the first slider bar.

The content of the slider control is defined in both an XML element and in a query, which can be defined either in native SQL or in Alfabet query language.

The following parts of the slider control are only defined in the XML definition and are static, that means they are identical for each slider bar:

- The width of the slider control reserved for the captions of the slider bars.
- Whether slider bars are independent of each other or sum up to a defined value.
- Whether each slider bar allows a single value or a range from and to defined values to be selected.
- Whether dates or numeric data shall be selected in the slider.
- A message with instructions for the user that will pop up if the slider definition depends on the setting in a master filter field and the master filter field is not set.

The following parts of the slider control can be defined either in the XML definition as static values or in the query:

- The range and step definition of the slider bars.
- The text at the right of each slider bar.

The following parts of the slider control are defined exclusively in the query:

- The number of slider bars is identical to the number of results found by the query.
- The caption of the slider bar is defined in the query.
- The primary location of the slider on the slider bar is defined in the query.

To define a slider control for a filter:

1) In the explorer, double-click the **Custom Report View** of the configured report or the presentation object of the class entry of the custom selector. The view editor opens with a toolbar for interface control creation on top.
2) In the toolbar, click the **Slider**  icon.
3) Drag the mouse on the filter panel in the report editor to create a box in the location and width of your choice.
4) In the attribute window of the new filter field, set the following attributes:
   - **Name**: Change the **Name** of the filter field to the parameter specified in the query of the report.
   - **Caption**: Enter a caption for the filter field. The caption is displayed on top of the field.
   - **Hint**: Optionally define a tooltip that is displayed for the filter field when the user moves the cursor over the field caption.
5) Click the **Range As Native SQL Query** or **Range As Alfabet Query** attribute and click the **Browse**  button at the right of the field to open the text editor or the Alfabet query builder respectively.
6) Define a query that returns one row for each slider bar, with the following columns in the result dataset:
   - If the query is a native SQL query, the first column in the dataset is ignored.
   - The string that shall be displayed as caption on the left of the slider bar must be returned. The column name must be `Name`, or, if another column name is returned, this column name must be specified in the **XML Definition** attribute.
   - The numeric or date value of the start position of the slider on the slider bar must be returned. The column name must be `Value`, or, if another column name is returned, this column name must be specified in the **XML Definition** attribute.

     For numeric values, the value must be a multiple of the defined step value. If you have defined a step of 10, you cannot define a value of 25, but of 20 or 30 only.

   - Optionally, the maximum and minimum value of the slider bar can be returned in two columns of the report. The column names must be `MinValue` and `MaxValue`. Alternatively, a fixed value can be defined for the slider bar range in the **XML Definition** attribute.

Please note the following:

- If the minimum and maximum value of the slider bar are both defined in the query and in the **XML Definition** attribute, the query definition supersedes the XML definition.
- If the slider is configured to select numerical values, and the minimum and maximum value are defined in the query, the step for setting the slider handle must also be defined in the query.

- Optionally, the step for setting the slider can be returned. The column name must be `Step`, or, if another column name is returned, this column name must be specified in the **XML Definition** attribute.

  For sliders configured to select dates, the step is always 1 and cannot be defined via the query or the **XML Definition** attribute.

  For sliders configured to select numeric values, the step definition is optional. It can be omitted in both the query and the **XML Definition** attribute. The default step is 1.

- Optionally, the unit to be displayed behind the slider bar can be returned as text string. The column name must be `Unit`, or, if another column name is returned, this column name must be specified in the **XML Definition** attribute. This feature is currently only available if the XML attribute `Mode` in the **XML Definition** attribute is set to `None` which means that the slider values are numerical values set independent from each other.
- Optionally, the color of the slider marking the current value in a slider bar can be returned in HTML compatible color code. The column name must be `Color`, or, if another column name is returned, this column name must be specified in the **XML Definition** attribute.

Alfabet parameter `@BASE` can be used in the query to refer to the current object the report or selector is opened for. For more information about Alfabet parameters, see Comparing Property Values with Other Property Values.

The following native SQL query defines two slider bars, using the default column names for all defined values:

```
SELECT NULL AS REFSTR, 'Business Relevance:Usability' AS
'Name', 3.0 AS 'Value' 1 AS 'MinValue', 5 AS 'MaxValue', 1 AS
Step, '#336699' AS 'Color'

UNION ALL

SELECT NULL AS REFSTR, 'Cost Assessment:Operating Costs
Score' AS 'Name', 4.0 AS 'Value', 1 AS 'MinValue', 5 AS
'MaxValue', 1 AS Step, '#3CC1B7' AS 'Color'
```

The following query defines two slider bars for a range of dates to be selected in each slider bar:

```
SELECT NULL AS 'REFSTR', 'Start Time' AS Name, cast('2015-01-
01' AS date) AS MinValue, cast('2024-12-31' AS date) AS
MaxValue, cast('2016-12-31' AS date) AS FromValue,
cast('2017-12-31' AS date) AS ToValue

UNION ALL
```



```
SELECT NULL AS 'REFSTR', 'End Time' AS Name, cast('2015-01-
01' AS date) AS MinValue, cast('2020-12-31' AS date) AS
MaxValue, cast('2018-01-01' AS date) AS FromValue,
cast('2020-12-31' AS date) AS ToValue
```

7) Click the **XML Definition** attribute and click the **Browse** button at the right of the field to open the text editor. A template with the XML code for the slider definition is displayed.

8) Set the following for the XML attributes in the XML elements of the **XML Definition** code, as needed:

`Slider` XML element:

- `Mode`:Enter one of the following modes:
  - `None`: The slider allows numeric values to be selected. All slider lines are independent from each other. This is the default value. This setting allows a slider to be defined with two slider handlers for setting a range from a defined value to a defined value.
  - `SumLessEqual`: The slider allows numeric values to be selected. The slider settings on all slider bars must be less than or equal to the value defined with the attribute `CompareValue`. If the maximum value is reached, a slider can only be moved to a higher value if another slider is set to a smaller value beforehand.
  - `SumEqual`: The slider allows numeric values to be selected. The sum of the slider settings on all slider bars must match the value defined with the attribute `CompareValue`. If one slider is moved, the sliders on the other bars are adapted automatically to keep the overall value.
  - `Date`: The slider allows dates to be selected. All slider lines are independent from each other.
- `CompareValue`: If the `Mode` is set to `SumLessEqual` or `SumEqual`, enter the sum that the slider settings shall match or be less than or equal to.
- `EmptyMessage`: If the rendering of the slider control depends on the setting in a master filter field, you can define a message to inform the user what he/she shall do to view the slider control. The message will be automatically displayed if the master filter field is empty.
- `NameColumn`: The text displayed to the left of each slider bar is defined in a query with the attribute **Range As Native SQL Query** or **Range As Alfabet Query** . Enter the name of the column in the result data set of the query that returns the text. The default column name is `Name`.
- `RangeType`: This XML attribute must be set to `true` if a range of values shall be selected per slider bar. It must be set to `false` or left out in the XML definition to configure a slider with a single value selection per slider line.
- `MinValueColumn`: If the minimum value of the slider bar is defined via the query in the **Range As Native SQL Query** or **Range As Alfabet Query** attribute. Enter the name of the column in the result data set of the query that returns the minimum value. The default column name is `MinValue`. The slider range can alternatively be defined statically via the XML element Range in this XML definition. A definition in the query overrides a definition in the XML element.
- `MaxValueColumn`: If the maximum value of the slider bar is defined via the query in the **Range As Native SQL Query** or **Range As Alfabet Query** attribute. Enter the name of the column in the result data set of the query that returns the maximum value. The default

column name is `MaxValue`. The slider range can alternatively be defined statically via the XML element `Range` in this XML definition. A definition in the query overrides a definition in the XML element.

- `StepColumn`: If the steps on the slider bar are defined via the query in the **Range As Native SQL Query** or **Range As Alfabet Query** attribute, enter the name of the column in the result data set of the query that returns the step of the slider handle on the slider bar. The default column name is `Step`. The step can alternatively be defined statically via the XML element `Range` in this XML definition. A definition in the query overrides a definition in the XML element.
- `UnitColumn`: If the unit that shall be displayed on the right side of the slider bar is defined via the query in the **Range As Native SQL Query** or **Range As Alfabet Query** attribute, enter the name of the column in the result data set of the query that returns the unit text. The default column name is `Unit`. The unit text can alternatively be defined statically via the XML element `ItemValueText` in this XML definition. A definition in the query overrides a definition in the XML element.
- `ValueColumn`: For a single value selection per slider bar, the initial placement of the slider handle on each slider bar is defined in a query in the **Range As Native SQL Query** or **Range As Alfabet Query** attribute. Enter the name of the column in the result data set of the query that returns the value on the slider bar that the slider handle is set to on first opening of the configured report. The default column name is `Value`.
- `FromValueColumn`: For selection of a range of values per slider bar, the initial placement of the slider handle for the start of the range on each slider bar is defined in a query in the **Range As Native SQL Query** or **Range As Alfabet Query** attribute. Enter the name of the column in the result data set of the query that returns the value on the slider bar that the slider handle for the minimum value selection is set to on first opening of the configured report. The default column name is `FromValue`.
- `ToValueColumn`: For selection of a range of values per slider bar, the initial placement of the slider handle for the end of the range on each slider bar is defined in a query in the **Range As Native SQL Query** or **Range As Alfabet Query** attribute. Enter the name of the column in the result data set of the query that returns the value on the slider bar that the slider handle for the maximum value selection is set to on first opening of the configured report. The default column name is `ToValue`
- `ColorColumn`: If the color of the slider handles can be defined via the query in the **Range As Native SQL Query** or **Range As Alfabet Query** attribute. Enter the name of the column in the result data set of the query that returns the text. The default column name is `Color`

`Range` XML element:

This XML element is optional. A definition in the query in the **Range As Native SQL Query** or **Range As Alfabet Query** attribute supersedes the XML definition.

- `MinValue`: Enter the minimum value for the slider bars. All slider bars have the same range. By default all sliders start with zero.
- `MaxValue`: Enter the maximum value for the slider bars. All slider bars have the same range. By default all sliders end with 100.
- `Step`: Enter the step for the settings of the slider on the slider bars. All slider bars have the same step definition. The default step width is 1.

`ItemName` XML element:

- `NameWidthInPercents`: Enter the percentage of the space in the slider control that will be used for display of the slider bar name in front of the slider bar. For sliders with the `Mode` set to `SumEqual`, a lock symbol is displayed in front of the name. In this case the space defined with `NameWidthInPercents` is used for both lock symbol and name. Enter the percentage of space as an integer between 0 and 100. The default value is 50, which means 50% of the slider width is used for the slider bar name.

`ItemValueText` XML element:

This XML element is optional. A definition in the query in the **Range As Native SQL Query** or **Range As Alfabet Query** attribute supersedes the XML definition.

- `Value`: Enter the text that shall be displayed behind the slider bars to inform the user about the units used in the slider. By default this XML attribute is empty and no text is displayed.

9) In the toolbar of Alfabet Expand, click the **Save** button to save your changes.

Re-Using Filter Fields Using the Copy and Paste Functionalities

Copy and paste functionalities are available to re-use filter fields. A filter field can be copied to another location in the same filter panel or to another filter panel. If the filter field is copied to the same filter panel, the Name attribute is changed, but all other attribute settings are kept. Copying a filter field into the same filter panel can be useful if two fields in the filter differ in a subset of the attribute settings and shall have the same design. One filter filed can be completely designed, copied into the same filter panel and the attributes defining the content of the filter field and the mapping to the report content can then be changed.

To copy and paste a filter field:

1) In the explorer, right-click the **Custom Report View** of the configured report containing the filter field that you want to copy and click **Design View**. The report view opens in the middle pane of Alfabet Expand and in the menu bar, the menus **Edit** and **Format** are displayed.
2) Click the filter field that you want to copy, click on the **Edit** menu and select **Copy**, if you want to copy the filter field or **Cut**, if you want to move the filter field to another location.
3) In the explorer, right-click the **Custom Report View** of the configured report to which you want to copy the filter field and click **Design View**. The report view opens in the middle pane of Alfabet Expand and in the menu bar, the menus **Edit** and **Format** are displayed.
4) In the menu bar, click the **Edit** menu and select **Paste**.
5) In the filter panel, click the location that you want to add the filter field to. The filter field is included into the filter panel.
6) In the toolbar of Alfabet Expand, click the **Save** button to save your changes.

In addition to the copy, cut and paste options the **Edit** menu allows to delete a selected filter field. Filter fields removed from a filter panel via the **Delete** option of the **Edit** menu are not available for paste actions.

Configuring Filter Layout

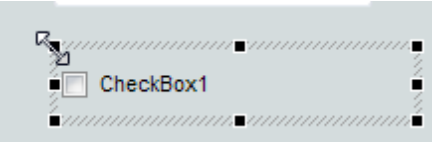
After having added filter fields to a filter panel, you can further design the filter to enhance usability. The following options are available:

- Changing the Size of a Single Filter Field
- Changing the Location of a Single Filter Field
- Aligning the Size and Position of Filter Fields
- Grouping Filters on the Filter Panel
- Adding Static Information to the Filter Panel
  - Adding Plain Text to the Filter Panel
  - Adding an HTML text to the Filter Panel
- Adding an Icon With or Without a Link to Another Alfabet View to the Filter Panel
- Configuring the Submit Button
- Changing the Sequence of the Focus on Filter Fields
- Configuring the Complete Filter Area to be Collapsible
- Configuring Part of the Filter Area to be Collapsible
- Allowing the User to Clear the Filter Area
- Changing the Position of the Filter Panel

Changing the Size of a Single Filter Field

There are two methods to change the size of a filter field on the panel:

- Click a filter field and pull the handles to resize it.



- Click the filter field in the panel to open the attribute window and change the following attributes in the section **Coordinates**:
  - **Height**: Define the height of the filter field.
  - **Width**: Define the width of the filter field.

After changing the size, click the **Save** button in the toolbar to save your changes.

You can combine both methods for easy and correct sizing of filter fields on the panel. After having resized the fields using the handles to define the basic layout of the filter panel, correct the size of the filter fields

in relation to each other using the attributes in the section **Coordinates**. For example, you can define an equal **Width** for filter fields that are placed below each other or an equal **Height** for filter fields displayed in the same row.

The attributes **Height** and **Width** cannot be defined for filter fields in configured reports of the type `Query`.

Changing the Location of a Single Filter Field

There are two methods to change the location of a filter field on the panel:

- Drag-and-drop the filter field to the correct location and click the **Save** button in the toolbar to save your changes.
- Click the filter field in the panel to open the attribute window and change the following attributes in the section **Coordinates**:
  - **Left**: Define the distance from the left of the panel
  - **Top**: Define the distance from the top of the panel

    If filter fields are placed in a group box, the values for **Left** and **Top** are marking the distance to the border of the group box instead to the border of the filter panel.

After changing the field location, click the **Save** button in the toolbar to save your changes.

You can combine both methods for easy and correct placement of filter fields on the panel. After having used drag-and-drop to define the basic layout of the filter panel, correct the location of the filter fields in relation to each other using the attributes in the section **Coordinates**. Filter fields that are placed below each other should have the same value for the attribute **Left**. Filter fields that are placed next to each other in the same row should have the same value for the attribute **Top**.

The attributes **Left** and **Top** cannot be defined for filter fields in configured reports of the type `Query`.

Aligning the Size and Position of Filter Fields

When you click an element in a filter panel, you can see a menu **Format** in the menu bar of Alfabet Expand. Via the options in the **Format** menu the design of multiple filter fields in the filter panel can be aligned to each other with regard to the position of the filter field in the filter panel and the filter size:

Aligning the Position of Multiple Filter Fields

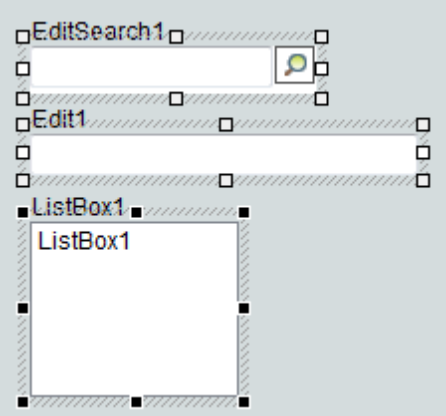
You can align multiple filter fields with one another to have an equal position for one of the borders of the field. For example, if you select multiple fields that are located next to each other in the same row, you can align the position of the top border to ensure that all fields are on the same horizontal axis.

1) In the filter panel, place one of the filter fields in the correct position.

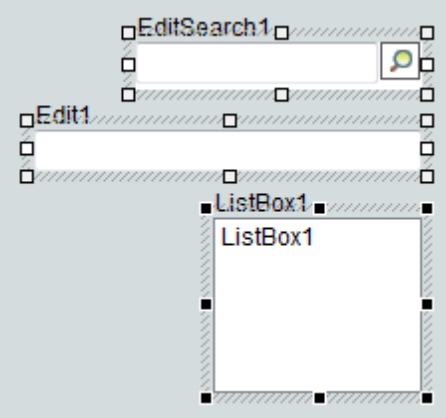
2) Select all filter fields that you want to align with each other holding CTR + SHIFT while clicking on the fields. The last field that you select is the field that determines the position.

3) In the toolbar, select:

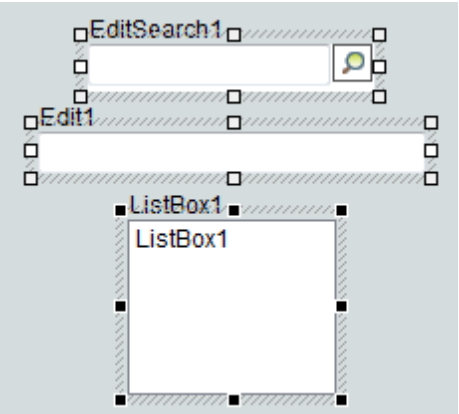
- **Format** > **Align Left** to align the position of the left border of all selected filter fields with the position of the last selected filter field.



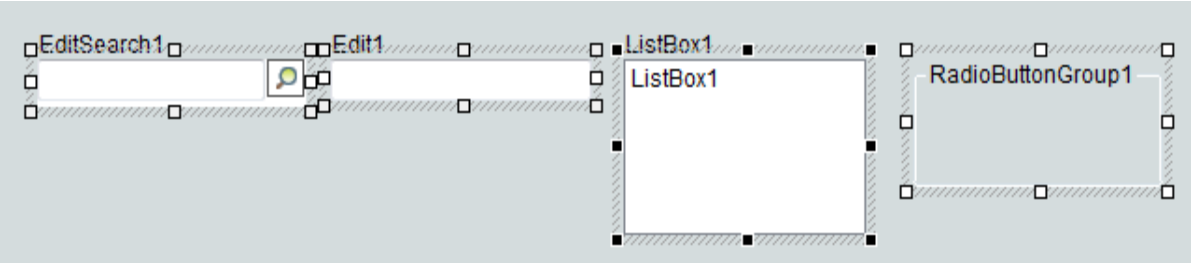
- **Format** > **Align Right** to align the position of the right border of all selected filter fields with the position of the last selected filter field.



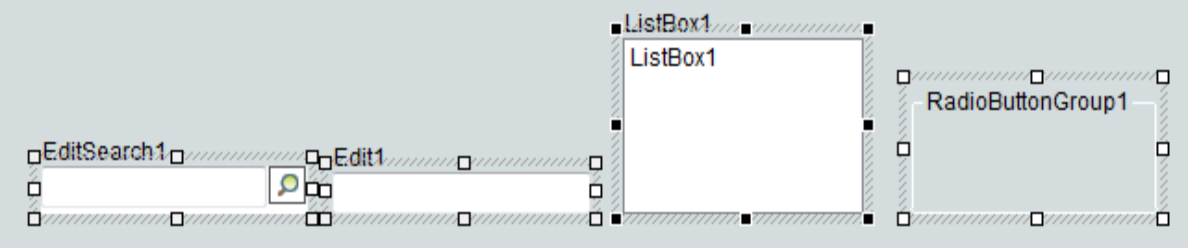
- **Format** > **Align Center** to align the horizontal middle position of all selected filter fields with the horizontal middle position of the last selected filter field.



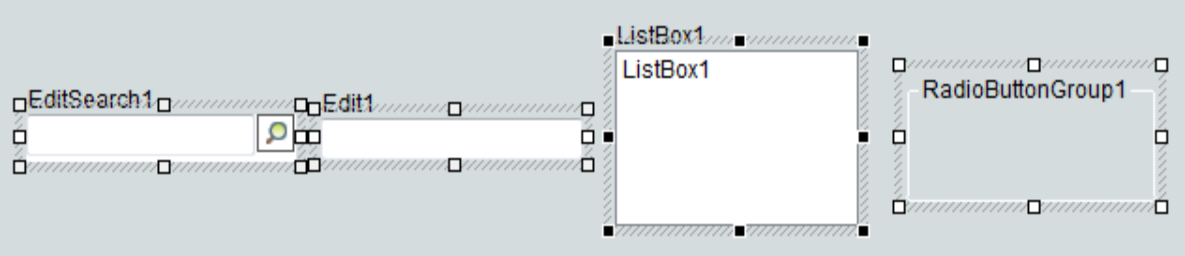
- **Format** > **Align Top** to align the position of the top border of all selected filter fields with the position of the last selected filter field.



- **Format** > **Align Bottom** to align the position of the bottom border of all selected filter fields with the position of the last selected filter field.



- **Format** > **Align Middle** to align the vertical middle position of all selected filter fields with the vertical middle position of the last selected filter field.



4) In the toolbar, click the **Save** button to save your changes.

Aligning the Size of All Filter Fields

The width or height or one or multiple selected filter fields can be aligned to the width or height of a selected filter field.

1) In the filter panel, select all filter fields that you want to resize holding CTR + SHIFT while clicking on the fields. The last field that you select is the field that determines the resulting width or height of all fields.

2) In the toolbar, select:

- **Format** > **Make Widths Equal** to adapt the widths of all selected fields to the width of the last selected field.

- **Format** > **Make Heights Equal** to adapt the heights of all selected fields to the height of the last selected field.

3) In the toolbar, click the **Save**  button to save your changes.

Aligning Spacing between Filter Fields

If multiple filter fields are placed in a horizontal row or a vertical column you can define an equal spacing between all fields in a single row or in a single column. The position of the first and the last field is maintained and the fields in between are distributed equally in between.

1) In the filter panel, select all filter fields that you want to distribute equally along a vertical or horizontal axis. Please note that only fields from a single column or row shall be selected.
2) In the toolbar, select:
   - **Format** > **Make Vertical Spacing Equal** to distribute the fields equally along the vertical axis.
   - **Format** > **Make Horizontal Spacing Equal** to distribute the fields equally along the horizontal axis.
3) In the toolbar, click the **Save**  button to save your changes.

Applying the Default Spacing between Filter Fields

When filters are created automatically, a default spacing is applied that takes font size for filter headings into account. You can apply the default spacing to your filter fields after having added design elements and fields:

1) Click anywhere in the filter panel.
2) In the toolbar, select **Format** > **Set Default Spacing**.
3) In the toolbar, click the **Save**  button to save your changes.

The default spacing is applied to all elements in the filter.

Aligning the Filter Fields Along a Grid

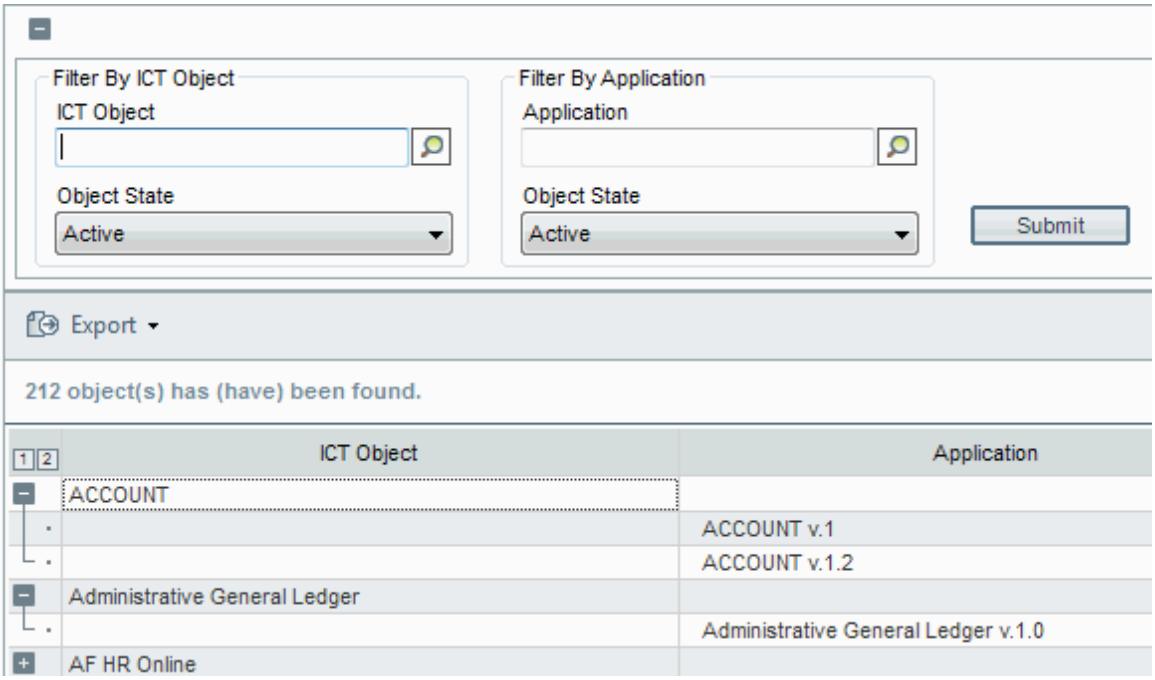
The filter fields can be aligned along a grid that is invisibly drawn on the filter panel. This method allows to align the position to the filter fields horizontally and vertically.

The top left corner of the filter fields are aligned with the respective crossing point of the grid lines left and above the filter field.

1) Select the filter fields that you want to align with along the grid by holding CTR + SHIFT while clicking the fields.
2) In the toolbar, select **Format** > **Snap to Grid**. The selected filter fields are positioned with the upper left corner matching the nearest crossing point of grid lines.
3) If the results are not as expected, you can alter the spacing between grid lines by selecting **Format** > **Grid Size...** in the toolbar and setting the spacing between grid lines in the window that opens. The default value is 2. It is recommended to select a number between 2 and 10 for a start and check the resulting effect on the filter first before moving on to higher numbers. If the spacing between grid lines is too high, the filter fields might be placed outside the filter panel.
4) In the toolbar, click the **Save**  button to save your changes.

Grouping Filters on the Filter Panel

You can optically group fields into group boxes:



To group filter fields:

1) In the toolbar, click the **Group Box** icon.
2) Drag the mouse on the filter panel in the report editor to create a text box in the location and width of your choice.

   The group box must be placed next to the filter fields that shall be part of the group. If you place the group box on top of the filter fields, the filter fields are not placed within the box but under the box and are not visible any longer.

3) In the attribute window of the new filter field, set the following attributes:
   - **Caption**: Enter a caption for the group box. The caption is displayed within the upper border of the field.
4) Place the filter fields that shall be part of the group into the group box using drag and drop.
5) In the toolbar, click the **Save** button to save your changes.

After having filled the group box, you can alter the size and location of both boxes and fields within the box using the mechanisms described in the sections Changing the Size of a Single Filter Field and Changing the Location of a Single Filter Field.

For filter fields that are placed in a group box, the values for the attributes **Left** and **Top** of the filter field are marking the distance to the border of the group box instead to the border of the filter panel.

Adding Static Information to the Filter Panel

You can add text to the filter panel, display current values stored in the Alfabet database for informational purposes or link from the filter panel to information available via a URL For example, to provide information about how to work with the filter or to provide information about the output of the configured report or selector. The following methods to add information to a filter panel are available:

- Adding Plain Text to the Filter Panel
- Adding an HTML text to the Filter Panel

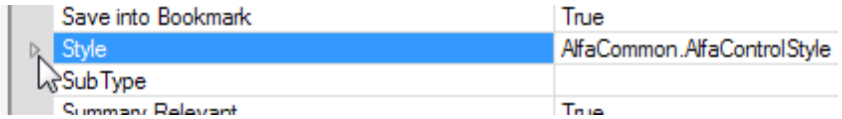
Adding Plain Text to the Filter Panel

To add a static text field to a filter panel:

1) In the toolbar, click the **Static Text** icon.
2) Drag the mouse on the filter panel in the report editor to create a text box in the location and width of your choice.
3) In the attribute window of the new filter field, set the following attributes:
   - **Caption**: Enter the text that shall be displayed in the text box.
4) In the toolbar, click the **Save** button to save your changes.

Optionally you can alter the style of the text by defining borders and a background color and by altering the text style and color:

1) Click the static text field in the filter panel.
2) In the attribute window, expand the **Style** attribute by clicking on the triangle in front of the attribute.

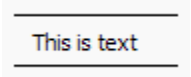


3) To alter a style, alter the style sub-attributes.

Please note that an attribute with a triangle in front of the attribute name cannot be set in the attribute itself. Instead, click the triangle to see a lower level of sub-attributes that can be used to define single dimensions of the attribute.

- **To define a border:**

Borders can be defined separately for each side of the text field. For example, you can set a border on top and bottom of the text only:



To define a border for one side of the box, expand one of the style attributes **Border Left**, **Border Right**, **Border Bottom** or **Border Top** and define both sub-attributes:

- **Border**: Define the width of the border as integer. If you define 0, no border is displayed. Setting the attribute to 1 results in display of a thin border. To define a wide margin, enter an integer higher than 1.
- **Color**: Click in the attribute field. A button appears on the right of the attribute field. Click the button to open a color picker and select a color. By default the color of the border is transparent.

- **To define a background color:**

To define a background color, click in the **Back Color** attribute field. A button appears on the right of the attribute field. Click the button to open a color picker and select a color. By default the background color is transparent.

The background color is applied to the whole text box when the filter is displayed on the Alfabet user interface, but in the editor in Alfabet Expand, the background color is only applied directly to the text.

- **To define the text style:**

To define the text style, expand the **Font** attribute and define the following sub-attributes:

- To alter the font style, click in the **(Name)** attribute field. A button appears on the right of the attribute field. Click the button and select a text style from the drop-down list.
- To alter the text color, click in the **Color** attribute field. A button appears on the right of the attribute field. Click the button and select a color from the color selector.
- To display text in bold, italic, underline or strike the text, set the corresponding attribute **Bold**, **Italic**, **Underline** or **Strikeout** to `True`.
- To alter the text size, change the real number in the **Size** attribute.

- **To define the text location in the text field:**

To define the location of the text in the text field, do one of the following:

- Expand the attribute **Font** and define the horizontal alignment of the text in the text box with the attribute **Horizontal Alignment** and the vertical alignment of the text in the text box with the attribute **Vertical Alignment**.
- Expand the attribute **Margin** and define the distance between the text and the border of the text box. If you want the same distance between the text and the border to be applied for all directions, you can define the value in the attribute **All**. Alternatively, you can define different values for the margin on top, bottom, left and right of the text by setting the attributes **Bottom**, **Top**, **Left** and **Right** to the individual values. The attribute **All** is then automatically set to -1.

4) In the toolbar, click the **Save**  button to save your changes.

Adding an HTML text to the Filter Panel

An **HTML Control** interface element can be added to a filter panel to display text with HTML formatting (font and color). A link to an URL providing additional information can also be added to the HTML text.

Embedded images may not be included in the HTML code.

To add HTML text to a customer editor:

1) In the toolbar, click the **HTML Control**  icon.
2) Drag the mouse on the filter panel in the report editor to create a text box in the location and width of your choice.
3) In the attribute window of the new filter field, set the following attributes:

- **Sub Type**: Select `EmbeddedHTML`.

  The HTML interface element has additional options in the drop-down list for the specification of the **Sub Type** attribute. These other options are currently not activated.

- **HTML Source**: Enter the HTML content that shall be displayed in the field. Note the following requirements for the definition of the HTML code:

  - The HTML must be compliant with XHTML, XML-conform HTML, compliant with HTML 5, and use standard HTML tags. The HTML header implements standard HTML elements.
  - The code must start with an `<xhtml>` tag and end with `</xhtml>`. Standard HTML elements must be written in lower-case letters in order to be correctly parsed: `<xhtml>`, `<html>`, `<head>`, `<body>`, and `<culture_>`. The definition of `<head>`, `<body>`, and `<culture_>` is optional.
  - The formatting of the HTML can either be written explicitly in the `<body>` element or can be specified via a stylesheet that is stored in the **Internal Document Selector** . In this case, the HTML must refer to the target CSS file in a `<link>` element. The CSS file should contain all necessary styles to display the content of the HTML. Please note that the CSS file may not be stored in the root folder of the **IDOC** explorer. The CSS file stored in the **Internal Document Selector** must be located in a document folder that is subordinate to the root folder of the **IDOC** explorer. To upload a file to the **Internal Document Selector**, see the section *Uploading Documents and Managing User Permissions to Document Folders in the Internal Document Selector* in the reference manual *User and Solution Administration*. For example,:

```
<xhtml>
    <culture_1033>
        <html>
        <body
        style="margin:4px;background:#3d4b60;overflow:hidden;">
            <link type="text/css" rel="stylesheet"
            href="IDOC:\CSS\my_style_file.css"></link>
            <p>This is the content for the header of the wizard
            step.</p>
```



```
            ...
        </body>
        </html>
    </culture_1033>
</xhtml>
```

- All CSS formatting instructions must end with `!important` to ensure that they are processed by the Alfabet application. For example,:

```
<xhtml>
    <culture_1033>
        <html>
        <body style="margin:4px !important;background:#3d4b60
        !important;overflow:hidden !important;">
            <style type="text/css">
                p.header
                {
                    font-family:verdana !important;
                    font-size:18px !important;
                    color:#ff0000 !important;
                    text-align: Left !important;
                }
            </style>
            <p class="header">This is the content for the header of
            the wizard step.</p>
            ...
        </body>
        </html>
    </culture_1033>
</xhtml>
```

- If translation to a secondary language is required, you must provide the translation in the HTML specification. Please note that HTML texts cannot be translated via the **Translation Editor** available in Alfabet Expand. Please consider the following:
  - If the HTML description is required in additional languages, each language text must be defined within language elements (For example, `<culture_1031>` for English, `<culture_1033>` for German, etc.)
  - The element `<culture_xxx>` must be specified as a child of the root element `<xhtml>`. The element `<html>` must be specified as a child of the element `<culture_xxx>`. The element `<html>` contains the HTML specification in the relevant language. For example, to provide information in English and German:

```
<xhtml>
    <culture_1033>
        <html>
```



```
                    <body>
                        <h3>Glossary: Application</h3>
                        <p>An application is a fully-functional integrated
                        IT product that provides functionality to end
                        users and/or to other applications. As such, an
                        application supports the business to accomplish
                        its mission. Applications operate on a platform
                        made up of hardware and software components
                        necessary to run the application.</p>
                    </body>
                </html>
            </culture_1033>
            </culture_1031>
                <html>
                    <body>
                        <h3>Glossar: Applikation</h3>
                        <p>Eine Applikation ist ein voll funktionsfähiges,
                        integriertes IT-Produkt, das Funktionalitäten für
                        Endanwender und/oder für andere Applikationen
                        bietet. Eine Applikation unterstützt das
                        Unternehmen bei der Zielerreichung. Applikationen
                        werden auf einer Plattform betrieben, die aus den
                        für die Ausführung der Applikation erforderlichen
                        Hardware- und Software-Komponenten besteht.</p>
                    </body>
                </html>
            </culture_1031>
        </xhtml>
```

4) In the toolbar, click the **Save** button to save your changes.

Instead of writing the text directly into the HTML control, you can alternatively store it as an HTML document and upload it to the **Internal Document Selector** in Alfabet. In the HTML control, you can link to the document in the **Internal Document Selector** and the text will be uploaded from the HTML document. This method is useful if you want to re-use the same text in various HTML controls.

For information about adding documents to the **Internal Document Selector** available in the **Internal Documents** functionality in Alfabet, see the section *Uploading Documents and Managing User Permissions to Document Folders in the Internal Document Selector* in the reference manual *User and Solution Administration*.

To add an HTML control reading content from an HTML file in the **Internal Document Selector**:

1) In the toolbar, click the **HTML Control** icon.
2) Drag the mouse on the filter panel in the report editor to create a text box in the location and width of your choice.
3) In the attribute window of the new filter field, set the following attributes:
   - **Sub Type**: Select `IDocument`.

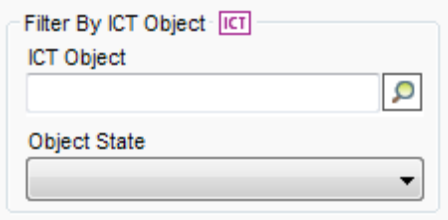
- **HTML Source**: Specify the full path to the HTML document in the **Internal Document Selector** . The path starts with `IDOC:\` followed by the folder structure within the **Internal Document Selector** defined with backslashes between folder names and file name. For example,:

  ```
  IDOC:\Documents\FilterHelp\General.html
  ```

4) In the toolbar, click the **Save** button to save your changes.

Adding an Icon With or Without a Link to Another Alfabet View to the Filter Panel

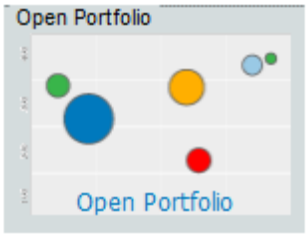
Icons can be added to the filter panel to enhance the design. Optionally, a link can be added to the image. When the user clicks the image with the underlying link, the configured target view in Alfabet opens. The link target can be either a standard graphic view or a configured report. For example, if a group box groups filter fields that allow to select properties of a defined object class, the icon used for the object class in explorers on the Alfabet user interface can be added next to the text in the group box:



Any icon that you want to add to the filter panel first must be imported to the icon gallery. For more information, see the section Adding and Maintaining Icons for the Alfabet Interface in the chapter Configuring the Visualization of the Alfabet Interface.

Icon fields can have a caption. The caption is displayed beneath the image in the Alfabet user interface, but when you design the view in Alfabet Expand you will see the caption within the image:

Icon field in Alfabet Expand:



Icon field displayed in the Alfabet user interface:

Open Portfolio

To add an icon to the filter panel:

1) In the toolbar, click the **Icon** icon.
2) Drag the mouse on the filter panel in the report editor to create a box in the location and width of your choice.
3) In the attribute window of the new filter field, set the following attributes:

   - **Sub Type**: Select the relevant option based on the size of the icon that you want to add:

- `Icon`: To add an icon that is 22x22 pixel in size.
- `IconLarge`: To add an icon that is 30x30 pixel in size.
- `IconFree`: To add an icon that has an arbitrary size.

- **Icon**: Click into the attribute field and then click the button on the right of the field to select an icon from the drop-down list.

4) To adjust the icon interface control to the general size of the icon, expand the **Coordinates** section of the property grid and enter pixel values in the **Height** and **Width** fields. For example, if you plan to add an icon that is 22x22 pixel in size, you could enter values 22 or larger in the **Height** and **Width** field to make handling the element easier while designing the custom editor.

5) In the **Caption** attribute, alter the default caption or delete it, if you do not want a text to be displayed beneath the icon.

6) In the toolbar, click the **Save** button to save your changes.

Optionally, you can configure the image to provide a link to a standard page view or configured report:

Please note the following about links from icons in filter panels:

- No base object is provided for the link. Make sure that a target standard page view is designed to provide meaningful results without a base object specification. For configured reports, filter settings of the current report can be provided to the link target using storage of filter results in the object class `UserGlobalData`. For more information, see Configuring Reusability of Filter Settings Across Reports.
- If a caption is displayed beneath the image, the font style of the caption is identical to the font style of all filter captions. If a link is provided with the icon, the caption style is the link font style defined for the user interface in the current GUI scheme. For more information about GUI schemes for defining the design of the Alfabet user interface, see Configuring GUI Scheme Definitions for the Alfabet Interface.

1) Click the icon field in the filter panel.

2) In the attribute window, click the **Source** attribute and click the **Browse** button at the end of the field. A new window opens.

3) Set the following attributes:

- **Source Type**: Select **Configured Reports** if the link shall open a configured report or **Page Views** if the link shall open a standard Alfabet page view.
- **Source Object**: Select a configured report or standard Alfabet page view from the drop-down list. The list is automatically adapting to the choice made in the field **Source Type**.

4) Click **OK** to close the window.

5) In the toolbar, click the **Save** button to save your changes.

Configuring the Submit Button

Both filter panels for configured reports and custom selectors have a preconfigured button **Submit** to apply the filter settings to the search result. On a filter, no other buttons are allowed.

You can adapt the text of the button according to your demands:

1) Click the button in the filter panel to open the attribute window of the button.
2) In the attribute window, edit the following attributes, if applicable:
   - **Caption**: Enter the text that shall be displayed in the button.
   - **Hint**: Define a tooltip that is displayed for the button when the user moves the cursor over the button.

The size and the position of the button can also be altered as described in the sections Changing the Size of a Single Filter Field and Changing the Location of a Single Filter Field.

The coloring and text style of the button is designed via the GUI scheme configuration for the user profile. The two different button styles **Primary Button Skin** and **Secondary Button Skin** can be configured in the GUI scheme to highlight the central buttons in a view with multiple buttons via a different design. The **Submit** button in filters is assigned the **Primary Button Skin**.

For more information about GUI scheme configuration, see Configuring GUI Scheme Definitions for the Alfabet Interface.

Usually, the styling of buttons is determined by the **Default Button** attribute of the button. If it is set to `True`, the **Primary Button Skin** style is used and if it is set to `False`, the **Secondary Button Skin** style is used. Please note that this setting is ignored for **Submit** buttons. **Submit** buttons are always displayed with the **Primary Button Skin** style.

The **Button** control can be used to re-create the **Submit** button if it has been deleted by mistake:

1) In the toolbar, click the **Button**  icon.
2) Drag the mouse on the filter panel in the report editor to create a box in the location and width of your choice.
3) In the attribute window of the new button, set the following attributes:
   - **Caption**: Enter the text that shall be displayed in the button.
   - **Hint**: Define a tooltip that is displayed for the button when the user moves the cursor over the button.
   - **Submit**: Select `True`.
4) In the toolbar on top of the attribute window, click the  button. The content of the attribute window changes to display available operations.

    To return to the attribute view, click the  button in the toolbar.
5) In the ***OnButtonClick*** field, select `OnClickSubmit` from the drop-down list.
6) In the toolbar, click the **Save**  button to save your changes.

Changing the Sequence of the Focus on Filter Fields

The user can use the TAB key to move the focus to another widget field, but by default, there is no order within the widget fields that is followed for each widget. You can define the sequence for moving the focus via the tab key:

1) Click anywhere in the filter panel.
2) In the toolbar, select **Format** > **Define Tab Order**. A blue box with a number is displayed in each filter field.
3) Click the blue boxes in the order that they should be sequenced.
4) In the toolbar, click the **Save** button to save your changes.
5) In the toolbar, select **Format** > **Define Tab Order** to remove the blue boxes from the panel.

Alternatively, the tab index of individual fields can be set via the attributes of the filter field. Please note that this alternative method is not available for filter fields in configured reports of the type `Query`.

1) Click the filter field that you want to set the tab order for to open the attribute window.
2) In the attribute window, set the following attribute:
   - **Tab Index**: Set the tab index of the field the user shall start browsing through the filter fields to 0. Set the **Tab Index** of the next filter field to 1. Increment the Tab Index by one for each sub-sequent filter field. The focus will move from the lowest number to the highest number following the order of numbers subsequently.
3) In the toolbar, click the **Save** button to save your changes.

Configuring the Complete Filter Area to be Collapsible

The complete filter area can be configured to be collapsible. The **Collapse Filter** option available in the **Options** menu on the top right side of the filter area allows users to close the filter area of the configured report or selector in order to increase the amount of space available to display the search results / report.

To configure the filter panel to be collapsible:

1) Open the attribute window of the filter panel:
   - For selectors: Click in the header panel of the graphic view to open its attribute window.
   - For reports: Click the parameter panel of the custom report view to open its attribute window.
2) In the **Collapsible** field, select `True` to implement the **Collapse Filter** button.

> The **Collapse Filter** option is automatically added to the **Options** menu when the **Clear Search Patterns** option is implemented. The setting of the **Collapsible** attribute to `True` is only required if the **Clear Search Patterns** option is not implemented, that means that the attribute **Has Clear Button** of the filter panel is set to `False`. The

Options menu is only available when either the **Clear Search Patterns** or the **Collapse Filter** option is implemented.

3) In the toolbar, click the **Save** button to save your changes.

Configuring Part of the Filter Area to be Collapsible

If some of the filter fields for a configured report should be set to meaningfully execute the configured report while others are only for advanced filtering, the filter panel can be split into a part that is always displayed and a part that will only be displayed if the user clicks a link. The user can collapse and expand the advanced filter fields according to demand.

To configure part of the filter to be collapsible, two panels must be added to the existing filter panel:

1) In the toolbar of the view editor, click the **Panel** icon.
2) Drag the mouse on the filter panel in the view editor to create a panel covering the top area of the filter panel in the size required for all filter fields that shall be always visible and the button to submit the filter values.
3) If filter fields and buttons have already been added to the filter panel prior to configuring part of the filter panel to be collapsible, move the filter fields and buttons that shall be always visible to the new panel.

   If you add a panel, it will be added on top of existing filter fields. To add the fields to the panel, change the location of the new panel in the mail filter panel area until you can see the filter fields, move them to the new panel and move the new panel back to the final location. for information about resizing and moving filter elements, see Changing the Size of a Single Filter Field and Changing the Location of a Single Filter Field.

   If the main filter area is left or right aligned and the **Options** menu shall be visible, the filter fields must be placed with enough spacing from top of the panel to allow the **Options** menu to be displayed.

4) Click into the new filter panel to open the attribute window of the new filter panel.
5) Set the **Dock** attribute to `Top`.
6) In the toolbar of the view editor, click the **Panel** icon.
7) Drag the mouse on the filter panel in the view editor to create a panel covering the lower area of the filter panel in the size required for all filter fields that shall be collapsible.
8) If filter fields and buttons have already been added to the filter panel prior to configuring part of the filter panel to be collapsible, move the filter fields that shall be hidden to the new panel.
9) Click into the filter panel that shall be collapsible to open the attribute window.
10) Set the **Dock** attribute to `Fill`.
11) Set the **Advanced Details Filter Panel** attribute to `True`.

    This attribute is only visible if the **Dock** attribute is set to `Fill`.

12) In the toolbar, click the **Save** button to save your changes.

Allowing the User to Clear the Filter Area

The filter panel can be configured to display a **Clear Search Patterns** option in the **Options** menu on the right side of the filter area. The **Clear Search Patterns** option allows users to clear all filter settings with a single click:

When the **Clear Search Pattern** option is activated, the setting of the **Collapsible** attribute of the filter panel is ignored and the **Collapse Filter** option is automatically added to the filter panel. It allows users to close the filter area of the selector in order to increase the amount of space available to display the search results/report. The **Options** menu is only available when either the **Clear Search Patterns** or the **Collapse Filter** option is implemented.

To include the **Clear Search Patterns** option in the **Options** menu of the filter panel of a configured report or custom selector:

1) Open the attribute window of the filter panel:
   - For custom selectors: Click in the header panel of the graphic view to open its attribute window.
   - For configured reports: Click the parameter panel of the custom report view to open its attribute window.
2) In the **Has Clear Button** field, select `True` to implement the **Clear Search Patterns** option.
3) In the toolbar, click the **Save** button to save your changes.

You can optionally configure individual filter fields to be excluded from the **Clear Search Patterns** action. To prevent a specified filter field to be cleared:

1) Click the filter field that you want to exclude from the **Clear Search Patterns** action to open its attribute window.
2) In the **Allow Clear Value** field, select **False** to prevent the field's value set by a user from being cleared when the user clicks the **Clear Search Patterns** option.

Changing the Position of the Filter Panel

By default, filter panels are added on top of a view for configured reports or selectors. The filter panel can optionally be moved to the left or right of the view:

1) In the view for a configured report or custom selector, click the filter panel.
2) In the attribute section of the filter panel, set the **Dock** attribute to `None`.
3) Click the ***Presentation*** object in the view.
4) In the attribute section of the ***Presentation*** object, set the **Dock** attribute to `None`.

5) Click the filter panel and pull the handles to change the size and position to include both the area the final filter field should cover and the position of existing buttons and filter fields.

6) If applicable, move existing filter fields and buttons to the correct position in the new filter field area.

> For information about moving filter elements, see Changing the Location of a Single Filter Field.

7) Click the filter panel and pull the handles to change the size and position the filter panel should have in the final view design.

8) In the attribute section of the filter panel, set the **Dock** attribute to `Left` for a filter panel positioned on the left of the view or to `Right` for a filter panel positioned on the right of the view.

9) Click the ***Presentation*** object in the view.

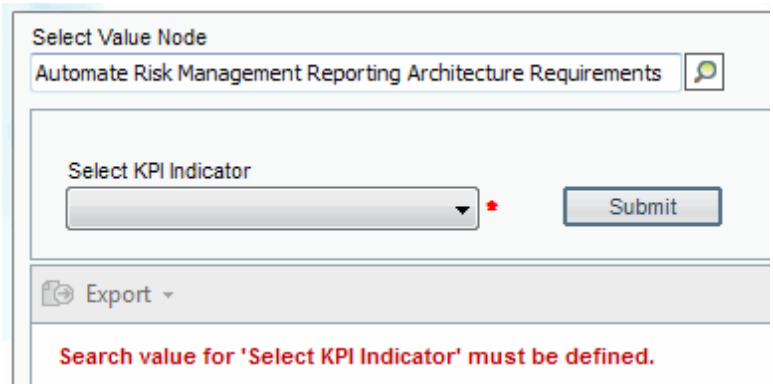
10) In the menu of Alfabet Expand, select **Edit** > **Cut**.

11) In the menu of Alfabet Expand, select **Edit** > **Paste** and click into the view area the ***Presentation*** object should be displayed in.

12) In the attribute section of the ***Presentation*** object, set the **Dock** attribute to `Fill`.

Defining Mandatory Filter Fields

Filter fields that were added manually to a report can be set to mandatory. The filter field is then marked with an asterisk. When the user clicks the **Submit** button without setting the filter field, a warning is displayed:



Please note the following:

- This feature is not available for automatically generated filter fields.
- Filters cannot be set to mandatory in console reports. The subordinate reports do not evaluate the filter field settings of the console report but read the stored filter data from a table in the Alfabet database.

To set a filter field to mandatory

1) Click the filter field that you want to edit.

2) In the attribute window, set the attribute **Presence** to `Always`.

3) In the toolbar, click the **Save**  button to save your changes.

Excluding Filters From Storage in Bookmarks

When a bookmark is set for a report, all filter settings are stored by default in the bookmark. When a user opens the report via the bookmark, the report opens with the filter settings that were selected when the bookmark was created. The filter field is displayed with the preset values in the filter fields.

A filter field can be excluded from being saved in bookmarks. A filter field that is excluded from the bookmark is not automatically filled with the settings stored in the bookmark. Instead, it is filled with the filter settings derived from the user context - in other words, with the values the user has entered in the field when he/she last executed the report. If no values have been stored in the user's personal settings, the field will be left empty when the user opens the report via the bookmark.

The user can change the settings of all filter fields to alter the predefined values for both filter fields included in bookmarks and excluded from bookmarks.

To exclude a filter field from storage in bookmarks:

1) Click the filter field that you want to edit.
2) In the attribute window, set the attribute **Save into Bookmark** to `False`.
3) In the toolbar, click the **Save**  button to save your changes.

For filter fields in Console Reports that are configured to store the user settings in an object of the object class `UserGlobalData`, the values currently stored for the properties of the `UserGlobalData` object are all stored in the bookmark independent from the setting for **Save into Bookmark**. For those filter fields, the values selected when creating the bookmark are reset when opening the bookmark even if **Save into Bookmark** is set to `False`. In addition, the values for all properties of the `UserGlobalData` object are overwritten with the data stored in the bookmark. That means that filter settings that have in the meantime been changed for other reports are also overwritten.

Excluding Filters from the Filter Summary in Exports

By default, a button **Export** is displayed in the toolbar of configured reports. Via the **Export** button the results currently displayed in the configured report can be exported for use outside of Alfabet. The available output formats depend on the type of configured report. When a configured report is exported via the **Export** functionality in Alfabet, the filter settings used to generate the exported report output can be included in the export.

For information on how to export configured reports, see the section *Exporting Data* in the reference manual *Getting Started with Alfabet*.

A filter field or the whole filter panel can be excluded from being displayed in the export.

To exclude a filter field from the exported filter summary:

1) Click the filter field that you want to edit.
2) In the attribute window, set the attribute **Summary Relevant** to `False`.
3) In the toolbar, click the **Save** button to save your changes.

To exclude all filter fields from the exported filter summary:

1) Click into the filter panel area.
2) In the attribute window, set the attribute **Summary Relevant** to `False`.
3) In the toolbar, click the **Save** button to save your changes.

Automatically Adding Wildcards to Search Strings

The setting of wildcards in filter fields depends on the configuration of the `AutoWildCard` element in the XML object ***SearchManager***. The search function can be configured to automatically add a wildcard before and after the search string entered in an edit field and Therefore, search for any string containing the search criteria. Alternatively, the search function can be configured to use the search string as explicitly entered in the field . In other words, the user entering the string in the edit field must add a wildcard to the string in the field if he/she wants to search for a string containing but not equal to the defined search string.

For more information about the configuration of the XML object ***SearchManager***, see Configuring the Wildcard for Standard and Custom Search Functionalities

If the XML object ***SearchManager*** is configured so that wildcards are automatically implemented, you can configure individual filter fields in a configured report or custom selector to be excluded from wildcard setting. Depending on the underlying query, the setting of wildcards might be undesirable if, For example, when comparing the entry in the filter field with a string using an equal to operator.

For example, a `WHERE` condition is defined that uses an equal to operator to compare the return value:

```
WHERE @Fieldvalue = 'Content'
```

When the user enters "`Content`" in the filter field and **Allow Wildcards** is set to `True`, the return string is "`%Content%`", which is not equal to "`Content`" and Therefore, the condition in the `WHERE` clause above is not fulfilled. You must either select a `LIKE` operator or include the wildcards in the compare value and define that the return value from the field must equal "`%Content%`". Alternatively, you can set **Allow Wildcards** to `False` for the filter field.

To exclude a filter field from automatic wildcard settings:

1) Click the filter field that you want to edit.
2) In the attribute window, set the attribute **Allow Wildcards** to `False`.
3) In the toolbar, click the **Save** button to save your changes.

Configuring Cascading Filters

In complex filters it can be useful to define the choice available to fill one control dependent on a selection already performed in another filter field. For example, a report that shows indicators for selected applications allows to select an application group in one filter field to show the indicators for all applications in the selected group and to select a single application in a second filter field to show the indicators of the selected application. The report can be configured to show For example, a drop-down list of available applications in the filter field for selection of the application. The content of the drop-down list depends on the settings performed for the selection of the application group. When the user has selected an application group, the drop-down list content is limited to the applications assigned to the selected application group.

When two filter fields are configured to be cascading, one must be defined as the master control that contains the content that the subordinate control depends on. Both master and subordinate control require a special configuration:

- The master control must be configured to submit changes when the user leaves the control after selecting/entering a search condition.
- The subordinate control must be configure to trigger a refresh of the field content on submit of changes of a defined master control.
- The query defined to find the objects displayed for selection in the subordinate filter field must define a dependency with the data from the master control.

Please note the following when defining cascading filters:

- The value type of the master control must be one of the following:
  - `Reference`
  - `ReferenceArray`
  - `String`
  - `Date`
  - `DateTime`
  - `Integer`
  - `Real`
- The following types of filter fields are not suitable for use as master control:
  - Check box
  - List box
  - Checked list box
  - Radio button group
  - Slider control
  - Value control

Edit fields can only be used as master controls if they are populated with a date.

- You can define more than one level of cascading filter fields. A filter field can depend on a master control and at the same time be the master control for another filter field.
- A filter can depend on multiple master controls. The query defined to find the objects displayed for selection in the subordinate filter field must then define dependencies with the data of all master controls.
- A filter field can be configured to depend on the base object of the report or selector instead of a master control. In that case, the query defining the filter field content must include the dependency to the current base object of the report.
- When defining cascading filters, make sure that no circular references occur between master and subordinate filter fields.

To configure an existing control as master control:

1) Click the filter field that you want to configure as master .
2) In the attribute window, edit the following attributes:
   - **Submit**: Select `True` to submit the current value when the user leaves the control after selecting/entering a search condition.
3) In the toolbar, click the **Save** button to save your changes.

To configure a control as subordinate control:

1) Click the filter field that you want to edit.
2) In the attribute window, edit the following attributes:
   - **Refresh on Submit**: Select `True` to re-evaluated the selection in the filter field each time the filter is submitted.
   - **Master Controls**: Click into the field and then click on the Browse button at the end of the field. A new window opens that lists all available controls. Select the checkbox of one or multiple controls to select them as master control fields. Click **OK** to save your selection.

If the control shall depend on the base object of the report or selector, the **Masters** attribute must remain undefined.

3) In the query defining the content of the filter field, define a dependency with the data from the master control. To refer to the content of the master control, the name of the master control must be used as parameter in a condition. The parameter returns the `REFSTR` of the object currently selected in the master control. To refer to the current base object of the report or selector, use the parameter `BASE`.

   The following example shows a native SQL query defined for a filter subordinate to the filter field `@APP` that allows to select an application:

```
SELECT COMP.REFSTR, COMP.NAME, COMP.VERSION FROM COMPONENT
COMP

INNER JOIN LOCALCOMPONENT AS LCOMP

   ON LCOMP.COMPONENT = COMP.REFSTR AND LCOMP.OWNER = @APP

/* Alfabet Instructions */
```



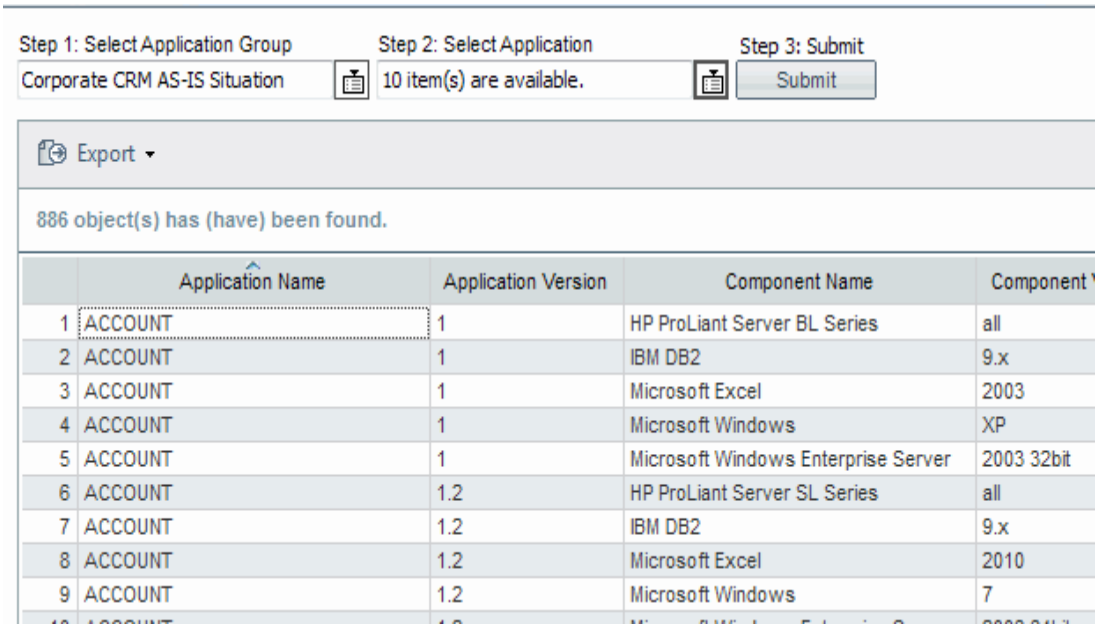
```
JoinColumns("COMP.NAME,COMP.VERSION", "COMP.NAME", " v.");
```

The following example shows an Alfabet query defined for a filter field that is filled with values dependent on the current base object. All applications in the application group the user is currently working with are listed in the selection:

```
ALFABET_QUERY_500

FIND Application

   WHERE Application.ApplicationGroups CONTAINS:BASE

Instructions

JoinColumns("Application.Name,Application.Version","Applicatio
n.Name", " v.");

EndOfInstructions

QUERY_XML

<QueryDef>

   <ShowProperty Type="Property" ClassName="Application"
   Name="Name" />

   <ShowProperty Type="Property" ClassName="Application"
   Name="Version" />

</QueryDef>
```

4) In the toolbar, click the **Save** button to save your changes.

In the following example, a tabular report shall display components that are assigned as local components to applications. Two filter fields of the type checked combo box for the selection of multiple values shall be available as a cascading filter. The user can first select one or multiple application groups to restrict the report results to local components of applications assigned to one of the selected application groups. A second checked combo box shall allow to further restrict display to a subset of the applications in the selected application groups. The content of the combo box for the selection of applications depends on the selection in the combo box for application groups.



	Application Name	Application Version	Component Name	Component
1	ACCOUNT	1	HP ProLiant Server BL Series	all
2	ACCOUNT	1	IBM DB2	9.x
3	ACCOUNT	1	Microsoft Excel	2003
4	ACCOUNT	1	Microsoft Windows	XP
5	ACCOUNT	1	Microsoft Windows Enterprise Server	2003 32bit
6	ACCOUNT	1.2	HP ProLiant Server SL Series	all
7	ACCOUNT	1.2	IBM DB2	9.x
8	ACCOUNT	1.2	Microsoft Excel	2010
9	ACCOUNT	1.2	Microsoft Windows	7

The definition of the report is described here both for an Alfabet query based report and for a native SQL based report in parallel.

Definition of the query for the report:

The query contains two parameters in the `WHERE` conditions, one that allows the user to restrict the report to data for selected application groups and one to restrict the report to data for selected applications in that application groups.

The report can be based on the following Alfabet query:

```
ALFABET_QUERY_500

FIND DISTINCT Component

InnerJoin LocalComponent ON LocalComponent.Component =
Component.REFSTR

InnerJoin Application ON LocalComponent.Owner = Application.REFSTR

InnerJoin ApplicationGroup ON Application.ApplicationGroups =
ApplicationGroup.REFSTR

WHERE

    (AND

       ApplicationGroup.REFSTR CONTAINSOR @APPG_REF

       Application.REFSTR CONTAINSOR @APP_REF

    )

QUERY_XML

<QueryDef>

<ShowProperty Type="Property" ClassName="Application" Name="Name" />

<ShowProperty Type="Property" ClassName="Application" Name="Version"
/>

<ShowProperty Type="Property" ClassName="Component" Name="Name" />
```



```
<ShowProperty Type="Property" ClassName="Component" Name="Version"
/>

<SortProperty Type="Property" ClassName="Application" Name="Name" />

<SortProperty Type="Property" ClassName="Application" Name="Version"
/>

<SortProperty Type="Property" ClassName="Component" Name="Name" />

<SortProperty Type="Property" ClassName="Component" Name="Version"
/>

</QueryDef>
```

In the `WHERE` clauses with the parameter the `CONTAINSOR` operator is used which means that the property is compared to a number of values from which one must apply. Therefore, when generating the custom report view, two filter fields of the type checked combo box for the selection of multiple values are created automatically.

The report can be based on the following native SQL query:

```
SELECT DISTINCT com.REFSTR, app.NAME As 'Application.Name',
app.VERSION As 'Application Version', com.NAME As 'Component Name',
com.VERSION As 'Component Version'

FROM COMPONENT com

INNER JOIN LOCALCOMPONENT lcom

   ON lcom.COMPONENT = com.REFSTR

INNER JOIN APPLICATION app

   ON lcom.OWNER = app.REFSTR

INNER JOIN RELATIONS rel

   ON rel.FROMREF = app.REFSTR

INNER JOIN APPLICATIONGROUP appg

   ON rel.TOREF = appg.REFSTR

WHERE rel.PROPERTY = 'ApplicationGroups'

   AND appg.REFSTR IN (@APPG_REF)

   AND app.REFSTR IN (@APP_REF)

ORDER BY app.NAME, app.VERSION, com.NAME, com.VERSION;
```

When generating the custom report view, combo boxes are automatically generated as filter fields. These are substituted by the corresponding checked combo boxes manually with the filter field attributes defined as described in the section Configuring Radio Button Groups, Combo-Boxes, Checked Combo-Boxes, List Boxes or Checked List Boxes. The following description of filter field configuration is limited to the query definition and the settings required for configuring the filter fields as cascading filter.

Definition of the master control:

The combo box for the selection of the application group is defined as master control, with the attribute **Submit** set to `True`.

The query defined for the combo box content searches for application groups. If a high number of application groups is defined, it is recommended to restrict the content of the combo box to a subset relevant for the user opening the report. In the example, a simple restriction to application groups with a name starting with '`Co%`' is added to the query.

In Alfabet query language, the following query is used:

```
ALFABET_QUERY_500

FIND ApplicationGroup

WHERE ApplicationGroup.Name LIKE 'Co%'

QUERY_XML

<QueryDef>

<ShowProperty Type="Property" ClassName="ApplicationGroup"
Name="Name" />

<SortProperty Type="Property" ClassName="ApplicationGroup"
Name="Name" />

</QueryDef>
```

In native SQL, the following query is used:

```
SELECT appg.REFSTR, appg.NAME

FROM APPLICATIONGROUP appg

WHERE appg.NAME LIKE 'Co%'

ORDER BY appg.NAME
```

Definition of the subordinate control:

The combo box for the selection of the applications is defined as subordinate control of the combo box for the selection of the application groups. The **Refresh on Submit** attributeis set to `True` and the **Master Controls** attribute defines the filter field `@APPG_REF` as master.

The query defined for the combo box content searches for all applications that are assigned to one of the selected first level application groups. An application can be assigned to multiple application groups. The assignment to application groups is stored via the property `ApplicationGroups` as a reference array, that means that the relations are stored in the relation table. In the Alfabet query language, it is nevertheless possible to define a simple `WHERE` clause that directly compares each assignment of an application group via the `ApplicationGroups` property with the values selected in the master control using a `CONTAINSOR` operator. The condition returns true if the application is assigned to at least one of the selected application groups:

```
ALFABET_QUERY_500

FIND Application

WHERE Application.ApplicationGroups CONTAINSOR @APPG_REF

QUERY_XML

<QueryDef>

<ShowProperty Type="Property" ClassName="Application" Name="Name" />

<ShowProperty Type="Property" ClassName="Application" Name="Version"
/>

<SortProperty Type="Property" ClassName="Application" Name="Name" />

<SortProperty Type="Property" ClassName="Application" Name="Version"
/>

</QueryDef>
```

In native SQL, the query must define the condition via a `JOIN` with the relations table, comparing the property `TOREF` of the relation with the values returned from the master control:


```
SELECT app.REFSTR, app.NAME, app.VERSION
FROM APPLICATION app
INNER JOIN RELATIONS rel
   ON rel.FROMREF = app.REFSTR
WHERE rel.PROPERTY = 'ApplicationGroups'
   AND rel.TOREF IN (@APPG_REF)
ORDER BY app.NAME, app.VERSION
```

Configuring Reusability of Filter Settings Across Reports

In Alfabet, filter settings done by the user in the report can be stored and re-used via the following mechanisms:

- For all filters in Alfabet, independent of whether they are standard or customer defined, the filter settings of the user are stored and when the user re-opens the report or selector at a later time, the report or selector opens with the stored filter settings.

Note the following regarding this mechanism:

  - The mechanism stores the filter settings in the context of the current user, the current user profile and the current report. When the same user logs in with different user profiles, the same report can open with different preset filter settings. Also, if a user was searching for an application with a defined name in one report, all other reports that include a filter searching for applications by name do not take over the filter setting.
  - Each time the user alters the filter settings, the stored settings are overwritten with the new settings and these are used when the user re-opens the report the next time.
  - The filter settings are stored either on submitting the report or on navigation to an object from the report. When a user alters the filter settings, but prior to re-submitting the new settings, decides to navigate to an object from the report, the altered settings are nevertheless stored and used to open the report when the user navigates back to the report.

It is recommended not to execute configured reports with filter fields storing the filter settings into the `UserGlobalData` object class offline. On rendering of the offline executed report results, the values stored in the `UserGlobalData` object class will be overwritten with the filter values defined on triggering the report execution of the current report results. The opening of offline executed report results in a new tab might change a filter setting depending on the same object of the `UserGlobalData` object class for a report opened in parallel in the original tab for working with Alfabet without further notice.

For more information about offline execution of long running configured reports, see Configuring Offline Execution for Long Running Tabular Reports with Filters.

- Filter settings made by a user in one configured report can be stored in the Alfabet database and referenced in `WHERE` conditions of queries in other reports to open one report dependent on the search performed in another report. A typical use case for this configuration is the design of a configured console report. A console report is a multi part report that allows to display results for multiple configured reports in one view. A filter can be defined for the console report. If filter fields of the filter in the console report are configured to be stored in the database, the reports displayed as

sub-ordinate reports can be configured to take over the filter settings from the database and open with the same context.

For example, a console report displays multiple reports about applications. In the filter, the user defines that only applications with a start date after a defined date are relevant. The sub-ordinate reports take over the filter settings and only display results for applications with a start date in the relevant period.

Please note the following:

- This mechanism is not available for filters defined in custom selectors or in configured reports of the type `Query`.
- Filters cannot be set to mandatory in console reports. The subordinate reports do not evaluate the filter field settings of the console report but read the stored filter data from a table in the Alfabet database.

The reuse of filter settings across reports requires a configuration of the meta-model and the filter in the configured report. It is based on the class `UserGlobalData`, that stores filter settings per filter field and user. The object class has one protected object property, `USERREF` that establishes a link to the object class Person.

An object of the object class `UserGlobalData` is automatically created the first time a new user logs in to Alfabet and the `USERREF` property is set to the `REFSTR` of the object of the object class Person that stores the data of that user.

To store filter settings in the class `UserGlobalData`, the solution designer must define the following:

- A custom property must be added to the object class `UserGlobalData` for each filter field for that the settings shall be applied to other reports.
- The configured report that includes the filter fields must be configured to find the instance of the object class `UserGlobalData` that is assigned to the current user.
- The filter fields of the filter for that the settings shall be re-used must be configured to store the settings performed by the user into the respective custom attribute of the `UserGlobalData` object of the user performing the setting on submit of the filter settings.

  The configuration of filter fields to store settings in `UserGlobalData` is not possible in configured reports of the type `Query` and custom selectors.

  If filter fields in a filter are defined to store values in `UserGlobalData`, all filter settings are submitted to the respective objects of the object class `UserGlobalData` even if only one or none of the filter settings has been changed.

- Reports that shall show results dependent on the settings from another report stored in the `UserGlobalData` objects must be configured to refer to values of the custom properties of the `UserGlobalData` object of the current user.

The following sections provide a detailed step by step instruction on how to define filter setting reusability:

- Configuring the Object Class UserGlobalData to Store Filter Settings
- Configuring the Configured Report to Store Filter Settings in UserGlobalData
- Configuring Filter Fields for Storage of Settings in UserGlobalData Properties

- Configuring Queries in Reports Referring To Global Filter Settings

Configuring the Object Class UserGlobalData to Store Filter Settings

The **Audit** attribute must be set to `False` for the object class `UserGlobalData`. Activation of auditing for the object class `UserGlobalData` significantly lowers performance.

To configure the object class `UserGlobalData` to store a filter setting:

1) In the **Meta Model** tab of Alfabet Expand, expand the **Classes** folder.
2) Right-click the object class `UserGlobalData` in the explorer and select **Add New Property**. The **Create New Property** editor opens. Define the following fields:
   - In the **Property Name** field, enter a unique name for the custom property. This value is used in the configuration of configured reports and custom selectors to refer to the current setting of a filter.

     A validation mechanism checks for correct syntax when defining a technical name. The technical name of a configuration object may not begin with an empty space nor include any of the following special characters: **\ / * ? " > < | ':**

     Furthermore, please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.

     Furthermore, if the technical name of a configuration object is changed, the name will be correctly updated in other configuration objects referencing the changed object during design time. Please note that the name of a changed configuration object will not be updated in guide pages nor if the guide pages reference a configuration object listed in the **Show Usage** functionality. If you plan to change the name of a configuration object, the reference in the guide page should be changed prior to changing the configuration object name.

   - In the **Tech Name** field, enter a unique technical name for the custom property that will be used in the database table. Click **OK** to save the definition.

     Please keep the following in mind:

     - The technical name may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.
     - The technical name should only contain standard `ASCII` characters.
     - The technical name should consist of upper-case letters only.
     - The maximum length of a technical name may not exceed 30 characters.
     - The technical name may not coincide with any of the reserved key words of the relational database management system.

A validation mechanism checks for correct syntax when defining a technical name. Please note that if the technical name of a configuration object is changed, the name will be correctly updated in other configuration objects referencing the changed object during design time. However, the name of a changed configuration object will not be updated in guide pages nor if the guide pages reference a configuration object listed in the **Show Usage** functionality. If you plan to change the name of a configuration object, the reference in the guide page should be changed prior to changing the configuration object name.

3) Click the custom property in the tree. You will see the attribute window in which you can define the custom property's attributes.

4) The custom property stores values added to a filter field. Filter fields allow to select values for a defined object class property from the Alfabet meta model. The data type of that object class property must be identical to the data type defined for the custom property of `UserGlobalData` storing the value. Therefore, set the following attributes to the values also defined for the object class property your filter field will target:

   - **Property Type**
   - **Type Info**
   - **Reference Support**

5) In the toolbar, click the **Save**  button to save the new custom property.

Configuring the Configured Report to Store Filter Settings in UserGlobalData

A configured report that shall store filter settings made in the filter setting of the configured report into the properties of the object class UserGlobalData must be configured to find the object of the class UserGlobalData referencing the user performing the filter settings as base class:

1) In the explorer of the ***Reports*** tab in Alfabet Expand, click the configured report for that you want to define reusable filter settings.

2) In the attribute window of the configured report, make sure that the **Apply to Class** attribute is not set.

3) In the **Base Object Query** attribute field, click the **Browse**  button. A text editor opens.

4) In the editor, enter a query finding the `UserGlobalData` object referencing the current user working with the configured report.

   You can use the following query in Alfabet query language:

   ```
   ALFABET_QUERY_500
   FIND UserGlobalData
   WHERE UserGlobalData.USERREF = @CURRENT_USER
   ```

   or in native SQL:

   ```
   SELECT REFSTR
   FROM USERGLOBALDATA
   WHERE USERREF = @CURRENT_USER
   ```


Configuring Filter Fields for Storage of Settings in UserGlobalData Properties

In addition to the settings required for the basic configuration of a filter field for a defined data type, the following configuration must be performed for each filter field for that the settings shall be stored in a custom property of the object class `UserGlobalData`:

1) In the explorer of the ***Reports*** tab in Alfabet Expand, click the custom report view for that the filter shall be defined.
2) In the attribute window of the custom report view, select `UserGlobalData` from the drop-down list in the **Class Name** field.
3) Double-click the custom report view to open the view.
4) In the editor area, click the filter control that you want to edit.
5) In the attribute window of the filter control, select the custom property of the object class `UserGlobalData` that is configured to store the filter value from the drop-down list in the attribute **Property**.
6) In the toolbar, click the **Save**  button to save your changes.

Configuring Queries in Reports Referring To Global Filter Settings

Note the following for the definition of the queries directly defined in the report containing the filters that are configured to store data in the object class `UserGlobalData`:

- The object class `UserGlobalData` must be defined as base class of the report with the attribute **Apply To Class** of the report. Therefore, the queries in the underlying report cannot refer to the object with that the user is currently working with the parameter `BASE`.
- The `WHERE` conditions in the query can refer to the filter fields by using the filter field name as parameter in the query independent from the storage of data into the `UserGlobalData` object class.

When filter settings of a report are stored in the object class `UserGlobalData`, all configured reports can refer to the filter settings by defining a JOIN to the `UserGlobalData` element of the current user and comparing the respective object data with the value stored in the `UserGlobalData` object.

The **Test Report** functionality in the context menu of the explorer cannot be used if a filter stores data in the object class UserGlobalData and the report reads the filter settings from the properties of UserGlobalData. Changing the filter settings in the test window will not update the reports that read data from `UserGlobalData`, because the test functionality will not change data in the Alfabet database. To test the report, set the **State** of the report to `Active`, open the Alfabet user interface and open the report via the **Reports** functionality. If the results are not as expected, set the **State** of the report back to `Plan` and edit the report until the output of the report meets your expectations.

Using Instructions to Format the Results of an Alfabet or Native SQL Query

Instructions are available in the Alfabet query language in order to allow solution designers to alter the display of datasets resulting from Alfabet queries or native SQL queries. Instructions can be used in the Alfabet query language and can be combined with native SQL.

The instructions must be defined in Alfabet query language even if the query itself is written in native SQL. If instructions may be combined with native SQL, a tab for the definition of instructions will be available in text editors and the report assistant. If a tab is not available to define instructions, then it is not permissible to define instructions with native SQL for the respective configuration.

Specifying an Alfabet Instruction in an Alfabet or Native SQL Query

When writing an Alfabet query directly in a text editor, all instructions are written in a section that starts with `Instructions` and ends with `EndOfInstructions`:

```
Instructions
any of the instructions specified below;
any of the instructions specified below;
EndOfInstructions
```

When writing a native SQL query directly in a text editor, the instructions must be separated from the native SQL query with the comment `/* Alfabet Instructions */`:

```
/* Alfabet Instructions */
any of the instructions specified below;
any of the instructions specified below;
```

However, in the **Alfabet Query Builder** and in the tabs for instructions definition in combination with native SQL, the instructions are defined without the delimiters `Instructions` and `EndOfInstructions`.

Please consider the following when defining instructions:

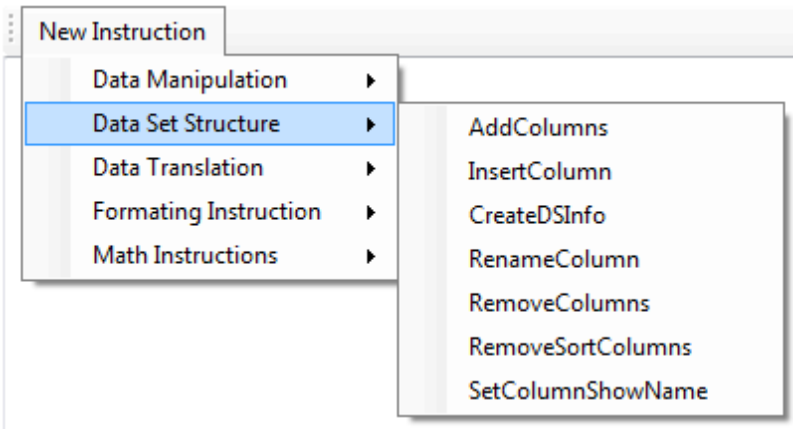
- To define instructions, the Show properties of the Alfabet query must be defined as an XML element. When using the **Alfabet Query Builder**, this is automatically done.
- It is not allowed to use whitespaces in the instruction outside of string specification of the delimiter.
- It is not allowed to use line breaks within the instruction.
- Each instruction must end with a semi-colon.
- You can specify multiple instructions in the same `Instructions` section.
- When defining instructions, you must specify column names to identify the columns that the instruction applies to. For information about the column names resulting from Alfabet query and native SQL query definitions, see the section Defining Column Names and Captions. Alternatively, instructions can refer to columns by their index number. For more information see Referencing Columns in Instructions by Index Number.
- Instructions are executed in the order that they are specified. This can be crucial for the instruction definition (For example, if an instruction refers to a column that has been renamed by an instruction that is executed earlier). An exception to this rule is the coloring of cells in the report via `ColorAssignment` and `RowColorAssignment` instructions. The `ColorAssignment` instruction supersedes the `RowColorAssignment` independent from the order of specification. If you define the `GroupBy_Ex` instruction in combination with instructions that are assigned to rows in a report, For example, to define link assignment or coloring of cells or rows, the `GroupBy_Ex` instruction must be defined first and the instruction for formatting of the rows must defined after the `GroupBy_Ex` instruction.

When defining native SQL queries in the context of the Alfabet configuration, the first column of a result set is not displayed in the query results. This applies to the result set after execution of the Alfabet instructions. The following sequence of execution of a native SQL query must be taken into account when defining the native SQL query:

1) Alfabet -specific parameters in the Alfabet query are translated to corresponding native SQL code and a result set is created.
2) Alfabet instructions are executed on the result set. This may influence the availability and display of columns in the result table.
3) The first column in the result table after the execution of instructions is not displayed.

To specify instructions in the **Alfabet Query Builder** or the **Instructions** tab of the SQL query editor of Alfabet Expand:

1) Go to the **Instructions** tab.
2) Click the **New Instruction** button, select the sub-menu containing the instruction and select the instruction from the drop-down list.



3) Either an example instruction will be written in the text editor or an editor will open. If the example is written in the text editor, change the example as needed. If an editor opens, enter the required values in the editor controls. For further information about the individual instructions that are available, see the relevant table row:

Instruction	Description	Described in Section...
Data Manipulation		
`GroupBy`	Allows an expandable report table by grouping results to be built.	Grouping Query Results in Expandable Reports
`JoinColumns`	Displays multiple Show properties in one row of the report.	Aggregating Multiple Query Results in One Column of a Configured Report
`CreateRefImage`	This instruction is for use in configured Data Table reports only. It adds the ability to navigate to the object	Configuring Cells in a Data Table Report to

Instruction	Description	Described in Section...
	profile of an object represented by a cell to all cells of a defined column in the configured report.	Provide Navigation to an Object Profile
RemoveEmptyRows	Removes empty rows from the report table. This instruction is only required if the report is an expandable report table built with the GROUPBY_EX instructions and if the root level of the report shows objects with no subordinate search results.	Grouping Query Results in Expandable Reports
SetClassCaption	Displays the class caption of the object class of the displayed objects in a defined column of the report. The REFSTR of the objects must be included in the query result dataset to execute the SetClassCaption instruction.	Displaying the Class Name and/or Class Caption of an Object Class That Is Not the Base Class
GetClassSettingInfo	Displays the name of the object class icon and the foreground and background color defined for an object class in the relevant class settings in defined columns of the report.	Inserting Columns with Information About the Class Icon And Class Colors
SetClassCaptionByRow	Displays the class caption of the base object class in the report in a defined column of the report.	Displaying the Class Name and/or Class Caption of the Base Object Class
SetClassNameByRow	Displays the class caption of the base object class in the report in a defined column of the report.	Displaying the Class Name and/or Class Caption of the Base Object Class
SetClassName	Displays the class name of the object class of the displayed objects in a defined column of the report. The REFSTR of the objects must be included into the query result dataset to execute the SetClassName instruction.	Displaying the Class Name and/or Class Caption of an Object Class That Is Not the Base Class
SetRowCategories	Creates categories based on values in a column of the result data set. The categories can be used to configure buttons in the toolbar of the report to be active for objects belonging to defined categories only.	Limiting Toolbar Button Functionality To a Subset of Objects in a Configured Report
SetRowReference	Defines an object class that was added to the result data set via a JOIN to be processed as the base object class of the result data set. For example, the **Navigate** button directs the user to the object profile of the	Changing the Navigation Behavior of the Base

Instruction	Description	Described in Section...
	selected class instead of the object profile of the `FIND` object class.	Object Class in a Configured Report
`SetStereotypeCaption`	Displays the caption of the object class stereotype of the displayed objects in a defined column of the report. Alternatively, the stereotype icon can be displayed next to or instead of the caption. The dataset resulting from the query must include the information about the stereotype name to execute the `SetStereotypeCaption` instruction.	Displaying the Stereotype Caption and/or Icon of Objects Displayed in the Report
`SetStereotypeIndex`	Defines the column in a dataset that lists the stereotype of an object. This instruction is exclusively used in custom selectors to create selectors that display only objects of a defined stereotype. When the search results are displayed in the explorer, the information provided in the column defined in the instruction is used to filter results by stereotype.	Enabling a Stereotype-Specific Search in Custom Selectors
`SortBy`	Defines the sorting of rows in the report overwriting the sort behavior determined by the database. This instruction is required to sort values in a column that is the result of a `JoinColumns` instruction and the column displays objects from multiple object classes.	Aggregating Multiple Query Results in One Column of a Configured Report
`ReplaceServerVariable`	Replaces the content of a defined column with the value of a server variable defined in the server alias of the Alfabet Web Application. For more information about the definition of server variables, see Using Server Variables in Web Link and Database Server-Related Specifications.	Adding the Value of a Server Variable to the Dataset
`ReplaceValues`	Replaces defined strings in the cells of a column in the dataset with replacement values defined per original string in the instruction. Only complete return values of the type string or text are replaced.	Replacing Values Returned in the Query with a Defined String
`RetrieveIDOCPath`	Triggers export of attachments specified in the result dataset to a temporary runtime folder. The path information is written into the result dataset.	Exporting Attachments to a Runtime Folder During Report Execution
`JoinURLLink`	If a configured report contains a link to an URL, this instruction can be used to display any text instead of the target URL specification as link text.	Masking a Link Target in A Configured Report With a Text

Instruction	Description	Described in Section...
Data Set Structure		
`AddColumns`	Adds multiple columns to the dataset. The instruction must be combined with a `SetColumnShowName` instruction or a `CreateDSInfo` command to display the columns in the report output.	Adding Multiple Columns with the AddColumns Instruction
`InsertColumn`	Inserts a single column to the dataset. The instruction must be combined with a `SetColumnShowName` instruction or a `CreateDSInfo` command to display the columns in the report output.	Adding A Single Column With the InsertColumn Instruction
`CreateDSInfo`	Redefines the complete dataset displayed to the user on the Alfabet interface when opening the report. The number and order of columns, and the column captions in the report are re-configured.	Changing the Order and Number of Columns Displayed in a Report
`RenameColumn`	Changes the column name.	Defining Column Names and Captions
`RemoveColumns`	Removes defined Show properties from the report. Some instructions require the setting of defined Show properties for technical reasons. The `RemoveColumns` command is designed to hide these technical show properties from the result table.	Hiding a Column in a Configured Report
`RemoveSortColumns`	Prevents sorting by the user for columns in the report. This instruction can only be used for reports based on an Alfabet query that display a grouped data set.	Preventing Users from Sorting According to Subordinate Objects in the section Aggregating Multiple Query Results in One Column of a Configured Report
`SetColumnShowName`	Changes the column caption after execution of instructions. This instruction is designed for use in native SQL queries. It is not required in Alfabet queries.	Defining Column Names and Captions

Instruction	Description	Described in Section...
`SetFreezeHeader`	Freezes one or multiple columns in a tabular dataset of a configured report. These columns will then stay visible during horizontal scrolling.	Freezing Columns in Tabular Datasets
Data Translation		
`TranslateEnums`	Enumeration, object state, and status values are translatable in Alfabet, but the translated values are not used per default in configured reports. The instruction triggers the translation of enumerations, object states, and statuses for defined columns of the configured report. The values are then displayed in the translation for the culture selected by the user viewing the configured report.	Translating Enumerations, Object States, Release Statuses, and Indicator Values in a Configured Report
`TranslateIndicators`	Indicator range values are translatable in Alfabet, but the translated values are not used in configured reports by default. The instruction triggers translation of indicator range values for defined columns of the configured report. The values are then displayed in the translation for the culture selected by the user viewing the configured report.	Translating Enumerations, Object States, Release Statuses, and Indicator Values in a Configured Report
`TranslateTimeStatus`	lifecycle status values are translatable in Alfabet, but the translated values are not used in configured reports by default. The instruction triggers translation of lifecycle status values for defined columns of the configured report. The values are then displayed in the translation for the culture selected by the user viewing the configured report.	Translating Enumerations, Object States, Release Statuses, and Indicator Values in a Configured Report
`SetTranslatedValue`	For object class properties for which data translation is enabled in Alfabet, translated values are automatically displayed in reports based on an Alfabet query in the language currently used to display the user interface. For native SQL queries the change to the current language must be included in the report with the instruction `SetTranslatedValue`.	Display Translated Values for Object Class Properties Subject to Data Translation in Configured Reports Based on Native SQL
Formating Instruction		
`CellColorAssignment`	Defines the coloring of the cells in a result table for cells matching a defined search condition.	Defining Coloring for Cells or Rows Containing Specific Data

Instruction	Description	Described in Section...
`Convert2Boolean`	Displays `String`, `Integer` and `Real` values as `Boolean` in the report.	Displaying String, Integer or Real Values as Boolean Values
`ConvertBoolean2String`	Displays defined strings instead of `True` or `False` for `Boolean` values in the report.	Changing the Output Format for Boolean Values
`Convert2Posix`	Displays `Date` and `DateTime` values in POSIX format.	Specification of Dates
`LinkAssignment`	Adds a link to a view in Alfabet to each cell of a defined column of the report. The link target can be an object profile, an object cockpit, a graphic view or a configured report.	Opening the Same Page View, Object Profile, Object Cockpit, or Configured Report for all Search Results
`EditLinkAssignment`	Adds a link to an editor or wizard to each cell of a defined column of the report. The editor can be opened at a defined tab and the wizard can be opened at a defined wizard step.	Opening the Same Wizard or Editor for all Search Results
`DynamicLinkAssignment`	Adds a link to a view, editor or wizard to each cell of a defined column of the report. The information which view, editor or wizard shall open is dynamically defined via the query the instruction is defined for.	Opening Diffent Views, Wizards, or Editors for Search Results
`PictureAssignment`	Defines a substitution or amendment of the return value in a cell with an icon from the icon gallery when the cell matches a defined search condition. The icon can be 22x22, 30x30, or an icon in the the free size icon gallery.	Displaying Graphics for Cells Containing Specific Data
`RowColorAssignment`	Defines the coloring of the rows in a result table for the cell in a defined column matching a defined search condition.	Defining Coloring for Cells or Rows Containing Specific Data
`SetColumnDateSubType`	Defines whether information of the data type `DateTime` is displayed as date only or date and time without defining an output format. The output format is then defined by the language settings of the user interface.	Changing the Output Format for Date and Time Information

Instruction	Description	Described in Section...
SetColumnFormat	Sets the output format for date and time information to a defined format. This format is used independent from the language settings of the user interface.	Changing the Output Format for Date and Time Information
SetColumnsAlignment	Displays content of cells in defined columns in the result table with a different alignment as the default. The cell content can be defined as left, right, or center aligned.	Changing the Alignment of Cell Content
FontStyleAssignment	Defines the formatting of the text in the cells in a result table for cells matching a defined search condition. Text can be formatted as italic, bold or underlined.	Defining Text Formatting for Cells Containing Specific Data
FontStyleColorAssignment	Defines the formatting of the text as well as the coloring of the cells in a result table for cells matching a defined search condition. Text can be formatted as italic, bold or underlined.	Defining Text Formatting for Cells Containing Specific Data
SetDynamicWebLink	Provides the ability to open a dynamic web link for an object selected in the configured report via a button interaction.	Adding Dynamic Web Links to a Configured Report
SetObjectIcon	Display the icons assigned to objects in a cell of a tabular configured report.	Displaying the Object Icon
Math Instructions		
CreateColumnSum	For a defined column of the data type Integer or Real all values in the query output are counted up. The resulting sum is displayed in an additional row added at the bottom of the data set. Optionally, a caption for the additional row can be defined for display in one of the other columns of the data set.	Displaying the Sum of All Values in a Table Column
CreateRowSum	Calculates the sum per row of multiple values of the type Real or Integer displayed in selected columns of the dataset. A new column must be added to the dataset for display of the result.	Displaying the Sum of Multiple Values in a Table Row

Instruction	Description	Described in Section...
`NTile_Weighted`	This instruction groups the results in the dataset according to the ascending order of values in a defined column of the data type `Integer` or `Real`. The number of groups can be defined in the instruction. The groups are represented by numbers, with 1 being the group with the lowest values. Each group is assigned the same number of results. If the number of results cannot be divided equally among the groups, the first group(s) have a higher count. A new column of the data type Integer or Real must be added to the data set to display the results.	Displaying the Sum of All Values in a Table Column

Defining Column Names and Captions

The name of the database column is used to identify the column results in technical processes. This is relevant, For example, for the following:

- to specify the column within an instruction.
- to enable the **Alfabet Report Assistant** (used for the creation of chart reports) to identify the columns used to display the X-axis or Y-axis of the report.

The caption of the database column is the information that is displayed as column header in the tabular output of the query.

When you define a simple Alfabet query or a native SQL query, the name of a database column and the caption displayed in the report are identical. This chapter provides information about the standard column names and captions resulting from query definitions and the methods available to alter the names with instructions.

Please note the following about the definition of column names and captions:

- The special characters # and: as well as commas (,) are not allowed in column names.
- Tabular reports in Alfabet have a fixed row high for the header cells. If the column caption is longer than the width of the header cell, a line break is added when the width of the header cell is reached, but only the first line of text is displayed in the header cell.
- The strings on the Alfabet user interface can be translated and the user interface can be rendered in other language cultures than the original English (USA) language culture. When query results are displayed on the Alfabet user interface in a translated version, column captions are only displayed translated if one of the following applies:
  - The string of the column caption is available in the standard vocabulary, For example, because it is identical to an object class property caption.
  - the string of the column caption has been added manually to the vocabulary by writing it into the XML object ***VocXML***. For more information about the translation of column captions in query results see Translating Configured Reports.

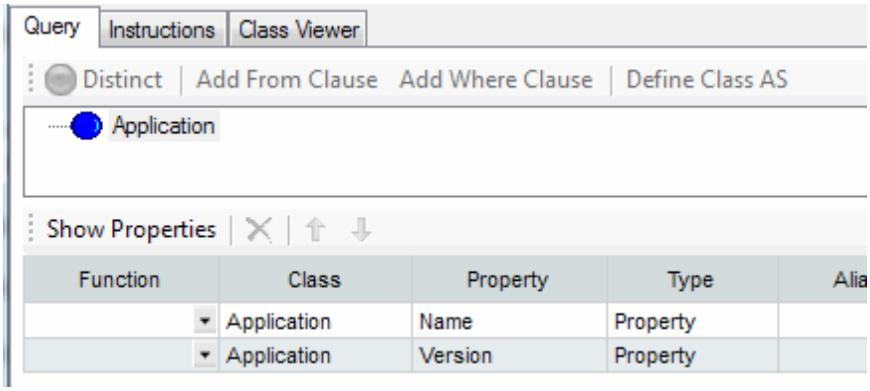
The following information is available:

- Defining Column Names and Captions in Alfabet Queries
- Defining Column Names and Captions in Native SQL Queries
- Defining Column Names and Captions with Instructions
- Referencing Columns in Instructions by Index Number

Defining Column Names and Captions in Alfabet Queries

In the Alfabet query language, the columns of the resulting data set are defined via the Show properties. The standard name of a column resulting from an Alfabet query is "ClassName.PropertyName" and the standard caption is "ClassName PropertyName".

A simple Alfabet query displays the name and the version of applications:



In the **Alfabet Query Builder**, you can see the object class name in the **Show Property** section in the column **Class** and the value of the property name in the column **Property**. In the Show properties of the resulting Alfabet query, the object class name is defined with the attribute ***ClassName*** and the object property name with the attribute ***Name***.

```
ALFABET_QUERY_500
FIND
Application
QUERY_XML
<QueryDef>
<ShowProperty Type="Property" ClassName="Application" Name="Name" />
<ShowProperty Type="Property" ClassName="Application" Name="Version"
/>
</QueryDef>
```

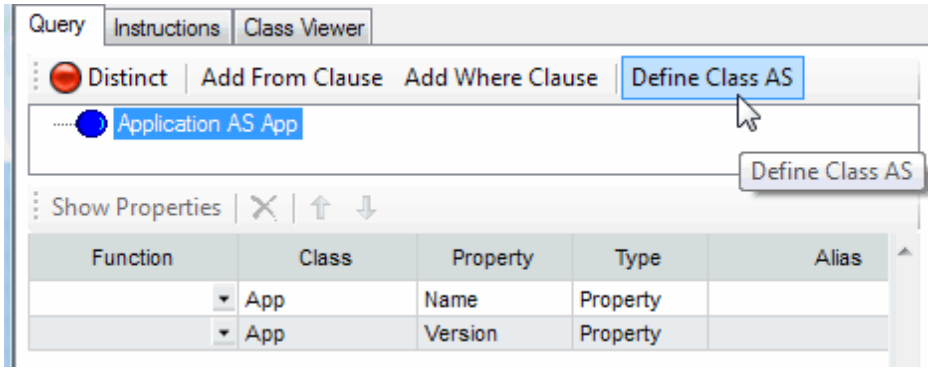
The resulting column names and captions are:

Column Name	Column Caption
Application.Name	Application Name
Application.Version	Application Version

The column caption is concatenated from the caption of the object class and the caption of the object class property. Usually these strings are available in all language cultures and are automatically displayed in the language used to render the user interface when displaying the query results. Please note however that the class name and the property name are translated separately and not as a combined string which may lead to differen translation results in some languages.

You can set an alias for the class name in the Alfabet query both for the `FIND` class and for classes added to the class via a `JOIN`. This alias changes the name, but not the caption of the column.

The alias `App` is defined for the object class `Application` in the Alfabet query:



In the **Alfabet Query Builder**, you can see the value of the object class alias in the **Show Property** section in the column **Class** and the value of the property name in the column **Property**. In the Show properties of the resulting Alfabet query, the object class alias is defined with the attribute ***ClassAlias*** and the property name with the attribute ***Name***.

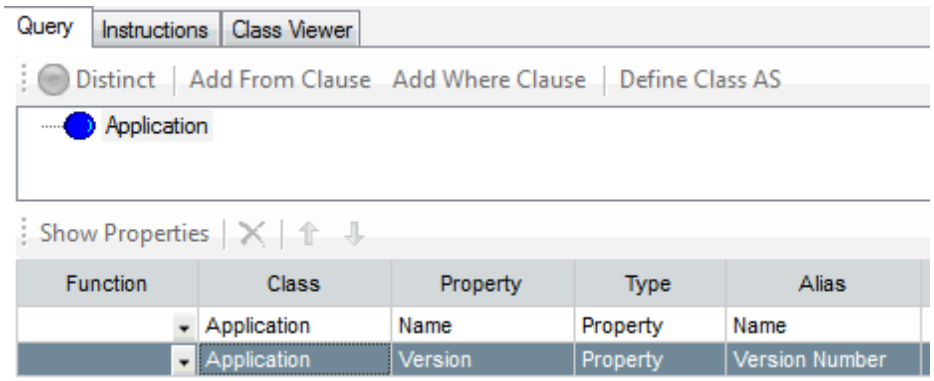
```
ALFABET_QUERY_500

FIND

Application AS App

QUERY_XML

<QueryDef>

<ShowProperty Type="Property" ClassName="Application"
ClassAlias="App" Name="Name" />

<ShowProperty Type="Property" ClassName="Application"
ClassAlias="App" Name="Version" />

</QueryDef>
```

The resulting column names and captions are:

Column Name	Column Caption
App.Name	Application Name
App.Version	Application Version

You can set an alias for a column in the Show properties. This alias does not change the name, but only the caption of the database column.

An alias is defined for the columns in the Show properties of the Alfabet query defined to find applications:



In the **Alfabet Query Builder**, the alias is defined in the **Show Property** section in the column **Alias**. In the Show properties of the resulting Alfabet query, the object class alias is defined with the attribute ***Alias***.

```
ALFABET_QUERY_500
FIND
Application
QUERY_XML
<QueryDef>
<ShowProperty Type="Property" ClassName="Application" Name="Name"
ShowName="Name" />
<ShowProperty Type="Property" ClassName="Application" Name="Version"
ShowName="Version Number" />
</QueryDef>
```

The resulting column names and captions are:

Column Name	Column Caption
Application.Name	Name

Application.Version	Version Number

Defining Column Names and Captions in Native SQL Queries

When you define a native SQL query, the column names and captions are derived from the `SELECT` clause. Column names and captions are identical.

When no alias is specified for a column header in the `SELECT` clause, the property name is used as a column header.

In native SQL queries, names of database columns and database tables in the Alfabet database must be written in upper-case letters.

The following native SQL query finds applications and returns the properties `Name` and `Version`:

```
SELECT APPLICATION.REFSTR, APPLICATION.NAME, APPLICATION.VERSION
FROM APPLICATION
```

The resulting column names and captions are:

Column Name	Column Caption
NAME	NAME
VERSION	VERSION

If two properties have the same name, the column headers in native SQL have the same name too. When specifying native SQL in the context of Alfabet, two column headers with the same name are not allowed. It is recommended that you specify an alias name for the columns. If you do not specify an alias, the first column is given the name of the property and the following column headers are named <property name><number>.

A native SQL query finds applications and returns the name and version of the application and the name of the ICT object the application is assigned to:

```
SELECT APPLICATION.REFSTR, APPLICATION.NAME, APPLICATION.VERSION,
ICTOBJECT.NAME
FROM APPLICATION, ICTOBJECT
WHERE APPLICATION.ICTOBJECT = ICTOBJECT.REFSTR
```

The resulting column names and captions are:

Column Name	Column Caption
NAME	NAME
VERSION	VERSION
NAME1	NAME1

Setting an alias in the SELECT clause not only changes the caption but also changes the name of the database column.

Aliases are defined for the native SQL query of the example above to avoid identical column names:

```
SELECT APPLICATION.REFSTR, APPLICATION.NAME AS 'Application Name',
APPLICATION.VERSION AS 'Application Version', ICTOBJECT.NAME AS 'ICT
Object Name'
FROM APPLICATION, ICTOBJECT
WHERE APPLICATION.ICTOBJECT = ICTOBJECT.REFSTR
```

The resulting column names and captions are:

Column Name	Column Caption
Application Name	Application Name
Application Version	Application Version
ICT Object Name	ICT Object Name

Defining Column Names and Captions with Instructions

For some Alfabet configurations, it is required or desirable to alter either the name of the column (which is not possible in the Show properties of an Alfabet query), or only the caption of the column (which is not possible in the SELECT clause of a native SQL query).

The Alfabet query language provides instructions that allow you to change only the name of the database column or only the caption of columns in a dataset.

The column name can be defined by adding the following instruction to the Alfabet query:

```
RENAMECOLUMN("ColumnName","NewColumnName");
```

Use the following parameter(s):

Code Parameter	Description
ColumnName	The name of the column that shall be renamed.
NewColumnName	The new name of the column.

The column caption can be defined by the following instruction:

```
SETCOLUMNSHOWNAME("ColumnName","ColumnCaption");
```

Use the following parameter(s):

Code Parameter	Description
ColumnName	The name of the column for that a new caption shall be defined.
ColumnCaption	The new caption of the column.

Referencing Columns in Instructions by Index Number

When writing instructions, the columns the instruction references are defined via their name in the instruction. Alternatively, instructions allow column definition via the position of the column in the column index. This mechanism is For example, required to write instructions for queries based on a stored procedure building dynamic datasets. For example, a return data set can include one column for each application currently defined in the Alfabet database.

The first column in the data set is position 1 and can be referenced within the instruction as `#1`. The next columns are numbered consecutively. For data sets created via a native SQL query, the first, invisible column defining the `REFSTR` of the current object is not included into the index.

Within the instruction, the index specification can either define one column or a range of columns by index when a single column specification is required. The instruction is then executed for each column in the range.

To define a range, the first and the last column of the range must be defined with a colon in between after the opening # sign:

```
#1:8
```

Alternatively, a range starting with a defined column and ending with the end of the dataset can be defined with the reserved keyword `end` defining the end of the range:

```
#1:end
```


Freezing Columns in Tabular Datasets

The first column or columns in a configured report displaying a tabular dataset can be defined as frozen to remain visible during horizontal scrolling.

Please note the following:

- The first columns in a dataset are the columns on the left of the dataset for most cultures. If you render the Alfabet user interface in Arabic language, the first columns are the columns on the right of the dataset.
- The freezing logic is applied to the final, visible dataset. If your configured report contains For example, instructions that are hiding columns, these columns are not to be taken into account when defining the number of frozen columns.

The following instruction freezes columns in a tabular dataset:

```
SetFreezeHeader(NumberOfFrozenColumns);
```

Use the following parameter(s):

Code Parameter	Description
`NumberOfFrozenColumns`	Define the number of columns that shall be frozen as an integer.

Aggregating Multiple Query Results in One Column of a Configured Report

In a configured report, each defined Show property of an Alfabet query or `SELECT` statement in the `SELECT` clause of a native SQL query will correspond to one column in the report table per default. By adding an instruction, you can define the Alfabet query to display multiple properties in one column. For example, the name and version of an application can be displayed in one column instead of two.

The following instruction aggregates two columns in one column:

```
JOINCOLUMNS("OldColumn1,Oldcolumn2","NewColumn","Delimiter");
```

Use the following parameter(s):

Code Parameter	Description
`OldCol-umn1,OldCol-umn2`	The names of the two columns that are aggregated. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions. The specifications of the two columns is delimited with a comma. White spaces are not allowed in the specification. You can specify more than two columns in a comma-separated format.

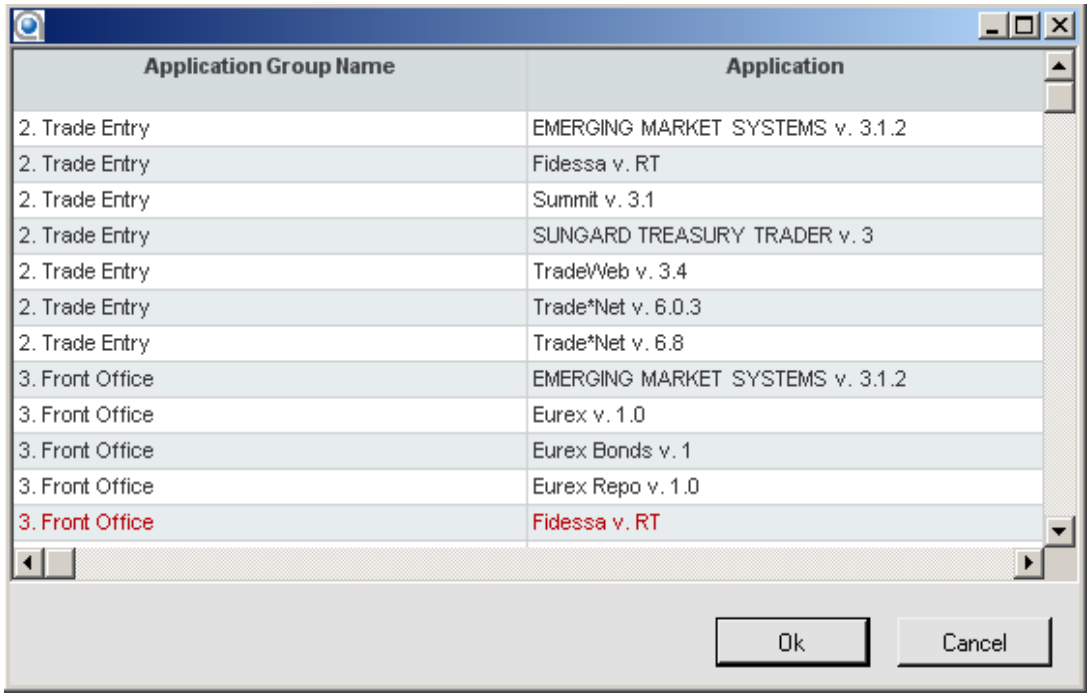
Code Parameter	Description
NewColumn	Define which of the two columns that are aggregated should be used to display the results. The specification of the new column must be identical to one of the column specifications of the old column.
Delimiter	The delimiter can be any string, including whitespaces. It will be written between the two values in the combined column in the report.

When joining columns, you should consider defining an alias for the column header to reflect both values in the column name. Otherwise, the default column header which specifies only one of the properties will be displayed.

A report shows application groups and the applications assigned to the application group. The name and version of the application shall be displayed in one column in the report. The code for the definition of the Show properties in the Alfabet query is:

```
Instructions

JOINCOLUMNS("Application.Name,Application.Version","Application.Name
"," v. ");

EndOfInstructions

QUERY_XML

<QueryDef>

   <ShowProperty Type="Property" ClassName="ApplicationGroup"
   Name="Name" />

   <ShowProperty Type="Property" ClassName="Application" Name="Name"
   ShowName="Application"/>

   <ShowProperty Type="Property" ClassName="Application"
   Name="Version" />

</QueryDef>
```

In the resulting report, the name and version of the applications is displayed in one column separated with `v.` The column header is `Application` because this is the header defined for the column `Application.Name`.



You can also use the `JoinColumns` instruction to narrow the display of a report having multiple columns where only one of the columns contain a result in a row. In other words, if one column displays a result, the other column will be empty in this row.

If the columns that are joined display results from different object classes, you cannot sort the results in the joined column using the Sort properties of the Alfabet query. Instead, a `SortBy` instruction must be added to the instruction section that triggers the sorting of values derived from various tables of the Alfabet database:

```
SortBy("ColumnName,ColumnName");
```

Use the following parameter(s):

Code Parameter	Description
`ColumnName`	The name of the column that shall be sorted. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions. You can define multiple column names comma-separated to sort multiple columns of the report. Columns are sorted in the order defined in the instruction.

For example, the configured report should display all architecture elements assigned to projects as well as the user that is the authorized user responsible for the architecture element. One row in the report will display each architecture element. A project can include objects from different object classes as architecture elements. The Alfabet query must be written as follows:


```
ALFABET_QUERY_500

FIND

ProjectArch

InnerJoin Project ON ProjectArch.Project = Project.REFSTR

LeftJoin Application ON ProjectArch.Object = Application.REFSTR

LeftJoin ICTObject ON ProjectArch.Object = ICTObject.REFSTR

LeftJoin Domain ON ProjectArch.Object = Domain.REFSTR

LeftJoin OrgaUnit ON ProjectArch.Object = OrgaUnit.REFSTR

LeftJoin BusinessProcess ON ProjectArch.Object =
BusinessProcess.REFSTR

LeftJoin BusinessFunction ON ProjectArch.Object =
BusinessFunction.REFSTR

LeftJoin Person ON (Or Person.REFSTR = Application.ResponsibleUser
Person.REFSTR = BusinessFunction.ResponsibleUser Person.REFSTR =
Domain.ResponsibleUser Person.REFSTR = BusinessProcess.ResponsibleUser
Person.REFSTR = ICTObject.ResponsibleUser Person.REFSTR =
OrgaUnit.ResponsibleUser)

Instructions

JoinColumns("Application.Name,Application.Version","Application.Name",
" v. ");

EndOfInstructions

QUERY_XML

<QueryDef>

<ShowProperty Type="Property" ClassName="Project" Name="Name" />

<ShowProperty Type="Property" ClassName="Application" Name="Name" />

<ShowProperty Type="Property" ClassName="Application" Name="Version"
/>

<ShowProperty Type="Property" ClassName="ICTObject" Name="Name" />

<ShowProperty Type="Property" ClassName="Domain" Name="Name" />

<ShowProperty Type="Property" ClassName="OrgaUnit" Name="Name" />

<ShowProperty Type="Property" ClassName="BusinessProcess" Name="Name"
/>

<ShowProperty Type="Property" ClassName="BusinessFunction" Name="Name"
/>

<ShowProperty Type="Property" ClassName="Person" Name="Name" />

<SortProperty Type="Property" ClassName="Project" Name="Name" />

</QueryDef>
```

The Alfabet query results in a report displaying architecture elements with one column per affected architecture object class:

	Project Name	Application Name	ICT Object Name	Domain Name	Organization Na...	Business Process Name	Business Function Na...	Person Name
47	Consolidate Trading Applic...					Bond Trading		Ngombe
48	Consolidate Trading Applic...					Custody Services		Ngombe
49	Consolidate Trading Applic...					Equity Trading		Ngombe
50	Consolidate Trading Applic...					Foreign Exchange Dealings		Ngombe
51	Consolidate Trading Applic...					Money Market Business		Ngombe
52	Consolidate Trading Applic...					Security Trading		Picard
53	Consolidate Trading Applic...				FD Trading			Customer
54	Consolidate Trading Applic...				OR Trading			Customer
55	Consolidate Trading Applic...				Wilmer & Partner			Picard
56	Consolidate Trading Applic...			The Enterprise				Customer

Adding a second `JoinColumns` instruction results in a report with three columns only. One for the project, one for the architecture element, and one for the authorized user of the architecture element. To enhance the usability of the report, you can add a `SetColumnShowName` instruction to rename the column caption to a neutral name (such as `Architecture Element`instead of `Application Name`, etc.) and a `SortBy` instruction to sort the result set by architecture element.

To do so, add the following instructions to the `Instruction` section of the Alfabet query defined above:

```
JoinColumns("Application.Name,ICTObject.Name,BusinessProcess.Name,Domain.Name,OrgaUnit.Name,BusinessFunction.Name","Domain.Name", "");

SetColumnShowName("Domain.Name","Architecture Element");

SortBy("Domain.Name");
```

The following columns are displayed in the configured report:

	Project Name	Architecture Element	Person Name
16	Consolidate HR Systems	AF HR Online US v. 2.2 Var.	Customer
17	CRM Analytics Center	AF HR Online US v. 2.2 Var.	Customer
18	Streamline CRM Applications	AF HR Online US v. 2.2 Var.	Customer
19	Consolidate HR Systems	AF HR Online v. 2.2	Customer
20	Consolidate HR Systems	AF HR Online v. 3.0	Customer
21	Replace AF HR Online by SAP HR HQ	AF HR Online v. 3.0	Customer
22	Enable Internet-based Trading with TradeNet	AI Reinsurances	Ngombe
23	Implement salesforce Greenfield strategy	AI Sales & Marketing	Alfabet
24	Rollout salesforce 9.2	AI Sales & Marketing	Alfabet
25	Rollout salesforce 9.3	AI Sales & Marketing	Alfabet
26	Integrate CRM with SAP	Analyze Turnover	Mustermann
27	Best in Breed Solution	Balance Analysis DB v. 1.2.2	Administrator
28	Central CRM Application	Balance Analysis DB v. 1.2.2	Administrator
29	CRM Architectural Study	Balance Analysis DB v. 1.2.2	Administrator
30	Implement Confidentiality and Integrity Mitiga...	Balance Analysis DB v. 1.2.2	Administrator
31	Streamline CRM Applications	Balance Analysis DB v. 1.2.2	Administrator
32	Make Bloomberg future compatible	BLOOMBERG	Administrator
33	Setup Bloomberg access	BLOOMBERG	Administrator
34	Upgrade Bloomberg	BLOOMBERG	Administrator

Hiding a Column in a Configured Report

By default, each property defined in the Show properties of an Alfabet query or the `SELECT` clause of a native SQL query (with the exception of the first statement in the `SELECT` clause) corresponds to one column in the report table. For some configurations (For example, when building a report with collapsed tables), it is

required to add properties for technical reasons only. In most cases the REFSTR property of an object class must be added to the report. This information is not relevant for the user viewing the report. You can hide a column in the report by adding the following instruction:

```
REMOVECOLUMNS("ColumnName");
```

Use the following parameter(s):

Code Parameter	Description
ColumnName	The name of the column that is hidden. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions. You can hide multiple columns with one command by specifying multiple column names delimited with a comma. White spaces are not allowed in the specification.

Grouping Query Results in Expandable Reports

By default, tabular reports about two related object classes display one row for each combination of the two object classes found by the Alfabet query or native SQL query. You can add an instruction to the query to group the results and display one object class as subordinate to the other object class in an expandable table section. The advantage of grouping is that the users will have a better overview of the results in the table. Users will see a report that displays one row for each object found for the superordinate object class, each with a + symbol in front. The objects of the subordinate object class are displayed each in a separate row beneath the selected object when the + symbol is clicked and the table section expanded.

For example, for a configured report that displays applications and their local components and shows the names and end dates of the objects in the resulting dataset, the normal result table would include one row for each combination of application and local component, with one column for each of the properties.

Application Name	Application End Date	Local Component Name	Local Component End Date
Groupware Services	2/18/2012	DB 2 Universal Database	9/29/2024
Mafo-Portal	7/3/2015	DB 2 Universal Database	9/29/2024
SAP@OptiRetail	1/24/2010	DB 2 Universal Database	9/29/2024
SAP@OptiRetail	1/24/2010	SAP PM	9/29/2024
SAP@OptiRetail	1/24/2010	MS Windows XP	9/29/2024
SAP@OptiRetail	1/24/2010	ORBIX	9/29/2024
SAP@OptiRetail	1/24/2010	MS Internet Explorer	9/29/2024
SAP@OptiRetail	1/24/2010	Ubuntu Linux	9/29/2024
SAP@OptiRetail	1/24/2010	ORBIX	9/29/2024
SAP@OptiRetail	1/24/2010	Apache	9/29/2024
SAP@OptiRetail	1/24/2010	WebSphere	9/29/2024
SAP@OptiRetail	1/24/2010	Ubuntu Linux	9/29/2024
SAP@OptiRetail	1/24/2010	ORBIX	9/29/2024
SAP@OptiRetail	1/24/2010	LDAP	9/29/2024
SAP@OptiRetail	1/24/2010	Ubuntu Linux	9/29/2024
SAP@OptiRetail	1/24/2010	ORBIX	9/29/2024
SAP@OptiRetail	1/24/2010	Ubuntu Linux	9/29/2024
SAP@OptiRetail	1/24/2010	IBM Power Systems	9/29/2024

With the grouping instruction, the outlook of the report can be changed to display local components subordinate to applications. When the user opens the report, he/she can see one row for each application with a + symbol in front to expand the table section, displaying the application's name and end date. When the user clicks the + symbol, one row for each local component will be displayed beneath the row with the parent application, displaying the component's name and end date.

1 2	Application Name	Application End Date	Local Component Name	Local Component End Date
+	Groupware Services	2/18/2012		
+	Mafo-Portal	7/3/2015		
+	SAP@OptiRetail	1/24/2010		
−	Corporate FI-CO	4/6/2011		
·			Adobe Acrobat Reader	9/29/2024
·			SAP GUI	9/29/2024
·			SAP CO	9/29/2024
·			SAP FI	9/29/2024
·			SAP MM	9/29/2024
·			SAP PM	9/29/2024
·			MS Windows XP	9/29/2024
·			ORBIX	9/29/2024
·			MS Internet Explorer	9/29/2024
·			Ubuntu Linux	9/29/2024
·			ORBIX	9/29/2024
·			Apache	9/29/2024
·			WebSphere	9/29/2024

Instructions are executed after the Alfabet query or native SQL query has been executed. Therefore, the information that is required to specify the relation between two object classes must not only be defined in the query, but also be reflected in the dataset resulting from the query. The instruction reads the technical data required to group the classes from the plain report table prior to instruction execution.

Relations between object classes are based on the following properties:

- Objects of the subordinate class are identified by their `REFSTR` property.

  For example, a configured report displays demands and their classification as well as the applications that are assigned to the demand as architecture objects. In this case, the class `Demand` is the superordinate object class. The data in the report can be grouped by demand `REFSTR`. The column with the `REFSTR` information is hidden from the report output via a `RemoveColumns` instruction. The classification information is the same in all rows for a demand and can Therefore, also be displayed in a grouped version of the report:

Demand Name	Demand Classification	Application	Application Version
CRM Consolidation	Strategic	CRM CSS	3.3
CRM Consolidation	Strategic	CRM	2.6
CRM Consolidation	Strategic	OptiRetail Marketing Solut	2.0
CRM Consolidation	Strategic	CRM Opti Retail	3.0
Development CRM Briefin	Application Mgmt.	CRM Opti Retail	3.0
Development M&R Portal	Operational,Strategic	Mafo-Portal	2.6
Development Portfolio Ma	Application Mgmt.,Strateg	CRM Opti Retail	3.0
Development Portfolio Ma	Application Mgmt.,Strateg	CRM AI	2.0
Development Portfolio Ma	Application Mgmt.,Strateg	CRM CSS	3.3
Development Portfolio Ma	Application Mgmt.,Strateg	CRM	2.6
Enhance Formblatt P2387	Operational,Application M	Credit Manager	2.1
Enhance Kreditnehmer R	Operational,Application M	Credit Manager	2.1
Enhance Reliability of Tra	Operational	Bysis GL	2.0
Enhance Reliability of Tra	Operational	GL Trade	2.3

  Although the grouping instruction is configured to work with the `REFSTR` property to find objects of the superordinate class, you may also define that grouping shall be performed based on an identical property value for the superordinate class. This is an exception to the normal specification of the report and no guarantee is given that the grouping can be performed on any property.

  If you define grouping based on a property that is equal for multiple objects, all subordinate objects of all superordinate objects belonging to the group are displayed in one group. The columns in the table providing information about the superordinate level must then be limited to information that is identical per group.

  Grouping by a property that is equal for multiple objects is best configured via an Alfabet query. Native SQL queries can only be used if defined in a special way. The grouping by property is described in the section Grouping By Category Instead of By Object.

- Objects of the subordinate class are identified by their `REFSTR` property.

To group two object classes in a report, the column names in the resulting dataset must match a specific scheme. They must be defined as

ClassName.PropertyName

If a class alias is defined in the query for the object class, the definition must be:

ClassAlias.PropertyName

For native SQL queries, the column names resulting from the Show property definition automatically match this criterion. A RenameColumn instruction must not be used.

For native SQL queries an alias must be defined for the properties in the SELECT clause that defines the column names as matching to the scheme.

For example, the Name property of the object class Demand must be specified in the SELECT statement as:

```
SELECT DEMAND.REFSTR, DEMAND.NAME As 'Demand.Name'
FROM DEMAND
```

If an alias specification is used for a class in the native SQL query, the alias instead of the class name must be used as class name in the column header. For example,:

```
SELECT dem.REFSTR, dem.NAME As 'dem.Name'
FROM DEMAND dem
```

To group two object classes in a report:

1) Add the REFSTR property of the subordinate object class and the REFSTR property of the superordinate object class to the Show properties of the Alfabet query or the SELECT clause of the native SQL query.

   In native SQL queries, you must add the REFSTR property of the superordinate class two times to the SELECT statement. Evaluation of native SQL requires that the REFSTR of the main object class found by the native SQL query is defined as first property in the SELECT statement. In the resulting report set, the first column containing the REFSTR is removed prior to executing instructions.

   To hide properties that are used for technical reasons only from display in the query results, use the REMOVECOLUMNS instruction described in the section Hiding a Column in a Configured Report.

2) For native SQL only: Define alias specification for the column headers in the SELECT clause. Each column header must follow the following syntax: <ClassName>.<PropertyName> or, if an alias is specified for the class in the FROM statement, <AliasName>.<PropertyName>.

3) Define Sort properties in the Alfabet query or ORDER BY clauses in the native SQL query that result in rows sorted by the grouping property of the superordinate class. In other words, all rows in the report with the same grouping property of the superordinate class must follow each other.

4) Go to the **Instructions** tab of the **Alfabet Query Builder** and write the following instruction:

   GROUPBY_EX("*SuperordinateObject*","*SubordinateObject*","*SuperordinateClass*",*Level*);

Use the following parameter(s):

Code Parameter	Description
SuperordinateObject	The name of the column that displays the REFSTR property of the superordinate object class used to identify the objects of the superordinate class in the report. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see *Defining Column Name And Caption*.
SubordinateObject	The name of the column that displays the REFSTR property of the subordinate object class used to identify the objects of the subordinate class in the report.
Superordinate Class	The object class that is displayed in the superordinate level. The class must be specified as If an alias specification is used for a class in the Alfabet query or native SQL query, the alias instead of the class name must be specified as superordinate class and used as class name in the column header specifications. ClassName All columns with headers starting with the specified class alias are displayed in the rows for the superordinate object.
Level	You can define multiple GROUPBY_EX instructions for one Alfabet query or native SQL query. The level defines the ranking of grouping commands. It is specified by a natural number. The first grouping has a level of 0, the next grouping below that has a level of 1.

The instruction is interpreted as follows:

- The class that is first defined in the GROUPBY_EX command with the level number 0 is the class that is displayed on the root level. All columns with headers starting with the specified class alias are displayed in the rows for the superordinate object. This is independent from the position of the columns displaying the attributes in the report.
- The columns with properties of the subordinate class are displayed in the second level result rows independent from their position in the report.
- The properties of any object classes that are joint to the subordinate object class are also displayed in the second level row, as long as no further grouping is specified.

You can combine the GROUPBY_EX instruction with other instructions:

Please note that the order of defining instructions is relevant. If you define the GroupBy_Ex instruction in combination with instructions that are assigned to rows in a report, For example, to define link assignment or coloring of cells or rows, the GroupBy_Ex instruction must be defined first and the instruction for formatting of the rows must defined after the GroupBy_Ex instruction.

- Use the REMOVECOLUMNS instructions to hide the columns added for technical reasons, like the REFSTR property of objects, from the results displayed to the user. For more information about the RemoveColumns instruction, see Hiding a Column in a Configured Report.

- Use the REMOVEEMPTYROWS instruction if your report is based on a left join between the grouped object classes. When objects of the superordinate class are displayed for which no objects are found, the objects of the superordinate class are nevertheless displayed with a + symbol to expand the table section, even though there are no subordinate rows displayed when you click the + symbol. You can remove the + symbol before rows that do not have subordinate object results by adding the REMOVEEMPTYROWS command as follows:

```
REMOVEEMPTYROWS();
```

- Use the REMOVESOERTCOLUMNS instruction to prevent that the user can sort the data in a column.
- Use the JOINCOLUMNS instruction to enhance readability of the configured report.
  - For example, a report includes two object classes that are displayed one subordinate to the other. For both objects, the end dates are displayed. These values are normally displayed in two columns in the report, one showing the end dates of one object class and the other the end dates of the second object class. Grouping the results make it possible to display all end dates of all objects on both hierarchy levels in one row.
  - For example, if you group the display of the names of the superordinate and subordinate objects in the first column, the report will appear to display both classes in one row. Values for subordinate objects will be indented.

For the report in the previous example where local components are grouped under applications and the start and end dates of both applications and local components are displayed, the basic configuration includes one grouping instruction to group local components under applications. An instruction to remove the columns for the REFSTR properties is also included.

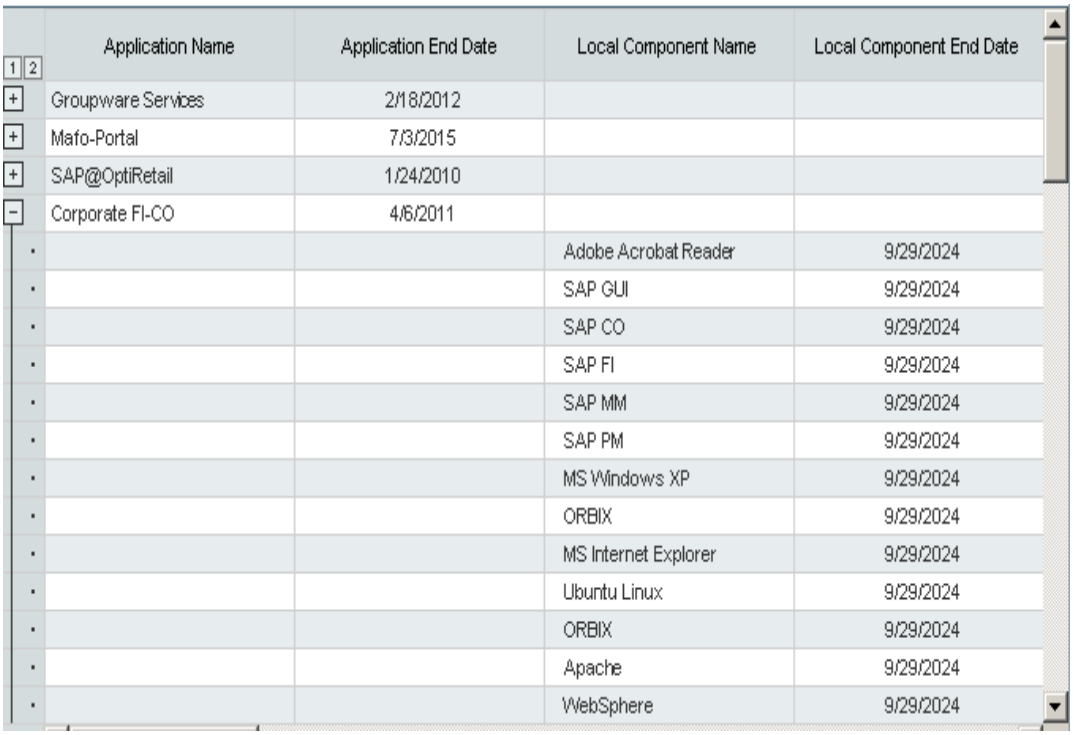
The report can be created with the following Alfabet query:

```
ALFABET_QUERY_500

FIND Application

   InnerJoin LocalComponent ON LocalComponent.Owner =
   Application.REFSTR

Instructions

GROUPBY_EX("Application.REFSTR","LocalComponent.REFSTR","Application
",0);

REMOVECOLUMNS("Application.REFSTR,LocalComponent.REFSTR");

EndOfInstructions

QUERY_XML

<QueryDef>

   <ShowProperty Type="Property" ClassName="Application" Name="Name"
   />

   <ShowProperty Type="Property" ClassName="Application"
   Name="EndDate" />

   <ShowProperty Type="Property" ClassName="LocalComponent"
   Name="Name" />

   <ShowProperty Type="Property" ClassName="LocalComponent"
   Name="EndDate" />
```



```
    <ShowProperty Type="Property" ClassName="LocalComponent"
    Name="REFSTR" />

    <ShowProperty Type="Property" ClassName="Application"
    Name="REFSTR" />

    <SortProperty Type="Property" ClassName="Application" Name="Name"
    />

</QueryDef>
```

In the resulting report, results for objects of each class are displayed in separate rows and separate columns. The results for local components can be collapsed in the table.



Application Name	Application End Date	Local Component Name	Local Component End Date
Groupware Services	2/18/2012		
Mafo-Portal	7/3/2015		
SAP@OptiRetail	1/24/2010		
Corporate FI-CO	4/6/2011		
		Adobe Acrobat Reader	9/29/2024
		SAP GUI	9/29/2024
		SAP CO	9/29/2024
		SAP FI	9/29/2024
		SAP MM	9/29/2024
		SAP PM	9/29/2024
		MS Windows XP	9/29/2024
		ORBIX	9/29/2024
		MS Internet Explorer	9/29/2024
		Ubuntu Linux	9/29/2024
		ORBIX	9/29/2024
		Apache	9/29/2024
		WebSphere	9/29/2024

To display the object names in the same column, a `JOINCOLUMN` instruction must be added to the Alfabet query's `Instructions` section. It is also recommended that you rename the column header for the resulting combined column:

```
ALFABET_QUERY_500

FIND Application

    InnerJoin LocalComponent ON LocalComponent.Owner =
    Application.REFSTR

Instructions

GROUPBY_EX("Application.REFSTR","LocalComponent.REFSTR","Application
",0);

REMOVECOLUMNS("Application.REFSTR,LocalComponent.REFSTR");

JOINCOLUMNS("Application.Name,LocalComponent.Name","Application.Name
","");

EndOfInstructions

QUERY_XML

<QueryDef>
```



```
    <ShowProperty Type="Property" ClassName="Application" Name="Name"
    ShowName="Application / local Component"/>

    <ShowProperty Type="Property" ClassName="Application"
    Name="EndDate" />

    <ShowProperty Type="Property" ClassName="LocalComponent"
    Name="Name" />

    <ShowProperty Type="Property" ClassName="LocalComponent"
    Name="EndDate" />

    <ShowProperty Type="Property" ClassName="LocalComponent"
    Name="REFSTR" />

    <ShowProperty Type="Property" ClassName="Application"
    Name="REFSTR" />

    <SortProperty Type="Property" ClassName="Application" Name="Name"
    />

    <SortProperty Type="Property" ClassName="LocalComponent"
    Name="Name" />

</QueryDef>
```

In the resulting report, results for objects of each class are displayed in separate rows and in the same column with regard to the `Name` property. The results for local components can be collapsed in the table.

1 2	Application / local Component	Application End Date	Local Component End Da
+	Groupware Services	2/18/2012	
−	Mafo-Portal	7/3/2015	
•	DB 2 Universal Database		9/29/2024
+	SAP@OptiRetail	1/24/2010	
+	Corporate FI-CO	4/6/2011	
+	eLead	10/11/2010	
+	CRM CSS	2/4/2013	

The same report results from the following native SQL query:

```
SELECT app.REFSTR, app.REFSTR as 'app.REFSTR', app.NAME AS
'app.Name', app.ENDDATE as 'app.EndDate', LOCALCOMPONENT.REFSTR as
'LocalComponent.REFSTR', LOCALCOMPONENT.NAME as
'LocalComponent.Name', LOCALCOMPONENT.ENDDATE As
'LocalComponent.EndDate'

FROM APPLICATION app

LEFT JOIN LOCALCOMPONENT

ON app.REFSTR =LOCALCOMPONENT.OWNER

ORDER BY app.REFSTR
```

Grouping is performed on basis of the `REFSTR` of the class Application. Therefore, the `REFSTR` property is added two times to the `SELECT` statement of the native SQL query.

In the native SQL query, the alias `app` is defined in the `FROM` clause for the object class Application. When defining the instructions, the columns containing information about the superordinate object class must be defined as `app.`<PropertyName>.

It is also recommended that you rename the column header for the resulting combined column.

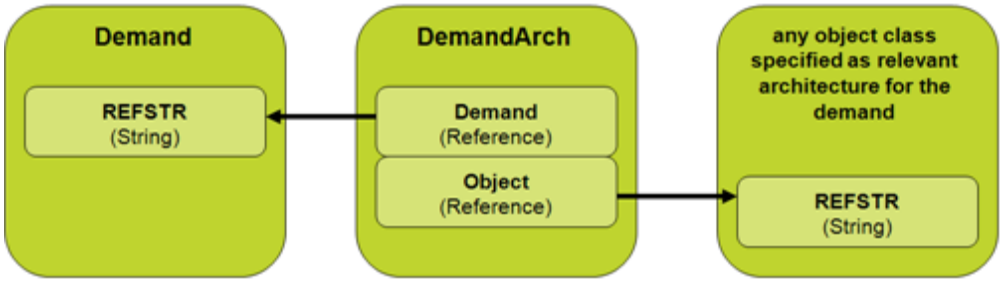
The following instructions must be defined for the native SQL query to create the grouped report displayed above:

```
GROUPBY_EX("app.REFSTR","LocalComponent.REFSTR","app",0);

REMOVECOLUMNS("app.REFSTR,LocalComponent.REFSTR");

JOINCOLUMNS("app.Name,LocalComponent.Name","app.Name","");

SETCOLUMNSHOWNAME("app.Name","Application / local Component");

SETCOLUMNSHOWNAME("app.EndDate","Application End Date");

SETCOLUMNSHOWNAME("LocalComponent.EndDate","Local Component End
Date");
```

Grouping Indirectly Related Object Classes

It is possible to group object classes that are indirectly related via a third object class. If the object class establishing the relation is used for grouping, but properties of that class are not included in the report results, the report looks as if the two indirectly related object classes are grouped.

For example, if you want to display all applications that are architecture elements relevant for demands, you can group applications under demands although they are only indirectly connected via the `DemandArch` object:



In the query, two `JOIN`s are required to display applications assigned to the demands.

In an Alfabet query:

```
ALFABET_QUERY_500

FIND Demand

   LeftJoin DemandArch ON DemandArch.Demand = Demand.REFSTR

   InnerJoin Application ON DemandArch.Object = Application.REFSTR

QUERY_XML

<QueryDef>

   <ShowProperty Type="Property" ClassName="Demand" Name="Name" />

   <ShowProperty Type="Property" ClassName="Application" Name="Name"
   ShowName="Application" />

   <ShowProperty Type="Property" ClassName="Application"
   Name="Version" />
```



```
    <SortProperty Type="Property" ClassName="Demand" Name="Name" />
</QueryDef>
```

In a native SQL query:

```
SELECT DEMAND.REFSTR, DEMAND.REFSTR as 'Demand.REFSTR', DEMAND.NAME
As 'Demand.Name', APPLICATION.REFSTR As 'Application.REFSTR',
APPLICATION.NAME As 'Application.Name', APPLICATION.VERSION As
'Application.Version'
FROM DEMAND
LEFT JOIN DEMAND_ARCH ON DEMAND_ARCH.DEMAND = DEMAND.REFSTR
INNER jOIN APPLICATION ON DEMAND_ARCH.OBJECT = APPLICATION.REFSTR
ORDER BY DEMAND.REFSTR;
```

Properties of the `DemandArch` object are not shown in the query results. Nevertheless, a grouping can be done that groups the class `DemandArch` as subordinate to the class `Demand`. The object class `Application` is joined to the object class `DemandArch`, and the properties for applications displayed in report columns are Therefore, displayed on the subordinate level:

1 2	Demand Name	Application
+	Revise Charting Capability	
−	Re-implement Monthly Consolidation Report	
.		eBank v. 1.2
+	Implement Sobolev algorithm	
−	Rework CRM for regional support	
.		CRM Opti Retail v. 3.0 Var.
.		CRM CSS v. 3.2
.		CRM Opti Retail v. 2.0
.		CRM v. 2.6
.		CRM AI v. 2.0
+	Roll-out SAP to the regions	
+	Update trading system to new version	
+	Retire Russia CL and CL Trade	

For this report, the Alfabet query was changed as follows:

```
ALFABET_QUERY_500
FIND Demand
   LeftJoin DemandArch ON DemandArch.Demand = Demand.REFSTR
   InnerJoin Application ON DemandArch.Object = Application.REFSTR
Instructions
JOINCOLUMNS("Application.Name,Application.Version","Application.Name
"," v. ");
GROUPBY_EX("Demand.REFSTR","Application.REFSTR","Demand",0);
REMOVECOLUMNS("Demand.REFSTR, Application.REFSTR");
EndOfInstructions
QUERY_XML
```

```
<QueryDef>
   <ShowProperty Type="Property" ClassName="Demand" Name="Name" />
   <ShowProperty Type="Property" ClassName="Application" Name="Name"
   ShowName="Application" />
   <ShowProperty Type="Property" ClassName="Application"
   Name="Version" />
   <ShowProperty Type="Property" ClassName="Application"
   Name="REFSTR" />
   <ShowProperty Type="Property" ClassName="Demand" Name="REFSTR" />
   <SortProperty Type="Property" ClassName="Demand" Name="Name" />
</QueryDef>
```

For this report, the following instructions must be combined with the native SQL query above:

```
JOINCOLUMNS("Application.Name,Application.Version","Application.Name
"," v. ");
GROUPBY_EX("Demand.REFSTR","Application.REFSTR","Demand",0);
REMOVECOLUMNS("Demand.REFSTR,Application.REFSTR");
SETCOLUMNSHOWNAME("Demand.Name","Demand Name");
SETCOLUMNSHOWNAME("Application.Name","Application");
```

Grouping By Category Instead of By Object

In the examples above, grouping was performed by specifying the superordinate class based on the `REFSTR` attribute of the object class. This ensures that the first level results are grouped to one group per object. But grouping does not have to be performed per object. You can use any other column in the tabular report to sort the results in the upper level. This allows grouping of sub-level items according to the value for a defined property of the superordinate class independent of the assignment of subordinate objects to superordinate objects.

The grouping by other than the `REFSTR` property of the superordinate object class is an exception that is not part of the guaranteed functionality of the `GroupBy_Ex` instruction. It may not work for some grouping scenarios.

The following example explains the possibilities and limits of grouping by non-unique property values for superordinate objects.

In the example, applications are displayed as subordinate to the demands to which they are assigned as architecture objects. Each demand was classified via the definition of the property `Classification`. If you add the `Classification` property to the Show properties of the Alfabet query or the `SELECT` clause of the native SQL query, and define the Sort properties of the Alfabet query or the `ORDER BY` clause of the native SQL query to sort demands by classification values, you can see in the result output that multiple demands share the same classification.

Demand Name	Demand Classification	Application
Development CRM Briefing Center Analytics	Application Mgmt.	
		CRM Opti Retail v. 3.0
Implement Sobolev algorithm	Application Mgmt.	
		eBank v. 1.2
Improve Commercial Order Management	Application Mgmt.	
		ARIS v. 6.2.1
Integration CRM CSS and Trade*Net	Application Mgmt.	
		CRM CSS v. 3.2
		Trade*Net v. 6.0.3
Rework CRM for regional support	Application Mgmt.	
		CRM Opti Retail v. 3.0 Var.
		CRM AI v. 2.0
		CRM CSS v. 3.2
		CRM Opti Retail v. 2.0
		CRM v. 2.6
Development Portfolio Management Tool	Application Mgmt.,Strategic	
Integrate Risk Management in Trade*Net	Application Mgmt.,Strategic	
Parcel Deliverables Optionization	Application Mgmt.,Strategic	
Rollout / Implementation of CRM onDemand	Application Mgmt.,Strategic	

If you want to see all applications that are relevant for the architecture assigned to any demand of a defined demand classification group, you can use the `Classification` property in the `GroupBy_Ex` command. The name of the demand must be excluded from the report because it is not identical for all rows within a group.

In an Alfabet query:

```
ALFABET_QUERY_500
FIND Demand
   LeftJoin DemandArch ON DemandArch.Demand = Demand.REFSTR
   InnerJoin Application ON DemandArch.Object = Application.REFSTR
Instructions
JOINCOLUMNS("Application.Name,Application.Version","Application.Name"," v.
");
GROUPBY_EX("Demand. Classification ","Application.REFSTR","Demand",0);
REMOVECOLUMNS("Application.REFSTR");
EndOfInstructions
QUERY_XML
<QueryDef>
   <ShowProperty Type="Property" ClassName="Demand" Name="Classification" />
   <ShowProperty Type="Property" ClassName="Application" Name="Name"
   ShowName="Application" />
   <ShowProperty Type="Property" ClassName="Application" Name="Version" />
   <ShowProperty Type="Property" ClassName="Application" Name="REFSTR" />
```



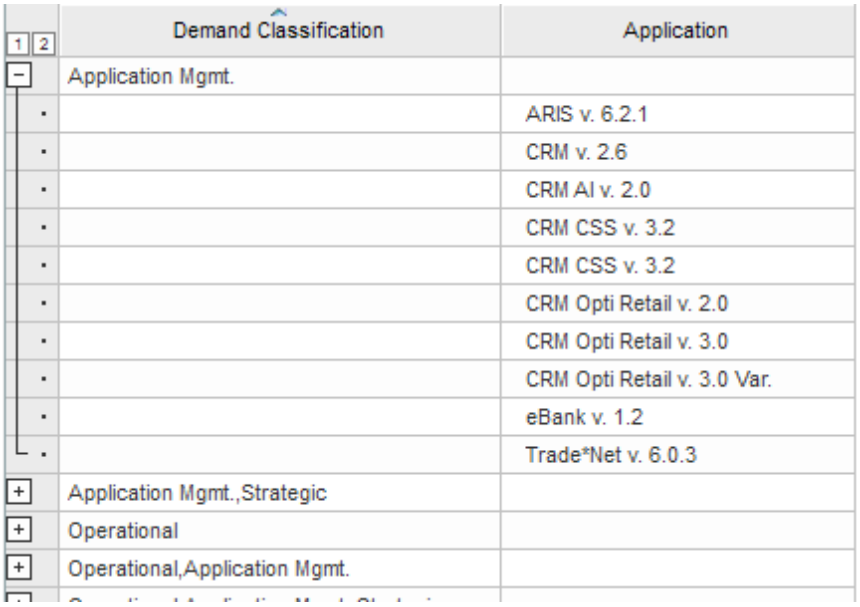
```
    <SortProperty Type="Property" ClassName="Demand" Name=" Classification "
    />
</QueryDef>
```

In native SQL, grouping strictly depends on the objects of the class in the FROM statement. If this class is an Alfabet object class, groups containing multiple objects are not allowed in the superordinate level. To solve this problem, the SQL query must be defined with a `FROM` clause not pointing to an object class in the database, but to an alias that is derived from a `WITH` statement collecting the required data from the Alfabet database. For the `SELECT` statement of the alias class, the required first argument returning a `REFSTR` is defined as `NULL`, because the alias class has no `REFSTR` property:

```
WITHAliasClassAs

(SELECTDEMAND.CLASSIFICATION As 'Name', APPLICATION.REFSTR As 'Application',
APPLICATION.NAME As 'AppName', APPLICATION.VERSION As 'AppVersion'

FROM DEMAND

LEFT JOIN DEMAND_ARCH ON DEMAND_ARCH.DEMAND = DEMAND.REFSTR

INNER JOIN APPLICATION ON DEMAND_ARCH.OBJECT = APPLICATION.REFSTR

)

SELECTNULL As 'REFSTR',Name As "AliasClass.Name ", Application As
'Application.REFSTR', AppName As 'Application.Name', AppVersion As
'Application.Version'

FROMAliasClass

ORDER BY AliasClass.Name ;
```

with the following instructions:

```
JOINCOLUMNS("Application.Name,Application.Version","Application.Name"," v.
");

GROUPBY(" AliasClass.Name ","Application.REFSTR","AliasClass",0);

REMOVECOLUMNS("Application.REFSTR");

SETCOLUMNSHOWNAME("Demand.Name","Demand Name");

SETCOLUMNSHOWNAME("Application.Name","Application");

SETCOLUMNSHOWNAME("AliasClass.Name","Demand Classification");
```

Demand Classification	Application
Application Mgmt.	
	ARIS v. 6.2.1
	CRM v. 2.6
	CRM AI v. 2.0
	CRM CSS v. 3.2
	CRM CSS v. 3.2
	CRM Opti Retail v. 2.0
	CRM Opti Retail v. 3.0
	CRM Opti Retail v. 3.0 Var.
	eBank v. 1.2
	Trade*Net v. 6.0.3
Application Mgmt.,Strategic	
Operational	
Operational,Application Mgmt.	

Preventing Users from Sorting According to Subordinate Objects

By default, users can click any header cell of a tabular report to sort the results in the report alphanumerically according to the information displayed in the selected column. Clicking the header again changes the sort order from ascending to descending. This is independent of the definition of Sort properties in Alfabet queries or `ORDER BY` clauses in native SQL queries that define presorting for the first display of data in the report.

If the user sorts results alphanumerically by a property of a subordinate object class, correct alphanumeric sorting is inhibited by the grouping by superordinate object. The sorting is done as follows:

- Within a group, the subordinate objects are sorted alphanumerically.
- The order of superordinate objects is also changed. The superordinate objects are sorted according to the alphanumeric order of the selected sort property of the first subordinate object in their group.

You can prevent sorting by the user by adding the following instruction to the report:

The report must be based on an Alfabet query to use the instruction. Sorting in reports based on native SQL queries cannot be prevented.

```
RemoveSortColumns("ColumnName,ColumnName");
```

Use the following parameter(s):

Code Parameter	Description
`ColumnName`	The name of the column in the tabular output of the report that shall not be sortable. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see *Defining Column Name And Caption*.

Code Parameter	Description
	If multiple columns shall not be sorted, the columns can be defined comma separated.

Changing the Order and Number of Columns Displayed in a Report

The instruction `CreateDSInfo` can be used to restructure per batch the output of a query. The following tasks can be performed within a single instruction:

- Changing the order of columns from the result dataset displayed in the report.
- Removing columns from display in the report.
- Adding multiple columns added via `InsertColumn` commands or an `AddColumns` instruction to the dataset.
- Changing all column captions in a single instruction.

The main use cases for the `CreateDSInfo` instruction are:

- Columns have been added to the dataset via `InsertColumn` or `AddColumns` instructions and the columns shall be displayed before or in the middle of the data set resulting from the query. By default, inserted columns are displayed at the end of the tabular report. For more information, see Inserting One or Multiple Columns to a Report.
- A report requires a high number of `SetColumnShowName` and `RemoveColumn` commands, For example, to display a complex grouped dataset. Instead, a `CreateDSInfo` command can be used to define the resulting data set in a single command. An example is given in this section.

Use the following instruction to restructure the output of your query:

```
CreateDSInfo("ColumnName,ColumnName","ColumnCaption,ColumnCaption");
```

Use the following parameter(s):

Code Parameter	Description
`ColumnName`	The comma-separated list of the columns that shall be displayed in the result dataset. Each column is identified by the column name. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see *Defining Column Name And Caption*. The order of specification in the comma-separated list of column names defines the order of the display of the columns in the resulting dataset.

Code Parameter	Description
`ColumnCaption`	The comma-separated list of captions to be displayed for the columns in the dataset. For each column specified in the comma-separated list of column names, a caption must be specified in the same order than the column definition.

Note the following about the use of the `CreateDSInfo` instruction:

- If you define a `CreateDSInfo` instruction for a native SQL query, you must include the first `SELECT` argument defining the `REFSTR` of the base object of the report in the result data set. This column is hidden in the report output, but removal of the first column from the data set is performed after the execution of the instruction (on the dataset resulting from the `CreateDSInfo` instruction).
- The `CreateDSInfo` instruction re-creates the complete result dataset. Even if only a subset of the result columns need re-configuration, you must include the complete result dataset that you want to display to the user ins the `CreateDSInfo` command.
- If you want to use a `PictureAssignment` instruction in combination with the `CreateDSInfo` instruction, the `CreateDSInfo` instruction must be defined prior to the `PictureAssignment` instruction.

For example, a report shall show all applications that are assigned to demands with defined classifications in a grouped dataset.

	Demand Classification	Application
−	Application Mgmt.	
		ARIS v. 6.2.1
		CRM v. 2.6
		CRM AI v. 2.0
		CRM CSS v. 3.2
		CRM CSS v. 3.2
		CRM Opti Retail v. 2.0
		CRM Opti Retail v. 3.0
		CRM Opti Retail v. 3.0 Var.
		eBank v. 1.2
		Trade*Net v. 6.0.3
+	Application Mgmt.,Strategic	
+	Operational	
+	Operational,Application Mgmt.	

The report is based on a native SQL query:

```
SELECT DEMAND.REFSTR, DEMAND.CLASSIFICATION As
'Demand.Classification', APPLICATION.REFSTR As 'Application.REFSTR',
```

```
APPLICATION.NAME As 'Application.Name', APPLICATION.VERSION As
'Application.Version'

FROM DEMAND

LEFT JOIN DEMAND_ARCH ON DEMAND_ARCH.DEMAND = DEMAND.REFSTR

INNER jOIN APPLICATION ON DEMAND_ARCH.OBJECT = APPLICATION.REFSTR

ORDER BY DEMAND.CLASSIFICATION;
```

with the following instructions:

```
JOINCOLUMNS("Application.Name,Application.Version","Application.Name"
," v. ");

GROUPBY("Demand.Classification","Application.REFSTR","Demand",0);

REMOVECOLUMNS("Application.REFSTR");

SETCOLUMNSHOWNAME("Application.Name","Application");

SETCOLUMNSHOWNAME("Demand.Classification","Demand Classification");
```

Alternatively, you can define the instructions with a `CreateDSInfo` instruction instead of using one `RemoveColumns` instruction and two `SetColumnShowName` instructions.

```
JOINCOLUMNS("Application.Name,Application.Version","Application.Name"
," v. ");

GROUPBY("Demand.Classification","Application.REFSTR","Demand",0);

CREATEDSINFO("DEMAND.REFSTR,Demand.Classification,Application.Name","
DEMAND.REFSTR,Demand Classification,Application");
```

Inserting One or Multiple Columns to a Report

The columns displayed in the tabular output of a query are defined with the Show properties of the Alfabet query or the `SELECT` statement of the native SQL query. Instructions not only allow to hide, change, or merge columns defined via the query, but also to add columns to the result dataset.

When the results of a query are displayed in the Alfabet interface, the processing of the query is done in multiple steps. The query results are gathered in an internal dataset in the background and the final report is generated from the internal dataset in a second step.

The specification of additional columns requires the definition of multiple instructions that do the following:

- Add the column to the internal dataset resulting from the query.
- Define the content to be filled in the column.
- Define the display of the column in the result dataset.

You must do the following to add new columns to the dataset:

1) Add the column to the query result dataset that is formed during query execution prior to displaying the results on the Alfabet interface:
   - Use `AddColumns` to add multiple columns to the end of the query result dataset. The columns are added to the internal dataset of query results at the end of the

dataset in the order of specification in the instruction. For information about the required syntax, see Adding Multiple Columns with the AddColumns Instruction.

- Use `InsertColumn` to add one column at a defined position of the internal query result dataset. For information about the required syntax, see Adding A Single Column With the InsertColumn Instruction.

2) Define the column caption and the position of the column in the dataset displayed in the report with one of the following instructions:

- `SetColumnShowName`: This instruction defines a caption for a new column. For each added column that shall be displayed in the results, a `SetColumnShowName` instruction must be added to the instructions section.

New columns added via an `AddColumns` instruction are displayed at the end of the tabular report, regardless of their position in the internal dataset. The order of display is exclusively defined by the order of `SetColumnShowName` definitions in the instructions. The first column for which a caption is defined is displayed first.

For more information about the `SetColumnShowName` instruction, see Defining Column Names and Captions with Instructions.

- `CreateDSInfo`: This instruction redefines the complete dataset. If `CreateDSInfo` is used, the columns resulting from the original query without instructions must also be redefined in the instruction to be displayed in the dataset. The `CreateDSInfo` instruction allows new columns to be displayed before or between columns resulting from the original query dataset.

For more information about the `CreateDSInfo` instruction, see Changing the Order and Number of Columns Displayed in a Report.

3) Define the content for the new column using instructions. For example,:

- Insert information about which object class or object class stereotype the objects in the report belong to. For more information, see Inserting a Column With Object Class Information in the Configured Report.

Complete examples regarding the definition of new columns are provided in the section Inserting a Column With Object Class Information in the Configured Report.

Adding A Single Column With the InsertColumn Instruction

To add a single column to the internal query result dataset at a defined position, add the following instruction to your query:

```
InsertColumn("Position","ColumnName");
```

Use the following parameter(s):

Code Parameter	Description
Position	The position of the new column in the internal query result dataset. The position is the number of the column after which the additional column is added. For example, if the position is set to 3, the column is added after the third column of the dataset. If the position is set to "0", the column is added before the first row of the report.  If the additional column is defined for a native SQL query-based report, the position must not be set to "0". The first column of a native SQL query report must define the REFSTR of the objects in the report for technical reasons and is excluded from display in the tabular report. The evaluation of the REFSTR and exclusion of the column is done after execution of instructions. Therefore, inserting the new column as first column of the report will cause the REFSTR information in the native SQL query to be displayed in the report and the mechanisms that depend on the REFSTR evaluation, like navigation to an object from the report, will not work.
ColumnName	The name of the column that shall be added to the internal dataset of query results that is generated in the background as a basis of report generation. The name is not identical to the column caption that is displayed in the result dataset of the report. The column caption must be defined separately with either a CreateDSInfo or SetColumnShowName instruction. Otherwise the column exists but is not displayed in the dataset displayed in the report.

Adding Multiple Columns with the AddColumns Instruction

To add multiple columns to the internal query result dataset, add the following instruction to your query:

```
AddColumns("ColumnName,ColumnName");
```

Use the following parameter(s):

Code Parameter	Description
ColumnName	The comma-separated list of the columns that shall be added to the internal dataset of query results that is generated in the background as a basis of report generation. A name must be defined for each column. The name is not identical to the column caption that is displayed in the result dataset of the report. The column caption must be defined separately with either a CreateDSInfo or SetColumnShowName instruction. Otherwise, the column will exist but is not displayed in the dataset displayed in the report.

Inserting a Column With Object Class Information in the Configured Report

If a report displays information about objects from different object classes within one column, For example, when the JoinColumns instruction is used in the report, the information about which object belongs to

which object class cannot be derived via the query because the object class name and caption are not a property of the object.

If a report displays information about objects from different object class stereotypes, the stereotype name can be included in the report because the property `Stereotype` of the object stores the stereotype name as string. The caption of the stereotype is defined via an XML object and cannot be derived by the query.

In the above mentioned use cases, you can add an additional column to the report using an `InsertColumn` or `AddColumns` command and additional instructions to display the class or stereotype information in the additional column.

You must do the following to add new columns displaying class or stereotype information to the dataset:

1) Add the column to the query result dataset that is formed during query execution prior to displaying the results on the Alfabet interface:

   - Use `AddColumns` to add multiple columns to the end of the query result dataset. The columns are added to the internal dataset of query results at the end of the dataset in the order of specification in the instruction. For information about the required syntax, see Adding Multiple Columns with the AddColumns Instruction.
   - Use `InsertColumn` to add one column at a defined position of the internal query result dataset. For information about the required syntax, see Adding A Single Column With the InsertColumn Instruction.

2) Define the column caption and the position of the column in the dataset displayed in the report with one of the following instructions:

   - `SetColumnShowName`: This instruction defines a caption for a new column. For each added column that shall be displayed in the results, a `SetColumnShowName` instruction must be added to the `Instructions` section. New columns are displayed at the end of the tabular report, regardless of their position in the internal dataset. The order of display is exclusively defined by the order of `SetColumnShowName` definitions in the instructions. The first column for that a caption is defined is displayed first.

For more information about the `SetColumnShowName` instruction, see Defining Column Names and Captions with Instructions.

   - `CreateDSInfo`: This instruction redefines the complete dataset. If `CreateDSInfo` is used, the columns resulting from the original query without instructions must also be redefined in the instruction to be displayed in the dataset. The `CreateDSInfo` instruction allows new columns before or between columns resulting from the original query dataset to be displayed.

For more information about the `CreateDSInfo` instruction, see Changing the Order and Number of Columns Displayed in a Report.

3) Define the content for the new column using instructions. The instructions to fill the column with data are described in this section. A complete configuration example is given for each instruction.

The class caption and stereotype caption are displayed in the translation for the language culture selected by the user viewing the report if a translation is provided via the mechanisms described in the section Localization and Multi-Language Support for the Alfabet Interface.

Displaying the Class Name and/or Class Caption of the Base Object Class

Use the instruction `SetClassNameByRow` to display the object class name of the base object of each row:

```
SetClassNameByRow("ColumnName");
```

Use the instruction `SetClassCaptionByRow` to display the object class caption of the base object for each row:

```
SetClassCaptionByRow("ColumnName");
```

Use the following parameter(s):

Code Parameter	Description
`ColumnName`	The name of the column inserted with an `InsertColumn` or `AddColumns` instruction to display the class name or caption.

For example, a report shall display all architecture elements assigned to projects and the user that is the authorized user responsible for the architecture element. One row in the report displays architecture elements. A project can include objects from different object classes as architecture elements. The column Architecture Element is defined via a `JoinColumns` instruction and displays objects from different object classes. The architecture elements are defined as base objects of the report to enable navigation to the objects with a `SetRowReference` instruction.

Two new columns are added to the result dataset with the `AddColumns` instruction to display the object class icon and caption for the base object of the result row. To display the caption of the object classes in one of the new columns, a `SetClassCaptionByRow` is used. To display the icon of the object class, the second new column is configured to display the object class name with a `SetClassName` instruction and `PictureAssignment` instructions are used to trigger display of the different names as icons.

A `CreateDSInfo` instruction is used to place the new columns between the `Project` and `Architecture Element` columns and to remove the `REFSTR` of the objects that was added to execute the `SetRowReference` instruction from the dataset displayed in the report.

The following Alfabet query would be implemented. Please note that the Alfabet query does not join all object classes that can be assigned to a project as architecture elements to enhance readability of the example. If you want to use the example as a template for a report, you should ensure that all object classes that might be assigned to one of your projects is included.

```
ALFABET_QUERY_500

FIND ProjectArch

InnerJoin Project ON ProjectArch.Project = Project.REFSTR
```



```
LeftJoin Application ON ProjectArch.Object = Application.REFSTR

LeftJoin ICTObject ON ProjectArch.Object = ICTObject.REFSTR

LeftJoin Domain ON ProjectArch.Object = Domain.REFSTR

LeftJoin OrgaUnit ON ProjectArch.Object = OrgaUnit.REFSTR

LeftJoin BusinessProcess ON ProjectArch.Object =
BusinessProcess.REFSTR

LeftJoin BusinessFunction ON ProjectArch.Object =
BusinessFunction.REFSTR

LeftJoin Person ON (Or Person.REFSTR = Application.ResponsibleUser
Person.REFSTR = BusinessFunction.ResponsibleUser Person.REFSTR =
Domain.ResponsibleUser Person.REFSTR = BusinessProcess.ResponsibleUser
Person.REFSTR = ICTObject.ResponsibleUser Person.REFSTR =
OrgaUnit.ResponsibleUser)

Instructions

JoinColumns("Application.Name,Application.Version","Application.Name",
" v. ");

JoinColumns("Application.Name,ICTObject.Name,BusinessProcess.Name,Doma
in.Name,OrgaUnit.Name,BusinessFunction.Name","Domain.Name", "");

JoinColumns("Application.REFSTR,ICTObject.REFSTR,BusinessProcess.REFST
R,Domain.REFSTR,OrgaUnit.REFSTR,BusinessFunction.REFSTR","Application.
REFSTR", "");

SetRowReference("Application.REFSTR");

AddColumns("ClassName,ClassCaption");

SetClassNameByRow("ClassName");

SetClassCaptionByRow("ClassCaption");

CreateDSInfo("Project.Name,ClassName,ClassCaption,Domain.Name,Person.N
ame","Project,Icon,Class,Architecture Element,Responsible User");

PICTUREASSIGNMENT(ClassName, EqualTo, "Application", Application);

PICTUREASSIGNMENT(ClassName, EqualTo, "ICTObject", ICTObject);

PICTUREASSIGNMENT(ClassName, EqualTo, "BusinessProcess",
BusinessProcess);

PICTUREASSIGNMENT(ClassName, EqualTo, "Domain", Domain);

PICTUREASSIGNMENT(ClassName, EqualTo, "OrgaUnit", OrgaUnit);

PICTUREASSIGNMENT(ClassName, EqualTo, "BusinessFunction",
BusinessFunction);

EndOfInstructions

QUERY_XML

<QueryDef>

<ShowProperty Type="Property" ClassName="Project" Name="Name" />

<ShowProperty Type="Property" ClassName="Application" Name="Name" />

<ShowProperty Type="Property" ClassName="Application" Name="Version"
/>

<ShowProperty Type="Property" ClassName="ICTObject" Name="Name" />
```



```
<ShowProperty Type="Property" ClassName="Domain" Name="Name" />
<ShowProperty Type="Property" ClassName="OrgaUnit" Name="Name" />
<ShowProperty Type="Property" ClassName="BusinessProcess" Name="Name"
/>
<ShowProperty Type="Property" ClassName="BusinessFunction" Name="Name"
/>
<ShowProperty Type="Property" ClassName="Person" Name="Name" />
<ShowProperty Type="Property" ClassName="Application" Name="REFSTR" />
<ShowProperty Type="Property" ClassName="ICTObject" Name="REFSTR" />
<ShowProperty Type="Property" ClassName="Domain" Name="REFSTR" />
<ShowProperty Type="Property" ClassName="OrgaUnit" Name="REFSTR" />
<ShowProperty Type="Property" ClassName="BusinessProcess"
Name="REFSTR" />
<ShowProperty Type="Property" ClassName="BusinessFunction"
Name="REFSTR" />
<SortProperty Type="Property" ClassName="Domain" Name="Name" />
<SortProperty Type="Property" ClassName="Project" Name="Name" />
<SortProperty Type="Property" ClassName="Person" Name="Name" />
</QueryDef>
```

The Alfabet query results in a report dataset displaying information about the object classes that the architecture elements belong to:

Project	Icon	Class	Architecture Element	Responsible User
Asset Management Rework	ICT	ICT Object	Asset Management	Customer
Business Analysis	ICT	ICT Object	BI Tools	Alfabet
Consolidate HR Systems	A	Application	AF HR Online EU v. 2.2 Var.	Customer
Consolidate HR Systems	A	Application	AF HR Online US v. 2.2 Var.	Customer
Consolidate HR Systems	A	Application	AF HR Online v. 2.2	Customer
Consolidate HR Systems	A	Application	AF HR Online v. 3.0	Customer
Consolidate HR Systems		Business Function	Perform Career Planning and Job Applica...	Customer
Consolidate HR Systems		Business Function	Perform Executive HR Analytics	Customer
Consolidate HR Systems		Business Function	Perform HR Management Controlling	Customer
Consolidate HR Systems		Business Process	Recruiting	Customer
Consolidate HR Systems	ICT	ICT Object	SAP R/3 Enterprise 4.7	Customer
Consolidate HR Systems		Business Function	Manage Employee Recruitment	Lee
Consolidate HR Systems		Business Process	Human Resources Management	Ngombe

Displaying the Class Name and/or Class Caption of an Object Class That Is Not the Base Class

If you want to add information about the object class that objects in the report belong to, the `REFSTR` of the objects must be added to the report data set. The information about the `REFSTR` is required to identify the object and evaluate which class the object belongs to.

The REFSTR, which added to the query results for technical reasons only, can be hidden from display in the tabular report by using a RemoveColumns instruction or by redefining the dataset with a CreateDSInfo instruction that does not include the REFSTR information.

Use the instruction SetClassName to display the object class name of the objects that are not the base object of the report:

```
SetClassName("SourceColumnName","TargetColumnName");
```

Use the instruction SetClassCaptionByRow to display the object class caption of objects that are not the base objects of the report:

```
SetClassCaption("SourceColumnName","TargetColumnName");
```

Use the following parameter(s):

Code Parameter	Description
SourceColumn-Name	The name of the column for the REFSTR of the objects for that the class information shall be added to the report. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions.
TargetColumn-Name	The name of the column inserted with an InsertColumn or AddColumns instruction to display the class name or caption.

For example, a report shall display all architecture elements assigned to projects and the user that is the authorized user responsible for the architecture element. There is one column in the report displaying architecture elements. A project can include objects from different object classes as architecture elements. The column Architecture Element is defined via a JoinColumns instruction and displays objects from different object classes. The architecture elements are not the base objects of the report. The base object class is the object class Project.

The REFSTR of the architecture elements is added to the report via the Show properties and joined into one column using a JoinColumns instruction.

Two new columns are added to the result dataset with the AddColumns instruction. A SetClassCaption instruction is used to display the caption of the class in one of the new columns. To display the icon of the object class, the second new column is configured to display the object class name with a SetClassName instruction and PictureAssignment instructions are used to trigger display of the different names as icons.

A CreateDSInfo instruction is used to hide the REFSTR information and to place the new columns between the Project and Architecture Element column.

The following Alfabet query would be implemented. Please note that the example of the Alfabet query does not join all object classes that can be assigned to a project as architecture elements to enhance readability. If you want to use the example as a template for a report, you should ensure that all object classes that might be assigned to one of your projects are included.

```
ALFABET_QUERY_500

FIND ProjectArch
```



```
InnerJoin Project ON ProjectArch.Project = Project.REFSTR

LeftJoin Application ON ProjectArch.Object = Application.REFSTR

LeftJoin ICTObject ON ProjectArch.Object = ICTObject.REFSTR

LeftJoin Domain ON ProjectArch.Object = Domain.REFSTR

LeftJoin OrgaUnit ON ProjectArch.Object = OrgaUnit.REFSTR

LeftJoin BusinessProcess ON ProjectArch.Object =
BusinessProcess.REFSTR

LeftJoin BusinessFunction ON ProjectArch.Object =
BusinessFunction.REFSTR

LeftJoin Person ON (Or Person.REFSTR = Application.ResponsibleUser
Person.REFSTR = BusinessFunction.ResponsibleUser Person.REFSTR =
Domain.ResponsibleUser Person.REFSTR = BusinessProcess.ResponsibleUser
Person.REFSTR = ICTObject.ResponsibleUser Person.REFSTR =
OrgaUnit.ResponsibleUser)

Instructions

JoinColumns("Application.Name,Application.Version","Application.Name",
" v. ");

JoinColumns("Application.Name,ICTObject.Name,BusinessProcess.Name,Doma
in.Name,OrgaUnit.Name,BusinessFunction.Name","Domain.Name", "");

JoinColumns("Application.REFSTR,ICTObject.REFSTR,BusinessProcess.REFST
R,Domain.REFSTR,OrgaUnit.REFSTR,BusinessFunction.REFSTR","Application.
REFSTR", "");

AddColumns("ClassName,ClassCaption");

SetClassName("Application.REFSTR","ClassName");

SetClassCaption("Application.REFSTR","ClassCaption");

CreateDSInfo("Project.Name,ClassName,ClassCaption,Domain.Name,Person.N
ame","Project,Icon,Class,Architecture Element,Responsible User");

PICTUREASSIGNMENT(ClassName, EqualTo, "Application", Application);

PICTUREASSIGNMENT(ClassName, EqualTo, "ICTObject", ICTObject);

PICTUREASSIGNMENT(ClassName, EqualTo, "BusinessProcess",
BusinessProcess);

PICTUREASSIGNMENT(ClassName, EqualTo, "Domain", Domain);

PICTUREASSIGNMENT(ClassName, EqualTo, "OrgaUnit", OrgaUnit);

PICTUREASSIGNMENT(ClassName, EqualTo, "BusinessFunction",
BusinessFunction);

EndOfInstructions

QUERY_XML

<QueryDef>

<ShowProperty Type="Property" ClassName="Project" Name="Name" />

<ShowProperty Type="Property" ClassName="Application" Name="Name" />

<ShowProperty Type="Property" ClassName="Application" Name="Version"
/>

<ShowProperty Type="Property" ClassName="ICTObject" Name="Name" />
```



```
<ShowProperty Type="Property" ClassName="Domain" Name="Name" />
<ShowProperty Type="Property" ClassName="OrgaUnit" Name="Name" />
<ShowProperty Type="Prope rty" ClassName="BusinessProcess" Name="Name"
/>
<ShowProperty Type="Property" ClassName="BusinessFunction" Name="Name"
/>
<ShowProperty Type="Property" ClassName="Person" Name="Name" />
<ShowProperty Type="Property" ClassName="Application" Name="REFSTR" />
<ShowProperty Type="Property" ClassName="ICTObject" Name="REFSTR" />
<ShowProperty Type="Property" ClassName="Domain" Name="REFSTR" />
<ShowProperty Type="Property" ClassName="OrgaUnit" Name="REFSTR" />
<ShowProperty Type="Property" ClassName="BusinessProcess"
Name="REFSTR" />
<ShowProperty Type="Property" ClassName="BusinessFunction"
Name="REFSTR" />
<SortProperty Type="Property" ClassName="Domain" Name="Name" />
<SortProperty Type="Property" ClassName="Project" Name="Name" />
<SortProperty Type="Property" ClassName="Person" Name="Name" />
</QueryDef>
```

The Alfabet query results in a report data set displaying information about the object classes that the architecture elements belong to:

Project	Icon	Class	Architecture Element	Responsible User
Asset Management Rework	ICT	ICT Object	Asset Management	Customer
Business Analysis	ICT	ICT Object	BI Tools	Alfabet
Consolidate HR Systems	A	Application	AF HR Online EU v. 2.2 Var.	Customer
Consolidate HR Systems	A	Application	AF HR Online US v. 2.2 Var.	Customer
Consolidate HR Systems	A	Application	AF HR Online v. 2.2	Customer
Consolidate HR Systems	A	Application	AF HR Online v. 3.0	Customer
Consolidate HR Systems		Business Function	Perform Career Planning and Job Applica...	Customer
Consolidate HR Systems		Business Function	Perform Executive HR Analytics	Customer
Consolidate HR Systems		Business Function	Perform HR Management Controlling	Customer
Consolidate HR Systems		Business Process	Recruiting	Customer
Consolidate HR Systems	ICT	ICT Object	SAP R/3 Enterprise 4.7	Customer
Consolidate HR Systems		Business Function	Manage Employee Recruitment	Lee
Consolidate HR Systems		Business Process	Human Resources Management	Ngombe

Displaying the Stereotype Caption and/or Icon of Objects Displayed in the Report

Stereotypes are defined in an XML stored in the attribute `Stereotype` of the respective object class. The stereotypes included in the definition can then be selected for each object. In the database table for the object class, the name of the stereotype assigned to an object is stored as a string in the column of the property `Stereotype` for the object class.

Object classes have both attributes and properties. Attributes are part of the meta-model definition. Most of the attributes of an object class are not editable but some can be changed in Alfabet Expand.

Properties define the columns of the database table for the object class. When you create an object of an object class on the Alfabet user interface, you define the values of the properties of the object class for that object.

Via a query, you can read the information about the stereotype name from the database table of the object class. But usually, the stereotype caption is displayed to the users in the Alfabet user interface and Therefore, it would be more convenient to display the stereotype caption in the configured report as well.

The instruction converting the stereotype name into the stereotype caption extracts the information about the object and the stereotype name of the object from the dataset resulting from the query. The dataset must provide the following information:

- The `REFSTR` of the object for that the stereotype information shall be displayed.
- The name of the stereotype assigned to the object.

The stereotype caption can be written into a new column that has been added to the report using an `AddColumns` or `InsertColumn` instruction. Alternatively, you can overwrite the stereotype name information. The additional column displaying the `REFSTR` of the object can be removed using a `RemoveColumns` instruction.

For more information about adding columns to a report dataset, see Inserting One or Multiple Columns to a Report.

For more information about removing columns from the report dataset, see Hiding a Column in a Configured Report.

Instead of or in addition to the stereotype caption, the icon defined for the stereotype via the class settings can be displayed in the configured report.

Use the following instruction to display the stereotype caption instead of the stereotype name in the data set displayed for a configured report:

```
SetStereotypeCaption("ReferenceColumn","StereotypeColumn","TargetColumn",RenderMode);
```

Use the following parameter(s):

Code Parameter	Description
`ReferenceColumn`	The name of the column containing the values of the `REFSTR` property of the objects for which the stereotype caption shall be displayed. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions.
`StereotypeColumn`	The name of the column containing the values of the `Stereotype` property of the objects for which the stereotype caption shall be displayed.

Code Parameter	Description
`TargetColumn`	The name of the column to which the stereotype caption shall be written. The target column can be added to the result dataset using an `AddColumns` or `InsertColumn` instruction. For more information, see Inserting One or Multiple Columns to a Report.
`RenderMode`	This parameter defines whether the stereotype caption or the icon defined for the stereotype are displayed. The following values can be set for `RenderMode`: • `Icon`: Only the icon is displayed. • `TextIcon`: Both the stereotype caption and the icon are displayed with the caption first. • `IconText`: Both the stereotype caption and the icon are displayed with the icon first. If `RenderMode` is not defined, only the stereotype caption will be displayed. If only an icon is displayed, a legend will be added to the report that explains which stereotype is represented by which icon.

For example, the following native SQL query results in a list of projects and displays the name and the stereotype of the projects:

```
SELECT proj.REFSTR, proj.NAME,proj.STEREOTYPE
FROM PROJECT proj
```

	NAME	STEREOTYPE
1	Consolidate Trading Applications	Program
2	Retire GL Applications	Project
3	Upgrade GenLManager	Project
4	Reshape Core Trading Applications	Project
5	Enhance TradeNet	Project
6	Implement Unified Trade Solution	Project
7	UTS phase 1	ProjectStep
8	UTS phase 2	ProjectStep
9	UTS phase 3	ProjectStep
10	Streamline CRM Applications	Program
11	Migrate CRM Opti Retail to CRM CSS	Project

To display the stereotype caption next to the stereotype name, the following instructions can be added to the query:

- An `InsertColumn` instruction that inserts a new column to the dataset. Please note that the column must be added to the dataset after the first column returning the `REFSTR` if the instruction is used in combination with native SQL.

- A `SetColumnShowName` instruction to trigger display of the new column in the result dataset and to define a caption for the column.

The column inserted via the `InsertColumn` instruction is displayed at the end of the dataset. New columns are always displayed at the end of the dataset regardless of the position defined in the `InsertColumn` instruction. If you want to display the column at any other position, use the `CreateDSInfo` instruction to re-define the complete dataset instead of triggering column display with the `SetColumnShowName` instruction.

- A `SetStereotypeCaption` instruction to display the stereotype caption in the new column.

The object reference is read from the first column of the report that is hidden in datasets resulting from native SQL. Therefore, it is not required to hide the information via a `RemoveColumns` instruction.

The following instructions are added to the example query:

```
InsertColumn("1","StereotypeCaption");
SetColumnShowName("StereotypeCaption","Stereotype Caption");
SetStereotypeCaption("REFSTR","STEREOTYPE", "StereotypeCaption");
```

resulting in the following report:

	NAME	STEREOTYPE	Stereotype Caption
1	Consolidate Trading Applications	Program	Program
2	Retire GL Applications	Project	Project
3	Upgrade GenLManager	Project	Project
4	Reshape Core Trading Applications	Project	Project
5	Enhance TradeNet	Project	Project
6	Implement Unified Trade Solution	Project	Project
7	UTS phase 1	ProjectStep	Project Step
8	UTS phase 2	ProjectStep	Project Step
9	UTS phase 3	ProjectStep	Project Step
10	Streamline CRM Applications	Program	Program
11	Migrate CRM Opti Retail to CRM CSS	Project	Project

Alternatively, you can display the stereotype caption instead of the stereotype name by doing one of the following:

- Add a `RemoveColumn` instruction to the instruction configuration displayed above that removes the column displaying the stereotype name from the result dataset:

  ```
  RemoveColumns("STEREOTYPE");
  ```

- Instead of defining the set of instructions displayed above, define one `SetStereotypeCaption` instruction that has an identical `StereotypeColumn` and `TargetColumn` definition, Therefore, overwriting the information about the stereotype name with the information about the stereotype caption in the result dataset:

  ```
  SetStereotypeCaption("REFSTR","STEREOTYPE", "STEREOTYPE");
  ```


Inserting Columns with Information About the Class Icon And Class Colors

The instruction `GetClassSettingInfo` allows information about the object class icon and coloring defined in the class settings of an object class to be returned in a configured report.

This information can For example, be used to color a configured graphic report that includes graphic representations of objects of multiple different classes. For user profiles, individual class settings with different colors and icons representing an object class can be defined. If the coloring and icon setting is derived from the class settings, the coloring and icons displayed for the graphics will automatically correspond to the coloring and icon that is used in general for the respective object class for the user profile the user is currently logged in with when opening the report.

The `GetClassSettingInfo` instruction requires the following preconditions to be met in the dataset of the query it is applied on:

- The dataset must contain a column that returns the object class name of the object class the current object belongs to. For information about how to add the object class name to the dataset, see Inserting a Column With Object Class Information in the Configured Report.
- Optionally, the stereotype can be taken into account. The dataset must then contain a column that returns the name of the stereotype of the object. The stereotype is stored in the object class property `Stereotype` of the object.
- For each information returned, an empty target column of the data type string must be included into the dataset. The instruction can return the icon name, foreground color and background color defined in the class settings. Whether all or only a subset of the information is returned, can be defined in the instruction. for information about how to add empty columns to the dataset, see .Inserting One or Multiple Columns to a Report.

After having configured the query to return the required dataset, the following instruction can be added to return foreground color, background color and icon defined in the class settings:

```
GetClassSettingInfo(ClassNameCol,StereotypeNameCol,TargetIconNameCol,TargetF
oreColorCol,TargetBackColorCol);
```

Use the following parameters:

Code Parameter	Description
`ClassNameCol`	The name of the column returning the object class name. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions. For information about how to add the object class name to the dataset, see Inserting a Column With Object Class Information in the Configured Report.
`Stereo-typeNameCol`	The name of the column returning the object class stereotype name. This definition is optional.

Code Parameter	Description
`TargetIcon-NameCol`	The name of the empty column in the result dataset that shall contain the information about the name of the icon of the object class. The data type of the empty column must be String. For information about how to add an empty column to the dataset, see Inserting One or Multiple Columns to a Report.
`TargetForeCol-orCol`	The name of the empty column in the result dataset that shall contain the information about the foreground color of the object class. The data type of the empty column must be String. For information about how to add an empty column to the dataset, see Inserting One or Multiple Columns to a Report.
`TargetBackCol-orCol`	The name of the empty column in the result dataset that shall contain the information about the background color of the object class. The data type of the empty column must be String. For information about how to add an empty column to the dataset, see Inserting One or Multiple Columns to a Report.

Most of the information that can be added to the instruction is optional. Please note that the columns are identified by their position in the comma separated list. If you do not want to define a column, provide no text in the position of the comma separated list.

For example, the instruction:

```
GetClassSettingInfo(ClassName,ClassStereotype,Target1,Target2,)
```

will return the object class icon name in the column `Target1` and the foreground color in the column `Target 2`, whereas the instruction

```
GetClassSettingInfo(ClassName,ClassStereotype,Target1,Target2)
```

will return the object class fore color in the column `Target1` and the background color in the column `Target2`.

The following Alfabet query returns a dataset that includes the information about the icon, foreground color and background color of the object classes that the objects in the as-is architecture of a selected project belong to. The query could be used in a graphic report to define coloring and icon display via the query that finds the object to be displayed in the graphic:

```
ALFABET_QUERY_500
FIND ProjectArch
InnerJoin Project ON ProjectArch.Project = Project.REFSTR
LeftJoin Application ON ProjectArch.Object = Application.REFSTR
LeftJoin ICTObject ON ProjectArch.Object = ICTObject.REFSTR
LeftJoin Domain ON ProjectArch.Object = Domain.REFSTR
```



```
LeftJoin OrgaUnit ON ProjectArch.Object = OrgaUnit.REFSTR

LeftJoin BusinessProcess ON ProjectArch.Object =
BusinessProcess.REFSTR

LeftJoin BusinessFunction ON ProjectArch.Object =
BusinessFunction.REFSTR

WHERE

   Project.REFSTR =:BASE

Instructions

   JoinColumns("Application.Name,Application.Version","Application.Nam
   e", " v. ");

   JoinColumns("Application.Name,ICTObject.Name,BusinessProcess.Name,D
   omain.Name,OrgaUnit.Name,BusinessFunction.Name","Domain.Name", "");

   JoinColumns("Application.REFSTR,ICTObject.REFSTR,BusinessProcess.RE
   FSTR,Domain.REFSTR,OrgaUnit.REFSTR,BusinessFunction.REFSTR","Applic
   ation.REFSTR", "");

   JoinColumns("Application.Stereotype,ICTObject.Stereotype,Domain.Ste
   reotype,OrgaUnit.Stereotype","Application.Stereotype", "");

   SetRowReference("Application.REFSTR");

   AddColumns("ClassName,ClassIcon,ClassForeColor,ClassBackColor");

   SetClassNameByRow("ClassName");

   GetClassSettingInfo(ClassName,Application.Stereotype,ClassIcon,Clas
   sForeColor,ClassBackColor);

   CreateDSInfo("Project.Name,Domain.Name,ClassName,Application.Stereo
   type,ClassIcon,ClassForeColor,ClassBackColor","Project,Architecture
   Element,Class,Stereotype,Icon,Foreground Color, Background Color");

EndOfInstructions

QUERY_XML

<QueryDef>

   <ShowProperty Type="Property" ClassName="Application" Name="Name"
   />

   <ShowProperty Type="Property" ClassName="Application"
   Name="Version" />

   <ShowProperty Type="Property" ClassName="ICTObject" Name="Name" />

   <ShowProperty Type="Property" ClassName="Domain" Name="Name" />

   <ShowProperty Type="Property" ClassName="OrgaUnit" Name="Name" />

   <ShowProperty Type="Property" ClassName="BusinessProcess"
   Name="Name" />

   <ShowProperty Type="Property" ClassName="BusinessFunction"
   Name="Name" />

   <ShowProperty Type="Property" ClassName="Application" Name="REFSTR"
   />

   <ShowProperty Type="Property" ClassName="ICTObject" Name="REFSTR"
   />

   <ShowProperty Type="Property" ClassName="Domain" Name="REFSTR" />
```



```
    <ShowProperty Type="Property" ClassName="OrgaUnit" Name="REFSTR" />
    <ShowProperty Type="Property" ClassName="BusinessProcess"
    Name="REFSTR" />
    <ShowProperty Type="Property" ClassName="BusinessFunction"
    Name="REFSTR" />
    <ShowProperty Type="Property" ClassName="Application"
    Name="Stereotype" />
    <ShowProperty Type="Property" ClassName="ICTObject"
    Name="Stereotype" />
    <ShowProperty Type="Property" ClassName="Domain" Name="Stereotype"
    />
    <ShowProperty Type="Property" ClassName="OrgaUnit"
    Name="Stereotype" />
    <SortProperty Type="Property" ClassName="Domain" Name="Name" />
    <SortProperty Type="Property" ClassName="Project" Name="Name" />
</QueryDef>
```

In a tabular representation, the following dataset is returned by the example query:

Architecture Element	Class	Stereotype	Icon	Foreground Color	Background Color
Balance Analysis DB v. 1.2.2	Application	Application	Application	#333333	#FEDAB3
CCM Management Tool v. 2.0	Application	Application	Application	#333333	#FEDAB3
CRM v. 2.6	Application	Application	Application	#333333	#FEDAB3
Customer Management&Storage v. 5.1	Application	Application	Application	#333333	#FEDAB3
SAS Enterprise Guide v. 1.0	Application	Application	Application	#333333	#FEDAB3
Links v. 1	Application	Application	Application	#333333	#FEDAB3
salesforce.com v. 9	Application	Application	Application	#333333	#FEDAB3
CRM CSS	ICTObject	ICTObject	ICTObject	#333333	#DAB8D8
CRM Opti Retail	ICTObject	ICTObject	ICTObject	#333333	#DAB8D8
CRM AI	ICTObject	ICTObject	ICTObject	#333333	#DAB8D8
OR Marketing Solution	ICTObject	ICTObject	ICTObject	#333333	#DAB8D8
CRM	ICTObject	ICTObject	ICTObject	#333333	#DAB8D8
OR Strategy, Marketing & Sales	OrgaUnit	OperatingEntity	OrgaUnit	Black	#F1E68C
Contact Management	Domain	BusinessCapability	ARIS_BusinessCapability	#333333	#E6FFC0
Customer Management	Domain	BusinessCapability	ARIS_BusinessCapability	#333333	#E6FFC0
Delivery	Domain	BusinessArea	ARIS_BusinessArea	#333333	#E6FFC0

Displaying the Object Icon

The instruction `SetObjectIcon` allows object icons to be displayed instead of or in combination with string values. The instruction returns the object icon for the object found in the current row of the query dataset. The following rule apply for the setting of the object icon:

- If an icon is assigned to the object via the standard property Icon, this icon is displayed.
- If no icon is assigned to the object and the object is based on a stereotype, the icon defined for the object class stereotype in the class settings of the object class stereotype is displayed.

- If no icon is assigned to the object and the object is not based on a stereotype, the icon defined in the class settings of the object class is displayed.

To display the object icon of the row reference object, add the following instruction to your query:

```
SetObjectIcon(ColumnName,IconPosition);
```

Use the following parameter(s):

Code Parameter	Description
`ColumnName`	The column containing the value that should be replaced by or amended with the object icon. The returned data type must be string. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions.
`IconPosition`	The icon can either be displayed instead of the string or in combination with the string. Allowed values for the icon position are: • `Icon`: The icon is displayed instead of the string. • `TextIcon`: The string is displayed in front of the icon. • `IconText`: The icon is displayed in front of the string. This parameter is optional. If it is not specified, the icon is displayed instead of the string.

Displaying Graphics for Cells Containing Specific Data

The instruction `PictureAssignment` allows values to be displayed via icons rather than or in combination with text or numeric values. For example, indicator values can be represented by icons. Or, if a report contains a column displaying the object class caption of the object found by the query with the instruction `InsertClassCaptionColumn`, the object class icon can be displayed rather than the object class caption.

In the legend of the report, the pictures are displayed with the column caption and the condition for picture assignment or with a text defined in the instruction.

To display an icon instead of or together with text or numbers in a column, add the following instruction to your query:

```
PictureAssignment(ColumnName,ConditionString,"Argument",IconPosition:IconNam
e,"LegendText", "IconAltText",Alignment);
```

Use the following parameter(s):

Code Parameter	Description
`ColumnName`	The column containing the value that constitutes the icon. For the `PictureAssignment` instruction, this is the column for which the cell icon is specified. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions.
`ConditionString`, "*Argument*"	The condition that the cell content must match to display the icon is defined by a condition string like `EqualTo` or `Contains`, and an argument. The specification of the argument is mandatory. The content of the argument depends on the condition string. All allowed filter condition strings and the argument specification required for the condition string are listed in the context of instructions for color assignments in the section Filter Condition Strings for Instructions. The argument must be written in quotation marks.
`IconPosition`	If the icon should be included into the cell in combination with the text or numeric value, this parameter defines whether the icon is located before or after the text. Allowed values are: • `TextIcon`: The text is displayed in front of the icon. • `IconText`: The icon is displayed in front of the text. This parameter is optional. If it is not specified, the icon is displayed instead of the text or numeric value. If the parameter is not specified, the `IconName` must be specified without the colon in front.
`IconName`	Name of an icon available in the icon gallery of Alfabet. For more information about the icon gallery see the section Adding and Maintaining Icons for the Alfabet Interface. If you define the instruction in the **Alfabet Query Builder**, you can select the icon from a drop-down list of available icons.
`LegendText`	The text that shall be displayed for the icon in the legend of the report. This parameter is optional. If no legend text is defined, the icons will be displayed with the column caption and the condition for picture assignment.
`IconAltText`	The text that will be displayed instead of the icon if the icon is not available. The text will also be read as description for the icon if a screen reader software is used. If the **Legend Item Text** shall not be defined but the **Icon Alt Text** shall be defined, you must enter an empty **Legend Item Text** to the instruction.
`Alignment`	The alignment of the icon in the cell. This parameter is only evaluated if `IconPosition` is set to `Icon` which means that only the icon is displayed without text. Allowed values are `Left`, `Center`, and `Right`. By default, the icon is center aligned.

Note the following when defining `PictureAssignment` instructions:

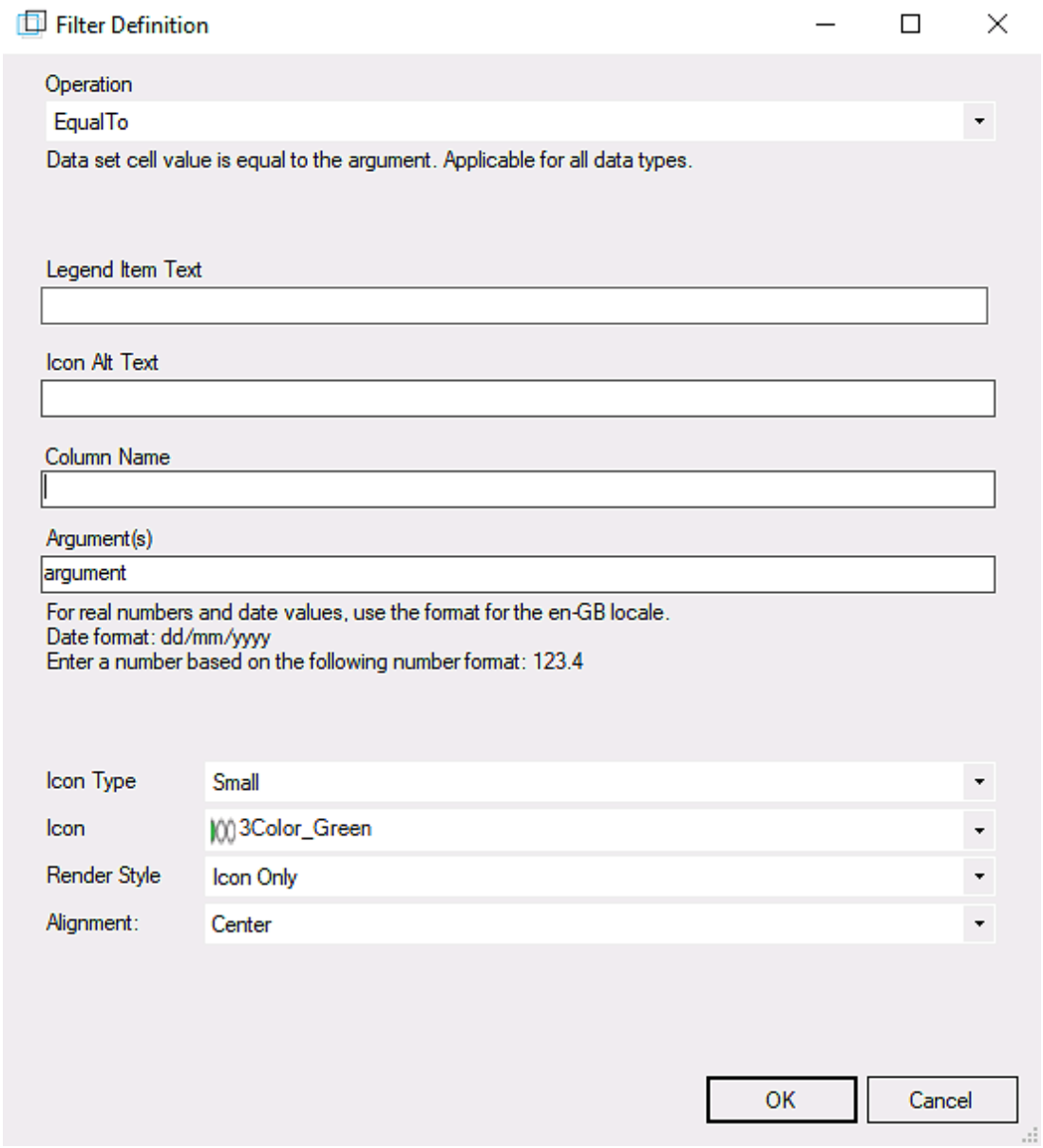
- If you do not specify optional parameters in the middle of the list of parameters, you must add a double comma between the previous and the next argument to maintain the order of parameters in the instruction.
- `PictureAssignment` instructions are executed in the order of their specification. If you define two `PictureAssignment` instructions that shall display a different icon in the same cell (For example, a specification that all numbers less than 3 are substituted with a red icon and another one that defines that all numbers greater than 2 are substituted with a red icon), the second instruction will overwrite the first one for the cells that correspond to both instructions.
- `PictureAssignment` instructions are not evaluated when testing an Alfabet query with the **Test Query**  button in the **Alfabet Query Builder** interface. Close the **Alfabet Query Builder** and use the **Test Report** functionality of the context menu of the configured report to see the result of your instruction configuration.
- If you combine `PictureAssignment` instructions with a `GroupBy_Ex` instruction, the grouping will split the results in one row into two rows, one containing the data for the superordinate object and one containing the data for the subordinate data. The empty cells in each resulting row are set to `NULL`, but color instructions are maintained and executed when the report is rendered in the client interface.

If you define a `PictureAssignment` instruction with an `IsNull` operator, the picture is displayed in the empty cells in all rows. For example, if the `PictureAssignment` instruction is defined to display an icon in a cell when a property of the subordinate class is set to `Null`, the respective empty cells in the rows containing the information about the superordinate class also display the icon. To avoid this problem, the query of the report shall return a value like "Not set" when the property of the subordinate class is set to `Null` and the picture assignment can than be applied to the string value instead of using an `IsNull` operator.

- If you combine `PictureAssignment` instructions with a `JoinColumns` instruction, the content of the source cells is concatenated and placed into the target cell. The picture assignment defined for the target cell is maintained and applied to the concatenated content when rendering the report.
- If you change the column caption with a `SetColumnShowName` instruction, the change is reflected in the legend of the picture assignment.

In the **Alfabet Query Builder**, `PictureAssignment` instructions can be defined via an editor:

1) In the **Instructions** tab of the **Alfabet Query Builder**, click the **New Instruction** button and select **Formatting Instructions** > `PictureAssignment` from the sub-menu. An editor opens.



2) Define the condition for icon display in the **Filter Definition** editor:

- **Column Name**: Enter the name of the column for which values shall be substituted with icons. The column name is specified by a combination of the attributes `ClassAlias` and `Name` in the ***ShowProperty*** XML element defining the column, delimited with a period.
- **Operation**: The condition that the cell content must match to display the icon is defined by a condition string like `EqualTo` or `Contains`, and an argument. Select the condition from the drop-down list. An overview of the conditions available for the instruction is given in the table above.
- **Argument(s)**: Define the argument(s) for the condition that the cell content must match to display the icon. The specification of arguments depend on the selection in the field **Operation**. An overview of the operations and argument specifications required by the argument are given in the table above.
- **Icon Type**: Select from the drop-down list whether the icon that you want to display belongs to the `Small`, `Large` or `Free` icon type.

- **Icon**: Select the icon displayed in all cells matching the defined condition. The drop-down list displays all icons of the icon type defined with the parameter **Icon Type**.
- **Render Style**: Select one of the following:
  - Icon Only: The icon will be displayed instead of the text.
  - Text Followed by Icon: Both text and icon will be displayed and the text is written in front of the icon.
  - Icon Followed by Text: Both text and icon will be displayd and the icon is displayed in front of the text.
- **Legend Item Text**: Define a text that will be displayed in the legend to explain the meaning of the icon. In the legend, the column caption and the defined text is displayed for the selected icon. If no legend text is defined, the icons will be displayed with the column caption and the condition for picture assignment.
- **Icon Alt Text**: Define a text that will be displayed instead of the icon if the icon is not available. The text will also be read as description for the icon if a screen reader software is used.
- **Alignment**: Select whether the icon should be center, left or right aligned in the cell of the dataset. This parameter is only evaluated if the **Render Style** parameter is set to `Icon Only`.

3) Click **OK** to save your settings. The resulting instruction is written into the text field of the **Instructions** tab.

Adding Graphic Icons from the Icon Gallery to a Tabular Report

An icon from any of the icon galleries can be displayed in cells of a configured tabular reports instead of or in addition to a string. The icon and all related information must be returned by the query of the configured report as specified below and the following instruction must be added to the query definition:

```
DynamicPictureAssignment(ColumnName,IconColumn,LegendItemTextColumn,RenderStyleColumn,AlignmentColumn,AltTextColumn);
```

Use the following parameters:

Code Parameter	Description
`ColumnName`	The name of the column in the result dataset the icon shall be displayed in either in addition to the cell content or in front of the cell content. This parameter is mandatory.
`IconColumn`	The name of the column in the result dataset returning the specification of the icon that shall be displayed. The icon must have been uploaded to the icon gallery. The icon must be defined as: `IconGalleryName:IconName`

Code Parameter	Description
	`IconGalleryName` must be `Small` (22x22), `Large`, or `Free`. The icon name is the name of the icon without extension as displayed in the icon gallery view. This parameter is mandatory.
`LegendItemTextColumn`	The name of the column in the result dataset returning the text to be displayed in the legend for the defined icon. The legend will show one entry for each differing combination of legend text and icon. The parameter is optional. If it is not defined, no legend will be displayed.
`RenderStyleColumn`	The name of the column in the result dataset returning whether the icon shall be displayed instead of or in addition to the cell content. This parameter is optional. The return value can be one of the following: • `Icon`: The icon will be displayed instead of the text. This is the default value. • `IconText`: the icon will be displayed in front of the text. • `TextIcon`: The icon will be displayed after the text.
`AlignmentColumn`	The name of the column in the result dataset returning the alignment of the icon or the icon and text in the cell it is displayed in. By default, the cell content is left aligned. Allowed return values are `Left`, `Center`, and `Right`.
`AltTextColumn`	The name of the column in the result dataset returning a text that will be displayed instead of the icon if the icon is not available. The text will also be read as description for the icon if a screen reader software is used.

Please note the following about the definition of the instruction:

- If you do not specify optional parameters in the middle of the list of parameters, you must add a double comma between the previous and the next argument to maintain the order of parameters in the instruction.
- Columns added to the report dataset to provide the picture rendering definition can be removed from the visible dataset via a `RemoveColumns` instruction.
- `PictureAssignment` and `DynamicPictureAssignement` instructions are executed in the order of their specification. If you define two instructions that shall display a different icon in the same cell, the second instruction will overwrite the first one for the cells that correspond to both instructions.
- If you combine `DynamicPictureAssignment` instructions with a `GroupBy_Ex` instruction, the grouping will split the results in one row into two rows, one containing the data for the superordinate object and one containing the data for the subordinate data. The empty cells in each resulting row are set to `NULL`, but picture instructions are maintained and executed when the report is rendered in the client interface.

- If you combine a `DynamicPictureAssignment` instruction with a `JoinColumns` instruction, the content of the source cells is concatenated and placed into the target cell. The picture assignment defined for the target cell is maintained and applied to the concatenated content when rendering the report.

Defining Coloring for Cells or Rows Containing Specific Data

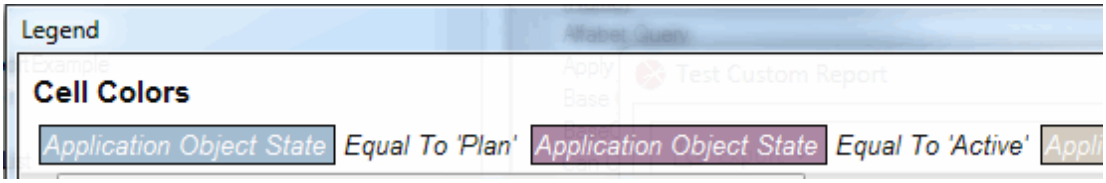
You can define colors to fill cells or entire rows in tabular reports based on a query. Colors are displayed in all cells or rows in the configured report that are found by a condition. A legend is displayed for the report that gives information about the color setting.

For example, a report is configured to display applications. The object state of the application shall be highlighted by cell coloring. There are two ways to highlight the object state:

- The color of the cells in the column containing the information about the object state depends on the application's current object state:

Application Name	Application Version	Application Object State
Business EAI Platform	2.2	Active
Groupware Services	2.2	Active
Mafo-Portal	2.6	Active
PS Global	2.5	Active
ALLFiance PISA	2.9	Retired
SAP@OptiRetail	2.0	Active
SAP@AI	4.0	Active
BookIT	2.9	Retired
Corporate FI-CO	2.2	Active
SAP International	2.8	Active
compas	2.2	Active
Electronic Personal Files	2.5	Active
Opti-SAP HR	2.X	Active
IT-Reporting Tool (SAP B	2.X	Retired
Prodi	2.7	Active
eLead	2.0	Active
KSK	2.0	Retired
CRM CSS	3.2	Active
CRM CSS	3.3	Plan
Compass	1.0	Active
SAP	4.6 c	Active

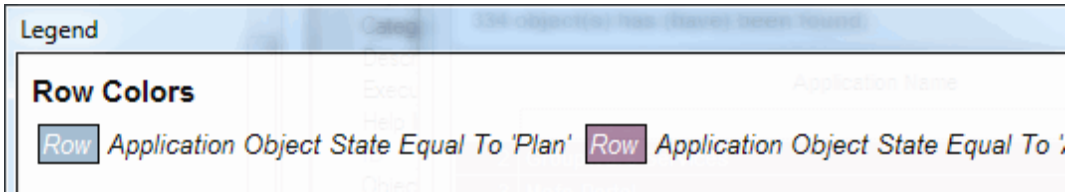
The legend displays all colors with the operator and argument leading to the specific coloring. The caption of the column is displayed in the colored field. If you do not like this information to be displayed in the legend, you can define a legend text in the instruction that will overwrite the default text.



- The color of the rows in the report depends on the application's current object state:

Application Name	Application Version	Application Object State
Business EAI Platform	2.2	Active
Groupware Services	2.2	Active
Mafo-Portal	2.6	Active
PS Global	2.5	Active
ALLFiance PISA	2.9	Retired
SAP@OptiRetail	2.0	Active
SAP@AI	4.0	Active
BookIT	2.9	Retired
Corporate FI-CO	2.2	Active
SAP International	2.8	Active
compas	2.2	Active
Electronic Personal Files	2.5	Active
Opti-SAP HR	2.X	Active
IT-Reporting Tool (SAP B	2.X	Retired
Prodi	2.7	Active
eLead	2.0	Active
KSK	2.0	Retired
CRM CSS	3.2	Active
CRM CSS	3.3	Plan
Compass	1.0	Active
SAP	4.6 c	Active

The legend displays all colors with the column caption, the operator and the argument leading to the specific coloring. In the colored field, Row is displayed to indicate that the color resulting from the data in one column is responsible for coloring of a complete row. If you do not like this information to be displayed in the legend, you can define a legend text in the instruction that will overwrite the default text.



The cell coloring based on a condition can be defined by adding one of the following instructions to the query:

- To color cells in one column based on the cell's content:

```
ColorAssignment(ColumnName,ConditionString,"Argument",TextColor,CellColor
,"LegendText");
```

- To color rows based on the content of the row's cell in a defined column:

```
RowColorAssignment(ColumnName,ConditionString,"Argument",TextColor,CellCo
lor,"LegendText");
```

Use the following parameter(s):

Code Parameter	Description
`ColumnName`	The column containing the value that constitutes the coloring. For the `ColorAssignment` instruction, this is the column for which the cell color is specified. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions.
`ConditionString`, "*Argument*"	The condition that the cell content must match to apply the coloring is defined by a condition string like `EqualTo` or `Contains`, and an argument. The specification of the argument is mandatory. The content of the argument depends on the condition string. All allowed filter condition strings and the argument specification required for the condition string are listed in the section Filter Condition Strings for Instructions. The argument must be written in quotation marks.
`TextColor`	HTML compliant definition of the color of the text in the cell matching the defined condition. For example, the color white can be defined as `White` or `#FFFFFF`.
`CellColor`	HTML compliant definition of the color of the background the cell matching the defined condition. For example, the color white can be defined as `White` or `#FFFFFF`.
`LegendText`	The text that shall be displayed for the color in the legend of the report. This parameter is optional. If no legend text is displayed, the color will be displayed with the column caption and the condition for picture assignment.

Note the following when defining `ColorAssignment` and `RowColorAssignment` instructions:

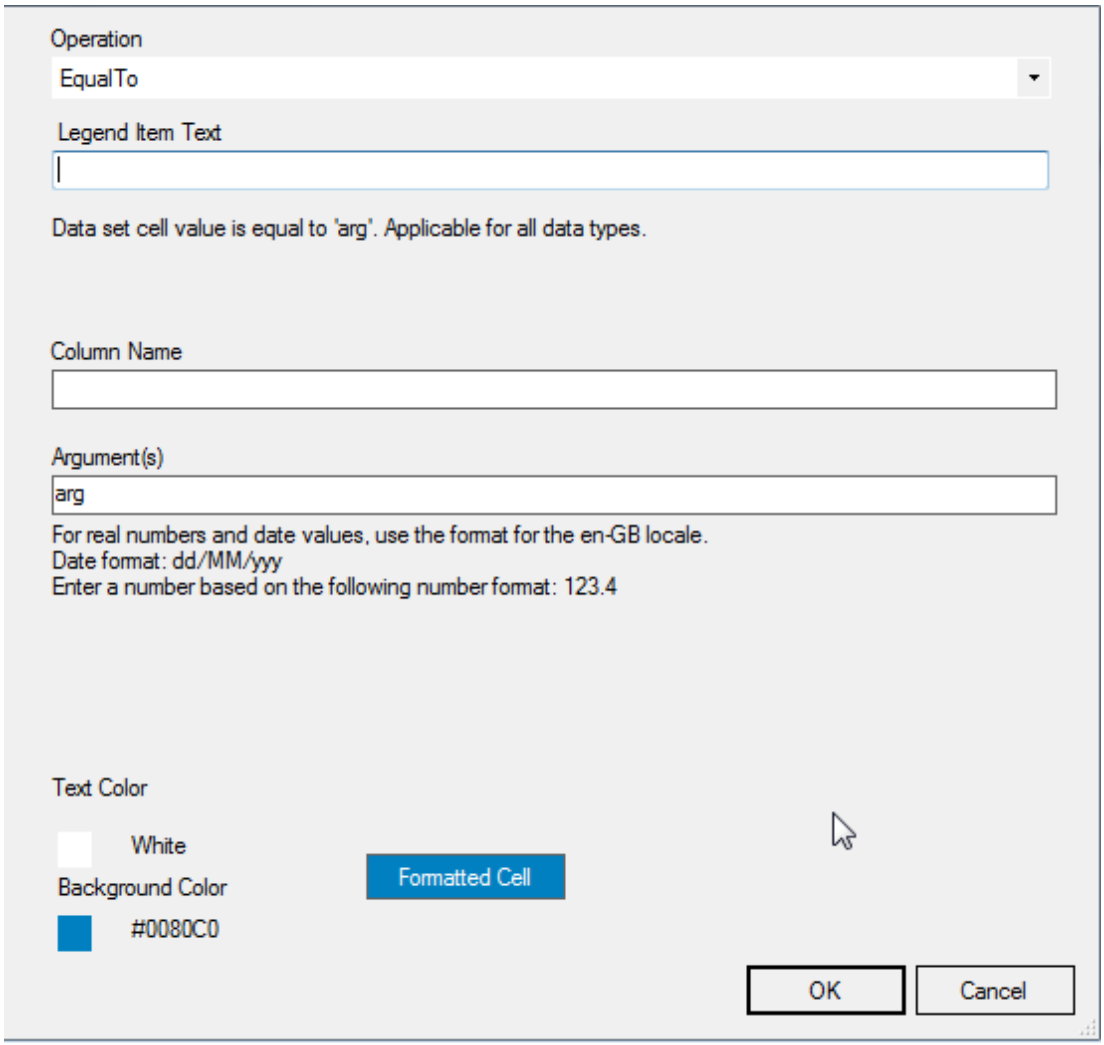
- `ColorAssignment` instructions supersede `RowColorAssignment` instructions independent from the order of definition.
- An additional instruction `FontStyleColorAssignment` is available to change both the coloring of a cell and the formatting of the text in the cell. For more information about this instruction see Defining Text Formatting for Cells Containing Specific Data.
- If you define two `ColorAssignment` instructions that shall color the same cell in different manners (For example, a specification that all numbers less than 3 are colored in blue and another one that defines that all numbers greater than 2 are colored in green), the second instruction overwrites the first one for the cells that correspond to both instructions.
- If you combine color assignments with a `GroupBy_Ex` instruction, the grouping will split the results in one row into two rows, one containing the data for the superordinate object and one containing the data for the subordinate object. The empty cells in each resulting row are set to `NULL`, but color instructions are maintained and executed when the report is rendered in the client interface.

If you define a color assignment instruction with an `IsNull` operator, the color assignment colors the empty cells in all rows with the defined color. For example, if the color assignment is defined to color cells when a property of the subordinate class is set to `Null`, the respective empty cells in the rows containing the information about the superordinate class are also colored. To avoid this problem, the query of the report shall return a value like "Not set" when the property of the subordinate class is set to `Null` and the coloring can than be applied to the string value instead of using an `IsNull` operator.

- If you combine color assignments with a `JoinColumns` instruction, the content of the source cells is concatenated and placed into the target cell. The color rule defined for the target cell is maintained and applied to the concatenated content when rendering the report.
- If you change the column caption with a `SetColumnShowName` instruction, the change is reflected in the legend of the color rules.

In the **Alfabet Query Builder**, color assignments can be defined via an editor:

1) In the **Instructions** tab of the **Alfabet Query Builder**, click the **New Instruction** button and select either **Formatting Instructions** > `ColorAssignment` or **Formatting Instructions** > `RowColorAssignment` in the sub-menu. An editor opens.



2) Define the condition for cell coloring in the **Filter Definition** editor:
   - **Column Name**: Enter the name of the column containing the value that constitutes the coloring. For the `ColorAssignment` instruction, this is also the column that will be colored. The column name is specified by a combination of the attributes `ClassAlias` and `Name` from the ***ShowProperty*** XML element defining the column, delimited with a period.
   - **Operation**: The condition that the cell content must match to apply the coloring is defined by a condition string like `EqualTo` or `Contains`, and an argument. Select the condition from the

drop-down list. An overview of the conditions available for the instruction is given in the table above.

- **Argument(s)**: Define the argument(s) for the condition that the cell content must match to apply the coloring. The specification of arguments depend on the selection in the filed **Operation**. An overview of the operations and argument specifications required by the argument are given in the table above.

3) Define the cell color and the text color that shall be used for cells or rows with a content matching the defined condition:

- **Text Color** / **Background Color**: Click on the colored rectangle to change the text color or cell background color respectively. A color selector opens. You can select one of the colors displayed in the selector by clicking the color field or define a new color by clicking the **Define Custom Color** button. The new color can either be selected from the colors displayed in the top area or defined as RGB color value. Click the **Add as New Color** button to add the color to the set of colors in the standard selector.

After color selection click **OK** to apply your selection. The selected color is displayed in the **Filter Definition** editor and the preview on the right of the color selection fields displays the resulting color format.

4) Optionally, define a text to be displayed for the color in the legend in the field **Legend ItemText**. If you do not define a text, the column name containing the data and the condition causing the coloring will be displayed in the legend with the color.

5) Click **OK** to save your settings. The resulting instruction is written into the text field of the **Instructions** tab.

Filter Condition Strings for Instructions

The following conditions can be specified for instructions that assign color to cells and rows of the report. The same conditions are also applicable to instructions defining cells that shall display a graphic icon. See Displaying Graphics for Cells Containing Specific Data for more information about instructions defining graphic cell content.

Condition	Description and Example
`Contains`	Data set cell value has the argument as part of the string. Applicable for columns of type `String` only. `ColorAssignment(Application.Name,Contains,"Trade",White,#336699);`
`DoesNotContain`	Data set cell value does not have the argument as part of the string. Applicable for columns of type `String` only. `ColorAssignment(Application.Name,DoesNotContain,"Trade",White, #336699);`
`StartsWith`	Data set cell value starts with the argument. Applicable for columns of type `String` only.

Condition	Description and Example
	`ColorAssignment(Application.Name,StartsWith,"Trade",White, #336699);`
`EndsWith`	Data set cell value ends with the argument. Applicable for columns of type `String` only. `ColorAssignment(Application.Name,EndsWith,"Trade",White, #336699);`
`EqualTo`	Data set cell value is equal to the argument Applicable for all column types. `ColorAssignment(Application.ObjectState,EqualTo,"Active",White, #336699);`
`NotEqualTo`	Data set cell value is not equal to the argument. Applicable for all column types. `ColorAssignment(Application.ObjectState,NotEqualTo,"Active",White, Red);`
`GreaterThan`	Data set cell value is greater than the argument. Applicable for column types `Real`, `Integer`, or `Date`. `ColorAssignment(Application.StartDate,GreaterThan,"13.02.2008",White, #336699);`
`GreaterThanOrEqualTo`	Data set cell value is greater than or equal to 'arg'. Applicable for column types `Real`, `Integer`, or `Date`. `ColorAssignment(Application.StartDate,GreaterThanOrEqualTo, "13/02/2008",White,#336699);`
`LessThan`	Data set cell value is less than the argument. Applicable for column types `Real`, `Integer`, or `Date`. `ColorAssignment(Application.StartDate,LessThan,"2008/02/13",White, #336699);`
`LessThanOrEqualTo`	Data set cell value is less than or equal to `'arg'`. Applicable for column types `Real`, `Integer`, or `Date`. `ColorAssignment(Application.StartDate,LessThanOrEqualTo, "2008/02/13",White,#336699);`
`Between`	The argument must contain two values separated by a comma. Data set cell value is greater than or equal to the value before the comma and less than or equal to the value after the comma. Applicable for column types `Real`, `Integer`, or `Date`. `ColorAssignment(Application.StartDate,Between, "2008/02/12,2008/09/12",White,#336699);`

Condition	Description and Example
`IsNull`	Data set cell value is not defined. Applicable for all column types. **NOTE**: `ColorAssignment` and `PictureAssignment` instructions require an argument definition to be processed. Although searching for undefined values does not require an argument, you must specify an argument, containing For example, a whitespace, for technical reasons. `ColorAssignment(Application.PreviousVersion,IsNull," ",White, #336699);`
`IsNotNull`	Data set cell value is defined. Applicable for all column types. **NOTE**: `ColorAssignment` instructions require an argument definition to be processed. Although searching for undefined values does not require an argument, you must specify an argument, containing For example, a whitespace, for technical reasons. `ColorAssignment(Application.Variant,IsNotNull," ",White, #336699);`
`In`	Data set cell value is equal to one of the values specified as comma separated list in the argument. Applicable for all column types. `ColorAssignment(Application.Version,In,"2.2,3.2,4.0",White, #336699);`
`NotIn`	Data set cell value is equal to one of the values specified as comma separated list in the argument. Applicable for all column types. `ColorAssignment(Application.Version,NotIn,"2.2,2.5,2.8",White , #336699);`

Defining Cell Coloring With Colors Defined in the Report's Query

The background and text color of cells in a tabular configured report can be defined via the query the configured report it is based on. The color definitions returned in the query will then be applied to a defined column of the dataset.

Define the following Instructions to apply colors defined in the query results to the cells in the result dataset:

```
DynamicColorAssignment(ColumnName,BackColorColumn,ForeColorColumn,LegendItem
TextColumn);
```

Use the following parameter(s):

Code Parameter	Description
`ColumnName`	The name of the column in the result dataset the coloring shall be applied to. This parameter is optional. If it is not specified, all columns are colored. Please note that the columns defined in the parameters of the instruction are excluded from coloring.
`BackColorColumn`	The name of the column in the result dataset returning the color in HTML compliant color coding (like #0FFFFFF for white coloring) that shall be applied to the background of the cells. This parameter is mandatory.
`ForeColorColumn`	The name of the column in the result dataset returning the color in HTML compliant color coding that shall be applied to the text in the cells. This parameter is optional.
`LegendItemTextColumn`	The name of the column in the result dataset returning the text to be displayed in the legend for the defined coloring. The legend will show one entry for each differing combination of text and color setting. The parameter is optional. If it is not defined, no legend will be displayed.

Please note the following about the definition of the instruction:

- If you do not specify optional parameters in the middle of the list of parameters, you must add a double comma between the previous and the next argument to maintain the order of parameters in the instruction.
- Columns added to the report dataset to provide the coloring definition can be removed from the visible dataset via a `RemoveColumns` instruction.
- `ColorAssignment` and `DynamicColorAssignment` instructions are executed in the order of their specification. If you define two instructions that shall display a different icon in the same cell, the second instruction will overwrite the first one for the cells that correspond to both instructions.
- If you combine `DynamicColorAssignment` instructions with a `GroupBy_Ex` instruction, the grouping will split the results in one row into two rows, one containing the data for the superordinate object and one containing the data for the subordinate data. The empty cells in each resulting row are set to `NULL`, but color instructions are maintained and executed when the report is rendered in the client interface.
- If you combine `DynamicColorAssignment` instructions with a `JoinColumns` instruction, the content of the source cells is concatenated and placed into the target cell. The picture assignment defined for the target cell is maintained and applied to the concatenated content when rendering the report.

Defining Text Formatting for Cells Containing Specific Data

You can define formatting of text as bold, undeline or italic in tabular reports based on a condition. The text format is changed in all cells of a defined column of the configured report that match the condition. A

legend is displayed for the report that gives information about the text formatting. There are two instructions for text formatting. One is for the definition of text formatting only. The other allows text formatting and coloring of the cell text and background to be configured in a single instruction.

For example, a report is configured to display applications. The object state of the application shall be highlighted by text formatting and cell coloring. For active applications, the text shall be written in bold and for retired applications, the text should be written in italic and at the same time the cells shall be displayed with a red background:

Application Name	Application Version	Application Object S
AF HR Online	3.0	**Active**
AF HR Online EU	2.2 Var.	**Active**
AF HR Online US	2.2 Var.	**Active**
AF WorkPortal	1.0	Plan
AI-Miner	1	Plan
Alfabet @ Cloud	9.8	Plan
ALLFinance PISA	2.9	*Retired*
Anno-Fact	1.0	**Active**
Anno-Fact	2.0	Plan
Announcements for Federal Reserve	1.0	**Active**

The legend displays the cell's text formatting and colors with the operator and argument leading to the specific coloring. The caption of the column is displayed in the cell.

If you do not like the information about the condition to be displayed in the legend, you can define a legend text in the instruction that will overwrite the default text.

The cell text formatting and optional coloring can be defined by adding one of the following instructions to the query:

- To format text of cells in a column based on the cell's content:

  ```
  FontStyleAssignment(ColumnName,ConditionString,"Argument",TextFormat,"LegendText");
  ```

- To format text of cells and color the text and background of cells in a column based on the cell's content:

  ```
  FontStyleColorAssignment(ColumnName,ConditionString,"Argument",TextFormat,TextColor,CellColor,"LegendText");
  ```

Use the following parameter(s):

Code Parameter	Description
`ColumnName`	The column containing the value that constitutes the formatting of text and coloring. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions.
`Condi-tionString`, " *Argument* "	The condition that the cell content must match to apply the formatting and coloring is defined by a condition string like `EqualTo` or `Contains`, and an argument. The specification of the argument is mandatory. The content of the argument depends on the condition string. All allowed filter condition strings and the argument specification required for the condition string are listed in the section Filter Condition Strings for Instructions. The argument must be written in quotation marks.
`TextFormat`	The formatting that shall be applied to the text. Allowed values are: • `bold` • `italic` • `underline` Multiple values can be defined comma separated. The text format has to be written in quotation marks if multiple values are selected.
`TextColor`	For `FontStyleColorAssignment` instructions only: HTML compliant definition of the color of the text in the cell matching the defined condition. For example, the color white can be defined as `White` or `#FFFFFF`.
`CellColor`	For `FontStyleColorAssignment` instructions only: HTML compliant definition of the color of the background the cell matching the defined condition. For example, the color white can be defined as `White` or `#FFFFFF`.
`LegendText`	The text that shall be displayed for the text formatting and color in the legend of the report. This parameter is optional. If no legend text is displayed, the color will be displayed with the column caption and the condition for picture assignment.

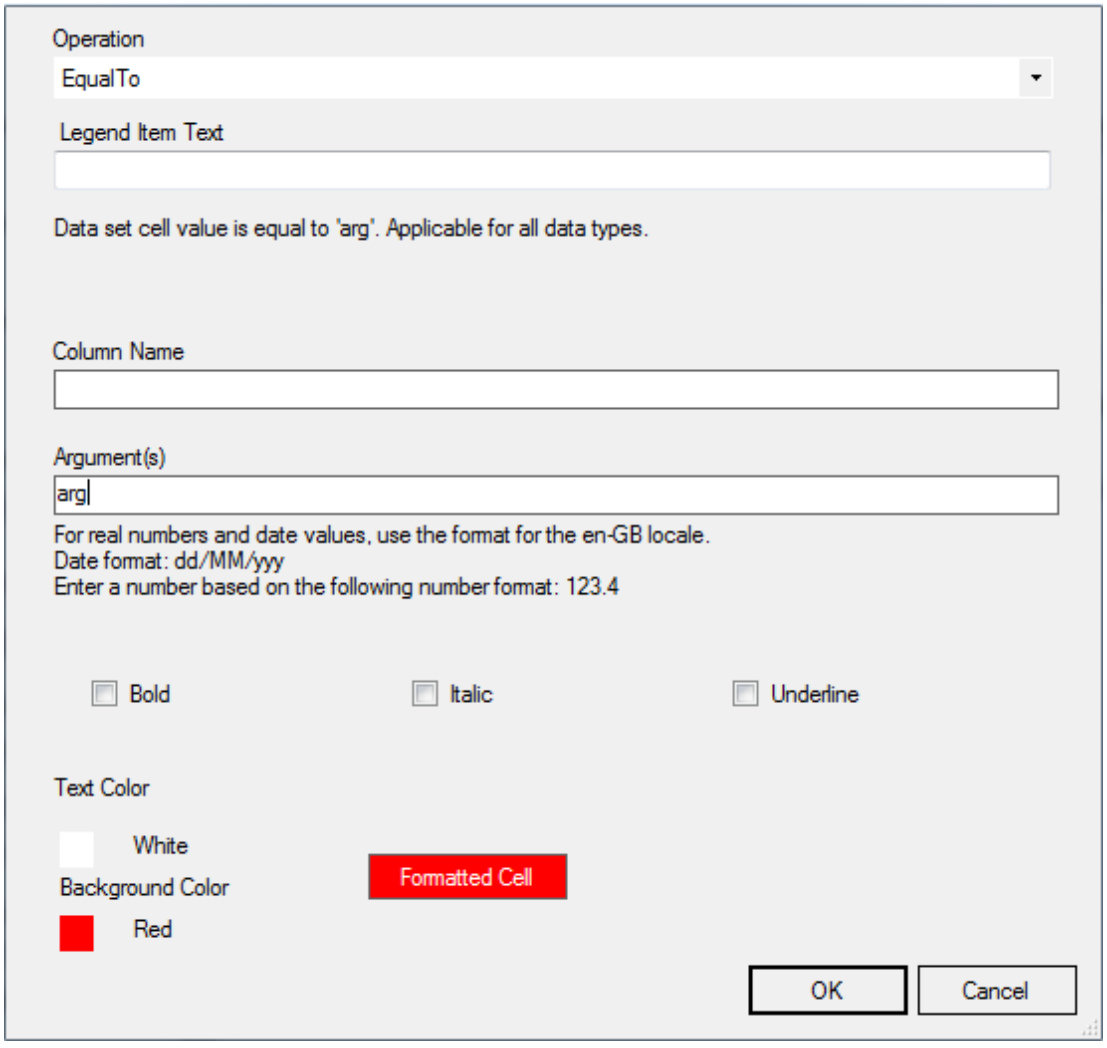
Note the following when defining `ColorAssignment` and `RowColorAssignment` instructions:

- If you define two instructions that shall format the same cell in different manners (For example, a specification that all numbers less than 3 are displayed in bold and another one that defines that all numbers greater than 2 are underlined), the second instruction overwrites the first one for the cells that correspond to both instructions.
- If you combine formatting instructions with a `GroupBy_Ex` instruction, the grouping will split the results in one row into two rows, one containing the data for the superordinate object and one containing the data for the subordinate data. The empty cells in each resulting row are set to `NULL`, but formatting instructions are maintained and executed when the report is rendered in the client interface.

- If you combine color assignments with a `JoinColumns` instruction, the content of the source cells is concatenated and placed into the target cell. The formatting defined for the target cell is maintained and applied to the concatenated content when rendering the report.
- If you change the column caption with a `SetColumnShowName` instruction, the change is reflected in the legend.

In the **Alfabet Query Builder**, formatting assignments can be defined via an editor:

1) In the **Instructions** tab of the **Alfabet Query Builder**, click the **New Instruction** button and select either **Formatting Instructions** > `FontStyleAssignment` or **Formatting Instructions** > `FontStyleColorAssignment` in the sub-menu. An editor opens.



2) Define the condition for cell formatting in the **Filter Definition** editor:

   - **Column Name**: Enter the name of the column containing the value that constitutes the coloring. For the `ColorAssignment` instruction, this is also the column that will be colored. The column name is specified by a combination of the attributes `ClassAlias` and `Name` from the ***ShowProperty*** XML element defining the column, delimited with a period.
   - **Operation**: The condition that the cell content must match to apply the coloring is defined by a condition string like `EqualTo` or `Contains`, and an argument. Select the condition from the

drop-down list. An overview of the conditions available for the instruction is given in the table in the section Filter Condition Strings for Instructions.

- **Argument(s)**: Define the argument(s) for the condition that the cell content must match to apply the coloring. The specification of arguments depend on the selection in the filed **Operation**. An overview of the operations and argument specifications required by the argument are given in the table above.

3) Define the formatting that shall be applied to the text by selecting one or multiple of the checkboxes **Bold**, **Italic**, and **Underline**.

4) For `FontStyleColorAssignement` only: Define the cell color and the text color that shall be used for cells or rows with a content matching the defined condition:

   - **Text Color** / **Background Color**: Click on the colored rectangle to change the text color or cell background color respectively. A color selector opens. You can select one of the colors displayed in the selector by clicking the color field or define a new color by clicking the **Define Custom Color** button. The new color can either be selected from the colors displayed in the top area or defined as RGB color value. Click the **Add as New Color** button to add the color to the set of colors in the standard selector.

After color selection click **OK** to apply your selection. The selected color is displayed in the **Filter Definition** editor and the preview on the right of the color selection fields displays the resulting color format.

5) Optionally, define a text to be displayed for the formatting in the legend in the field **Legend ItemText**. If you do not define a text, the condition causing the formatting will be displayed in the legend with the cell.

6) Click **OK** to save your settings. The resulting instruction is written into the text field of the **Instructions** tab.

Changing the Alignment of Cell Content

By default, integer and real values are right aligned in cells of a tabular report, icons are center aligned and all other data types are displayed left aligned. The default alignment of the cell content can be changed for all or a subset of columns returning a string value in a result data set. For all other data types, the alignment cannot be changed.

Use the one of the following instructions to change the default cell content alignment for a configured report:

- To change alignment of defined columns:

```
SetColumnsAlignment("Left:Column1,Column3","Right:Column5,Column6","Center:Column2,Column4");
```

- To change the alignment of cells in a defined column:

```
SetCellAlignment("Left:Column1,Column3","Right:Column5,Column6","Center:Column2,Column4");
```

Use the following parameter(s):

Code Parameter	Description
`Left: Column1,Column2`	The prefix `Left:` followed by a comma separated list of the names of columns that shall be displayed left aligned. The parameter must be written in quotation marks. The selected columns must return string values. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions.
`Right: Column1,Column2`	The prefix `Right:` followed by a comma separated list of the names of columns that shall be displayed right aligned. The parameter must be written in quotation marks.
`Center: Column1,Column2`	The prefix `Center:` followed by a comma separated list of the names of columns that shall be displayed center aligned. The parameter must be written in quotation marks.

Note the following when defining `SetCellAlignment` and `SetColumnsAlignment` instructions:

- `SetCellAlignment` instructions supersede `SetColumnsAlignment` instructions, independent of the order of definition.
- `SetColumnsAlignment` instructions only affect column rendering. If another instruction specific to cell rendering is applied on the same column (For example, if both `COLORASSIGNMENT` and `SetColumnsAlignment` are used on the same column), the formatting of the cell-level instruction will take precedence and `SetColumnsAlignment` will be ignored.
- The order of the parameters to be defined in these instructions is not relevant. If a parameter is not required, it can be left out.

Changing the Output Format for Boolean Values

Boolean values are stored as `true` or `false` in the Alfabet database.

Theses values can be changed to any other string in the report output using the following instruction:

```
ConvertBoolean2String(ColumnName,StringForTrue,StringForFalse,StringForUndefined);
```

Use the following parameter(s):

Code Parameter	Description
`ColumnName`	The column in the tabular output containing the boolean values that shall be changed to a string.

Code Parameter	Description
	For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions.
`StringForTrue`	Define the string that shall be displayed if the boolean property is set to `true`.
`StringFor-False`	Define the string that shall be displayed if the boolean property is set to `false`.
`StringForUn-defined`	Define the string that shall be displayed if the boolean property is undefined, that means NULL in the database table. This parameter is optional.

For example, a report displays devices and the information, whether the device is a physical hardware device or a virtual host. The information, whether the device is physical or virtual is defined by the `IsPhysical` property of the object class `Device`. The `IsPhysical` property has the **Property Type** attribute set to `Boolean`.

A simple report showing the `Name` and the `IsPhysical` property looks as follows:

Device	Device Is Physical
New York Collaboration Server	x
Cisco Router X189	x
Switch, Baracuda N8000	x
Information Server	x
Transaction Server	x
Oracle Exadata OnPremise Machine	x
Sun Server - Fire Wall	
Sun OS DB Server	x
Windows 2000 Server	x
Backup Server	x
Web Server	x
cRM Server	x

The `ConvertBoolean2String` instruction can be used to alter the output to display the information '`physical host`' or '`virtual host`' instead of an x or an empty cell in the column for the **Is Physical** property:

Device	Host Type
New York Collaboration Server	physical host
Cisco Router X189	physical host
Switch, Baracuda N8000	physical host
Information Server	physical host
Transaction Server	physical host
Oracle Exadata OnPremise Machine	physical host
Sun Server - Fire Wall	virtual host
Sun OS DB Server	physical host
Windows 2000 Server	physical host
Backup Server	physical host
Web Server	physical host
cRM Server	physical host
e-Peopel	physical host

The report is based on the following Alfabet query:

```
ALFABET_QUERY_500

FIND Device

InnerJoin Location ON Device.Location = Location.REFSTR

Instructions

ConvertBoolean2String(Device.IsPhysical,physical
host,virtualhost,undefined);

EndOfInstructions

AUTODSINFO

QUERY_XML

<QueryDef>

   <ShowProperty Type="Property" ClassName="Device" Name="Name"
   ShowName="Device" />

   <ShowProperty Type="Property" ClassName="Device"
   Name="IsPhysical" ShowName="Host Type" />

</QueryDef>
```

The same output can be defined by the following SQL query:

```
SELECT DEVICE.REFSTR, DEVICE.NAME As 'Device', DEVICE.ISPHYSICAL As
'Host Type'

FROM DEVICE
```

in combination with the following instruction definition:

```
ConvertBoolean2String("Host Type","physical host","virtual host");
```

Displaying String, Integer or Real Values as Boolean Values

For object class properties of the **Data Type** `String`, `Integer` or `Real`, the values stored in the database can be displayed as boolean values in a report.

Conversion of values to boolean is done dependent on the property type:

- For properties of the type `Integer` or `Real`, all values that are not equal to zero (=0) are set to true. If the value equals zero, it is set to false. Undefined values are left undefined.
- For properties of the type `String`, the string that shall be set to true must be defined in the instruction. All other strings are set to false. Undefined values are left undefined.

In the resulting report output, an x is displayed in the table cells for that a `String`, `Integer` or `Real` is set to true. If the value is set to false or is undefined, the table cell is empty.

Integer or real values can be changed to boolean in the report output using the following instruction:

```
Convert2Boolean("ColumnName(s)");
```

String values can be changed to boolean in the report output using the following instruction:

```
Convert2Boolean("ColumnName(s)",StringForTrue);
```

Use the following parameter(s):

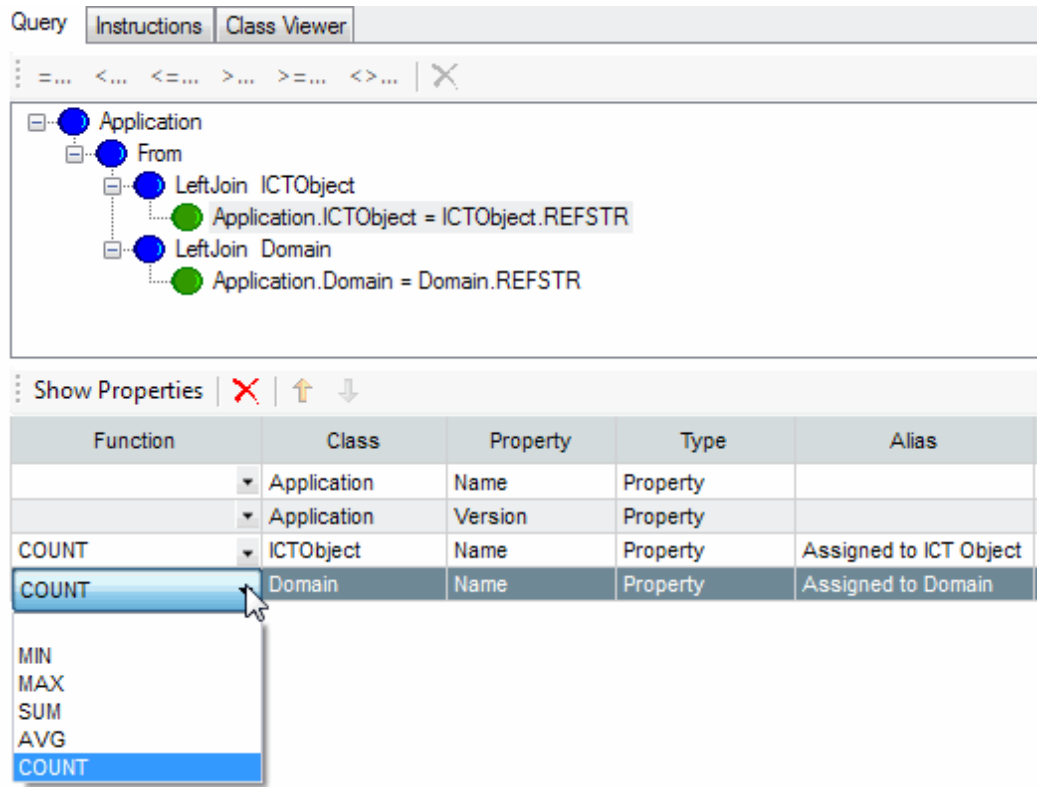
Code Parameter	Description
`ColumnName(s)`	The column in the tabular output containing the values of the **Data Type** `String`, `Integer` or `Real` that shall be displayed as boolean. If multiple columns contain values that shall be displayed as boolean, the columns can be defined comma separated. If multiple columns are defined, and a `StringForTrue` parameter is also defined, this parameter is applied to all columns with string values defined in the comma separated list. For all columns containing integer and real values, the `StringForTrue` parameter is ignored. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions.
`StringForTrue`	Define the string that shall be set to true.

For example, a report displays the assignment of applications to ICT objects and domains.

Application Name	ICT Object Name	Domain Name
Business EAI Platform v.2.2	Business EAI Platform	Deliver and Run IT Services
Groupware Services v.2.2	Groupware Services	
Mafo-Portal v.2.6	Mafo-Portal	Customer Management
PS Global v.2.5	SON	
ALLFiance PISA v.2.9	ALLFinance PISA	Short- and Mid-Term Financial Plan
SAP@OptiRetail v.2.0	SAP@OptiRetail	Finance and Controlling
SAP@AI v.4.0	SAP@AI	Finance and Controlling
BookIT v.2.9	BookIT	Stake Holder Management
Corporate FI-CO v.2.2	Corporate FI-CO	Internal Audit
SAP International v.2.8	SAP Template	
compas v.2.2	compas	Regulatory Compliance Manageme
Electronic Personal Files v.2.5		
Opti-SAP HR v.2.X	Opti-SAP HR	Human Resources
IT-Reporting Tool (SAP BW) v.2.X		Fraud and Incident Management
Prodi v.2.7	Prodi	
eLead v.2.0	eLead	
KSK v.2.0	KSK	
CRM CSS v.3.2	CRM CSS	Customer Management
CRM CSS v.3.3	CRM CSS	Customer Management

The report is generated to check whether all applications are assigned to both a domain and an ICT object. To provide a better overview, the assignment to ICT Objects and Domains shall be displayed as a boolean value.

For that purpose, the information about the assignment to an ICT Object or Domain is first converted into an integer by using the `COUNT` function. To enhance readability, an alias is defined for the column headers reflecting the change in the information displayed in the column:



This change results in the following output:

Application Name	Assigned to ICT Object	Assigned to Domain
Business EAI Platform v.2.2	1	1
Groupware Services v.2.2	1	0
Mafo-Portal v.2.6	1	1
PS Global v.2.5	1	0
ALLFiance PISA v.2.9	1	1
SAP@OptiRetail v.2.0	1	1
SAP@AI v.4.0	1	1
BookIT v.2.9	1	1
Corporate FI-CO v.2.2	1	1
SAP International v.2.8	1	0
compas v.2.2	1	1
Electronic Personal Files v.2.5	0	0
Opti-SAP HR v.2.X	1	1

The use of the `Convert2Boolean` instruction further enhances the readability of the information:

Application Name	Assigned to ICT Object	Assigned to Domain
Business EAI Platform v.2.2	x	x
Groupware Services v.2.2	x	
Mafo-Portal v.2.6	x	x
PS Global v.2.5	x	
ALLFiance PISA v.2.9	x	x
SAP@OptiRetail v.2.0	x	x
SAP@AI v.4.0	x	x
BookIT v.2.9	x	x
Corporate FI-CO v.2.2	x	x
SAP International v.2.8	x	
compas v.2.2	x	x
Electronic Personal Files v.2.		
Opti-SAP HR v.2.X	x	x
IT-Reporting Tool (SAP BW)		x
Prodi v.2.7	x	
eLead v.2.0	x	

The report is based on the following Alfabet query:

```
ALFABET_QUERY_500

FIND

Application

LeftJoin ICTObject ON Application.ICTObject = ICTObject.REFSTR

LeftJoin Domain ON Application.Domain = Domain.REFSTR

Instructions

JoinColumns("Application.Name,Application.Version","Application.Name
", " v.");

Convert2Boolean("ICTObject.Name,Domain.Name");

EndOfInstructions

AUTODSINFO

QUERY_XML
```



```
<QueryDef>
<ShowProperty Type="Property" ClassName="Application" Name="Name" />
<ShowProperty Type="Property" ClassName="Application" Name="Version"
/>
<ShowProperty Type="Property" ClassName="ICTObject" Name="Name"
ShowName="Assigned to ICT Object" Function="COUNT" />
<ShowProperty Type="Property" ClassName="Domain" Name="Name"
ShowName="Assigned to Domain" Function="COUNT" />
</QueryDef>
```

For example, a report displays the object state and status of applications.

	Application Name	Object State	Status
1	Business EAI Platform v.2.2	Retired	Approved
2	Groupware Services v.2.2	Active	Approved
3	Mafo-Portal v.2.6	Active	Approved
4	PS Global v.2.5	Retired	Approved
5	ALLFiance PISA v.2.9	Retired	Approved
6	SAP@OptiRetail v.2.0	Active	Approved
7	SAP@AI v.4.0	Retired	Closed
8	BookIT v.2.9	Retired	Approved
9	Corporate FI-CO v.2.2	Active	Draft
10	SAP International v.2.8	Retired	Approved

The report is generated to check whether all applications with the object state set to Retired have a status Closed assigned. To provide a better overview, the report is configured to display object state and status as boolean values. True is returned if the object state is set to Retired and if the status is set to Closed.

	Application Name	Is Retired	Is Closed
1	Business EAI Platform v.2.2	x	
2	Groupware Services v.2.2		
3	Mafo-Portal v.2.6		
4	PS Global v.2.5	x	
5	ALLFiance PISA v.2.9	x	
6	SAP@OptiRetail v.2.0		
7	SAP@AI v.4.0	x	x
8	BookIT v.2.9	x	
9	Corporate FI-CO v.2.2		
10	SAP International v.2.8	x	

Two `Convert2Boolean` instructions must be defined for the query, because two different strings must be displayed as true.

The report is based on the following native SQL query:

```
SELECT APPLICATION.REFSTR, APPLICATION.NAME AS 'Application Name',
APPLICATION.VERSION, APPLICATION.OBJECTSTATE AS 'Is
Retired',APPLICATION.STATUS AS 'Is Closed'
FROM APPLICATION
```


in combination with the following instruction definition:

```
JoinColumns("Application Name,VERSION","Application Name", " v.");
Convert2Boolean("Is Retired","Retired");
Convert2Boolean("Is Closed","Closed");
```

Changing the Output Format for Date and Time Information

Dates are stored as a timestamp with the date and time information in the Alfabet database tables. If only date information is specified by selecting a date via the calendar button in an editor, For example, the time information will be set to 00:00:00:000 (hh:mm:ss:ms).

The output format of date and time information depends on the culture definition for the client and the property displayed. If the timestamp of a property is irrelevant to the user, it will not be displayed in the results even though it is stored in the database.

Three instruction can be configured to display a non-standard date and/or time format in a tabular report.

Whether date only, date and time, or time only are displayed in the result dataset is defined using the following instruction:

```
SetColumnDateSubType("ColumnName",DateType);
```

The date and time format used depends on the culture definition of the client.

Use the following parameter(s):

Code Parameter	Description
`ColumnName`	The name of the column for which the output format is specified. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions.
`DateType`	The following values can be set: • `Date`: Only the date is displayed with the date pattern defined in the culture settings for the client. • `Time`: Only the time information is displayed with the time pattern defined in the culture settings for the client. • `DateTime`: Date and time information are displayed with the date and time patterns defined in the culture settings for the client separated by a white space.

The output format of date/time information can be changed using the following instruction:

```
SetColumnFormat("ColumnName","DateFormat");
```


The format defined with the instruction is used independent from the culture settings for the client.

Use the following parameter(s):

Code Parameter	Description
`ColumnName`	The name of the column for which the output format is specified. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions.
`DateFormat`	The date format that shall be used to display the results in the specified column. Allowed values are listed in the section Changing the Output Format for Date and Time Information.

The output format can be changed to POSIX using the following instruction:

```
Convert2Posix("ColumnName",ConversionMethod);
```

Use the following parameter(s):

Code Parameter	Description
`ColumnName`	The name of the column for which the output format is specified. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions.
`ConversionMethod`	The method for conversion to POSIX. Allowed values are: • `UTC`: Posix is calculated based on UTC time. • `undefined`: The conversion standard is used. This parameter is optional. The default is `undefined`.

Permissible Date and Time Formats in Alfabet

Please note that the dates in editor fields are always displayed as a numeric value, regardless of the date format. For example, whereas the date format dddd, dd MMMM yyyy will be displayed as Tuesday, 22 August 2006 in the object profile, it will be displayed as 22/08/2006 in the editor field.

Please consider the following syntax for the specification of date and time formats.

- To specify the 24-hour clock system, use "H" or "HH" for hours.
- To specify the 12-hour clock system, use "h" or "hh". Users will need to specify AM or PM.
- To specify a time format, use "M" for month and "m" for minute.

The following formats can be specified:

Date/Time/Separator Syntax	Date/Time/Separator Output
MM/dd/yyyy	08/22/2006
dddd, dd MMMM yyyy	Tuesday, 22 August 2006
dddd, dd MMMM yyyy HH:mm	Tuesday, 22 August 2006 06:30
dddd, dd MMMM yyyy hh:mm tt	Tuesday, 22 August 2006 06:30 AM
dddd, dd MMMM yyyy H:mm	Tuesday, 22 August 2006 6:30
dddd, dd MMMM yyyy h:mm tt	Tuesday, 22 August 2006 6:30 AM
dddd, dd MMMM yyyy HH:mm:ss	Tuesday, 22 August 2006 06:30:07
MM/dd/yyyy HH:mm	08/22/2006 06:30
MM/dd/yyyy hh:mm tt	08/22/2006 06:30 AM
MM/dd/yyyy H:mm	08/22/2006 6:30
MM/dd/yyyy h:mm tt	08/22/2006 6:30 AM
MM/dd/yyyy h:mm tt	08/22/2006 6:30 AM
MM/dd/yyyy h:mm tt	08/22/2006 6:30 AM
MM/dd/yyyy HH:mm:ss	08/22/2006 06:30:07

Date/Time/Separator Syntax	Date/Time/Separator Output
MMMM dd	August 22
MMMM dd	August 22
yyyy'-'MM'-'dd'T'HH':'mm':'ss.fffffffK	2006-08-22T06:30:07.7199222-04:00
yyyy'-'MM'-'dd'T'HH':'mm':'ss.fffffffK	2006-08-22T06:30:07.7199222-04:00
ddd, dd MMM yyyy HH':'mm':'ss 'GMT'	Tue, 22 Aug 2006 06:30:07 GMT
ddd, dd MMM yyyy HH':'mm':'ss 'GMT'	Tue, 22 Aug 2006 06:30:07 GMT
yyyy'-'MM'-'dd'T'HH':'mm':'ss	2006-08-22T06:30:07
HH:mm	06:30
hh:mm tt	06:30 AM
H:mm	6:30
h:mm tt	6:30 AM
HH:mm:ss	06:30:07
yyyy'-'MM'-'dd HH':'mm':'ss'Z'	2006-08-22 06:30:07Z
dddd, dd MMMM yyyy HH:mm:ss	Tuesday, 22 August 2006 06:30:07
yyyy MMMM	2006 August
yyyy MMMM	2006 August

Replacing Values Returned in the Query with a Defined String

For object class properties of the **Data Type** `String` or `Text`, the values stored in the database can be replaced with defined strings in the result dataset of a query. Only the complete return value can be replaced with this instruction. Substrings in a text or string cannot be replaced.

The following instruction replaces values returned in a defined column of the result dataset of the query:

```
ReplaceValues("ColumnName","value1,...,valueN","replacementvalue1,...,replacementvalueN");
```

Use the following parameters:

Code Parameter	Description
`ColumnName`	The name of the column for which the output format is specified. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions.
`value1,...,valueN`	A comma separated list of values that shall be replaced. If a cell in the column returns one of the defined values, the value in the cell is replaced.
`replacementvalue1,...,replacementvalueN`	A comma separated list of values that shall be added to the dataset instead of the originally returned values. The position in the comma separated list must be identical to the position of the value to be replaced in the comma separated list of original values.

For example, a configured report returns the name and version of applications in a selected application group and information about the object state of the applications to inform about the availability of the applications for the user:

Application Name	Application Version	Application Object Stat
AF HR Online EU	2.2 Var.	Active
AF HR Online US	2.2 Var.	Active
AF WorkPortal	1.0	Plan
ALLFinance PISA	2.9	Retired
ARBI	1.2.1	Active

Instead of the object state value **Plan**, the string **For future use** shall be displayed.

Instead of the object state value **Active**, the string **Available** shall be displayed.

The object state value **Retired** shall be displayed unchanged.

The ReplaceValues instruction is added to the Alfabet query of the configured report. The instruction replaces the values for the object states **Plan** and **Active**.

```
ALFABET_QUERY_500

FIND Application

InnerJoin ApplicationGroup ON Application.ApplicationGroups =
ApplicationGroup.REFSTR

Instructions

   ReplaceValues("Application.ObjectState","Plan,Active","For future
   use,Available");

EndOfInstructions

QUERY_XML

<QueryDef>

   <ShowProperty Type="Property" ClassName="Application" Name="Name"
   />

   <ShowProperty Type="Property" ClassName="Application"
   Name="Version" />

   <ShowProperty Type="Property" ClassName="Application"
   Name="ObjectState" />

   <ShowProperty Type="Property" ClassName="Application"
   Name="Status" />

</QueryDef>
```

The resulting report displays the replaced values in the Application Object State column:

Application Name	Application Version	Application Object State
AF HR Online EU	2.2 Var.	Available
AF HR Online US	2.2 Var.	Available
AF WorkPortal	1.0	For future use
ALLFinance PISA	2.9	Retired
ARBI	1.2.1	Available

Adding the Value of a Server Variable to the Dataset

The instruction `ReplaceServerVariable` replaces the content of a specified column of the data set returned by a query with the value of a server variable. Server variables are defined in the server alias configuration in the tab **Variables** and provide a means to include environment specific data into configurations. If a specification is different in a test and production environment, the environment specific value can be defined in the server alias as required and the same, unchanged report can be used in all environments, returning different values. The `ReplaceServerVariable` instruction requires the specification of a column in the result dataset of the query and of a server variable defined as `$ ServerVariableName`.

For information about the specification of server variables, see Using Server Variables in Web Link and Database Server-Related Specifications .

The content of all cells in the result dataset is overwritten with the value of the server variable defined in the server alias of the Alfabet Web Application.

The following instruction adds a server variable value to a result dataset:

```
ReplaceServerVariable(ColumnName,ServerVariableName);
```

Use the following parameters:

Code Parameter	Description
`ColumnName`	The name of the column that shall return the server variable value. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions.
`ServerVariableName`	The name of the server variable defined in the server alias configuration of the Alfabet Web Application with the prefix `$`. For example, the server variable defined in the server alias as `ExternalHelpServer` must be defined as `$ExternalHelpServer`.

Limiting Toolbar Button Functionality To a Subset of Objects in a Configured Report

Usually, the functionality of buttons in the toolbar of configured reports is applied to the base object of the selected row in the report. The base object is either the `FIND` object in an Alfabet query or the object defined by its `REFSTR` as the first argument of the `SELECT` clause in a native SQL query. For example, when the user selects a row in a tabular report and clicks the **Navigate** button, the object profile of the base object of the report opens.

The buttons in the toolbar can be configured to work with a subset of the base objects only. For example, when the report is about objects for which object class stereotypes are defined, you can limit the button functionality to objects of a defined stereotype. Or, when the report displays objects with different object states, you may want a button to work only for objects with a defined state.

The instruction `SetRowCategory` allows a toolbar button functionality to be defined as dependent on property values. The instruction points to a column in the query results. The values in the column are the categories that specify the subset of objects a button functionality is applied to:

```
SetRowCategory("ColumnName");
```

Use the following parameter(s):

Code Parameter	Description
`ColumnName`	The name of the column that shall be used for the definition of the object categories. Each column value is a different category. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions.

For example, a configured report displays applications with different object states:

Application Name	Application Version	Application Object State
Xetra	0.10	Retired
ACCOUNT	1	Active
Brokertec	1	Active
CAPRICE	1	Active
CashLine Web - Germany	1	Active
Cognotec	1	Active
Configuration	1	Plan
Course	1	Active

The configured report is based on the following Alfabet query:

```
ALFABET_QUERY_500

FIND Application

Instructions

   SetRowCategories("Application.ObjectState");

EndOfInstructions

QUERY_XML

<QueryDef>

   <ShowProperty Type="Property" ClassName="Application" Name="Name"
   />

   <ShowProperty Type="Property" ClassName="Application"
   Name="Version" />

   <ShowProperty Type="Property" ClassName="Application"
   Name="ObjectState" />

</QueryDef>
```

The user shall only be able to navigate to applications that are in the state `Active`. The `SetRowCategory` instruction added to the report Therefore, defines the column `Application.ObjectState` as relevant for the definition of categories. The resulting categories are all object states that can be defined for an application:

- `Plan`
- `Active`
- `Retired`

In the example above, you could limit the button functionality to objects with the object state `Plan` by setting the **Apply to Category** attribute of the button to "`Plan`". Or, you could limit the button functionality to objects with the object state `Plan` or `Active` by setting the **Apply to Category** attribute to "`Plan,Active`".

After object categories have been defined for a configured report via the `SetRowCategory` instruction, the button of the toolbar must be configured to be active for either one or multiple of the defined categories. This is done by defining the category in the button's attribute **Apply to Category**. The button can be defined to be active for objects that match multiple categories by defining a comma-separated category list.

The configuration of the button is exclusively performed by Software AG Support. If you want a button to be applied to objects of defined categories, you must define the categories in the underlying query of the report and provide Software AG Support with the information about which values shall be entered in the **Apply to Category** attribute of the button. Please note that Software AG Support must be informed about the required value of the attribute and which button shall be configured.

Note the following when defining categories:

- You can add multiple `SetRowCategory` instructions to a query. All values of all columns defined in a `SetRowCategory` instruction can be used as category. For example, you can define that the activity of a button depends on both the stereotype and the object state of an object.
- When multiple categories are listed in the **Apply to Category** attribute of a button, the button is active when at least one of the categories applies. For example, when both object class stereotype and object state of an object are defined as categories and the **Apply to Category** attribute of the button is set to "`Area,Active`", the button is active for all objects with either the stereotype `Area` or the object state `Active` or objects for that both conditions apply.
- To define that a button is active when an object applies to two categories, you must join the columns that contain the information about the two conditions into one column using the `JoinColumns` instruction and define the resulting combined column as category definition column.
- If a table is grouped via a `GroupBy_Ex` instruction, the category definition can be applied to all levels of the grouped dataset. The `GroupBy_Ex` instruction must be defined first. The columns returning the category for the different levels must then be joined into a single column with a `JoinColumns` instruction. The `SetRowCategories` instruction must be defined after the `GroupBy_Ex` and `JoinColumns` instructions.

Changing the Navigation Behavior of the Base Object Class in a Configured Report

Typically, the user will navigate to the object view of the base object class of the configured report when double-clicking an object in a row. It is also possible to configured a query so that the user can navigate to the object view of an object class that is not the base object.

The base object of a query is by default the `FIND` object of the Alfabet query or the object for which the `REFSTR` is defined as the first argument in the `SELECT` clause of the native SQL query. The navigate functionality of configured reports refers per default to the base object. This is independent of the information displayed in the tabular output of the report. Therefore, it is possible that a user might even be directed to an object class that is not visible in the report results.

You can use the SetRowReference instruction to define that the object defined in another row of the report is used as base object:

```
SetRowReference("ColumnName");
```

Use the following parameter(s):

Code Parameter	Description
ColumnName	The name of the column that shall be used as base object definition. The column value must be the REFSTR of the object. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions.

The SetRowReference instruction requires that the selected column returns the REFSTR of the objects from the object class that has been selected as new base class. Usually, the REFSTR of the objects should not be displayed in the configured report because it is useless information for the user. You can hide the column in the report output by using the RemoveColumns instruction described in the section Hiding a Column in a Configured Report.

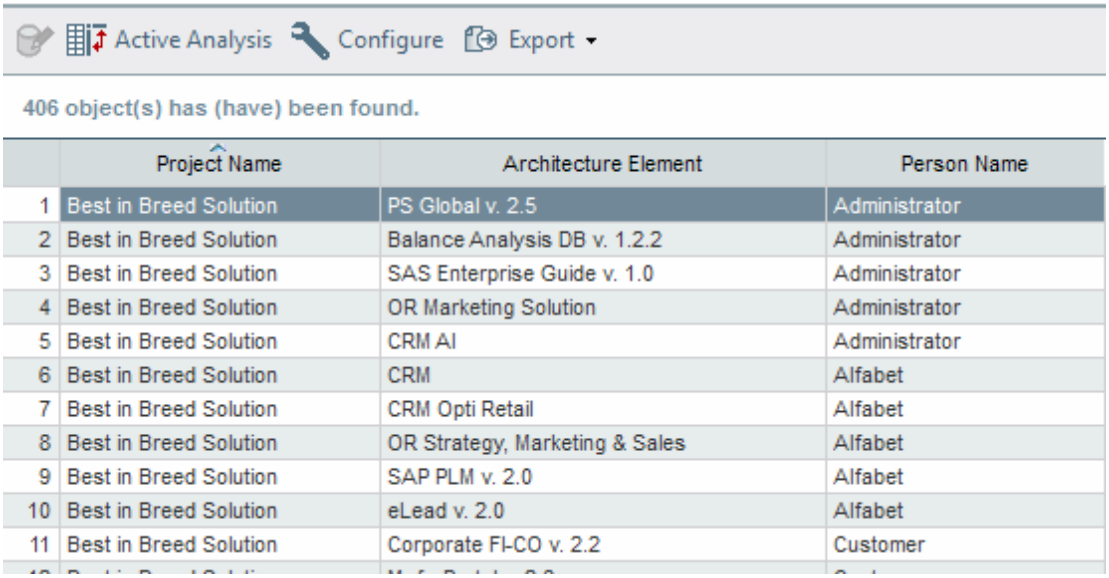
For example, a configured report shall display all architecture elements assigned to projects and the user that is the authorized user responsible for the architecture element. There is one row in the report for each architecture element. A project can include objects from different object classes as architecture elements. The report is based on the following Alfabet query:

```
ALFABET_QUERY_500

FIND

ProjectArch

InnerJoin Project ON ProjectArch.Project = Project.REFSTR

LeftJoin Application ON Project

tArch.Object = Application.REFSTR

LeftJoin ICTObject ON ProjectArch.Object = ICTObject.REFSTR

LeftJoin Domain ON ProjectArch.Object = Domain.REFSTR

LeftJoin OrgaUnit ON ProjectArch.Object = OrgaUnit.REFSTR

LeftJoin BusinessProcess ON ProjectArch.Object =
BusinessProcess.REFSTR

LeftJoin BusinessFunction ON ProjectArch.Object =
BusinessFunction.REFSTR

LeftJoin Person ON (Or Person.REFSTR = Application.ResponsibleUser
Person.REFSTR = BusinessFunction.ResponsibleUser Person.REFSTR =
Domain.ResponsibleUser Person.REFSTR = BusinessProcess.ResponsibleUser
Person.REFSTR = ICTObject.ResponsibleUser Person.REFSTR =
OrgaUnit.ResponsibleUser)

Instructions

JoinColumns("Application.Name,Application.Version","Application.Name",
" v. ");
```



```
JoinColumns("Application.Name,ICTObject.Name,BusinessProcess.Name,Doma
in.Name,OrgaUnit.Name,BusinessFunction.Name","Domain.Name", "");

EndOfInstructions

QUERY_XML

<QueryDef>

<ShowProperty Type="Property" ClassName="Project" Name="Name" />

<ShowProperty Type="Property" ClassName="Application" Name="Name" />

<ShowProperty Type="Property" ClassName="Application" Name="Version"
/>

<ShowProperty Type="Property" ClassName="ICTObject" Name="Name" />

<ShowProperty Type="Property" ClassName="Domain" Name="Name" />

<ShowProperty Type="Property" ClassName="OrgaUnit" Name="Name" />

<ShowProperty Type="Property" ClassName="BusinessProcess" Name="Name"
/>

<ShowProperty Type="Property" ClassName="BusinessFunction" Name="Name"
/>

<ShowProperty Type="Property" ClassName="Person" Name="Name" />

<SortProperty Type="Property" ClassName="Project" Name="Name" />

<SortProperty Type="Property" ClassName="Person" Name="Name" />

</QueryDef>
```

The resulting report has a deactivated **Navigate** button, because the `FIND` class of the Alfabet query is the class `ProjectArch` that does not have an object view. Additionally, the `ProjectArch` object class is a pure representation of the link between the project and an architecture element. The user clicking the **Navigate** button should be directed to either the project or the architecture element.



	Project Name	Architecture Element	Person Name
1	Best in Breed Solution	PS Global v. 2.5	Administrator
2	Best in Breed Solution	Balance Analysis DB v. 1.2.2	Administrator
3	Best in Breed Solution	SAS Enterprise Guide v. 1.0	Administrator
4	Best in Breed Solution	OR Marketing Solution	Administrator
5	Best in Breed Solution	CRM AI	Administrator
6	Best in Breed Solution	CRM	Alfabet
7	Best in Breed Solution	CRM Opti Retail	Alfabet
8	Best in Breed Solution	OR Strategy, Marketing & Sales	Alfabet
9	Best in Breed Solution	SAP PLM v. 2.0	Alfabet
10	Best in Breed Solution	eLead v. 2.0	Alfabet
11	Best in Breed Solution	Corporate FI-CO v. 2.2	Customer

In the following example, the query is changed to allow navigation to the architecture element. There are architecture elements from 6 different object classes displayed in the `Architecture Element` column of the report. To navigate to the objects of either of these classes, a new column returning the `REFSTR` of the object class must be added to the Alfabet query for each of the joined object classes. The columns are then joined into one column using a `JoinColumns` instruction and

this column is used in the `SetRowReference` instruction to select the object as the base object of the result row. The column is then hidden from the configured report using a `RemoveColumns` instruction because the `REFSTR` of the objects is not relevant for the user viewing the report.

```
ALFABET_QUERY_500

FIND

ProjectArch

InnerJoin Project ON ProjectArch.Project = Project.REFSTR

LeftJoin Application ON Project

tArch.Object = Application.REFSTR

LeftJoin ICTObject ON ProjectArch.Object = ICTObject.REFSTR

LeftJoin Domain ON ProjectArch.Object = Domain.REFSTR

LeftJoin OrgaUnit ON ProjectArch.Object = OrgaUnit.REFSTR

LeftJoin BusinessProcess ON ProjectArch.Object =
BusinessProcess.REFSTR

LeftJoin BusinessFunction ON ProjectArch.Object =
BusinessFunction.REFSTR

LeftJoin Person ON (Or Person.REFSTR = Application.ResponsibleUser
Person.REFSTR = BusinessFunction.ResponsibleUser Person.REFSTR =
Domain.ResponsibleUser Person.REFSTR = BusinessProcess.ResponsibleUser
Person.REFSTR = ICTObject.ResponsibleUser Person.REFSTR =
OrgaUnit.ResponsibleUser)

Instructions

JoinColumns("Application.Name,Application.Version","Application.Name",
" v. ");

JoinColumns("Application.Name,ICTObject.Name,BusinessProcess.Name,Doma
in.Name,OrgaUnit.Name,BusinessFunction.Name","Domain.Name", "");

JoinColumns("Application.REFSTR,ICTObject.REFSTR,BusinessProcess.REFST
R,Domain.REFSTR,OrgaUnit.REFSTR,BusinessFunction.REFSTR","Application.
REFSTR", "");

SetRowReference("Application.REFSTR");

RemoveColumns("Application.REFSTR");

EndOfInstructions

QUERY_XML

<QueryDef>

<ShowProperty Type="Property" ClassName="Project" Name="Name" />

<ShowProperty Type="Property" ClassName="Application" Name="Name" />

<ShowProperty Type="Property" ClassName="Application" Name="Version"
/>

<ShowProperty Type="Property" ClassName="ICTObject" Name="Name" />

<ShowProperty Type="Property" ClassName="Domain" Name="Name" />

<ShowProperty Type="Property" ClassName="OrgaUnit" Name="Name" />

<ShowProperty Type="Property" ClassName="BusinessProcess" Name="Name"
/>
```



```
<ShowProperty Type="Property" ClassName="BusinessFunction" Name="Name"
/>
<ShowProperty Type="Property" ClassName="Person" Name="Name" />
<ShowProperty Type="Property" ClassName="Application" Name="REFSTR" />
<ShowProperty Type="Property" ClassName="ICTObject" Name="REFSTR" />
<ShowProperty Type="Property" ClassName="Domain" Name="REFSTR" />
<ShowProperty Type="Property" ClassName="OrgaUnit" Name="REFSTR" />
<ShowProperty Type="Property" ClassName="BusinessProcess"
Name="REFSTR" />
<ShowProperty Type="Property" ClassName="BusinessFunction"
Name="REFSTR" />
<SortProperty Type="Property" ClassName="Domain" Name="Name" />
<SortProperty Type="Property" ClassName="Project" Name="Name" />
<SortProperty Type="Property" ClassName="Person" Name="Name" />
</QueryDef>
```

Providing a Link to Alfabet Views, Editors or Wizards from Cells in a Report

In a configured report displaying a table, cells in the data set can be configured to open either a page view, an object view or object cockpit, an editor, or a wizard when a user clicks the cell in the table. The text in a cell that opens the view, editor or wizard is formatted as a link and a tooltip can optionally be defined that is displayed to a user moving the mouse over the link.

The view, editor or wizard opens for the base object of the current row of the table. By default, this is the `FIND` class of an Alfabet query or the object found via the `REFSTR` defined as first argument of the `SELECT` statement of a native SQL query, but the base class can be changed for a table via a `SetRowReference` instruction.

There is no mechanism that controls whether the editor or wizard that is defined for the link in the cell is valid for the base object class of the configured report. If you define an editor or wizard for the wrong object class, you may cause damage to the storage of object data.

If the link target is an editor or wizard, the editor or wizard will open in a modal window. you can optionally configure the link to open a defined tab of the editor or a defined step of the wizard. Editors can only be opened with any other than the first tab opened if they are not embedded in a wizard. If you want a specific tab of the editor to open when opening an editor embedded in a wizard, you must configure the wizard to open the editor with that specific tab.

If the link target is an Alfabet view, the user will be redirected to the new view. The behavior is the same as for all navigation operations in Alfabet. Optionally filters can be defined for opening the view.

A link can either be defined statically to open the same view, wizard, or editor for all results or dynamically, opening a different type of view, editor or wizard for each row of the result dataset.

- Opening the Same Page View, Object Profile, Object Cockpit, or Configured Report for all Search Results
- Opening the Same Wizard or Editor for all Search Results

- Opening Diffent Views, Wizards, or Editors for Search Results

Opening the Same Page View, Object Profile, Object Cockpit, or Configured Report for all Search Results

Double-clicking on a cell in a configured report opens the object profile of the base object of the current row. In addition to this basic navigation, additional navigation can be configured for tabular reports by using the `LinkAssignment` instruction. A link can be assigned to the content of the cells in a defined column of the report. The link target can be an object profile, an object cockpit, a configured report or a standard Alfabet page view. The linked view opens either for the object that is the base object of the current row in the report containing the link or for an object defined via a `REFSTR` value in another column of the report. If the link target is a configured report, this object is the base object of the report referred to with the parameter `BASE`. The `LinkAssignment` instruction can contain definitions to populate values for other parameters defined for the configured report opened by the link.

The link is defined using the following instruction:

```
LinkAssignment("LinkColumnName",ViewType:ViewName:ViewDetail,"BaseObjectColumnName","ParameterColumnNames");
```

Use the following parameter(s):

Code Parameter	Description
`LinkColumnName`	The name of the column that shall contain the link. The content of the cell defined via the query of the report is displayed as link text. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions. Alternatively, the position of the column in the index of columns can be used to specify the column. For more information see Referencing Columns in Instructions by Index Number.
`ViewType`	The kind of view that opens when the user clicks the link. Allowed values are: • `ObjectView` to open an object profile or object cockpit. • `GraphicView` to open a standard page view. • `Report` to open a configured report.
`ViewName`	The name of the view that shall open. The name is the value for the **Name** attribute of the view. For object profiles and object views, the name of the object view configuration element that contains the object profile or object cockpit configuration must be defined.
`ViewDetail`	This parameter is only relevant for opening an object cockpit and defines the name of the object cockpit that shall open. If the `ViewType` is `ObjectView` and the

Code Parameter	Description
	`ViewDetail` parameter is not defined, the object profile configured for the object view is the target of the link.
`BaseObjectColumnName`	This parameter defines the object for that the link target view is opened for. Allowed values are: • The name of the column that contains the value of the `REFSTR` property of the object that shall be the base object for that the link target opens. Alternatively, the position of the column in the index of columns can be used to specify the column. For more information see Referencing Columns in Instructions by Index Number. • `ROW`: If the parameter is set to "`ROW`", the base object found for the row in the report containing the link is used as base object for the link target. If the link target is a configured report, the object can be referenced in the query of the report with the parameter `@BASE`.
`ParameterColumnNames`	This parameter is only relevant if the link target is a configured report. It defines the column names of the columns each containing the values for an Alfabet parameter used in the query of the configured report that opens. The name of the column must be identical to the name of the parameter without the prefix @ . Alternatively, the position of the column in the index of columns can be used to specify the column. For more information see Referencing Columns in Instructions by Index Number.

If the link target is a configured report, the query defined for the report can use Alfabet parameters that are substituted with values from the report containing the `LinkAssignment` instruction during runtime. Next to the parameters defined via the instruction, the name of the column containing the link can be used as parameter value.

The parameters in the link target report must be defined as `@<ParameterName>`. The alternative syntax:`ParameterName` cannot be targeted by a `LinkAssignment` instruction.

Please note that the Alfabet Query Builder creates parameters starting with a colon. This has to be changed manually using the attribute " **Query As Text** " of the configured report.

The following table lists the Alfabet parameters that can be used in the configured reports and the values from the report containing the `LinkAssignment` instruction that they represent:

Parameter of the query in the target report	Value from the report containing the link
`@BASE`	The REFSTR value of the object that the link target view is opened for. In the `LinkAssignment` instruction, the specification of the `BaseObjectColumnName` defines whether this is the base object of the row of the report containing the link or a REFSTR value read from a defined cell of the row.
`@COLUMN`	The name of the column containing the link. This column is defined with the `LinkColumnName` definition in the `LinkAssignment` instruction. Instead of the column name, the position of the column in the index of columns can be used to specify the column. (For more information see Referencing Columns in Instructions by Index Number.) In this case, the `@COLUMN` is still substituted with the name, and not the index position of the column containing the link. The definition of the column by index position also allows to define a range of columns for that the instruction will be processed. In that case, the value substituting the parameter `@COLUMN` is the name of the column containing the link that the user clicked on. An example is given below.
`@<Parameter-Name>`	The `LinkAssignment` instruction can contain a `ParameterColumnNames` definition, that specifies the names of one or multiple columns of the report containing the link. The values in this columns are substituting the parameters that are defined as the column name.

The following example shows a configured report that opens a configured report via a link defined with a `LinkAssignment` instruction. The main report shows application groups and the number of applications in the group that are in the object state Active, Plan, or Retired. The number of applications is calculated via a `COUNT` command. The column displaying the number of applications contains a link that opens a configured report displaying all applications in the application group in the respective object state:

	Application Group Name	Application Object State	Number of Applications
1	1. Market Data	Active	12
2	1. Market Data	Plan	4
3	2. Trade Entry	Active	6
4	3. Front Office	Active	9
5	3. Front Office	Plan	4
6	3. Front Office	Retired	1
7	4. Back Office	Active	13

The report is based on the following Alfabet query:

```
ALFABET_QUERY_500

FIND ApplicationGroup
```



```
InnerJoin Application ON Application.ApplicationGroups =
ApplicationGroup.REFSTR

Instructions

   RenameColumn("Application.ObjectState","OState");

   LinkAssignment("Application.REFSTR",
   "Report:LinkAssignmentTarget", ROW," OState ");

EndOfInstructions

QUERY_XML

<QueryDef>

   <ShowProperty Type="Property" ClassName="ApplicationGroup"
   Name="Name" />

   <ShowProperty Type="Property" ClassName="Application"
   Name="ObjectState" />

   <ShowProperty Type="Property" ClassName="Application"
   Name="REFSTR" Function="COUNT" />

   <SortProperty Type="Property" ClassName="ApplicationGroup"
   Name="Name" />

   <SortProperty Type="Property" ClassName="Application"
   Name="ObjectState" />

</QueryDef>
```

In the `LinkAssignment` instruction, the configured report `LinkAssignmentTarget` is defined as target of the link. The configured report `LinkAssignmentTarget` shows all applications in the respective application group and with the respective object state. It is based on the following Alfabet query:

```
ALFABET_QUERY_500

FIND Application

WHERE

   (AND

      Application.ApplicationGroups CONTAINS@BASE

      Application.ObjectState =@OState

   )

QUERY_XML

<QueryDef>

   <ShowProperty Type="Property" ClassName="Application" Name="Name"
   />

   <ShowProperty Type="Property" ClassName="Application"
   Name="Version" />

   <ShowProperty Type="Property" ClassName="Application"
   Name="CommitedRecoveryPoint" />

   <ShowProperty Type="Property" ClassName="Application"
   Name="CommitedRecoveryTime" />

</QueryDef>
```


The query contains two parameters that are populated with values from the configured report containing the `LinkAssignment` instruction: `@BASE` and `@OState`.

The `@BASE` refers to the `REFSTR` property of the base object of the row containing the link opening the report. In the `LinkAssignment` instruction, this is defined with the specification of the `BaseObjectColumnName` as `ROW`.

The `@OState` refers to the value of the column named `OState` for the row containing the link opening the report. In the `LinkAssignment` instruction, this is defined with the specification of `ParameterColumnNames` as `OState`.

The following report is a dynamically generated report that is based on a stored procedure. The columns of the report are dynamically generated. For each application of a selected application group a column is added to the report. The column name is identical to the name and version of the application. As a result, the overall number of columns and the names of the individual columns differ for each application group for that the report is opened. The rows of the report show costs for different cost types together with an indicator that evaluates whether the costs are critical.

Select Application Group

6. Risk Reporting

Year*

2014 Submit

Export

7 object(s) has (have) been found.

	Cost Type	ARBI 1.2.1	Credient 2.2.1 SP4	CRES 9.0.0
1	Software Subscriptions	55.00	60.00	45.00
2	Other Operational Costs	35.00	33.00	27.00
3	Hardware Replacement Costs			
4	Other Deployment Costs			
5	Internal Maintenance Costs	303.00	263.00	239.00
6	External Maintenance Costs	82.00	70.00	69.00
7	TOTAL	475.00	426.00	380.00

The report contains links in all cells of the dynamically generated columns. The `LinkAssignment` instruction added to the report refers to the column containing the link as a range of index numbers starting with the second column of the report, which is the first column that is created dynamically and including all columns until the last one:

```
LinkAssignment(#2:end, Report:Demands4SelectedApplication, ROW);
```

The target report for the links is a report that shows all demands that are pending for the selected application. The application name and version are given in the name of the column containing the link. Therefore, the Alfabet parameter `@COLUNMN` is used in the native SQL query of the report that opens when a user clicks the link:

```
SELECT dem.REFSTR, dem.ID, dem.NAME, dem.STATUS, dem.TARGETDATE,
dem.DESCRIPTION
```



```
FROM DEMAND dem, DEMAND_ARCH dema, APPLICATION app
WHERE dema.DEMAND=dem.REFSTR
AND dema.OBJECT=app.REFSTR
AND (app.NAME+' '+app.VERSION) LIKE @COLUMN
```

Opening the Same Wizard or Editor for all Search Results

Use the following instruction to set a link to an editor or wizard statically defined in the instruction definition:

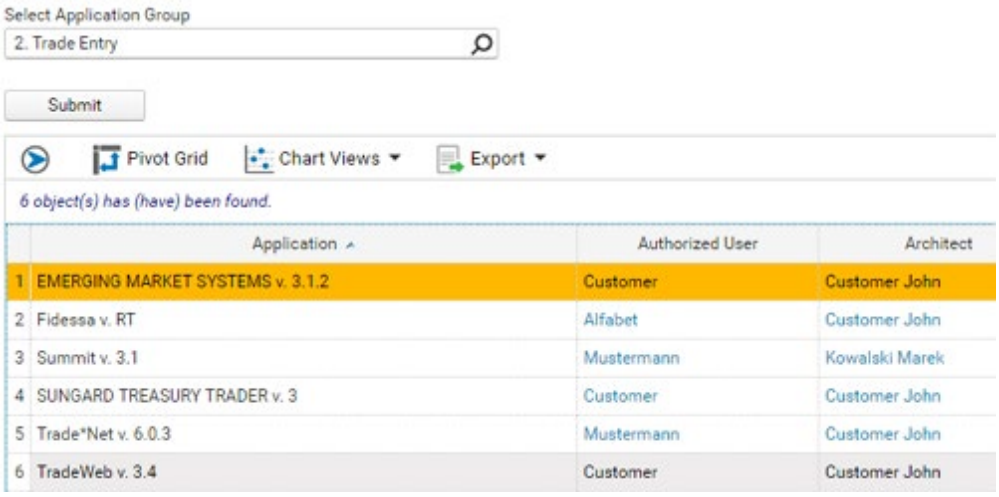
```
LinkAssignment_Edit("LinkColumnName",EditViewType:EditViewName,
"Step:WizardStepName","Page:EditorTabPosition", "TooltipText");
```

Use the following parameters:

Code Parameter	Description
`LinkColumnName`	The name of the column that shall contain the link. The content of the cell defined via the query of the report is displayed as link text. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions. Alternatively, the position of the column in the index of columns can be used to specify the column. For more information see Referencing Columns in Instructions by Index Number.
`EditViewType`	The kind of edit view that opens when the user clicks the link. Allowed values are: • `Editor` to open an editor. • `Wizard` to open a wizard.
`ViewName`	The name of the edit view that shall open. The name is the value for the **Name** attribute of the standard editor or wizard. For custom editors, the name of the standard editor of the object class that the custom editor is defined for must be specified. Custom editors are additional tabs amended to standard editors. They can be opened by defining a landing tab for opening the editor with the `EditorTabPosition` parameter.
`WizardStepName`	This parameter is only relevant for opening a wizard and defines the name of the wizard step at which the wizard shall open. When the `EditViewType` is `Editor`, an empty definition must be added to the instruction to maintain the order of parameters. An empty parameter can be defined by adding two commas between `EditViewType:EditViewName` and `Page:EditorTabPosition`.

Code Parameter	Description
`EditorTabPosition`	This parameter is only relevant for opening an editor and defines the position of the tab in the tab order of the editor at which the editor shall open. When the `EditViewType` is `Wizard` and a tooltip text shall be defined, an empty definition must be added to the instruction to maintain the order of parameters. An empty parameter can be defined by adding two commas between `Step:WizardStepName` and `TooltipText`.
`TooltipText`	A string that shall be displayed when the user moves the mouse over the link text. This parameter is optional.

For example, a configured report shall be created for checking and redefining application responsibilities. The report displays all applications in an application group and informs about the current authorized user setting and the assignment of the role 'Architect' for the application:



The configured report is based on an Alfabet query and the `FIND` class is `Application`:

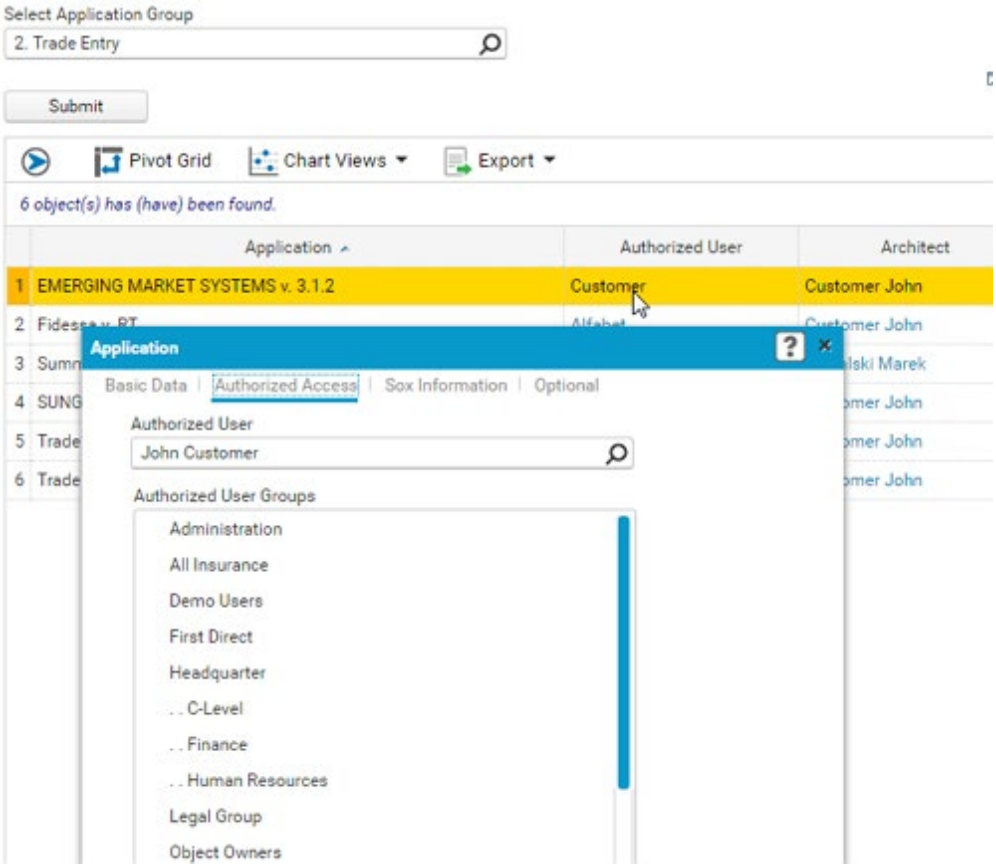
```
ALFABET_QUERY_500

FIND Application

InnerJoin Person ON Application.ResponsibleUser = Person.REFSTR

WHERE Application.ApplicationGroups CONTAINS:BASE

Instructions

   JoinColumns("Application.Name,Application.Version","Application.N
   ame", " v. ");

EndOfInstructions

QUERY_XML

<QueryDef>
```



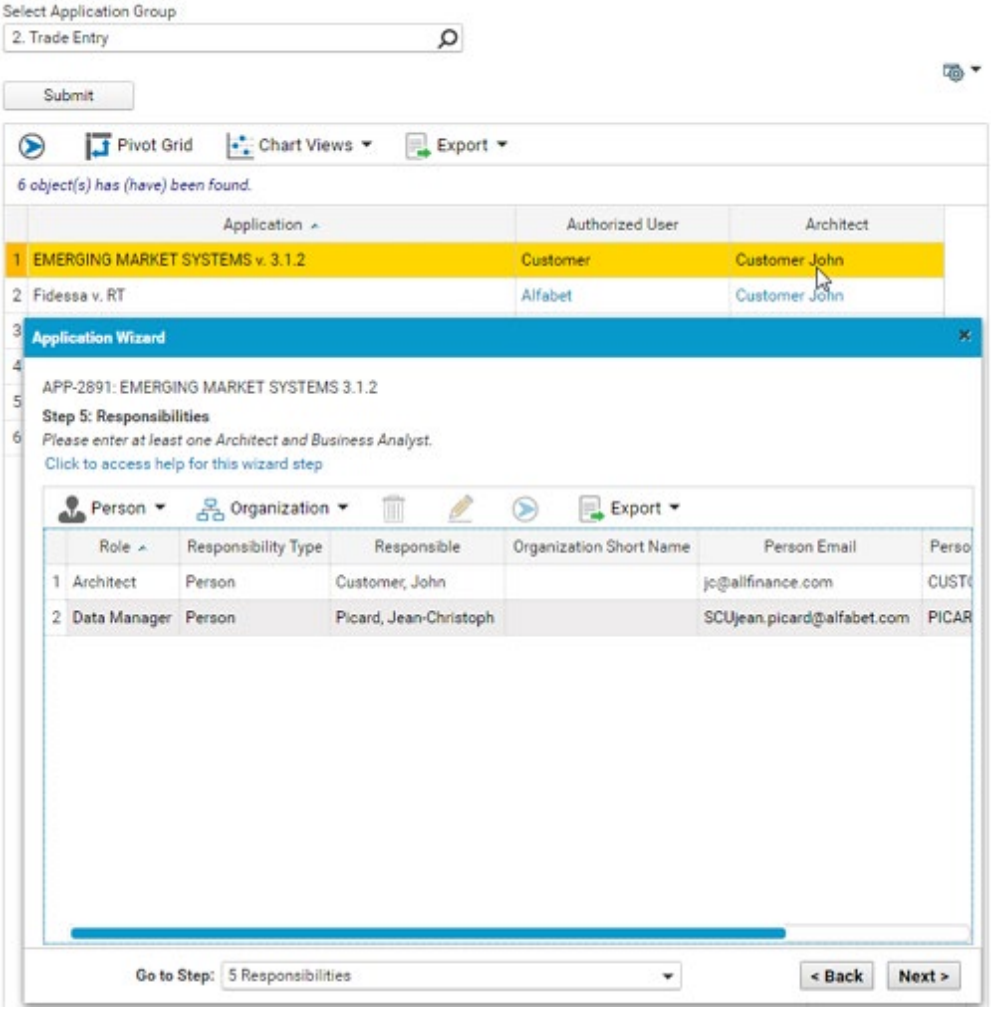
```
        <ShowProperty Type="Property" ClassName="Application" Name="Name"
        ShowName="Application" />

        <ShowProperty Type="Property" ClassName="Application"
        Name="Version" />

        <ShowProperty Type="Property" ClassName="Person" Name="Name"
        ShowName="Authorized User" />

        <ShowProperty Type="RoleType" ClassName="Application"
        Name="Architect" ShowName="Architect"
        ImageProps1="Name,FirstName" ImageProps2="Name" />

    </QueryDef>
```

Therefore, all editor or wizard links must be configured to open an editor or wizard for the class Application.

The user shall be able to open the standard editor for applications at the tab **Authorized Access**, which is the second tab of the editor, from all cells of the column **Authorized User**:



The user shall be able to open a customer configured wizard for the class Application at step 5, which displays the standard **Responsibilities** page view:



To open the editors, the following two `LinkAssignment_Edit` instructions are added to the query:

```
ALFABET_QUERY_500

FIND Application

InnerJoin Person ON Application.ResponsibleUser = Person.REFSTR

WHERE Application.ApplicationGroups CONTAINS:BASE

Instructions

    JoinColumns("Application.Name,Application.Version","Application.N
    ame", " v. ");

    LinkAssignment_Edit("Person.Name", Editor:APP_Editor,Page:2,
    "Change Responsible User");

    LinkAssignment_Edit(Application.Architect, Wizard:APP_Wizard,
    "Step:View_Responsibilities", "Change Architect");

EndOfInstructions

QUERY_XML

<QueryDef>

    <ShowProperty Type="Property" ClassName="Application" Name="Name"
    ShowName="Application" />
```



```
    <ShowProperty Type="Property" ClassName="Application"
    Name="Version" />

    <ShowProperty Type="Property" ClassName="Person" Name="Name"
    ShowName="Authorized User" />

    <ShowProperty Type="RoleType" ClassName="Application"
    Name="Architect" ShowName="Architect"
    ImageProps1="Name,FirstName" ImageProps2="Name" />

</QueryDef>
```

Opening Diffent Views, Wizards, or Editors for Search Results

Use the following instruction set a link to views, editors or wizards dynamically defined via the query of the report:

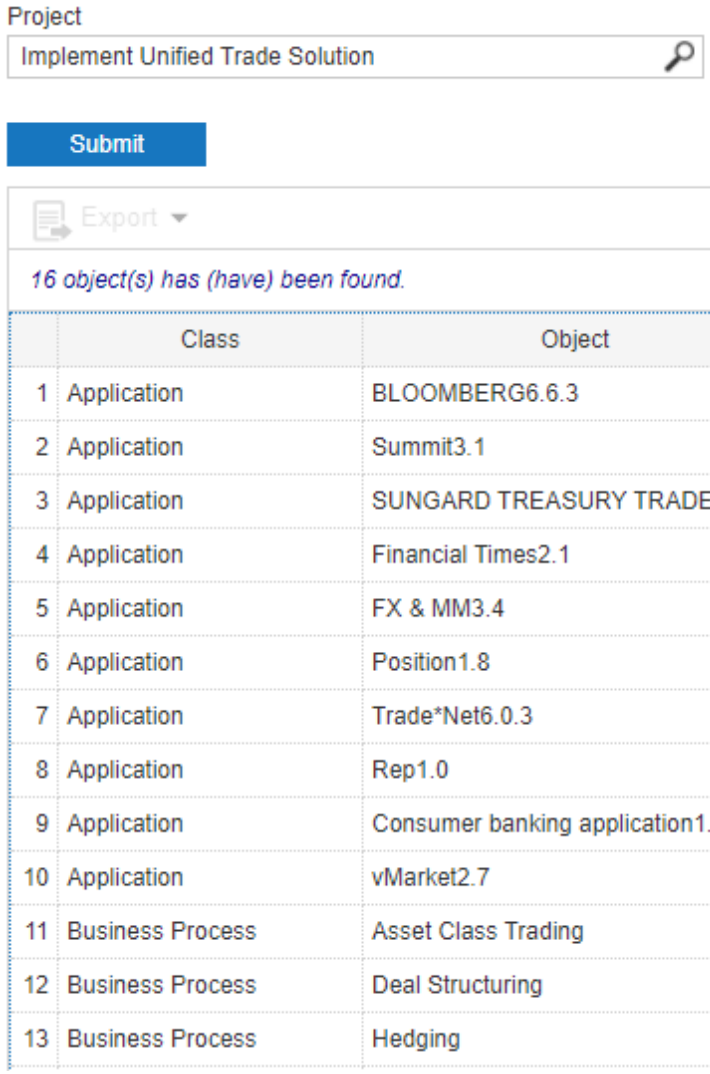
```
DynamicLinkAssignment("LinkColumnName",ViewColumnName,NavigationDetailsColum
nName,EditorTabPositionColumnName,TooltipTextColumnName);

DynamicLinkAssignment(aLinkCol,
Editor|Wizard|GraphicView|Report|ObjectView:Name Col, [Step
Col(Editor/Wizard)]|["Col1,...,ColN"(GraphicView/Report)], [Page Number
Col(Editor/Wizard)], [Cell Tooltip Text Col]);
```

Use the following parameters:

Code Parameter	Description
`LinkColumnName`	The name of the column that shall contain the link. The content of the cell defined via the query of the report is displayed as link text. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions. Alternatively, the position of the column in the index of columns can be used to specify the column. For more information see Referencing Columns in Instructions by Index Number.
`ViewColumnName`	The name of the column in the report dataset that returns the view, wizard or editor that opens when the user clicks the link. The query must return the type of view and the name of the view separated by a colon, like For example,: `Editor:APP_Editor` Allowed view types are • `Editor` to open an editor. The view name is the value of the **Name** attribute of a standard editor. To open a custom editor, the name of the standard editor of the object class that the custom editor is defined for must be specified. Custom editors are additional tabs amended to standard editors. They can be opened by defining

Code Parameter	Description
	a landing tab for opening the editor with the `NavigationDetailsColumnName` parameter. • `Wizard` to open a wizard. The view name is the value of the **Name** attribute of a wizard. • `GraphicView` to open a page view. The view name is the value of the **Name** attribute of a standard page view. • `Report` to open a configured report. The view name is the value of the **Name** attribute of a configured report. • `ObjectView` to open an object profile or object cockpit. The definition of the view name is not required, but the colon is required as part of the definition. The object view or object cockpit defined to open for the object class via the configuration of the user profile opens.
`NavigationDetailsColumnName`	The name of the column in the report dataset that returns details for opening the view. Details are optional and only applicable for the following: • For opening a wizard you can define the wizard step at which the wizard shall open. The column must return the string `Step:` followed by the the value of the **Name** attribute of the wizard step. • For opening an editor you can define the position of the editor tab in the tab order of the editor at which the editor shall open. The column must return the string `Page:` followed by the the tab number. The first tab in the editor has the number 1. • For opening a page view or configured report with filters, you can set filter values on opening of the link target. The filter value must be returned in a column with a name identical to the filter parameter without the starting character @. Multiple columns with filter values can be defined in a comma-separated list.
`EditorTabPositionColumnName`	If a wizard shall open at a wizard step that is an editor with multiple tabs, you can define the editor tab that shall open in the query. The column must return the string `Page:` followed by the the tab number. The first tab in the editor has the number 1. This parameter must be set to the name of the column in the report dataset returning the information.
`TooltipTextColumnName`	The name of the column in the report dataset that returns a string that shall be displayed when the user moves the mouse over the link text. This parameter is optional.

For example, a configured report shall be created for redefining the authorized user for both applications and business processes relevant for a project. The report displays all applications and business processes of a selected project defined as base object of the report and informs about the current authorized user setting:


Project

Implement Unified Trade Solution

Submit

Export

16 object(s) has (have) been found.

	Class	Object	Authorized User
1	Application	BLOOMBERG6.6.3	PICARD
2	Application	Summit3.1	MUSTERMANN
3	Application	SUNGARD TREASURY TRADER3	CUSTOMER
4	Application	Financial Times2.1	PICARD
5	Application	FX & MM3.4	LEE
6	Application	Position1.8	LEE
7	Application	Trade*Net6.0.3	MUSTERMANN
8	Application	Rep1.0	GOSSARAH
9	Application	Consumer banking application1.2	CLIENTE
10	Application	vMarket2.7	KOWALSKI
11	Business Process	Asset Class Trading	NGOMBE
12	Business Process	Deal Structuring	NGOMBE
13	Business Process	Hedging	NGOMBE

To change the authorized user, a link from the Authorized User column shall open a wizard for the definition of applications on step one, opening the application editor, and the editor for the definition of business processes on tab two, which is the tab for the definition of the authorized user.

The configured report is based on the following native SQL query containing an instruction for dynamic link assignment that activates the link for both applications and business processes:

```
SELECT app.REFSTR, 'Application' As 'Class', app.NAME + app.VERSION
AS 'Object', per.USER_NAME AS 'Authorized User', 'Wizard:APP_Wizard'
As 'EditView', 'Step:Editor' As 'WizardStep', '' As 'EditorPage',
'Open Application Editor' AS 'EditTooltip'

FROM APPLICATION app

INNER JOIN PROJECT_ARCH projarch ON projarch.OBJECT = app.REFSTR

INNER JOIN PROJECT proj ON projarch.PROJECT = proj.REFSTR

INNER JOIN PERSON per on per.REFSTR = app.RESPONSIBLEUSER
```

```
WHERE proj.REFSTR = @BASE

UNION ALL

SELECT bp.REFSTR, 'Business Process' As 'Class', bp.NAME AS
'Object', per.USER_NAME AS 'Authorized User', 'Editor:PROC_Editor'
As 'EditView', NULL AS 'WizardStep', 'Page:2' AS 'EditorPage', 'Open
Business Process Editor' As 'EditTooltip'

FROM BUSINESSPROCESS bp

INNER JOIN PROJECT_ARCH projarch ON projarch.OBJECT = bp.REFSTR

INNER JOIN PROJECT proj ON projarch.PROJECT = proj.REFSTR

INNER JOIN PERSON per on per.REFSTR = bp.RESPONSIBLEUSER

WHERE proj.REFSTR = @BASE

/* Alfabet Instructions */

DynamicLinkAssignment_Edit("Authorized User",
EditView,WizardStep,EditorPage,EditTooltip);

RemoveColumns("EditView,WizardStep,EditorPage,EditTooltip");
```

Enabling a Stereotype-Specific Search in Custom Selectors

Custom selectors allows objects in an object class that are defined in the `ClassEntry` element in the custom selector definition to be searched. Objects of other classes cannot be selected but are nevertheless displayed in, For example, a **Browse** search to structure the objects in the explorer.

When object class stereotypes are defined for an object class, custom selectors can be defined to limit the search to only objects of a defined object class stereotype. For example, for views implementing a **Browse** functionality, the hierarchy can display a hierarchy of domains of different stereotypes per hierarchy level, but only domains of the object class stereotype specified in the class entry definition can be selected by the user.

When the `ClassEntry` element in the custom selector defines an object class stereotype, all queries defined for the pages of the class entry must include a stereotype definition as described below:

- The stereotype of the object class must be added to the Show properties of Alfabet queries or the `SELECT` statement of native SQL queries. That means that the dataset resulting from the query contains a column that lists the stereotype of the objects in the dataset. The column must be available for technical reasons to allow the software to control whether an object is selectable. The `RemoveColumns` instruction can be used to hide the column from display in the explorer.
- The instruction `SetRowsStereotypeIndex` must be included in the query to identify the column containing the stereotype information in the dataset for the process that checks whether an object is selectable in the explorer.

The column in the dataset listing the stereotype of the objects is defined using the following instruction:

```
SetRowsStereotypeIndex("ColumnName");
```

Use the following parameter(s):

Code Parameter	Description
`ColumnName`	The name of the column listing the stereotypes. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions.

Configuring Cells in a Data Table Report to Provide Navigation to an Object Profile

Data table reports are based on a large dataset that the user can build own reports from according to his/her current demand. The resulting reports are highly individual and it is Therefore, difficult to determine a base object per row. Navigation can Therefore, not be based on the same mechanisms as in other tabular reports.

Instead, drill down from this report to the object profile of the current object is provided per cell. Drill-down is not provided for all cells of the report but only for cells that contain information about the `REFSTR` of an object. While the `REFSTR` is required as technical information for the engine to provide the navigation, it is not suitable for the user to identify the object. The `CreateREfImage` instruction allows rows containing any string, For example, information about object name and version, to be merged with the row containing the technically required `REFSTR` information. The resulting column displays the string information about the object and at the same time provides navigation in data table reports because it contains the `REFSTR` in the background. The resulting column is the column that originally contained the `REFSTR`. The columns containing the string information will be removed from the visible dataset.

To define columns providing drill-down to object profiles from data table reports, the dataset must contain a column that returns the `REFSTR` and one or multiple other columns returning the property values that shall be displayed in the navigable cell. The columns can then be merged with the following instruction:

```
CreateRefImage("StringColumn1,StringColumn2","RefColumn","Delimiter")
```

Use the following parameters:

Code Parameter	Description
`StringColumn1,StringColumn2`	The name of the column containing the information that shall be visible to the user in the result dataset. Multiple columns can be defined in a comma separated list. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions.
`RefColumn`	The name of the column containing the `REFSTR` of the objects for that drill down to the object profile shall be enabled. This column is the column that will be visible and selectable in the data table report.

Code Parameter	Description
`Delimiter`	If multiple string columns have been defined with the first parameter, you can optionally define a delimiter string to be displayed between the values from the string columns.

The data table report shall contain a column that provides navigation to the object class Application and displays the name and version of the appplication separated by the string " v. ".

First, a dataset is created that contains a column returning the REFSTR, one returning the name and one returning the version of the application:

Application REFSTR	Application Name	Application Version	Applic
76-2518-0	Business EAI Platform	2.2	14/11/
76-2525-0	Groupware Services	2.2	12/02/
76-2538-0	Mafo-Portal	2.6	25/08/
76-2566-0	PS Global	2.5	24/03/

The `CreateRefImage` instruction is added to the Alfabet query, merging the columns for the application's name and version into the column for the application's REFSTR with the delimiter " v. ".

```
ALFABET_QUERY_500

FIND Application

Instructions

   CreateRefImage("Application.Name,Application.Version","Applicatio
   n.REFSTR", " v. ");

EndOfInstructions

QUERY_XML

<QueryDef>

   <ShowProperty Type="Property" ClassName="Application"
   Name="REFSTR" />

   <ShowProperty Type="Property" ClassName="Application" Name="Name"
   />

   <ShowProperty Type="Property" ClassName="Application"
   Name="Version" />

   <ShowProperty Type="Property" ClassName="Application"
   Name="StartDate" />

</QueryDef>
```

The resulting dataset no longer displays three columns for REFSTR, name and version, but the column for the REFSTR only, with the name and version information visible in the column.

Application REFSTR	Application Start Date
Business EAI Platform v. 2.2	14/11/2015
Groupware Services v. 2.2	12/02/2015
Mafo-Portal v. 2.6	25/08/2015
PS Global v. 2.5	24/03/2016
ALLFinance PISA v. 2.9	24/01/2015
SAP@OptiRetail v. 2.0	10/07/2015

Masking a Link Target in A Configured Report With a Text

Links in tabular configured reports can be masked with a text that is different than the link target URL.

Please note however that the object preview window will show the link with the original URL.

The dataset of the query the report is based on must return the URL and the text in two different columns. The text can then be displayed in the column containing the URL retaining the navigation capability to the underlying URL by adding the following instruction to the query:

```
JoinUrlLink(URL_Column, Column_with_replacing_text);
```

Use the following parameters:

Code Parameter	Description
`URL_Column`	The name of the column containing the link to the target URL. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions.
`Column_with_replacing_text`	The name of the column containing the text to mask the target URL in the column defined as `URL_Column`.

The column returning the text is not removed from the dataset by this instruction. Use a `RemoveColumns` instruction to remove this column. The `RemoveColumns` instruction must be defined after the `JoinUrlLink` instruction. For information about the `RemoveColumns` instruction, see Hiding a Column in a Configured Report.

Converting a String into a Link in a Configured Report using Reference Value

The dataset instruction `MakeReferenceValue` allows you to assign a reference link to a label string to be displayed in a defined column of a configured report. This link navigates to the object view or object cockpit for the object defined via a `REFSTR` value in the query.

The `MakeReferenceValue` instruction behaves identical to the `LinkAssignment` instruction in ordinary datasets.

Converting a label string into a link is done by adding the following instruction to the query:

```
MakeReferenceValue(RefStringColName, ValueStringColName);
```

The parameters to be defined are described in detail:

Code Parameter	Description
`RefStringColName`	The name of the column defined via the query which contains the string to be converted to the reference. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions.
`ValueStringColName`	The name of the column containing the string to be displayed as the value i.e. the label.

When the instruction is used, the dataset of the query the report is based on must return the resulting reference link and the label in two different columns. The label can then be displayed both in the column containing the reference link and in its original column which does not retain the navigation capability to the underlying link.

The column returning the label i.e. `ValueStringColName` is not removed from the resulting dataset by the `MakeReferenceValue` instruction automatically, but can be removed using a `RemoveColumns` instruction. The `RemoveColumns` instruction must be defined after the `MakeReferenceValue` instruction. For information about the `RemoveColumns` instruction, seeHiding a Column in a Configured Report.

See Creating a Card View Report for an example on how to use the `MakeReferenceValue` instruction along with the `RemoveColumns` instruction in a simple query for a Card View report.

Adding Dynamic Web Links to a Configured Report

The XML object ***WebViewManager*** allows you to configure dynamic web links referencing applications that are accessible via a URL. The dynamic web link references a specific location in the application, allowing you to access information in the application that is relevant to the object that the user is currently working with in Alfabet.

Dynamic web links configured in the XML object ***WebViewManager*** can be opened from a configured report.

For information about configuring dynamic web links in the XML object ***WebViewManager***, see Configuring Dynamic Web Links That Users Can Access.

The instruction `SetDynamicWebLink` will add an **Open Web Link** button to the report that opens the dynamic web link for the object selected in the configured report via a sub-menu option. Only one web link can be defined per object, but different objects can open different web links. For example, a configured report displaying applications and components can contain a link to an internal report about the application for each application and a link to a technical service repository for the components. The **Open Web Link** button will show a sub-menu option for each kind of dynamic web link added to the configured report. Only the sub-menu option valid for the currently selected object will be active.

In addition to the button interaction, the link can be opened via the **Operations** button in the object preview window.

The configured report must return the following data in the columns of the dataset returned by the query the configured report is based on:

- Each row must have a base object the row result is about. Please note that the report must be configured to display only one row for each found object. If the result dataset returns multiple rows with the same base object but different data, the dynamic web links cannot be set.

  In Alfabet queries, the base object of a result is the `FIND` class of the Alfabet query. In native SQL queries, the first argument in the `SELECT` statement must return the `REFSTR` of an object. This object is the base object of the row result. This first argument is not displayed in the configured report. For both Alfabet queries and native SQL queries, the base object of a row can be changed via a `SetRowReference` instruction. For more information, see Changing the Navigation Behavior of the Base Object Class in a Configured Report.

  For example,:

  - A configured report finds components as a base class and displays the component and the component category that the component is assigned to in a tabular data set. There will be only one row per component because each component can only be assigned to one component category. Dynamic web links can be added this configured report.
  - A configured report finds components as a base class and displays the component and the component groups that the component is assigned to in a tabular data set. Multiple rows might be displayed per component because each component can be assigned to multiple component groups. Dynamic web links cannot be added to this configured report.

- The value of the XML attribute `Caption` of an XML element `WebViewDef` in the XML object ***WebViewManager***. Please note that the attribute `ClassNames` of that XML element `WebViewDef` must include the base object of the row results in the configured report the dynamic web link shall be opened from.
- The value of the XML attribute `Caption` of one of the subordinate XML objects `WebLinkDef` of the defined XML element `WebViewDef`.
- The text that shall be displayed as the sub-menu item of the **Open Web Link** button in the toolbar.

- The text that shall be displayed as the sub-menu item of the **Operations** button of the preview window.
- The `REFSTR` of the object for which the link shall contain property values. The placeholders in the URL of the dynamic web link will be replaced with the values of the object class properties for the object referenced in this column of the configured report. Make sure that the object class properties defined in the XML attribute `PropNames` of the relevant `WebLinkDef` are available for the object class referenced by this column.
- A string to be used as a category name to differentiate different kind of dynamic web links. The following must apply for the category setting:
  - The toolbar button and preview button menu text must be different for different categories.
  - The toolbar button and preview button menu text as well as the selected `WebViewDef` and `WebLinkDef` must be identical for all rows having the same category definition. If For example, multiple toolbar button texts are defined for a category, each text will be displayed as separate sub-menu item. When the user selects an object assigned to that category, all sub-menu items that are created for the dynamic web link category will be active and perform the same action.

The column names must be added to the instruction that has the following syntax:

```
SetDynamicWebLink(ObjectReferenceColumn,WebViewCaptionColumn,WebLinkCaptionC
olumn,ToolbarButtonMenuTextColumn,PreviewButtonMenuTextColumn,CategoryColumn
)
```

The parameters are all mandatory. Use the following parameters:

Code Parameter	Description
`ObjectRefer-enceColumn`	The name of the column containing the `REFSTR` values of the objects for that the dynamic web link shall be opened. That means that the dynamic web link will open with object class property values for the object referenced in the `Object Refer-enceColumn`. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions.
`WebViewCap-tionColumn`	The name of the column returning the value of the XML attribute `Caption` of an XML element `WebViewDef` in the XML object ***WebViewManager***
`WebLinkCap-tionColumn`	The name of the column returning the value of the XML attribute `Caption` of one of the subordinate XML objects `WebLinkDef` of the defined XML element `Web-ViewDef`.
`ToolbarButton-MenuTextColumn`	The name of the column returning the string to be displayed as sub-menu item of the button **Open Web Link** in the toolbar of the configured report.
`PreviewButton-MenuTextColumn`	The name of the column returning the string to be displayed as sub-menu item of the button **Operations** in the preview window of the configured report.

Code Parameter	Description
`CategoryColumn`	The name of the column returning the category name to enable the instruction with the given category name for the row.

If dynamic web links shall be opened for the leaf level results in a grouped dataset, the `GroupBy_Ex` instruction for grouping of the dataset must be defined prior to the `SetDynamicWebLink` instruction in the order of instructions to ensure that dynamic web links are only applied to the subordinate rows.

Exporting Attachments to a Runtime Folder During Report Execution

The instruction `RetrieveIDOCPath` triggers the export of attachments to the runtime folder of the Alfabet Web Application during execution of a query that provides information about the attachments that are specified in the query . The path to the runtime directory is written in the configured report to inform about the location the file can be retrieved from.

This instruction is explicitly implemented for use with the JIRA integration feature for export of attachment with JIRA issues. Storage time and location are handled then handled by the JIRA integration mechanisms. If the configured report is executed directly in the Alfabet user interface, documents are stored in the temporary directory during the active user sesssion only and are removed automatically from the runtime folder on logout of the user.

The attachments are exported each time the report is executed. Each report will be stored with the file name of the attachment amended with a GUID that ensures that the file name is unique. The whitelist and blacklist defined for export actions is taken into account. If a file has an extension that is not allowed for export, the REFSTR of the ALFA_IDOCUMENT is written into the respective cell instead of the file path.

For information about whitelist and blacklist, see Configuring the Permissibility of Files and Web Links in Alfabet .

Attachments are stored in the Alfabet database as objects of the object class `ALFA_IDOCUMENT`. The query for that the instruction is defined must return the `REFSTR` of the relevant `ALFA_IDOCUMENT`s in a column of the result dataset. The following instruction can then be applied to that column:

```
RetrieveIDCOPath(ColumnName);
```

Use the following parameter:

Code Parameter	Description
`ColumnName`	The name of the column containing the `REFSTR` values of the `ALFA_IDOCUMENT`s to be exported. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions.

Displaying Translated Values of Enumerations, Object States, Statuses, Milestones and Indicators in Configured Reports

Values defined for object state and status definitions, indicator values, lifecycle status definitions, enterprise release and project milestone definitions and enumerations are translatable in the Alfabet vocabularies.

For a detailed description about the globalization capability, see Localization and Multi-Language Support for the Alfabet Interface

The translated values are displayed on the Alfabet interface when the user opens the user interface in the respective language with the following exceptions, whereby the values are displayed in English by default:

- The history tracking report provided by Software AG
- Configured reports.

For configured reports, instructions are provided that trigger the display of translated enumeration, indicator range, object state and status value, and lifecycle status values in the configured report.

If values that shall be translated are later concatenated with other values, For example, with a `JoinColumns` instruction, the instruction for translating the value must be added prior to the instruction concatenating the value.

To display history tracking information with translated enumeration, indicator range value, object state and status values, you can create a configured report displaying the history tracking information and use it instead of the standard history tracking report. For more information about creating a configured report to display history tracking information, see Defining Audit Management Related Configured Reports in the chapter Configuring Reports.

The translation of enumeration, indicator range, object state, milestone captions, lifecycle status values, and status values displayed in a configured report can be defined by adding one of the following instructions to the query:

- To display enumeration, object state, milestone caption and status values in the translation provided for the culture that the user is currently using:

  ```
  TranslateEnums("ColumnName,ColumnName,...");
  ```

- To display indicator range values in the translation provided for the culture that the user is currently using:

  ```
  TranslateIndicators("ColumnName,ColumnName,...");
  ```

  Reports based on Alfabet query language with indicators added via the Show properties does not require the use of the `TranslateIndicators` instruction to be displayed translated in the Alfabet user interface.

- To display lifecycle status values, that means the value of the property `Status` of the object class `TimeStatus`, in the translation provided for the culture that the user is currently using:

  ```
  TranslateTimeStatus("ColumnName,ColumnName,...");
  ```

Use the following parameter(s):

Code Parameter	Description
`ColumnName`	The name of the column in the tabular output of the report that contains the values to be translated. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions. If multiple columns contain values to be translated, the columns can be defined comma-separated. Alternatively, the position of the column in the index of columns can be used to specify the column. For more information see Referencing Columns in Instructions by Index Number.

Display Translated Values for Object Class Properties Subject to Data Translation in Configured Reports Based on Native SQL

Some predefined protected properties and custom properties of the type `String` and `Text` can be translated to the secondary languages for which data translation is enabled in the cultures.

For more information about object data translation, see the section Configuring the Translation of Object Data in the chapter Localization and Multi-Language Support for the Alfabet Interface.

For configured reports of the type `Query` the translated object data is automatically displayed in the report in the language currently displayed in the user interface. For native SQL queries the change to the current language must be included in the report. One of the available methods is the instruction `SetTranslatedValue`. Please note however, that a more convenient implementation method is available. This method does not involve an instruction and is described in the section Displaying Translated Object Data In the Current Language of the User Interface.

In the Alfabet database, the translated values for translatable object class properties are written to additional columns of the database table for the respective object class. The columns are named <ClassTechName>_<local ID>. For example, for a German translation of the obects's name and description, the columns would be named `NAME_1031` and `DESCRIPTION_1031`.

If you want a configured report based on a native SQL query to display the translations for the current language of the user interface, you must add all columns containing the translated values to the `SELECT` statement of your query. In addition, the `SetTranslatedValue` instruction must be added to the query definition:

```
SetTranslatedValue("ColumnName1,ColumnName2");
```

Use the following parameter(s):

Code Parameter	Description
`ColumnName`	The name of the column in the tabular output of the report that contains the value in the original language. For information about the names of the columns resulting from Alfabet queries and native SQL queries, see Defining Column Names and Captions. If multiple columns contain data to be translated, the columns can be defined comma-separated.

If the instruction is added, the following behavior applies:

- If the user opens the report in the original language, the columns containing translated values will be removed from the dataset.
- If the user opens the report in any other language for which a translation is included in the query, the values in the column for the base culture value will be substituted with the values from the respective language column. After that, columns containing translated values will be removed from the dataset

For example, if you translate data to German (locale ID = 1031) and French (locale ID = 1036), and the name of an application shall be displayed in the current language of the user interface, you must define the following for the `SELECT` statement to display the application name in the current language:

```
SELECT REFSTR, NAME, NAME_1031, NAME_1036, VERSION
FROM APPLICATION
```

In addition, the `SetTranslatedValue` instruction must be added to the query definition with the definition of the `NAME` column as the column in the report that shall contain the relevant translated data:

```
SetTranslatedValue("NAME");
```

Translating Text Defined Directly in the SELECT Statement of Native SQL Queries

In native SQL queries, static text can be defined as return value for a column in the `SELECT` statement.

For example, the following native SQL query contains the fixed strings `Action required` and `You are not in charge` in the column `Importance`:

```
SELECT app.REFSTR, app.NAME As 'Application', app.VERSION As
'Version', CASE WHEN app.RESPONSIBLEUSER = @CURRENT_USER THEN
'Action required' ELSE 'You are not in charge' END As 'Importance'
FROM APPLICATION app WHERE app.STATUS = 'Draft'
```

The strings are by default not translated if the user interface is rendered in a secondary language.

If you want a configured report based on a native SQL query to display translations for static strings defined in the `SELECT` argument as values for the cells in a column, the `SetTranslatedValue` instruction must be added to the query definition:


```
TranslateColumnContent("ColumnName","Untranslated value1,Untranslated
value2,...");
```

Use the following parameter(s):

Code Parameter	Description
`ColumnName`	The name of the column in the tabular output of the report that contains the strings. For information about the names of the columns resulting from native SQL queries, see Defining Column Names and Captions. Translation is limited to the defined column in the instruction. If the same strings are available in a second column as well, they will not be translated in that column, but only in the column defined in the instruction. You must add another `TranslateColumnContent` instruction for the second column to add translation to it.
`Untranslated value1, Untranslated value 2`	The values in the column that shall be translated in a comma separated list of strings. Please note the following: • Only strings that are listed in the instruction are displayed translated. If you do not add a string to the list, its translation will not be displayed, even if the translation is available in the `METAMODEL` vocabulary due to another context using an identical string. • A translation must be available for the string in the `METAMODEL` vocabulary. Translations in other vocabularies are ignored. If strings listed here are not available in the `METAMODEL` vocabulary due to another context using an identical string, they will automatically be added to the `METAMODEL` vocabulary on storage of the query containing the instruction. You must then provide a translation in the `METAMODEL` vocabulary for all applicable cultures to view a translation in the column in the configured report. For information about translating vocabulary strings, see Modifying, Translating and Managing the Vocabularies.

The instruction can be used in all native SQL queries defined for reports of the type `NativeSQL` or `Custom`.

Displaying the Sum of Multiple Values in a Table Row

For tabular configured reports with multiple columns that display data of the data type Real or Integer, the sum of the values in multiple columns can be calculated per row and displayed in a separate row in bold text.

Do the following to display the sum of multiple values in a row:

1) Add a new, empty column to the dataset.

   • If the report is based on an Alfabet query, a new column can be added using instructions. See Inserting One or Multiple Columns to a Report for information about the required instructions.

- If the report is based on a native SQL query, a new column can either be added using instructions (see Inserting One or Multiple Columns to a Report for details) or by adding a new column with static content directly in the `SELECT` statement.

In the following dataset, a column "`Risk Factor`" with the integer 1 as fixed value is added to the dataset in addition to the columns containing data from the Alfabet database.

```
SELECT RISK.REFSTR, RISK.NAME As 'Risk', RISK.DAMAGE As
'Damage', RISK.PROBABILITY As 'Probability',1 AS 'Risk
Factor'

FROM RISK
```

2) Add the following instruction to the report to fill the new column with the sum of the values of selected values of the type Real and Integer in each row:

```
CreateRowSum("ValueColumnNames","ResultColumnName");
```

Use the following parameter(s):

Code Parameter	Description
`ValueColumn-Names`	The column names of the columns containing the values for that the sum is calculated delimited with commas. The values must be of the data type Real or Integer. Alternatively, the position of the column in the index of columns can be used to specify the column. For more information see Referencing Columns in Instructions by Index Number.
`ResultColumn-Name`	The name of the column displaying the calculated sum. If you define a column that already contains data, the data is overwritten. Alternatively, the position of the column in the index of columns can be used to specify the column. For more information see Referencing Columns in Instructions by Index Number.

In the following Alfabet query based report, the risk evaluation for applications is displayed. The evaluations concerning the risk damage and risk probability are summed up in the last row to highlight the relevance of the risk.

Risk	Object	Damage	Probability	Ri
CVE-2013-1813	BLOOMBERG	1.00	3.00	4
Database-driven Application Failure	BLOOMBERG	1.00	2.00	3
Faulty Code Changes	BLOOMBERG	2.00	1.00	3
Lack of Storage	BLOOMBERG	2.00	2.00	4

The report is based on the following query:

```
ALFABET_QUERY_500
```



```
FIND Risk
InnerJoin Application ON Risk.Object = Application.REFSTR
Instructions
   AddColumns("Sum");
   CreateDSInfo("Risk.Name,Application.Name,Risk.Damage,Risk.Probabi
   lity,Sum", "Risk,Object,Damage,Probability,Risk Factor");
   CreateRowSum("Risk.Damage,Risk.Probability","Sum");
EndOfInstructions
AUTODSINFO
QUERY_XML
<QueryDef>
   <ShowProperty Type="Property" ClassName="Risk" Name="Name" />
   <ShowProperty Type="Property" ClassName="Application" Name="Name"
   />
   <ShowProperty Type="Property" ClassName="Risk" Name="Damage" />
   <ShowProperty Type="Property" ClassName="Risk" Name="Probability"
   />
</QueryDef>
```

Displaying the Sum of All Values in a Table Column

For tabular configured reports, the sum of the values in a column displaying Real or Integer values can be calculated per row and displayed in an additional row added below the result rows. Optionally, a text can be specified that is displayed in a selected column of the report. Both the text and the sum of values are displayed in bold text.

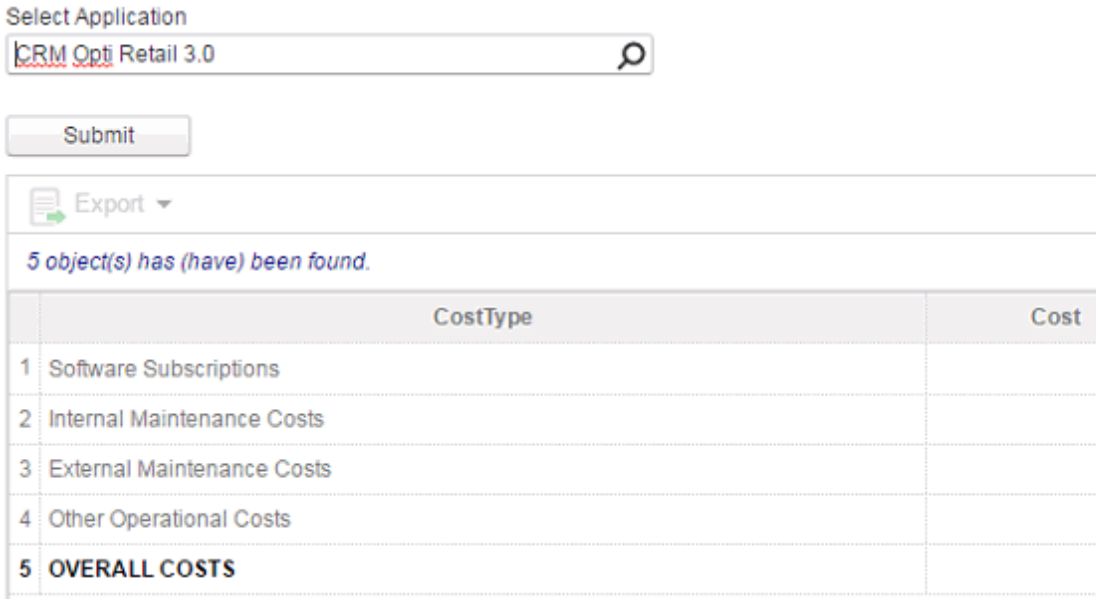
Use the following instruction to display the sum of all results in a column below the result column:

```
CreateColumnSum("ValueColumnName","CaptionColumnName","Caption");
```

Use the following parameter(s):

Code Parameter	Description
`ValueColumn-Name`	The name of the column containing the values for that the sum is calculated. The values must be of the data type Real or Integer.
`Caption-ColumnName`	The name of the column displaying the caption in the additional data set row for the sum.
`Caption`	The text displayed in the additional data set row for the sum in the column defined by `CaptionColumnName`.

In the following native SQL query based report, the current costs of an application are displayed. The report is applied to the class `Application`. The user can select the application for that the costs are displayed in the filter field on top of the report. The costs for each cost type is listed separately and the sum of all costs is displayed in the last row. The sum is added to the report via the `CreateColumnSum` instruction. The caption `OVERALL COSTS` is displayed in the row for the sum in the `CostType` column.



	CostType	Cost
1	Software Subscriptions	
2	Internal Maintenance Costs	
3	External Maintenance Costs	
4	Other Operational Costs	
5	**OVERALL COSTS**	

The report is based on the following native SQL query:

```
SELECT BV.REFSTR, ct.NAME As 'CostType', BV.VALUE As 'Cost'
FROM BUDGETVALUE BV, APPLICATION app, COSTTYPE ct
WHERE ct.REFSTR = BV.MONETARYTYPE
    AND BV.OWNER = app.REFSTR
    AND BV.MONETARYCODEID='Current'
    AND BV.YEAR = '2014'
    AND app.REFSTR = @BASE
```

The query is combined with the following instruction:

```
CreateColumnSum(Cost, CostType, "OVERALL COSTS");
```

Grouping Results by Weighting of Values in a Defined Column

For tabular configured reports with a column displaying data of the type Real or Integer, the results can be grouped according to the results in that column. The number of groups is configurable. The overall number of rows is divided by the number of groups to evaluate how many results belong to each group. If the number of results cannot be divided equally among the groups, the first groups each get one more member assigned until all value are assigned.

The assignments of the results to each group depends on the values in the column used for grouping. The results are sorted in descending order. The results with the lowest numbers are members of the first group. The next groups are each having higher numbers in the result data set.

Each of the resulting groups has a number, starting with 1. The group number is displayed in the report in a separate column.

Do the following to divide the data set into groups formed by weighting of values in a defined column.

1) Add a new, empty column to the dataset. The column must be of the data type Integer or Real.

It is not possible to add the new column to the dataset using instructions. The empty columns added to the data set via instructions do not have the data type Integer or Real.

- If the report is based on a native SQL query, add the new column directly in the `SELECT` statement. The static value for the column must be an integer or real number to set the data type of the column to Integer or Real.

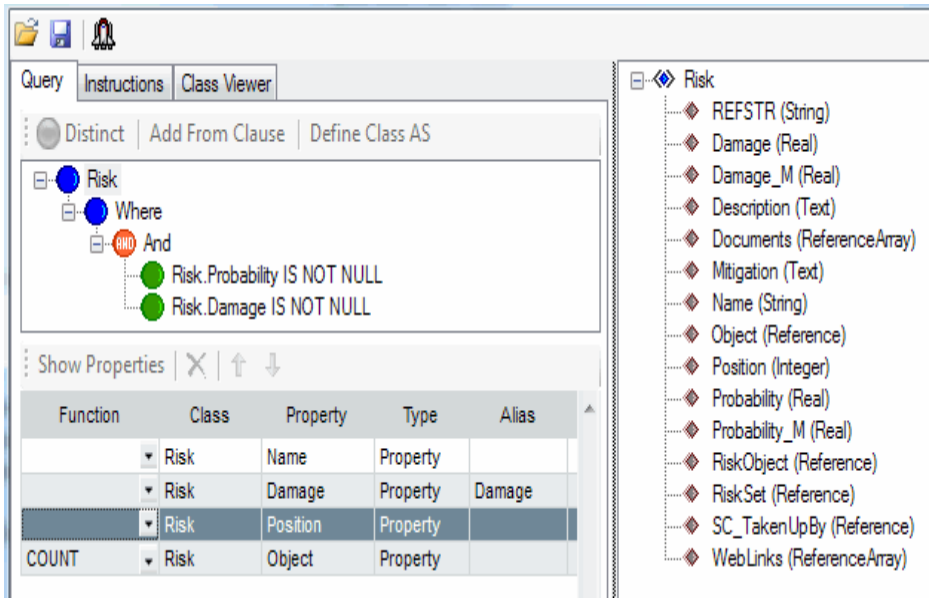
In the following dataset, a column "`Weighting`" with the integer 1 as fixed value is added to the dataset in addition to the columns containing data from the Alfabet database.

```
SELECT RISK.REFSTR, RISK.NAME As 'Risk', RISK.DAMAGE As
'Damage', RISK.PROBABILITY As 'Probability',1 AS
'Weighting'

FROM RISK
```

- If the report is based on an Alfabet query, a column containing integer data from the Alfabet database must be added to the report via a Show property. The data in the column is later overwritten by the grouping information. If the `FIND` class do not have a property of the type Integer or Real, you can For example, use a `COUNT` function on a property of the **Data Type** `Reference` or `ReferenceArray`.

For example, to add a new column to the data set defined for the class `Risk` that shall be used to display grouping performed by weighting of the risk damage value, there are two possibilities:

  - Add one of the other object class properties of the **Data Type** `Integer` or `Real` of the object class `Risk` to the Show properties, For example, the object class property `Position`.
  - Add an object class property of the **Data Type** `Reference`, For example, the property `Object`, to the Show properties and apply a `COUNT` function on that Show property.



2) Add the following instruction to the report to fill the new column with the weighted group information:

```
NTile_Weighted("ValueColumnName","GroupColumnName","Caption");
```

Use the following parameter(s):

Code Parameter	Description
`Value-ColumnName`	The column name of the columns containing the values used to define group assignment. The values must be of the data type Real or Integer.
`Group-ColumnName`	The name of the column displaying the group number. The data type of the column must be Real or Integer.
`Tile`	The number of groups the result set is split into.

In the following Alfabet query based report, the risk evaluation for applications is displayed. The evaluations concerning the risk probability are used to group the results into four groups. The group number is displayed in the column `Tile`. Sorting was performed on the column `Tile`.

	Application	NAME	Damage	Probability	Tile
1	Eurex	Availability	1.00	1.00	1
2	Eurex	Availability	1.00	1.00	1
3	Eurex	Integrity	1.00	1.00	1
4	Eurex	Integrity	1.00	1.00	1
5	GLOSS BUREAU	Confidentiality	1.00	1.00	1
6	SUNGARD TREASURY TRADER	Lack of Storage	1.00	1.00	1
7	BLOOMBERG	Faulty Code Changes	2.00	1.00	1
8	CRES	Integrity	2.00	1.00	1
9	GGM	Integrity	2.00	1.00	1
10	SUNGARD TREASURY TRADER	Spying of Critical Data	2.00	1.00	1
11	TradeThru	Spying of Critical Data	2.00	1.00	1
12	Wagner Classifier	Configuration Errors	2.00	1.00	1
13	EasiTrade Web	Accidental Grant of Access	3.00	1.00	1
14	EasiTrade Web	Power Failure	3.00	1.00	1
15	Wagner Classifier	Accidental Grant of Access	3.00	1.00	1
16	Wagner Classifier	Catastrophic Events	3.00	1.00	1
17	Links	Catastrophic Events	4.00	1.00	1
18	Trade*Net	Catastrophic Events	4.00	1.00	1
19	Trade*Net	Spying of Critical Data	4.00	1.00	1
20	BLOOMBERG	Database-driven Application Failure	1.00	2.00	2
21	CRES	Availability	1.00	2.00	2
22	BLOOMBERG	Lack of Storage	2.00	2.00	2
23	Eurex	Confidentiality	2.00	2.00	2
24	Fed Ex	Confidentiality	2.00	2.00	2

The report is based on the following native SQL query:

```
SELECT RISK.REFSTR, app.NAME As 'Application', RISK.NAME,
RISK.DAMAGE As 'Damage', RISK.PROBABILITY As 'Probability', 1 AS
Tile FROM RISK, APPLICATION app WHERE RISK.DAMAGE IS NOT NULL AND
app.REFSTR = RISK.OBJECT;
```

The query is combined with the following instruction:

```
NTile_Weighted(Probability,Tile, 4);
```

Defining Queries in XML Elements

The configuration of some Alfabet functionalities requires the definition of queries in XML objects. For example, this applies to the definition of custom selectors, the full-text search index, and rule-based access permissions. In this case, the query is defined as a value of an attribute of the relevant XML element.

For example,:

```
<Query
   Class="Application"
   Query="ALFABET_QUERY_500 FIND Application WHERE Application.Name
   LIKE 'CRM%'
   SHOW Application.Name"/>
```


Please take the following into account when specifying a query in an XML element.

- If you want to enter a string that contains special characters (For example, a greater then (>) or lesser then (<) symbol), you must replace the special characters with respective XML compliant code. For example:
  - > for >
  - < for <
  - " for "
  - [for [
  -] for]
- If you define an Alfabet query, the default specification of show and sort properties that is the standard output of the **Alfabet Query Builder** is an XML definition that cannot be pasted as-is into an XML attribute. Therefore, the show and sort properties have either to be defined escaped, in other words, all brackets must be written in XML compliant code, or defined as follows:

`SHOW` (for show properties) or `SORT` (for sort properties) followed by *ObjectClassName.PropertyName*.

You can define multiple properties separated by a whitespace.

For example,:

```
SHOW Application.Name Application.ID Application.StartDate
SORT Application.Name
```

When instructions are used in a query, you must define the show property as XML and substitute all brackets with either < for < and > for >. The alternative syntax listed above cannot be used in combination with instructions.

Creating an Index to Improve Performance for Query Searches in Database Tables

A database index is an ordered and thus quickly searched part of a database table. If the database must search in a particular table for entries matching a specified value, then this process will be significantly sped up if the database can use an index for finding the data instead of requiring a full table scan.

You can configure the Alfabet database to create indices for all object data that is frequently used. Index creation is a side effect of a class key definition. Usually, class keys are defined to implement uniqueness constraints. For each class key that is configured for an object class in Alfabet Expand, an index is created. Each class key has a **Unique** attribute that can be set to `False` to allow an index to be created for the class key without implementing a uniqueness constraint for the class. Thus, the definition of class keys can be used to speed up query-based searches.

In addition to using class keys to define an index created for data in a database table, audit keys can be defined to trigger index creation for audit tables. Each object class for which auditing is activated can have audit keys defined. For example, if you have specific custom reports that take a significant amount of time to be executed, the execution of the report can often be significantly sped-up by creating proper indices.

Please be aware that every index defined for a database table slightly reduces the data update and insert performance for entries in the table. Therefore, you should create indices only where it is specifically useful or necessary.

Class keys and audit keys must be unique within an object class. You cannot create a key that duplicates existing public or preconfigured key of the same type for the same object class. Two class keys are identical if the values for their attributes **Properties**, **Unique** and **Strict Null Handling** are identical. For audit keys, the value of the **Columns** attribute must be unique.

The two sections below explain the creation of indices via a class key and the creation of indices via an audit key.

To create a class key to define an index without uniqueness constraints for a selected object class:

1) In the **Meta-Model** tab, right-click the protected class  that you want to create the class key for and select **Add New Class Key**. You will see the **Class Key Group** folder and the new class key subordinate to the selected object class.

2) Click the new class key  to open the attribute window. In the attribute window, edit the name for the class key in the **Name** field, if necessary.

3) Click the cell to the right of the **Properties** field and click the **Browse**  button. You will see the dialog box in which you can define the properties to be included into the index.

4) Click each property that you want to include in the index. After all necessary properties have been selected, click **OK**. You will see the new key below the **Class Key Group** folder.

5) In the attribute window, set the **Unique** attribute to `False`.

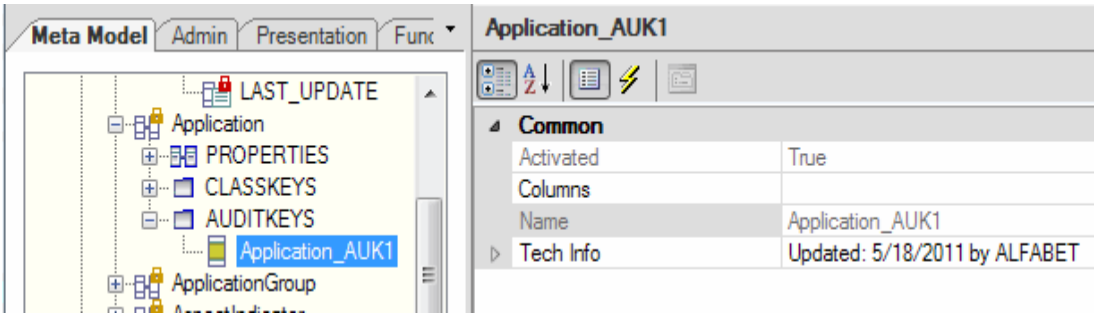
6) In the toolbar, click the **Save**  button to save your changes.

To delete all protected and custom class keys that have been configured for a protected or custom object class, click the **Class Key Group** node below the relevant object class and select **Delete All Public/Protected Class Keys**. The protected and public class keys will be deleted and no longer displayed below the **Class Key Group** node. Private class keys that have been configured by Software AG will not be deleted.

To create an audit key defining an index for audit table data:

1) In the **Meta-Model** tab, right-click the protected class  that you want to create the audit key for and select **Add New Audit Key**. You will see the **Audit Key Group** folder and the new audit key subordinate to the selected object class.

The **Audit Key Group** node is only displayed if the **Audit** attribute of the object class is set to `True`.



The **Audit** attribute can be set to `True` for all object classes by clicking the **Classes** node and selecting **Turn Audit On for All Protected/Public Classes**. If selected The **Audit** attribute will be set to `True` for all protected and public object classes. The **Audit** attribute can be set to `False` for individual classes or set to `False` for all protected and public classes by selecting **Turn Audit Off for All Protected/Public Classes**.

2) Click the new audit key to open the attribute window.

3) Click the cell to the right of the **Columns** field and click the **Browse** button. You will see the dialog box in which you can select the data from the audit table for the index created on basis of the audit key.

4) Click each property that you want to include in the index. After all necessary properties have been selected, click **OK**. You will see the new key below the **Audit Key Group** folder.

Please note that the attribute **Columns** must be unique for audit keys of the same object class.

5) In the toolbar, click the **Save** button to save your changes.

Some processes like ADIF imports or rescan of indicators either via the command line tool `RescanIndicatorsConsole.exe` or via the **Job Scheduler** functionality may trigger a high number of single database insert and delete transactions. This can have a negative impact on fragmentation of indices. As a result, CPU usage is increased and performance is reduced. It is recommended to rebuilt indexes in regular intervals for object classes subject to ADIF import or rescan of indicators. This can be done using the command line tool `AlfaAdministratorConsole.exe`. For more information, see *Rebuilding Indexes on Database Tables* in the reference manual *System Administration*.

To delete all protected and custom audit keys that have been configured for a protected or custom object class, click the **Audit Key Group** node below the relevant object class and select **Delete All Public/Protected Audit Keys**. The protected and public audit keys will be deleted and no longer displayed below the **Audit Key Group** node. Private audit keys that have been configured by Software AG will not be deleted.

Creating Database Views To Enhance Performance And Support Search Functionalities

Database views are searchable database constructs based on a native SQL query with a `SELECT` statement or an Alfabet query. A database view is like a virtual table represented by the native SQL query or Alfabet query. The query syntax used to define database views in native SQL on the database server level is:

```
CREATE VIEW ViewTableName AS SELECT ViewContentDefinition
```

For the Alfabet database, database views can be created and maintained in the tool Alfabet Expand without the need to directly access and configure the database server.

A configuration object **Database View** is available for defining the database view. The name of the database view table and the `SELECT` statement for the content of the database view are defined in separate attributes of the configuration object. When the configuration is saved, the `CREATE_VIEW` statement for creating the view is automatically built and executed based on the defined information.

The content of the table resulting from the output of the native SQL query or Alfabet query defined as `ViewContentDefinition` is not stored in the database. Instead, the native SQL query is executed each time data is requested from the database view table defined with `ViewTableName` by any other native SQL query targeting the database view table. Targeting the database view table in a native SQL query is done in the same way as targeting a standard database table. In the Alfabet query language targeting of a database view table is not supported.

Database views do not store data and Therefore, take up very little database space. They offer the following advantages for building queries For example, for reporting:

- **Simplicity**: Complex parts of queries with For example, multiple joins across multiple database tables can be stored in a database view. Query definition during solution configuration can then be based on simple queries referencing the columns of the database view without any joins required.
- **Reusability**: If multiple queries in the configuration gather the same information from the tables in the database as part of the query output, a database view can be created to implement the part that is identical for all the queries.
- **Consistency**: The way For example, complex KPI calculations are done in queries can be defined in database views and the results are then consistent in all queries defined for Alfabet.
- **Easy debugging in case of database changes**: Changes to the class model, For example, when upgrading to a new Alfabet release or when deleting custom object class properties, may require re-definition of queries defined for configuring Alfabet. If information is first gathered in a database view and then re-used in many queries that are defined for configuring Alfabet, you do not have to change all queries but only the underlying database view query if a change in the class model requires a re-definition of the query content. In addition, a mechanism is implemented in Alfabet Expand that can be used to detect errors in database view configurations after changes to the class model.

In addition, database views can be used to support the automatic generation of reports for the faceted semantic search provided via the `Analyze` intent of the AlfaBot.

The content of a database view can either be defined as a native SQL query or an Alfabet query. The result dataset is built as follows depending on the type of query defined:

- **Native SQL**

For native SQL queries, the columns and column name of the database view table are derived from the `SELECT` statement of the native SQL query. Unlike all other native SQL query definitions for configuring Alfabet, the first definition of the `SELECT` statement is NOT removed from the resulting dataset. The complete definition in the `SELECT` statement results in database view table columns.

The table names defined as For example, `APPLICATION.NAME` will result in a column name `NAME` only. To avoid problems with name specifications it is recommended to define an alias with the desired name in the `SELECT` statement for the database view.

- **Alfabet Query**

Which properties of the object classes added to the query are added to the database view as table columns depend on the following settings:

- If Show properties are defined for the Alfabet query, the number of columns correspond to the number of show properties defined. Please note that Show Properties of the type `RoleType` and `Indicator` are ignored. Only Show Properties of the type `Property` are used to create columns in the database view.
- If no show properties are defined, all properties except for properties of the type `ReferenceArray` are added to the database view. Optionally, properties of defined data types can be excluded from the database view table by setting the attribute **Filter Show Properties** of the database view.

The column names in the database view are derived from the query in the following way:

- The column name is concatenated as object class caption followed by property caption delimited by a whitespace. For example, for the property `Start Date` of an `Application`, the resulting column name is `Application Start Date`. Please note that special characters are not stripped from the captions when building the database view column captions. If no caption is defined, the Name of the object class or object class property is used.
- If Show Properties are defined and an Alias is defined for a property in the Show Properties, the defined Alias is used instead of the concatenated column name.
- If an Alias is defined for the object class in the Alfabet query, the alias is used in the column name instead of the class name. An alias can be used For example, to avoid name abbreviation by the system for column names longer than 30 characters.
- If the resulting column name is longer than 30 characters, the name will be cut after 30 characters, corresponding to the string length restrictions that apply on Oracle® database servers. Depending on the property caption, this might cause problems if the resulting column names are identical in the first 30 characters. Alias definitions in the query can help solving such problems.

The following information is available:

- Creating a Database View Based on a Native SQL Query
- Creating a Database View Based on an Alfabet Query
- Targeting a Database View in Queries and Other Database Views
- Changing the Order of Execution for Database Views
- Checking the Consistency of the Database View With the Meta-Model

Creating a Database View Based on a Native SQL Query

To create a database view based on native SQL in Alfabet Expand:

1) Open the **Meta-Model** tab
2) Right-click the explorer node **Database Views** and select **Add New Database View** from the context menu. The new database view is displayed as new explorer node under the **Database Views** node and the attribute window of the new database view node opens.
3) In the attribute window of the new database view, set the following attributes:
   - **Type**: Select `NativeSQL`.
   - **Name**: Define a name for the Database View configuration object. The name must be unique. It is used to identify the Database View object in the explorer of Alfabet Expand.
   - **Technical Name**: Define the name of the virtual database table that shall be created via the database view. Queries that want to read data from the database view must refer to the database view with this name. The technical name must be in compliance with the rules that apply to table names on your database server, that means special characters and reserved keywords as well as names of already existing database tables must not be used.
   - **Query/Native SQL Query as Text**: Define the content of the database view via a native SQL query in a simple text editor (**Native SQL Query as Text**) or in an Alfabet specific text editor with integrated help that provides information about the Alfabet meta-model (**Query**). The query must have a `SELECT` statement. The column names defined in the `SELECT` statements are the column names for the database view. Queries defined to read data from the database view must use the defined names to refer to columns of the database view table.

     Please note the following:
     - Unlike all other native SQL query definitions for configuring Alfabet, the first definition of the `SELECT` statement is NOT removed from the resulting dataset. The complete definition in the SELECT statement results in database view table columns.
     - The table names defined as For example, `APPLICATION.NAME` will result in a column name `NAME` only. To avoid problems with name specifications it is recommended to define an alias with the desired name in the `SELECT` statement for the database view.
   - **Description**: Optionally, enter information about the purpose of the database view. This information is displayed to solution designers in Alfabet Expand only.

- **Applicable for AlfaBot**: Set this attribute to `True` if the database view should be used to automatically generate reports for the `Analyze` intent of the AlfaBot which can be used to search for information in configured reports. The mechanism for generation of ad-hoc reports will then scan the **Semantic Analysis** explorer of the database view to find out whether the database view provides information a user is currently searching for. If the scan is positiv, a configured tabular report targeting the database view will be generated and provided to the user as one of the search results. By default, the attribute is set to `False`.

For more information about the faceted semantic search provided by the `Analyze` intent of the AlfaBot, see Define AlfaBot Search Capabilities for Configured Reports.

- **Base Class**: If the **Applicable for AlfaBot** attribute is set to `True`, select a base class from the list of object classes identified from the query of the database view. Automatically generated reports for the `Analyze` intent of the AlfaBot will only be created on basis of this database view if the **Base Class** attribute is set to the object class identified to be the object class the user is currently searching for. If the base class is not defined, the database view will not be used for automatic generation of reports.

4) In the toolbar, click the **Save** button to save your changes.
5) Right-click the explorer node **Database Views** and select **Recreate Database Views** from the context menu to create the database view in the Alfabet database. The attribute **Created** of the database view should then be set to `True`. If it is set to `False`, and an error message is displayed in the attribute **Last Error**, the query defined for the query attribute needs correction.

Clicking the **Save** button will store your configuration and create the database view in the Alfabet database. The attribute **Created** of the database view should then be set to `True`. If it is set to `False`, and an error message is displayed in the attribute **Last Error**, the query defined for the query attribute needs correction.

Creating a Database View Based on an Alfabet Query

To create a database view based on an Alfabet query in Alfabet Expand:

1) Open the **Meta-Model** tab
2) Right-click the explorer node **Database Views** and select **Add New Database View** from the context menu. The new database view is displayed as new explorer node under the **Database Views** node and the attribute window of the new database view node opens.
3) In the attribute window of the new database view, set the **Type** attribute to `AQL`. The attributes are changing to the attributes relevant for Alfabet query language based attributes.
4) Set the following attributes:

- **Name**: Define a name for the Database View configuration object. The name must be unique. It is used to identify the Database View object in the explorer of Alfabet Expand.
- **Technical Name**: Define the name of the virtual database table that shall be created via the database view. Queries that want to read data from the database view must refer to the database view with this name. The technical name must be in compliance with the rules that apply to table names on your database server, that means special characters and reserved keywords as well as names of already existing database tables must not be used.

- **Description**: Optionally, enter information about the purpose of the database view. This information is displayed to solution designers in Alfabet Expand only.
- **Query/AQL Query as Text**: Define the content of the database view via an Alfabet query in a simple text editor (**AQL Query as Text**) or in the Alfabet query builder (**Query**). Sort properties and Alfabet query language instructions must not be defined for the Alfabet query. The definition of Show Properties is optional.

Which properties of the object classes added to the query are added to the database view as table columns depend on the following settings:

- If Show properties are defined for the Alfabet query, the number of columns correspond to the number of show properties defined. Please note that Show Properties of the type `RoleType` and `Indicator` are ignored. Only Show Properties of the type `Property` are used to create columns in the database view.
- If no show properties are defined, all properties except for properties of the type `ReferenceArray` are added to the database view. Optionally, properties of defined data types can be excluded from the database view table by setting the attribute **Filter Show Properties** of the database view.

The column names in the database view are derived from the query in the following way:

- The column name is concatenated as object class caption followed by property caption delimited by a whitespace. For example, for the property `Start Date` of an `Application`, the resulting column name is `Application Start Date`. Please note that special characters are not stripped from the captions when building the database view column captions. If no caption is defined, the Name of the object class or object class property is used.
- If Show Properties are defined and an Alias is defined for a property in the Show Properties, the defined Alias is used instead of the concatenated column name.
- If an Alias is defined for the object class in the Alfabet query, the alias is used in the column name instead of the class name. An alias can be used For example, to avoid name abbreviation by the system for column names longer than 30 characters.
- If the resulting column name is longer than 30 characters, the name will be cut after 30 characters, corresponding to the string length restrictions that apply on Oracle® database servers. Depending on the property caption, this might cause problems if the resulting column names are identical in the first 30 characters. Alias definitions in the query can help solving such problems.

- **Filter Show Properties**: This attribute is only relevant if the Alfabet query defining the database view has no Show Properties. If set to `True`, the following additional attributes are added to the attribute section. You can use the attributes to exclude object class properties with a defined data type from the database table view:
  - **Exclude Boolean Properties**: Select `True` to exclude all object class properties with the attribute **Property Type** set to `Boolean` from the database view.
  - **Exclude Color Properties**: Select `True` to exclude all object class properties with the attribute **Name** containing the string `Color` from the database view.
  - **Exclude Icons**: Select `True` to exclude all object class properties with the attribute **Name** containing the string `Icon` from the database view.

- **Exclude Last Update**: Select `True` to exclude all object class properties the attribute **Name** set to either `LAST_UPDATE`, `LAST_UPDATE_USER`, `CREATION_DATE` or `CREATION_USER` from the database view.
- **Exclude Reference Properties**: Select `True` to exclude all object class properties with the attribute **Property Type** set to `Reference` from the database view.

- **Applicable for AlfaBot**: Set this attribute to `True` if the database view should be used to automatically generate reports for the `Analyze` intent of the AlfaBot which can be used to search for information in configured reports. The mechanism for generation of ad-hoc reports will then scan the **Semantic Analysis** explorer of the database view to find out whether the database view provides information a user is currently searching for. If the scan is positiv, a configured tabular report targeting the database view will be generated and provided to the user as one of the search results. By default, the attribute is set to `False`.

For more information about the faceted semantic search provided by the `Analyze` intent of the AlfaBot, see Define AlfaBot Search Capabilities for Configured Reports.

- **Base Class**: If the **Applicable for AlfaBot** attribute is set to `True`, select a base class from the list of object classes identified from the query of the database view. Automatically generated reports for the `Analyze` intent of the AlfaBot will only be created on basis of this database view if the **Base Class** attribute is set to the object class identified to be the object class the user is currently searching for. If the base class is not defined, the database view will not be used for automatic generation of reports.

5) In the toolbar, click the **Save** button to save your changes.

Targeting a Database View in Queries and Other Database Views

Database views can be used as searchable data set in native SQL queries. Use of database views in Alfabet query language is not supported.

When data from a database view is included into the `SELECT` statement of a native SQL query, the call to the database view causes the query of the database view to be executed with the filter conditions defined in the native SQL query that calls the database view data.

The column names in database views based on an Alfabet query usually contain white spaces and might also contain special characters. Column names should be written into the select statement of the native SQL query targeting the database view in inverted commas.

```
SELECT "Application Name", "Application Property #1" FROM
ApplicationBasedDatabaseView
```

A native SQL query defined for creating a database view can also call data from other database views. If you create a database view that calls data from another database view, you must make sure that the configuration object for that database view is located beneath the configuration object for the called database view in the **Database Views** node of the **Meta-Model** explorer in Alfabet Expand.

Changing the Order of Execution for Database Views

Database views are not sorted alphabetically in the explorer, but are listed in the order of definition. The order of definition equals the order of execution of the database views For example, when recreating all database views for debugging. A database view can target another database view and these views must be executed after the targeted database views are executed. This can be ensured by execution in definition order.

You can change the order of execution by re-defining the sort order in the explorer:

1) Open the **Meta-Model** tab
2) Click the explorer node **Database Views**. The attribute window of the node opens.
3) Click the **Browse** button in the field of the **Database Views** attributes. The **Sort Entries** button window opens.
4) Click a database view in the list and move it in the list with the **Up** and **Down** buttons at the upper right of the **Sort Entries** window.
5) Click **OK** to save your changes.

Checking the Consistency of the Database View With the Meta-Model

After changes have been made to the class model, database views might require adaptations to the changes.

To check whether the database views are still valid, a mechanism is implemented in Alfabet Expand that recreates all database views in the database and displays error messages for errors that occurred during recreation.

To check the database views for consistency with the meta-model:

- Open the **Meta-Model** tab
- Right-click the explorer node **Database Views** and select **Recreate Database Views** from the context menu. All database views are recreated in the Alfabet database in the order of definition.

  A database view can target another database view and these views must be executed after the targeted database views are executed. This can be ensured by execution in definition order.

- Click on each database view in the explorer to check whether errors occurred. Errors can be read from the following read only attributes of the database view configuration object:
  - **Created**: Displays `True` if the database view has been created. Displays `False` if the database view has not been created. Database views with the attribute **Created** set to `False` need to be corrected.
  - **Last Error**: Displays the error that occurred during database view creation.

If you have defined a high number of database views, you can avoid clicking each node in the explorer by creating a configured report based on a native SQL query that reads the information which database views were not created because of an error during creation from the database

table that stores the Database View configuration object. The following native SQL query will return the name of all database views that require re-configuration:

```
SELECT REFSTR, NAME
FROM ALFA_DB_EXT
WHERE ISCREATED = 'False'
```

You can then directly open the explorer nodes of the database views returned by the report to see the error message and correct the query of the database view accordingly.

Database views are recreated after each **Update Meta-Model** or **Restore Database Archive** action and need to be checked as described above afterwards. The recreation ensures that new database views added to the configuration via the **Update Meta-Model** or **Restore Database Archive** action are created and that database views that are not compatible with the changes to the meta-model applied via Meta-Model Update are removed. For more information about update of the meta-model or restore of a database archive see the reference manual *System Administration*.

Maintaining the Information in the Semantic Analysis Sub-Node of the Database View

The semantic analysis functionality for database views scans all queries in a database view for references to Alfabet object classes and object class properties thereof.

The information is used for the following functionalities:

- The **Show Usage** functionality in the context menu of object class properties evaluates the use of the object class property in configured reports on basis of the semantic analysis results.
- The AlfaBot uses the semantic analysis to build automatically generated reports from the database view to provide additional search results for the `Analyze` intent finding configured reports displaying information about defined object class properties and values thereof.

The results are listed in a node **Semantic Analysis** beneath the node of each datase view in a hierarchical structure going down from object classes to object class properties to values of object class properties. The Name property of object classes and object class properties is used in the explorer.

If object class properties restrict the content of the database view via a `WHERE` clause definition in the underlying query, the values will be included into the semantic analysis for object class properties returning one of the following:

- The value of a standard `Stereotype` object class property.

The semantic analysis ignores value definitions including a wildcard and checks whether the object class stereotype name equals a stereotype name defined for the object class. Only existing object class stereotypes are listed.

- The value of a the `Name` of an object.

The semantic analysis ignores value definitions including a wildcard.

- A Boolean value.

- A value from an enumeration.

The semantic analysis ignores value definitions including a wildcard and checks whether the enumeration value is defined for the enumeration defined for the object class property. Only existing enumeration items are listed.

In addition, the semantic analysis maps all column names in the database view with the object class and object class property for which data is returned in the column. The results are listed in a sub-folder **Aliases**. this information is used by the mechanism for automatic generation reports to find the correct database views for generation of meaningful ad-hoc reports for the current user request.

Scanning is done automatically after the query for the database view has been defined.

Please note the following:

- If the **Semantic Analysis** node is empty, this is an idicator for an error in the underlying query, For example, a typo in an object class name.

The analysis scans both Alfabet queries and native SQL queries. It comprises object class properties referenced by any part of the query including joins and the object class properties of the type `ReferenceArray` which are included in native SQL queries via the `PROPERTY` column of the `RELATIONS` database table.

Alfabet queries and native SQL queries are handled differently:

- The semantic analysis for database views based on Alfabet queries is not editable.

Indicator and role definitions added via the show properties of the type `Indicator` or `Role` are not included into the semantic analysis results.

The `REFSTR` object class property of the `FIND` class of the Alfabet Alfabet query is added as object class property to the semantic analysis.

- Native SQL queries can be very complex and in some cases, the scanning mechanism may not find all relevant object classes and object class properties. Therefore, the solution designer can add missing information to the semantic analysis via the **Add Properties** option in the context menu of the **Semantic Analysis** node. Object class property values included in the semantic analysis are ReadOnly and excluded from editing. The semantic analysis is executed each time the native SQL query of the database view is changed. The manually-added semantic analysis results will be left unchanged during a subsequent semantic analysis.

Information about object classes and object class properties may be missing in the analysis for one of the following reasons:

- The query definition is incorrect. For example, an object class or object class property name may have been added with a typo. It is recommended to check the query definition for correctness if the semantic analysis is missing or incomplete.
- Native SQL queries can be very complex and the scanning mechanism for semantic analysis may fail to identify object class or object class property specification.

If the semantic analysis of a correct native SQL query is incomplete, missing information can be added manually to complete the analysis:

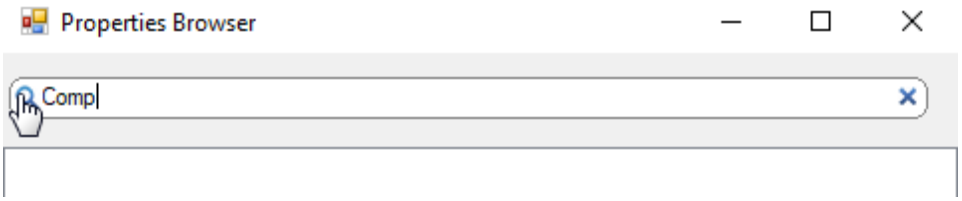
- Adding an Object Class to the Semantic Analysis
- Adding an Object Class Property to the Semantic Analysis

- Removing Manually Added Entries From the Semantic Analysis

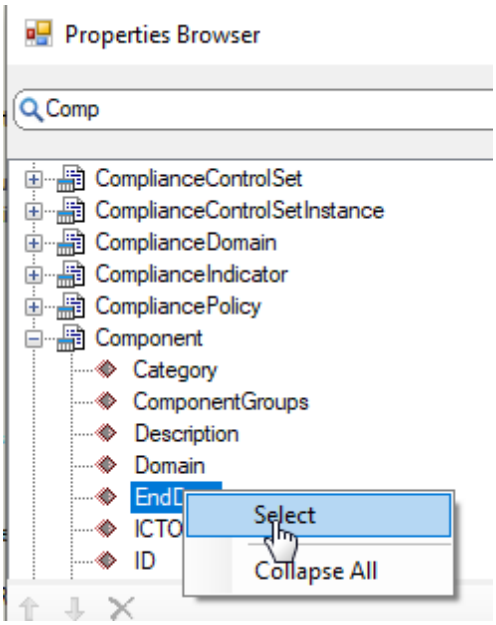
Adding an Object Class to the Semantic Analysis

To add information about an object class missing in the automatic semantic analysis output:

1) Right-click the **Semantic Analysis** node and select **Add Properties**.
2) In the selector window, type at least the first three letters of the object class name into the search field and click the search icon on the left of the field.



3) All object classes matching your search are listed in the middle field of the selector. Expand the relevant object class node, right-click the relevant property and select **Select** from the context menu.



4) Click OK. The object class and the selected object class property is added to the semantic analysis explorer persistently.

Adding an Object Class Property to the Semantic Analysis

To add information about an object class property missing in the automatic semantic analysis output for an object class already included in the semantic analysis:

1) Right-click the object class node and select **Add Properties**.
2) In the selector window, right-click the relevant property and select **Select** from the context menu.

3) Click **OK**. The object class property is added to the semantic analysis explorer persistently.

Removing Manually Added Entries From the Semantic Analysis

If you added an object class property manually to the semantic analysis, it will not be removed the next time the native SQL query of the database view is updated, even if it is no longer used in the query. Manually added entries in the semantic analysis can only be removed manually:

- To remove a manually added object class property from the semantic analysis, right-click the object class property node and select **Remove Property**.
- To remove a manually added object class including all object class properties thereof from the semantic analysis, right-click the object class node and select **Remove Class**

Deleting a Database View

If you delete a database view configuration object from the Meta-Model tab in Alfabet Expand, the database view is irrevocably deleted from the Alfabet database and the configuration object is removed from the Alfabet database table that stores the database view configuration objects.

Prior to deleting a database view, you should check that neither queries nor other database views read data from the database view.

To delete a database view:

1) Open the **Meta-Model** tab and expand the explorer node **Database Views**.
2) Right-click the database view that you want to delete and select **Delete** from the context menu.

Testing Queries for Compliance with the Current Release

Alfabet queries and native SQL queries are defined for a number of custom configurations, including For example, workflows, custom explorers, custom object profiles and custom reports.

After replacing or merging a configuration with the configuration in an XML file and after upgrading to another Alfabet release, it is recommended that you test whether changes to the Alfabet query language require you to adapt existing Alfabet queries in the configuration or whether changes in the version of the SQL parser require you to adapt existing native SQL queries.

To ease testing of the queries, a test and export functionality is available for queries defined in the following parts of the configuration:

- Color Rules
- Compliance Reports
- Custom Explorers
- Custom Editors
- Editors

- Export
- Indicator Types
- Monitors
- Object Selectors
- Object Views including Object Cockpits
- Reports including Report Views
- Rights Manager
- Search Manager
- Selectors
- Text Templates
- Wizards
- Workflow Templates
- XML Objects

Alfabet queries defined in these configuration objects can both be checked directly in Alfabet Expand or exported for test purposes.

Native SQL queries cannot be checked for compliance with the current parser version directly in Alfabet Expand. Nevertheless, native SQL queries are included into the list of queries to enable use of the export mechansims provided by the test functionality for native SQL queries.

Via the export mechanisms available in the test functionality all queries in the configuration can be exported to a single Microsoft Excel file or written into a database table to ease analysis of existing queries by mechanisms outside of Alfabet.

The following information is available:

- Testing Alfabet Queries In Alfabet Expand
- Saving All Queries For External Testing

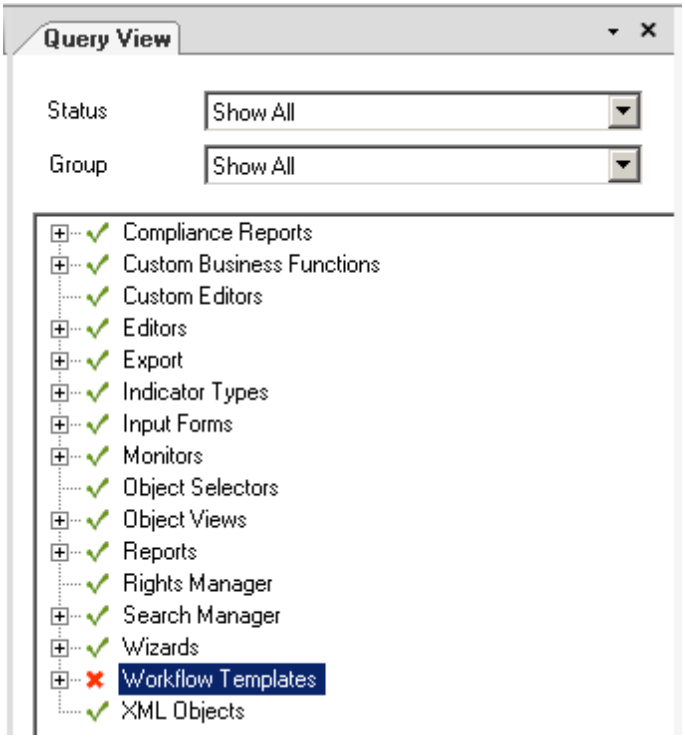
Testing Alfabet Queries In Alfabet Expand

The testing capability exclusively tests whether the Alfabet query matches the current syntax of the Alfabet query language and whether object class and object class properties written into the query are available in the Alfabet meta-model. It is not designed to test the validity of the query in the context of the configuration it is written for. Even if the Alfabet query is correct, a configuration may not work because the results delivered by the query do not match the requirements or the surrounding configuration made in an XML object or in attributes of the configuration object may not be correct.

To test the Alfabet queries in your configuration:

1) In the Alfabet Expand menu bar, select **Meta-Model** > **Check All Queries**. The **Query View** opens in the editor window of Alfabet Expand.

2) Click the **Refresh** button. The editor window displays all types of configurations for which queries are defined. If all Alfabet queries match the current Alfabet query language and the current meta-model, the section will be checked green, otherwise, a red cross will be displayed.



3) Optionally, set the following filters:

- **Status**: Select **Show All** to view all queries including native SQL queries, select **Show Errors** to view only Alfabet queries that need correction or select **Show Passed** to view only Alfabet queries that are correct.
- **Group**: Select the part of the configuration you are interested in or select **Show All** to see all queries in all parts of the configuration.
- **Include Retired Reports**: By default, retired reports are regarded as irrelevant and are not included into the view. Select the checkbox to also check and display queries in retired reports.

4) When a red cross is displayed in the report, click the + symbol in front of the red cross to expand the relevant section. If required, further expand the tree until the level of a single Alfabet query is reached at the leaf level.

5) Click the Alfabet query that needs correction. In the property section of Alfabet Expand, a report about the Alfabet query is displayed that provides the following information:

- **Caption**: The caption of the configuration object for which the Alfabet query is defined.
- **Error Code**: The type of error that occurred. The error code for wrong Alfabet queries is `ErrorInQuery`. for correct Alfabet queries it is `None` and for native SQL queries it is `NotApplicable`, because native SQL queries are not tested.
- **Message**: The kind of error that occurred in the Alfabet query. For example, if an object class that is no longer part of the meta-model is referenced in the Alfabet query, the message "Can't find class: <class name>" is displayed.

- **Owner Type**: The owner type is only specified for a subset of Alfabet queries and gives information about the use of the Alfabet query. In workflows, For example, an Alfabet query defined for to find responsible users for a workflow step has the **Owner Type** `WorkflowObjectQuery`.
- **Path**: The path to the Alfabet query in the configuration.
- **Query**: Allows to open the Alfabet query in a text editor that is read only.

6) To correct the Alfabet query, double-click the Alfabet query in the report tree. In the explorer window of Alfabet Expand, the explorer containing the relevant configuration opens with the location of the Alfabet query expanded. You can now access the Alfabet query from the explorer and make the required corrections.

   If the query is defined via the user interface, which is the case for queries in color rules, For example, the Alfabet interface opens in a new window when you double-click the Alfabet query in the report tree. The object view of the object for which the Alfabet query was defined is displayed. .

   Note the following regarding access to queries via the **Check All AQL Queries** capability:

   - The **Alfabet Query Builder** cannot build an incorrect Alfabet query and will not open when the Alfabet query definition is incorrect. To correct an Alfabet query that contains an error, use the text editor to alter the existing Alfabet query or delete the complete Alfabet query from the text editor and rewrite the Alfabet query in the **Alfabet Query Builder**.

7) In the **Query View**, click the **Refresh** button. The corrected Alfabet query is now displayed with a green checkmark.

Saving All Queries For External Testing

Alfabet Expand provides export mechanisms to save a list of all Alfabet queries and native SQL queries found in Alfabet configuration objects.

The export includes the following:

- The queries are stored in a database table in the Alfabet database for access via a configured report or external test tools.
- The queries are stored in a Microsoft® Excel® file for external analysis.

To store a list of configured reports for analysis:

1) In the Alfabet Expand menu bar, select **Meta-Model** > **Check All Queries**. The **Query View** opens in the editor window of Alfabet Expand.
2) Click the **Refresh** button. The editor window displays all types of configurations for which queries are defined.
3) Optionally, set the following filters. All other filters will have no effect on export:
   - **Include Alfabet Queries**: Select this option to include Alfabet queries into the export.
   - **Include Native Sql Queries**: Select this option to include native SQL queries into the export.

- **Include Retired Reports**: By default, retired reports are regarded as irrelevant and are not included. Select this option if you want queries from retired reports to be included.

4) Do one of the following:

- Click the **Save into Database**  button to save the queries to the `ALFA_QUERY_CHECK` database table only.

- Click the **Export As Excel**  button to save the queries to a Microsoft® Excel® file and to the `ALFA_QUERY_CHECK` database table.

The following information is saved: The title displayed below corresponds to the column name in the `ALFA_QUERY_CHECK` database table and the column headers in the Microsoft® Excel® sheet:

- `ID`: The `ID` is a consecutive number assigned to each query found in the configuration. Saving of the queries overwrites the information in the `ALFA_QUERY_CHECK` database table and re-generates the information written to the Microsoft® Excel® file. The `ID` of a query can Therefore, be different after each re-use of the export functionality.

- `OBJECT_TYPE`: The type of configuration object that the query is found in. The type refers to the parent containing the query. For example, the `OBJECT_TYPE` for queries defined within a workflow template, like For example, a query to find responsible users or a query defined for a pre-condition, the `OBJECT_TYPE` is **Workflow Templates**, independent from the subordinate configuration object containing the query.

- `OBJECT_NAME`: The name of the configuration object. For configuration objects defined in Alfabet Expand, this is the value of the **Name** attribute. For the parts of the configuration that are defined as configuration data on the Alfabet user interface, like For example, indicator types or color rule groups, this is the value of the **Name** property of the respective object class.

- `QUERY_PATH`: If the query location beneath the parent object can be further specified, the subordinate configuration object is returned. For example, for wizards the wizard step containing the query is returned as path information.

- `QUERY_TYPE`: The query type is Native for native SQL queries and AQL for Alfabet queries.

- `QUERY_CAPTION`: If a caption is defined for the part of the configuration containing the query, it is returned in this column. For example, for custom editors, the caption of the editor control containing the query is returned.

- `QUERY_NAME`: The name of the subordinate configuration object or attribute containing the query. For example, for custom editors, the name of the editor control containing the query is returned.

- `ORIGINAL_QUERY`: The query in the configuration object. Please note that only 32767 characters can be written into a Cell in Microsoft® Excel®. If the query exceeds the allowed number of characters in Excel, it is truncated.

Chapter 20: Configuring Surveys for Data Capture Campaigns

The Surveys capability allows for stakeholder surveys or data acquisition campaigns to be configured. Such surveys are typically driven by regulatory or senior management requests. Alfabet Expand provides a capability that allows surveys to be configured so that the data collection process is standardized by a high degree of automation and can be conducted in a timely manner.

Surveys integrates several standard capabilities available in Alfabet Expand including the configuration of wizards for the structured collection of data as well as the configuration of workflow templates that allow the Alfabet objects targeted by the survey to be identified and a workflow initiated for each surveyed object. The configured workflow ensures that the data collection tasks required for the survey are distributed to responsible users, that if/then scenarios can be implemented for the data input, and that all data collection tasks are completed by their specified deadlines.

To configure a survey, you must create a custom object class that serves as the container in which to collect data about objects in the Alfabet database. If one survey object is to be generated for the standard Alfabet object (in other worlds, one survey object per authorized user), then a workflow template should be configured for the survey. In the context of a workflow template, the custom survey class is associated with a standard Alfabet object class that the data inquiry targets. An object based on the custom survey class will be generated for each Alfabet object determined via a query to be relevant for the survey.

Alternatively, if multiple survey objects are to be created for a standard Alfabet object (in other words, if multiple survey objects are needed because multiple users will be providing input), then an ADIF scheme should be configured to create the survey objects. A workflow template should be configured for the data inquiry. For more information about the configuration of an ADIF scheme, see the reference manual *Alfabet Data Integration Framework*.

Users responsible for a workflow step may access the Alfabet object in order to gather the necessary information required for the data input. Because the data collection occurs on the object created for the custom survey class, users with access permissions to only the base objects will not have access to the survey data unless they are explicitly granted access permissions via the workflow template or relevant user profile.

The enterprise decides it is necessary to conduct a survey to assess all applications that are not assigned to a domain assignment to determine their readiness to migrate to a Windows® 7 client.

To this end, a survey is created with the custom survey class named Windows7Readiness. The custom survey class has 3 custom object class properties defined to evaluate compatibility for a 64-bit system, 32-bit system, and XP-emulation system. Enumerations are created for each custom object class property that allows the user evaluating the application to select a pre-configured value. A fourth object class property allows the evaluating user to provide comments regarding, For example, installation, execution, or run-time behavior. The user will be provided with a custom editor defined for the custom survey class Windows7Readiness in which the values can be defined.

A workflow template is also configured whereby the custom survey class Windows7Readiness is defined in the **Start Base Class** attribute and the class Application is defined in the **Source Base Class** attribute. A query is defined that finds all applications that have no domain assignment. A workflow is thus generated for each application found by the query and a new survey object based on the custom object class Windows7Readiness will be created for every application. A user evaluating an application in the context of the survey can provide the relevant data input about the application in an editor or wizard created for the custom survey class Windows7Readiness.

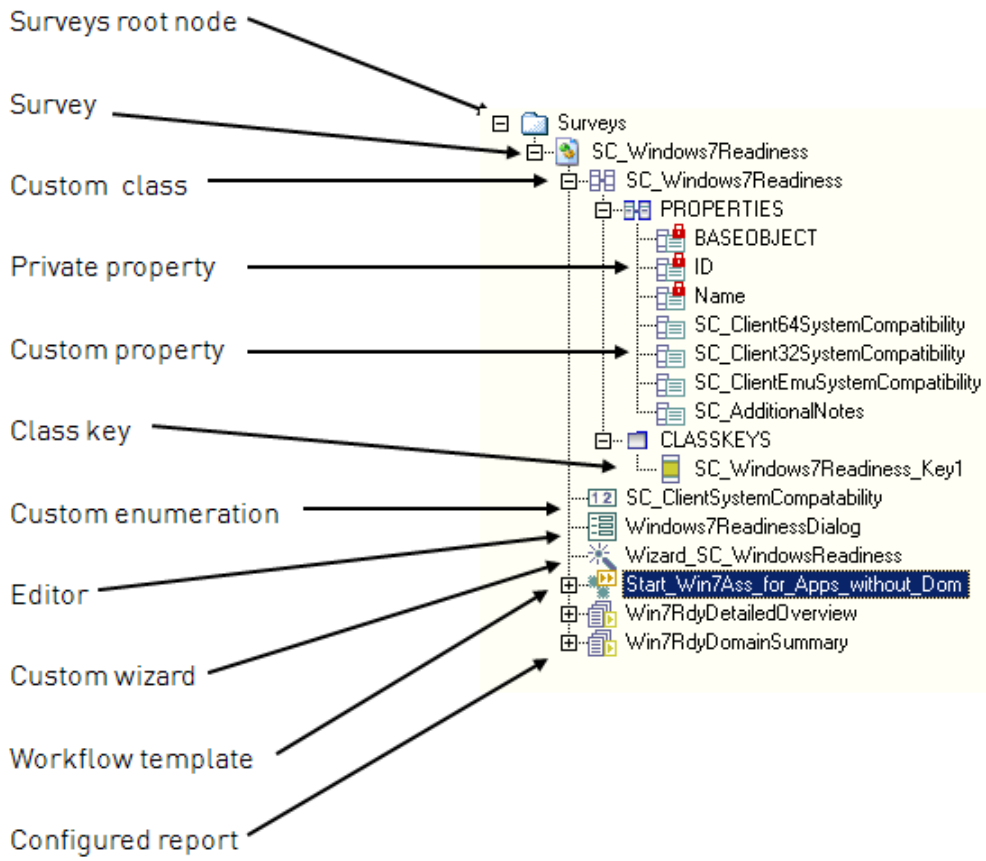
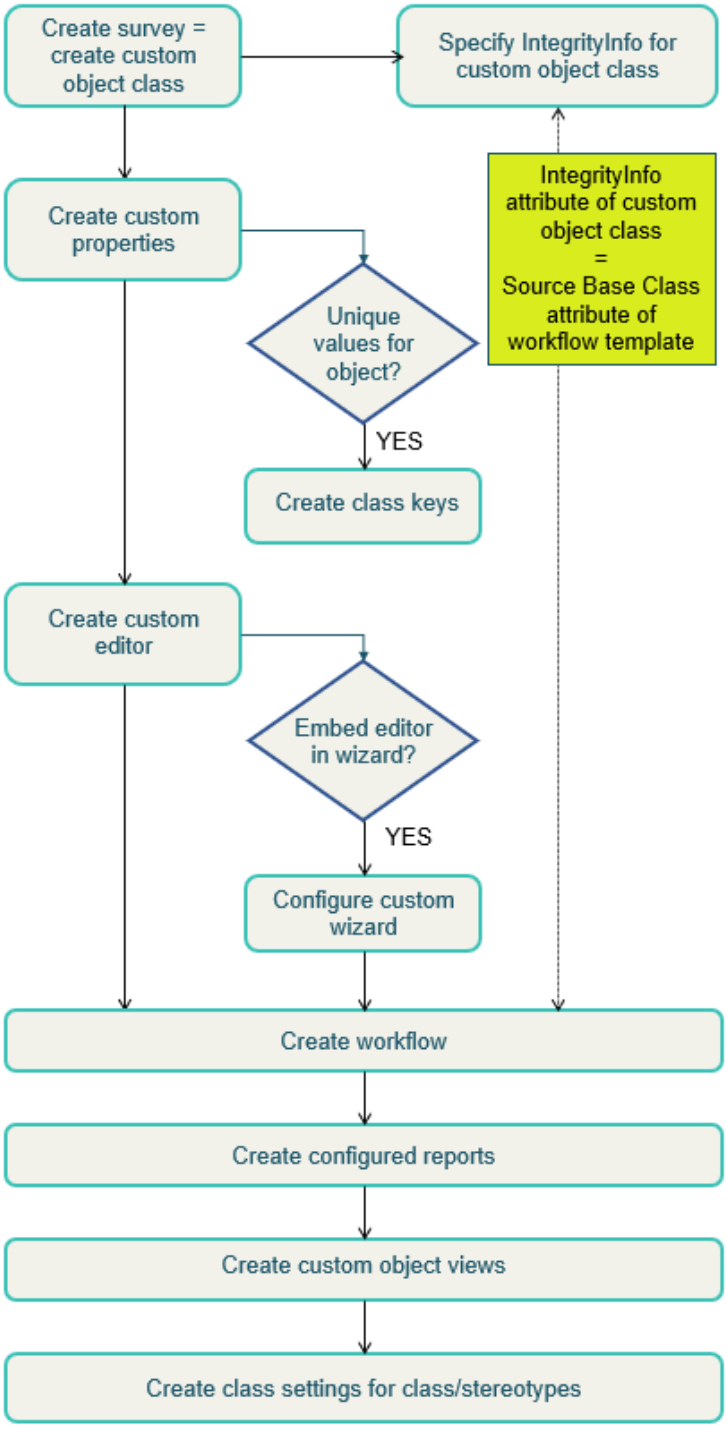


FIGURE: The Surveys tab in Alfabet Expand

The following steps must be carried out in order to configure and implement a survey:



- The information model must be configured. In this phase, you must determine which data is to be captured in the survey. This entails the following configuration:
  - The creation of the survey and specification of the custom survey class for which the data will be captured. A default object profile will be available in the Alfabet interface for each survey class with the following buttons potentially available:
    - **Workflow**: This button will only be available if workflows exist for the survey objects based on the custom object class.

        - **Edit**: This button will only be available if a custom editor or custom wizard is assigned to the class setting relevant for the custom object class.
        - **Audit Trail**: This button will only be available if auditing is configured as permissible for the custom object class.
    - The creation of all necessary custom object class properties and enumerations required to capture the information about the objects surveyed.
- The means to capture the data required for the information model. This entails the following configuration:
    - The creation and specification of one or more custom editors to capture the data.
    - If necessary, the configuration of a wizard in order to structure the data capture process. Pre-conditions can be stipulated for each wizard step and the completion of data input can be checked before the user can move on to the next step in the wizard.
- A workflow to implement, standardize and manage the collection of data for the survey. The configuration of a workflow template allows you to determine the Alfabet objects that should be assessed in the survey, the various tasks involved in the data collection process, and the users who are responsible for providing input at each stage of the survey. This entails the following configuration:
    - The creation and specification of one or more workflow templates made up of workflow steps.
    - The workflow template associates the custom survey class created for the survey with the object class in Alfabet that is targeted by the survey. The workflow template must include a query that finds the Alfabet objects that are to be surveyed. A new survey object based on the custom survey class will be automatically created for each Alfabet object found by the query.
    - Pre-conditions can be stipulated for each workflow step and the completion of data input can be checked before the user can move on to the next step in the workflow.
- If necessary, the configuration of reports to display, For example, the results of the survey or the progress of the data capture campaign. A custom object view can be created if the data collected for individual survey objects should be displayed on an object-by-object basis.

The custom object classes configured for surveys will be available in the **Class Configuration** functionality in the Alfabet user interface. Indicators and roles can be configured for the custom object classes that are targeted by a survey. For more information, see the sections *Configuring Evaluations, Prioritization Schemes, and Portfolios* and *Configuring Role Types to Define Roles in the Responsibilities Page View* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.

The definition of the information model as well as the execution of the workflows to capture the data occurs at server run-time.

The following information is available:

- Creating the Survey and Custom Survey Class
- Creating Custom Object Class Properties and Enums for the Custom Survey Class
- Creating Uniqueness Constraints for the Custom Survey Class

- Configuring Editors and Wizards for the Survey
- Configuring a Workflow for the Survey
- Configuring Reports for the Survey
- Configuring Object Views for the Custom Survey Class
- Making the Survey Class Searchable
- Configuring Class Settings for the Custom Survey Class

Creating the Survey and Custom Survey Class

When you create a survey, you automatically generate a new custom survey class. Only one custom survey class can be created per survey. When the survey is executed, survey objects will be created based on the custom survey class via a configured workflow template.

Any custom object class you create for a survey will also be displayed below the **Classes** node in the **Meta-Model** tab in Alfabet Expand.

To create a survey:

1) Go to the **Surveys** tab, right-click the root node **Surveys**  and select **New Survey**. The **Create New Class** editor will open.

   If you want to copy an existing survey as the basis of a new survey, right-click the survey node  and select **New Survey as Copy**. Proceed as described below.

2) In the **Class Name** field, enter the name of the custom survey class. This value is displayed in the solution interface if the **Caption** attribute has not been defined for the custom survey class.
3) Click in the **Technical Name** field. The class name will appear in capital letters. The technical name can be edited, if necessary. Click **OK** to save the custom survey class to the database.

   Please keep the following in mind:

   - The technical name may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the section Defining Attributes for Configuration Objects in the chapter Getting Started with Alfabet Expand.
   - The technical name should only contain standard `ASCII` characters.
   - The technical name should consist of upper-case letters only.
   - The maximum length of a technical name may not exceed 30 characters.
   - The technical name may not coincide with any of the reserved key words of the relational database management system.

   A validation mechanism checks for correct syntax when defining a technical name. Please note that if the technical name of a configuration object is changed, the name will be correctly updated in other configuration objects referencing the changed object

during design time. However, the name of a changed configuration object will not be updated in guide pages nor if the guide pages reference a configuration object listed in the **Show Usage** functionality. If you plan to change the name of a configuration object, the reference in the guide page should be changed prior to changing the configuration object name.

4) The survey is displayed in the **Surveys** explorer. Expand the custom survey class node and then expand the **PROPERTIES** node . You will see three private object class properties labelled `BASEOBJECT`, `ID`, and `NAME`. These standard object class properties have been automatically generated for the custom survey class and can be implemented later in queries defined for the workflow template. These object class properties cannot be deleted. Typically, no further definition is required for these object class properties.

5) Click the custom survey class node to activate the attribute window. With the exception of the **Integrity Info** attribute, the available attributes can be optionally defined, as described for a conventional object class.

6) To define the **Integrity Info** attribute, click the **Browse** button to open the XML editor. In this editor, you specify the dependency between the custom object class and the object class that will be defined in the workflow template as the **Source Base Class**. The definition of the **Integrity Info** attribute is important so that if a source object is deleted from the Alfabet database, the associated survey object will also be deleted.

For example, if a survey object is created for the custom survey class Windows7Readiness and the associated application targeted by the survey is deleted, then the associated Windows7Readiness object should also be automatically deleted.

7) Define the following XML attributes:

Definition of the **Integrity Info** attribute for a custom object class:

```
<ClassIntegrityInfo>
   <IntegrityInfo MasterClassName="Application"
   ClassName="Windows7Readiness" PropertyNames="BASEOBJECT"
   />
</ClassIntegrityInfo>
```

- In the XML attribute `IntegrityInfo MasterClassName`, enter the Alfabet object class that is the target of the survey. This is the object class that you will later define as the source base class for the workflow template.
- In the XML attribute `ClassName`, enter the name of the custom survey object class.
- In the XML attribute `PropertyNames`, ensure that the value `BASEOBJECT` is entered.

8) Click **OK** to save the XML definition and to close the editor.

9) In the toolbar, click the **Save** button to save your changes. The survey with a custom survey class has been created. You should now create the custom object class properties and enumerations required to capture the data collected in the survey.

Creating Custom Object Class Properties and Enums for the Custom Survey Class

The procedure to create custom object class properties and enumerations for a custom survey class is the same as the procedure to create custom object class properties and enumerations for a standard object class. The attributes available in the attribute window will depend on the property type that you select for the custom property.

It is highly recommended that you create all custom object class properties required for the survey in the context of the survey configuration in the **Surveys** tab. To create a custom object class property for the survey, right-click the custom survey class node  and selecting **Add New Property**. Define the **Property Name** and **Technical Name** fields as described for the survey class. The custom object class property  will be displayed below the **PROPERTIES** node  . The custom object class property should be defined according to the standard conventions described in this reference manual.

Any custom object class property you create for a survey will also be displayed below the **PROPERTIES** node of the custom object class in the **Meta-Model** tab in Alfabet Expand.

The following information is available to help you:

- For an overview of all property types, see the section Overview of Data Types Available for Custom Properties in the chapter Configuring the Class Model.
- For a general description about how to create a custom object class property, see the section Configuring Custom Properties for Protected or Public Object Classes in the chapter Configuring the Class Model.

If enumerations are required for one or more custom object class properties, then it is highly recommended that you create the enumerations in the context of the survey configuration in the **Surveys** tab. To create an enumeration for the survey, right-click the survey node  and select **New Enum** . The public enumeration  will be displayed below the survey node  . The enumeration should be defined according to the standard conventions described in this reference manual and then assigned to the relevant custom object class property in the **Enum** field of that object class property.

Any public enumeration you create for a survey will also be displayed below the **Enums** node in the **Meta-Model** tab in Alfabet Expand. For information about how to create an enumeration, see the section Defining Protected and Custom Enumerations in the chapter Configuring the Class Model.

Creating Uniqueness Constraints for the Custom Survey Class

If a class key is required in order to require users to enter a unique combination of values for a specified set of custom object class properties when defining data for the custom survey class, then you must create the class key in the context of the survey. To create a class key for the survey, right-click the custom survey class node  and select **Add New Class Key** . The class key should be defined according to the standard conventions described in this reference manual. The class key  will be displayed below the **CLASSKEYS** node below the custom survey class  .

Any class key you create for a survey will also be displayed below the **CLASSKEYS** node of the custom object class in the **Meta-Model** tab in Alfabet Expand. For information about how to create a class key, see the section Configuring Class Keys for Object Classes in the chapter Configuring the Class Model.

Configuring Editors and Wizards for the Survey

The configuration of editors and wizards provides the means for users to capture the data required for the survey. It is highly recommended that you create all editors and custom wizards required for the survey in the context of the survey configuration in the **Surveys** tab. You can define multiple editors and wizards for the custom object class.

As with the general configuration of custom editors, the editors you create for a survey can be assigned to wizards and workflow steps. Please note that in the context of surveys, you will be creating an editor and not a custom editor (which is always appended to a standard editor). Any editor you create for a survey will also be displayed below the **Editors** node in the **Presentation** tab in Alfabet Expand.

To create an editor for the survey, right click the survey node  and select **New Editor** . The editor  is displayed below the survey node  . The editor should be defined according to the standard conventions for custom editors described in this reference manual. For a general description about how to create an editor, see the chapter Configuring Custom Editors.

To implement an editor in the survey, you must assign the editor to either a custom wizard or the relevant workflow step. To assign the editor to a workflow step, select `Editor` in the **Type** attribute of the relevant step and select the editor in the **View** attribute. Please note that you cannot select the editor in the **Custom Editor** attribute of the workflow step.

To create a custom wizard for the survey, right click the survey node  and select **New Wizard** . The custom wizard  is displayed below the survey node  . The custom wizards should also be defined according to the standard conventions for custom wizards described in this reference manual. Pre-conditions can be stipulated for each wizard step and the completion of data input can be checked before the user can move on to the next step in the wizard. For a general description about how to create a custom wizard, see the chapter Configuring Wizards.

To implement the custom wizard in the survey, you must assign the custom wizard to the relevant workflow step. To do so, select `Wizard` in the **Type** attribute of the relevant step and select the custom wizard in the **View** attribute.

Configuring a Workflow for the Survey

A workflow template is required to implement the survey capability. The configuration of a workflow template allows you to determine the Alfabet objects that should be assessed in the survey, the various tasks involved in the data collection process, and the users who are responsible for providing input at each stage of the survey.

It is highly recommended that you create all workflow templates required for the survey in the context of the survey configuration in the **Surveys** tab. You can define one or more workflow templates per survey. Any workflow template you create for a survey will also be displayed below the **Workflow Templates** node of the custom object class in the **Workflows** tab in Alfabet Expand.

Please note the following information relevant to the configuration of a workflow template in the context of a survey:

- To create a workflow template for the survey, right-click the survey node and select **New Workflow** . The workflow template is displayed below the survey node . The **Workflow State** attribute for the workflow template will automatically be set to `Plan`.
- Click the new workflow template and in the attribute window, set the **Automatic Start** attribute to `True`.
- Define the base class for the workflow template. This is the survey class that the workflow begins with when the initial workflow step is performed. Survey objects will be initially created for this object class. In the **Start Base Class** attribute, select the base class for the workflow template.
- Next, define the source base class for the workflow template. This is the object class containing objects in the Alfabet inventory that the survey targets. A workflow will be started for each object found by the query in the source base class. The data for the source base objects will be copied to the survey objects (based on the start base class) when the workflows are created. In the **Source Base Class** field, select the source base class for the workflow template.
- Define an Alfabet query in the **Source Objects via Query** attribute or native SQL query in the **Source Objects via Query as Text** attribute to find the Alfabet objects that should be surveyed. A workflow will be started for each object found by the query.

  For example, the survey is targeted at the creation of workflows for all applications that are not yet allocated domains in the enterprise. The **Start Base Class** attribute of the workflow template specifies the survey class. The **Source Base Class** attribute is specified as `Application`. The following Alfabet query searches for existing active applications in the inventory that have no domain assignment.

  ```
  ALFABET_QUERY_500
  FIND Application
  WHERE (AND Application.Domain IS NULL
      Application.ResponsibleUser IS NOT NULL
      Application.ObjectState = 'Active')
  ```

  For general information about defining an Alfabet query or SQL query, see the chapter Defining Queries.

- Next, create and define the initial workflow step . For the workflow step, create a workflow step action via the **Action On Entered Step** node. For the workflow step action, set the **Type** attribute to `Script`. In the **Statements** attribute, specify the object class properties of the source object class that should be automatically written to the survey objects.

  For example, to copy the applications Name and ID to the survey object.

  ```
  Set(WORKFLOWBASE.BASEOBJECT,SOURCE.REFSTR);
  Set(WORKFLOWBASE.ID,SOURCE.ID);
  Set(WORKFLOWBASE.Name,SOURCE.Name);
  ```

  For more configuration of a object class property update of the type `Script`, see the section Configuring Automatic Property Updates for a Workflow Step.

- To implement an editor created for the survey, select `Editor` in the **Type** attribute of the relevant workflow step and select the editor in the **View** attribute. Please note that you cannot select the editor in the **Custom Editor** attribute of the workflow step.
- To implement a custom wizard created for the survey, select `Wizard` in the **Type** attribute of the relevant step and select the custom wizard in the **View** attribute.
- All other aspects of the workflow template configuration should be defined according to the standard conventions described in this reference manual for workflow templates. For example, pre-conditions can be stipulated for each workflow step and the completion of data input can be checked before the user can move on to the next step in the workflow. Likewise, other workflow templates configured for the survey can be triggered based on the fulfillment of a workflow step's pre- and post-conditions. For information about the configuration of a workflow template, see the chapter Configuring Workflows.
- The **Workflow State** of the workflow must be set to `Retired` in order to delete the workflow from the survey.

Configuring Reports for the Survey

Configured reports of any type can be configured for the survey. You can configure, For example, to display the results of the survey or the progress of the data capture campaign. Any reports configured for the survey can be accessed in the **Reports** functionality in Alfabet.

It is highly recommended that you create all configured reports required for the survey in the context of the survey configuration in the **Surveys** tab. You can define one or more configured reports per survey.

To create a configured report for the survey, right-click the survey node  and select **New Report** . The configured reports  is displayed below the survey node  . The **State** attribute for the configured reports will automatically be set to `Plan`. Any configured report you create for a survey will also be displayed below the **Reports** node of the custom object class in the **Reports** tab in Alfabet Expand. Configured reports should be defined according to the standard conventions described in this reference manual. For information about the specification of a configured reports, see the chapter Configuring Reports.

Configuring Object Views for the Custom Survey Class

A custom object view can be created if the data collected for individual survey objects should be displayed on an object-by-object basis. The object view can be implemented to display an Attributes section and configured reports. Views cannot be added to the object view since no views for the custom object class exist in the configuration.

To ensure that the users can navigate to the object view of the source objects found by the query defined for the workflow template, you should configure a link in the Attributes section to the base class (**Start Base Class** attribute).

It is highly recommended that you create custom object views required for the survey in the context of the survey configuration in the **Surveys** tab. You can define multiple custom object views for the custom object class. Each custom object view should have a unique name. Only one custom object view can be assigned per object class to a view scheme. The configuration of a custom object view is not mandatory.

To create a custom object view for the survey, right-click the survey node  and select **New Object View** . The custom object view  is displayed below the survey node  .

Custom object views should be defined according to the standard conventions described in this reference manual. For information about the configuration of a custom object view, see the chapter Configuring Object Views.

Making the Survey Class Searchable

Survey classes may be configured to be searchable in the **Simple Search** functionalities. A custom selector, custom object view, and custom editor must be configured for the survey class.

The following should be specified in the class setting of the survey class:

- Survey classes may be configured to be searchable in the **Simple Search** functionalities. A custom selector, custom object view, and custom editor must be configured for the survey class. The following should be specified in the class setting of the survey class:
  - **Selector Definition** attribute must specify the custom selector defined for the survey class
  - **Object View** attribute must specify the custom object view defined for the survey class
  - **Edit View** attribute must specify the custom editor defined for the survey class
  - **Searchable** attribute must be set to `True`
  - **Preview Properties** attribute and **Icon** attribute should be defined

Configuring Class Settings for the Custom Survey Class

The configuration of class settings allows you to specify information about the standard or custom object view to implement for the source object class. This will allow users executing the survey to access the relevant information about the object that is the target of the survey in the **Workflow Activities** views.

For example, if users are evaluating applications in the context of a survey object created for the custom object class Windows7Readiness, then they may need to navigate to the application for which they are providing information. If no class settings are defined, then users cannot navigate to the application in the workflow. In this case, the **Navigate to Artifact** option in the **Workflow Activities** views will provide navigation to the object view for the current workflow step.

It is highly recommended that you create class settings required for the survey in the context of the survey configuration in the **Surveys** tab. You can define multiple class settings for the custom object class. Each class setting should have a unique name. Only one class setting can be assigned per object class to a view scheme. The configuration of a class setting is not mandatory.

To create a class setting for the survey, right-click the custom survey class node and select **Add New Class Settings**. The class setting is displayed below the custom object class node.

Please note that when a custom object class is created, the custom object class will automatically be displayed below the **Class Settings** node in the **Presentation** tab only after Alfabet Expand has been closed and reopened.

Any class setting you create for a survey will then be displayed below the relevant custom object class under the **Class Settings** node in the **Presentation** tab in Alfabet Expand.

The class setting should be defined according to the standard conventions described in this reference manual. The following information is available to help you: For a general description about how to create a custom object class property, see the section Configuring Class Settings for Object Classes and Object Class Stereotypes in the chapter Configuring User Profiles for the User Community.

Appendix 1: Glossary

This glossary defines terms that are often used in the Alfabet Expand Online Help or when working with Alfabet Expand.

activity monitor 

An activity monitor is a type of monitor that alerts subscribed users about changes that have been made to objects in a specified object class. The monitor owner must specify a set of properties that are to be monitored and a set of users that are to be alerted if the monitor is triggered. These users as well as the monitor owner will be informed via email notification if a specified attribute for any object in the class is changed. Monitoring is typically performed in regular intervals over a specific period of time.

Alfabet query

An Alfabet query is a means to efficiently search and retrieve information in Alfabet by means of object attributes and relationships. Alfabet queries are based on a class-oriented query language and may be configured in a variety of contexts in Alfabet, including rule-based permissions, pre- and post-conditions for workflow steps, full-text searches, configured reports, computation rules, notification monitors, and consistency monitors, for example. Typically, an Alfabet query is defined for a base class. However, it can also be defined to traverse object classes.

As an alternative to the Alfabet query language, native SQL queries may be specified for many of the configurable functionalities in Alfabet.

analytics dashboard

An analytic dashboard is an ad-hoc information-rich data visualization designed by a user based on the embedded third-party tool DevExpress® Dashboard. One or more dashboard items such as charts, scatter charts, grids, cards, gauges, pivots, range maps, treemaps, etc. can be added to the analytics dashboard and filtering options such as combo-boxes, list boxes, and tree views can be leveraged. A configured analytics dashboard data provider is specified for the analytics dashboard in order to fetch the data to display in the analytics dashboard. Analytics dashboards may be shared with other users.

architecture element

An architecture element is an object in the inventory defined to capture the as-is architecture.

archive object

An archive object is a snapshot of a deleted Alfabet object. When an Alfabet object is archived, a ZIP file is created containing HTML files that display the object profile for the archived object as well as the object

profiles of its dependent objects. Each archived object profile displays a preconfigured set of page views, whereby the visibility of these views will depend on the class setting configured for the object class. If a page view displays dependent objects, a user can click the dependent object in the HTML view in order to open another HTML file showing the archived object profile of the selected dependent object.

Alfabet objects are typically archived by a solution administrator. The ZIP file may be downloaded to a local disk and extracted. The relevant HTML file can then be viewed in a browser window. An archive object will be created for each culture setting supported by your enterprise.

aspect evaluation

An aspect evaluation is a qualitative assessment of the performance of applications or components assigned to one or more respective application groups or component groups according to defined evaluation criteria. Because an application, for example, may be a member of several different application groups, different qualitative assessments of the same application using the same evaluation criteria are possible for each application group that the application is assigned to.

For example, the component TradeNet may be relevant for different purposes in the enterprise architecture. The component may be suitable to varying degrees both as a business intelligence solution and as an OLAP database. An aspect evaluation would allow you to examine components in a component group based on specific aspect evaluation types, such as business importance or usage. Like conventional evaluation types, aspect evaluation type comprises one or more indicator types that are used to evaluate objects.

An aspect evaluation is based on one or more evaluation types, which are then assigned as aspect evaluation types to the object classes Application or Component. Like conventional evaluation types, aspect evaluation types comprise one or more indicator types that are used to evaluate objects.

Aspect evaluations are currently only available for the object classes Application or Component.

assignment

An assignment is a task that is defined for a selected object and assigned to a specific user. The assignee is expected to provide the required input for the object by a specified due date. Assignments can be defined to be optional or mandatory. Email notifications may be configured to be sent in the context of the assignment capability.

authorized user

An authorized user is the Alfabet user that has primary responsibility for an object. The authorized user has Read/Write access permissions to the objects that he/she is defined to be the authorized user of. The authorized user is often referred to as the owner of the object and the objects that he/she owns are commonly referred to as his/her personal objects.

The authorized user may specify another user to be a deputy for any object that he/she owns. Any user that is defined as a member of an authorized user group will automatically have authorization to the objects assigned to the authorized user group.

In addition to authorized users, the concept of roles allows persons and organizations with a functional relationship to an object to be defined.

authorized user group

An authorized user group is a user group that has been assigned to an object. Like the object's authorized user, all users in the authorized user group have Read/Write access permissions to the object. A user group may have an unlimited number of subordinate user groups.

A solution designer may configure the Inheritance and/or propagation of access permissions for authorized user groups. If the inheritance of access permissions has been specified, then all user groups in the user group hierarchy that are descendant to a user group should have the same access permissions to that object. If the propagation of access permissions has been specified, then all user groups in the user group hierarchy that are ascendant to a selected user group should have the same access permissions to that object.

base class

The base class is the object class for which a user is seeking results for. For example, a base class is relevant for an alfabet query, native SQL query, or computation rule.

blueprint

A blueprint is a master plan or IT strategy that serves as a guideline for planning business support across organizations in the enterprise. In the blueprint planning process, master planners and strategic planners can refer to a blueprint to plan tactical business supports or strategic business supports in business support maps. The blueprint planning process supports the standardization of business support and efficiency in the roll-out of IT support in the enterprise.

business appraisal

Business appraisal is a means for an enterprise to gain an initial understanding of the alignment between the business and the supporting IT. Business appraisals are defined in the context of a business support map. The definition of business supports is not required to conduct a business appraisal.

The business appraisal is an evaluation of the as-is and to-be support for the business process or domain for a selected organization or market product in regard to a specific operational aspect. Once values have been defined for the indicator types configured for the business appraisal, the gaps between the to-be and as-is support can be assessed allowing the enterprise to determine priorities for future projects or its IT strategy in order to align the business and IT.

business capability map 

A business capability map allows you to define the guidelines to evaluate the business capabilities defined for the enterprise. The performance of domains and business functions associated with a business capability can be evaluated according to specific business capability aspects and evaluation types by a defined group of evaluators. Mandates are not supported in the context of business capability maps.

business case

The business case is defined for a project and provides a structured approach to estimating the investment and operating costs and benefits associated with a project, and computes standard performance measures such as return on investment or depreciated cash flow.

business function 

See functionality.

business service 

A business service is an IT service that can be provided by an application, component, local component, organization, market product, business process, ICT object, or solution building block in order to realize a specific business function. Depending on the solution configuration, applications and components may provide one or more business services for the same business function. A business service thus fulfills a specific business service request that is posed by a business process for that specific business function.

business support

A business support defines an object such as an application that provides support to the enterprise to carry out its business. Business supports typically provide support to business processes, although an enterprise may describe its business support to domains instead. Furthermore, business supports are typically executed by organizations responsible for the business processes/domains. Some enterprises may specify market products instead of organizations. Alfabet supports the definition and analysis of operational business supports, solution business supports, strategic business supports, and tactical business supports.

business support map

A business support map is the graphic visualization of the business support provided by specified objects in order to support the planning of the short-term, medium term, and long-term to-be architecture. The map is visualized as a matrix with an X-axis and a Y-axis. The X-axis of a standard business support map will display business processes and the Y-axis will display organizations that are supported. In some industry segments, it is more relevant to analyze the business support for market products than for organizations. If

this is the case, market products may be configured for the Y-dimension of the business support. Furthermore, some enterprises may describe business supports to provide support to domains of the business rather than business processes. If this is the case, domains may be configured for the X-dimension of the business support.

The matrix cells consist of the business supports that support the corresponding X-axis object and Y-axis object. In the context of master plans or IT strategies, business support matrices may display operational business supports, tactical business supports, strategic business supports. In the case of a solution map, solution business supports will be displayed.

A business support map may be specified for an IT strategy or master plan that is specified as a blueprint. The definition of the business support map for a blueprint strategy or master plan may be embedded in another business support map to serve as a guideline for planning and standardizing business support across organizations when planning the enterprise's business architecture.

Alfabet supports the configuration of customized business support matrices to allow large numbers of business supports to be simultaneously captured for a specified set of organizations/market products or business processes/domains.

class

See object class.

class key

A class key is the specification of one or a combination of standard and custom object class property for an object class. Typically, class keys are defined to implement uniqueness constraints. A standard or configured class key provides an enforcement mechanism for the definition of standard properties and custom object class properties when an object is created. In this case, the combination of properties must be unique for all objects in the object class. If the uniqueness constraint is violated by a user when defining data, a message will be displayed prompting the user to provide a unique combination of values. For example, class keys for the object class Application could require a unique combination of values for the object class properties Name, Version, and Object State.

Index creation is a side effect of a class key definition. For each class key that is configured for the object classes in Alfabet Expand, an index is created. The definition of class keys can therefore be used to speed up query-based searches and thus increase performance.

Please note the following:

- Multiple class keys can be defined as needed for an object class. All custom class keys are public class keys.
- Private class keys  cannot be edited or deleted. Private class keys are typically preconfigured for indexing purposes.
- Protected class keys  may be edited but not deleted.
- Public class keys  may be edited and deleted.

class setting

A class setting is a specification about an object class or object class stereotype. A class setting is associated with a view scheme which is assigned to a user profile. The class setting therefore specifies what the user can see and do with an object class in the context of the user profile that he/she uses to access Alfabet. The class setting includes such information as which standard or custom object view is displayed, whether users will enter data in a standard or custom editor vs. a standard or custom wizard, whether a standard or custom selector will be used to find objects in the object class, and which standard and custom object class properties will be hidden from view.

Multiple settings can be defined for an object class. However, only one class setting per object class can be assigned to a view scheme so that only one class setting will be available for an object class in a given user profile.

color rule

A color rule is based on one or more Alfabet queries or native SQL queries that are configured to color a found set of objects in standard Alfabet business support maps and diagrams. If the color rule functionality is activated for a business support map, all activated color rules will be executed and the matrix cells colored accordingly. In order to visualize color rules in diagrams, color rules must be assigned to a diagram view. When the diagram view is selected in a diagram, all activated color rules will be executed and the diagram objects will be colored accordingly. The color rules are automatically added to legends where they are applicable.

Additionally, color rule specifications can be defined for configured reports and color instructions can be configured to color cells based on property values in configured reports and object cockpits.

consistency monitor 

A consistency monitor is a type of monitor that supports the system-wide maintenance of objects in the Alfabet database. The consistency monitor is specified to periodically search for inconsistencies among objects. Each consistency monitor is based on an alfabet query or native SQL query that defines the object classes targeted by the query as well as the inconsistent properties to be detected. If an inconsistency is found by the query, the monitor alerts the authorized user of the object via an assignment about the inconsistency. The timely completion of the assignment triggered by a consistency monitor can be tracked by the solution administrator.

contract

A contract is a legal document that stipulates the terms of agreement between organizations buying products and services and organizations or vendors providing the products and services.

A contract may be referenced as a master contract by multiple contracts.

cost type

A cost type is a classification of costs. Cost types are defined by the enterprise to ensure comparability in the definition and evaluation of the investment and operation costs associated with ICT objects, applications, deployments, projects, and service products. Cost types are critical for the definition of business cases as well as for cost planning for projects. An unlimited number of subordinate cost types may be defined for any cost type in order to allow for bottom-up cost estimation and analysis.

culture

A culture constitutes the base configuration of the default primary language displayed on the user interface when the user first logs in as well as the date, time, and number formats, etc. used to capture and render dates, times and numbers in the Alfabet user interface.

custom editor

A custom editor is a customized data entry view that allows users to capture data for the custom object class properties that have been created for a protected or public (custom) object class. A custom editor is typically configured for the needs of users working with a specific user profile. A custom editor can only be created for an object class that has custom object class properties defined. A custom editor cannot be created for a protected class that has no custom object class properties defined.

A custom editor typically consists of one or more tabbed pages that are automatically appended to the standard editor available for the object class. The solution designer designs the interface controls such as combo boxes, checkboxes, etc. that users need to capture the required data. Additional text fields or URLs may be included in the custom editor in order to provide users with additional information to help them define data. All captions created for the interface controls can be translated for the languages implemented in the Alfabet interface.

Custom editors are displayed in the context of standard data capture editors but can also be implemented in wizards and workflows configured for the enterprise. Multiple custom editors can be created for an object class, but only one custom editor can be assigned to a class setting. Therefore, only one custom editor can be implemented per object class per user profile.

custom property

A custom property is an object class property that has been configured in order to capture enterprise-specific data for a specified object class.

Multiple custom properties can be created for each object class. Like private properties, custom properties (also referred to as public properties) can be excluded from a class setting and therefore be hidden for users accessing Alfabet with the associated user profiles. Custom properties can also be included in the definition of a class key requiring users to enter unique values for a set of standard and custom properties when defining objects in the object class.

In order for users in the user community to define values for the custom properties, the properties must be associated with data entry fields in custom editors. Custom properties can be displayed in object views and

object cockpits. Additionally, custom properties can be used to search for objects in Alfabet 's search functionalities and can be specified in alfabet queries and native SQL queries that are used in, for example, configured reports.

date monitor 

A date monitor is a type of monitor that alerts subscribed users about approaching dates that have been defined for objects in a specified object class (for example, the approach of an object's start or end date). Two kinds of date monitors are available in Alfabet: object-specific date monitors and system-wide date monitors that target all objects in a specified object class.

Object-specific date monitors are defined by an individual user to keep track of the approach of a date for specified objects. The monitor owner must specify a set of properties that are to be monitored and a set of users that are to be alerted if the monitor is triggered. These users as well as the monitor owner are informed when the target date approaches for the specified attribute and can then decide upon the appropriate action to take. Monitoring is performed in regular intervals over a specified period of time.

In addition to the conventional Alfabet date monitors, system-wide defined date monitors can be configured for an object class. When a specified date approaches, all authorized users responsible for objects in the relevant object class will be sent notifications per email asking them to review the objects that they are responsible for.

deputy

A deputy is an Alfabet user granted Read/Write access permissions to an object in order to act on behalf of the authorized user.

diagram view

A diagram view is a configuration that is associated with a diagram. It allows users to superimpose qualitative information - such as aggregated indicators or attribute values - associated with these architectural elements. Diagram views can be implemented in diagrams displaying applications, business processes, devices, domains, frameworks, platforms, and solution building blocks. Multiple diagram views may be defined and made available for a diagram. A diagram view may be reused for multiple diagrams.

discussion group 

A discussion group is a preconfigured group of users that have been defined permission to discuss objects in specified object classes.

domain glossary

A domain glossary allows users to define and search for domain-specific terminology defined for objects that are assigned to a domain. Domain-specific terminology includes aliases and descriptions of the objects.

The solution designer must specify a search group including an alfabet query for each object class that can be associated with a domain and for which users will be defining glossary items. Possible classes include: Application, Business Function, Business Object, Component, Functional Module, ICT Object, Business Process, Standard Platform, and Vendor Product.

enterprise architecture

An enterprise architecture is a comprehensive method and framework used to manage and align an enterprise's business processes, organizations, information technology (IT), software and hardware, local and wide area networks, people, operations, and projects with the enterprise's overall strategy.

enterprise release

An enterprise release is the process in which changes to the enterprise architecture are planned and managed via a governed process. For each enterprise release, a set of milestones are specified that constitute the stage gates for the approval and execution of the release and thus allow the enterprise release cycle to be managed.

enterprise release item

An enterprise release item expresses the deliverables that will be provided for an enterprise release. Typically, an enterprise release item is based on a project that has been defined to provide the architectural change. Alternatively, enterprise release items can be based on applications, components, or standard platforms. In this case, the architecture element constitutes the deliverable to the enterprise release.

enumeration

An enumeration is a set of preconfigured enumeration items available to users when defining an object class property in Alfabet. An enumeration is associated with a standard or custom object class property. Enumerations are reusable and the same enumeration may be assigned to multiple custom object class properties in different object classes. An enumeration has two or more enumeration items.

The enumeration items represent the values that users can select in a combo-box, checked combo box, list box or checked list box in a standard or custom editor or in standard object filters available in relevant views. For example, an enumeration could consist of the four enumeration items `Mainframe`, `Client Server`, `eBusiness`, and `Other` that are assigned to the custom object class property `Application Type`.

There are two types of enumerations displayed in the **Class Model** explorer in Alfabet Expand:

- A protected enumeration  is an enumeration that has been preconfigured by Alfabet Expand. A protected enumeration cannot be deleted but the enumeration items can be modified according to the needs of your enterprise.
- A public enumeration  is a custom enumeration created by your enterprise. A custom enumeration can be created and configured for custom object class properties of the type `String`, `Text`, `Real`, `Integer`, or `StringArray`. A custom enumeration can be edited and deleted, if necessary.

evaluation type

An evaluation type bundles one or more indicator types for use in the evaluation of a specific dimension of an object's performance. Users define an indicator for each indicator type bundled in the evaluation. For example, typical evaluation types could be Complexity and Standardization Status. The evaluation type Complexity might have the indicator types Number of Interfaces, Number of Modules, and State of the Art, and the evaluation type Standardization Status might have one indicator type also named Standardization Status.

An evaluation type may be configured once and reused for multiple object classes. In this way, all system-related classes such as Application, Component, and Standard Platform, for example, could be evaluated for their complexity, standardization, etc.

Evaluation types can also be grouped into prioritization schemes that, in turn, may be implemented as axes in portfolios.

Access to the evaluation types used in Alfabet evaluations can be controlled via user profiles. A user can view any evaluation type that has either no user profile defined or the same user profile as that which he/she is currently logged in with.

Evaluation types can be used in Alfabet queries and native SQL queries written for configured reports. Evaluation types are configured in the Configuration module.

express view

An express view is a link to a specific Alfabet view or explorer displaying data that allows individuals inside or outside of the Alfabet user community to view Alfabet information. When the express view is created, an email notification is automatically mailed to the specified recipient who receives a URL that allows him/her to access the current page view in Alfabet.

federated architecture

A federated architecture is an architecture that is typically managed in silos. Such an architecture allows a holding company to structure and communicate the elements that are shared across the enterprise, the elements that are common to some but not all organizations in the enterprise, and the elements that are exclusive to a specific organization. The assignment of mandates to Alfabet objects allows the holding company to structure the objects in the enterprise architecture and regulate their visibility to only those users who should see them.

functionality

A functionality is a set of capabilities that allow users to carry out a specific tasks in the Alfabet solution. The functionalities (technically called business functions) available in Alfabet Expand are preconfigured and their content cannot be redefined.

Functionalities are made accessible to a user profile via menu items or a navigation page. For each user profile, the assigned view scheme determines the visibility of specified object classes as well as the exclusion of custom and non-mandatory standard properties, workspaces and page views, and functionalities available in toolbars in page views and object views.

icon gallery

An icon gallery is a configurable collection of icons that can be used to graphically visualize the quantitative value of indicators in evaluations and diagrams. An icon gallery is typically a set of mutually related images whereby each icon represents a specific value range. For example, the three stages of a traffic light signal could represent 3 different indicators.

icon group

An icon group allows custom and standard icons to be bundled and made available in an icon gallery.

ICT object

An ICT object (ICT = Information and Communication Technology) is an abstract object that can represent either an architecture element (for example, an application, solution building block, component, device, standard platform, or vendor product) regardless of its versioning, or multiple architecture elements that are related for business or financial reasons.

An ICT object is owned by an organization that is usually responsible for the budget of the architecture elements assigned to the ICT object. As such, ICT objects are key to enterprise strategy and master planning. The use of ICT objects is advantageous in that the planner must not initially commit him/herself to a certain version of, for example, an application. Later, at the stage of detailed planning, the ICT object can be replaced by a specific concrete version.

Object class stereotypes may be configured for the class **ICT Object**.

inactivity monitor

An inactivity monitor is a type of monitor that alerts subscribed users about the absence of activity occurring to objects in a specified object class. The monitor owner must specify a set of objects that are to be monitored and a set of users that are defined as listeners to be alerted if the monitor is triggered. The monitor owner is typically the user responsible for the objects defined for the monitor. The monitor owner as well as the listeners will be informed via email notification if a specified object has not been edited or

reviewed within a specified period of time. If the monitored object does not need to be changed, the monitor owner can mark the object as reviewed.

income type

An income type is a classification of income. Income types typically represent benefits in a business case and are defined by the enterprise to ensure comparability in the monetary evaluation of activities and services for projects. Income types are critical for the definition of business cases in Alfabet.

indicator

An indicator is the qualitative or quantitative measurement of an object's performance. Indicators may be entered manually, by means of an external system interfacing with Alfabet, or computed or aggregated within Alfabet. Each indicator is based on an indicator type that defines the dimension and scale of measurement. One or more indicator types are bundled in an evaluation type and may be implemented in evaluations, prioritization schemes, and portfolios.

indicator type

An indicator type defines a dimension of measurement regarding the performance of an object in the IT landscape. The indicator is the value defined for an indicator type in the context of an object in the IT landscape.

A single indicator type or multiple indicator types can be bundled in an evaluation type that is used to evaluate an object. For example, the indicator types Number of Interfaces, Number of Modules, and State of the Art could be assigned to the evaluation type Complexity, and the indicator type Standardization Status could be assigned to the evaluation type also named Standardization Status.

Each evaluation type may have one indicator or more assigned to it. An indicator can be either:

- manually entered by a user
- selected from a predefined set of values by a user
- calculated by Alfabet according to defined computation rules
- calculated based on customer-specific code provided by Alfabet
- imported via an interfacing system such as ADIF

Indicator types having a predefined set of values may be associated with an icon gallery, allowing for indicators to be visualized in diagram views as well as other configured reports by means of an icon rather than a numeric value.

Indicator types are configured in the Configuration module in Alfabet.

information flow 

An information flow describes the exchange of business data between source and target applications, components, devices, or peripherals. The interface logic that is required for the exchange of information is an integral part of the information flow. Information flows between applications are realized through concrete interface systems which are typically embedded in the application's platform as a platform component. Within an application's platform, a local component can also be specified as a source or target interface enabling the data transfer of the information flow. An information flow may possess source and target interfaces as well as interface systems. Information flows can be classified by the connection type, connection method, connection frequency of the information exchange, and the connection data format describing the data exchange.

IT landscape

The IT landscape describes the entirety of architecture elements and their relationships in the enterprise, including the application architecture, business architecture, information architecture, technical architecture, and deployment architecture.

lifecycle

A lifecycle describes the succession of stages that an architecture element goes through. Many objects (for example, applications, components, standard platforms, business supports) in Alfabet have a lifecycle, although a lifecycle does not have to be defined for all objects. A lifecycle is comprised of lifecycle phases that describe the object's status of activity or production over time. Each lifecycle phase is aligned with its proceeding and succeeding lifecycle phase.

The lifecycle definition also includes the definition of the object's active period. The active period of an object is considered the period that the object is in production. Therefore, the active period of an object corresponds to the object's start and end dates. The active period may be configured to be aligned with one or more specified lifecycle phases that represent the period when the object is in production. Typically, the object state of the object will be specified as Active in the period between the start and end date.

The lifecycle definitions of dependent objects should be aligned with the lifecycle of the object that they have been defined for. For example, the lifecycles of any local components, information flows, and business supports should be aligned with the lifecycle of the application that they are assigned to. In the case of components, the lifecycles of any local components, information flows, and technical services should be aligned with the component's lifecycle.

Furthermore, the lifecycle definition for an application may include the specification of predecessor and successor versions and the lifecycle definition of components may have successor versions. Successor versions will be assigned per default to the same ICT object as the original application/component, but this can be modified, as needed.

lifecycle phase

A lifecycle phase describes a status of activity or production in the lifecycle of an object. Typical lifecycle phases include, for example, Evaluation, Pilot, Production, Limited Production and Shut Down. The active

period of the lifecycle is the period that the object is in production and is thus aligned with the object's start and end dates. One or more lifecycle phases may be left undefined. In this case, they are skipped. Each lifecycle phase is aligned with the proceeding and succeeding lifecycle phase that has been defined.

Lifecycle phases are configurable per object class. The number of lifecycle phases, their sequence, and their mapping to the active period of an object can be specified per object class by a solution designer. The active period may be mapped to one or more consecutive lifecycle phases. The active period as well as the object's start and end dates will thus align with the start date of the first lifecycle phase mapped to the active period and the end date of the last lifecycle phase mapped to the active period.

lifecycle status

See lifecycle phase.

locale text template

A locale text template is a localized version of a standard or custom text template in which the text can be translated to another language. Text templates can be generated in multiple languages, allowing the translated email notifications generated within Alfabet to be displayed for a selected culture.

mandate

A mandate is a means to organize and structure the federated architecture of a holding company. The assignment of mandates to objects allows the holding company to structure the objects in the enterprise architecture in order to regulate visibility to objects across some or all subsidiaries. Only users explicitly assigned to a mandate will see objects with that mandate definition. An object that has not been assigned to a mandate is considered not to be owned by a mandate and is thus visible throughout the holding company to users with relevant access permissions.

The use of mandates in the Alfabet solution is optional. If mandates are implemented in an enterprise, the visibility of an object with a mandate assignment will take precedence over other concepts of access permissions in Alfabet. For example, an authorized user of an object must be assigned the relevant mandate to access the object that he/she is the owner of.

Within the context of mandates, the conventional rules governing access permissions apply. Thus, a user assigned to a mandate will only have Read/Write access permission to the objects made visible by the mandate if he/she has authorized access to the object as an authorized user, deputy, member of an authorized user group or discussion group or via rule-based access permissions, workflow contributor or assignee of an assignment.

mandatory property

Each object class in the class model has a preconfigured set of properties that serve to semantically describe the object class. Some standard properties are mandatory properties and are highlighted with a red star in editors. Mandatory properties are highlighted yellow in the class model explorer.

No enforcement mechanism is implemented if a mandatory property is undefined unless the property is included in a standard or configured class key that enforces the unique definition of a new object. If the mandatory property is included in a class key, then an error message will be displayed if the property is not defined.

monitor 

Alfabet provides a variety of monitors to alert users about changes that have occurred to specific objects and may require further activity, reviews that did not occur as intended, or transactions that were planned to occur according to the documented plans. Such monitors alert the subscribed users through e-mail notifications. Monitoring is performed in regular intervals for a specific period of time. Alfabet currently provides activity monitors, date monitors, inactivity monitors, notification monitors, and consistency monitors.

monitor template

A monitor templates is configuration file that allows you to specify the use of monitors by the user community. You can enable or disable a specific monitor template, thus allowing or disallowing monitors to be created by users for the object class that the monitor template is associated with. Monitor templates are available for activity monitors, inactivity monitors, and date monitors.

A monitor alerts user about changes that have occurred to specific objects which may require further activity (activity monitor), reviews that did not occur as intended (inactivity monitor), or transactions that were planned to occur according to the documented plans (date monitor). Such monitors alert the subscribed users through e-mail notifications. Monitoring is performed in regular intervals for a specific period of time.

The monitor template allows you to specify the object classes for which monitors can be created, the properties that can be monitored for an object class as well as the monitored contexts that users can select to be monitored. Monitored contexts allow users to monitor a specific context, such as the lifecycle or information flows associated with an object. Monitor contexts are predefined.

navigation page

Navigation pages allow an enterprise to configure a pages with links and information for the enterprise's users in order to efficiently guide them to specific functionalities in the Alfabet interface. The navigation page that serves as the start page is assigned to a user profile. Navigations page can be configured to provide a variety of different kinds of support to users in their work in Alfabet. A navigation page can be freely configured and could include, for example, links to specific functionalities in Alfabet, links to other navigation pages, informational text or images about enterprise-specific workflows in the Alfabet solution, standard framework approaches like Zachmann, or URL links to internal documents or the Web.

The technical term for a navigation page is a guide page. One guide page can be assigned to a user profile as the start page for the user profile.

notification monitor

A notification monitor is a type of monitor that allows email notifications to be automatically triggered based on configured Alfabet queries or native SQL queries. The queries specify the targeted objects and their object class properties as well as the users who shall be notified about the objects found by the queries.

object

An object is a constituent of the Alfabet architecture that generally describes an element in the enterprise architecture. An object belongs to an object class defined for the Alfabet meta-model. Specific objects may also be referred to as architecture elements, artifacts, or instances. Many objects in Alfabet have a start and end date that specifies the planned period of activity for the object, a lifecycle definition, and an object state that distinguishes an object as actively used, planned to be used, or to have been used in the past.

object class

An object class is a construct that semantically describes the objects that are instantiated for that class. Object classes in the class model have a specified set of standard object class properties and may also have custom properties that have been configured by a solution designer. Object classes typically reference other object classes, thus creating a semantic network that can, for example, describe the enterprise's IT landscape.

The Alfabet class model constitutes a standard set of object classes. Please note the following information about the types of classes available:

- Private classes  cannot be edited but are visible in the class model in order to make them available for definition in an Alfabet query.
- Protected classes  may have both private properties  that cannot be edited and protected properties  that may be edited. Furthermore, public (custom) properties  may be created for a protected class.
- Public (custom) classes  should only be created in conjunction with Software AG Support . Any class that is not part of the standard class model is a public class. Public (custom) properties  must be created for a public class.

object class property

Each object class in the class model has a preconfigured set of properties that serve to semantically describe the object class. Some standard properties are mandatory properties and are highlighted yellow in Alfabet editors. Mandatory properties are highlighted yellow in the class model explorer.

Please note the following about the types of object class properties available:

- Private properties  cannot be edited. Private properties can be excluded for specified user profiles.
- Protected properties  are typically standard properties with limited editability. For example, you can edit the caption or hint attributes and specify whether a protected property can be translated. Protected properties can be excluded for specified user profiles.
- Public properties  are also referred to as custom properties and are typically created by a solution designer. Thus, custom properties may be defined, edited and deleted in accordance with the needs of the enterprise. Like private and protected properties, custom properties can also be excluded for specified user profiles.

object class stereotype

An object class stereotype is a sub-classification of an object class. A permissible object class can have multiple object class stereotypes, each of which captures a specified set of attributes, reference data, and class configurations. For example, the object class Application that may have the object class stereotypes Business Applications and Technical Applications.

Only a limited number of object classes support the configuration of object class stereotypes.

object cockpit

An object cockpit provides users with an immediate and transparent overview of data for a selected object. It is as an abbreviated and focused presentation of object data, typically from a particular perspective such as an architecture perspective or a strategy perspective.

Multiple object cockpits can be available per object profile. For example, one object cockpit for the object class Application might display data relevant for understanding the application in the as-is architecture, another cockpit might display data relevant to master planning issues, and a third cockpit might display relevant data for cost and budget issues. The object cockpit is a configuration that is available in addition to the more comprehensive object profile.

An object cockpit can be created and configured in the context of a custom object view. Although an object view may have multiple object cockpits defined, it is possible to hide object cockpits in the context of the view scheme configuration. In this case, only the object cockpits visible for the view scheme will be available to the associated user profile.

object profile

The object profile summarizes information that is relevant to a selected object. The object profile typically displays object class properties and their values for the object as well as workspaces with views and reports relevant to the selected object. Users can open the available views and define or edit the data about the selected object.

Typically, object profiles are configured specifically for the needs of the user community. Thus, various object profiles with different data and functionality may be available for the same object class, depending on

the user profile with which users access Alfabet. Each object profile may have multiple object cockpits associated with it that have been configured to visualize a specific set of data about the object.

The object profile can be configured in the context of a custom object view. It is possible to hide the object profile in the context of the view scheme configuration. In this case, only the object cockpits visible for the view scheme will be available to the associated user profile.

object state

An object state describes the operational status of an object in the enterprise. The object state indicates whether an object is actively used, planned to be used, or has been used in the past. Because an object's start and end dates specify the planned period of activity for the object, the object state should be changed from **Plan** to **Active** once the object's start date is reached. Equally, the object state should be changed from **Active** to **Retired** when the object's end date is reached. If the lifecycle concept is available for an object class the object's start and end dates will initially be aligned with the active period of the object.

An object's object state can be changed by a user with Read/Write access permissions. For objects like applications or components, relevant object dependencies must be taken into consideration when the object state is changed. For applications, for example, the user changing the object state should consider whether the object state should be propagated to the application's associated information flows or business supports.

The number of available states Retired, Active, and Plan cannot be changed, however the names of the preconfigured object states can be customized per object class.

object view

An object view is the configuration object that typically consists of an object profile as well as any object cockpits that have been configured for the object view. A standard object view  is available for all object classes that require an object profile. The object profile is a standard overview that displays the basic information about an object with topical workspaces grouping all standard page views available for the object class. Object cockpits are not available in standard object views. Multiple custom object views  may be configured for an object class or object class stereotype, whereby only one object view can be assigned per class setting per user profile.

The solution design can specify which standard and custom object class properties, page views, configured reports, and object cockpits should be visible or hidden for a custom object view. Each custom object view may include the standard object profile or a customized object profile as well as multiple configured object cockpits. The object cockpit is typically a focused and abbreviated set of data for a particular topic that includes pertinent attribute information as well as a few relevant page views or configured reports that are embedded in the object cockpit.

operational aspect

An operational aspect is a facet of the business that may be supported by the IT. Standard classes that represent operational aspects in Alfabet include Brand, Customer Segment, Market, and Sales Channel although other object classes or object class stereotypes may be implemented as operational aspects.

Multiple operational aspects can be created for each configured aspect class to express the individuals facets of business that require support. For example, aspects for the class Market could be Capital Markets, Retail Banking, Emerging Markets, and Direct Banking.

Operational aspects are relevant in the context of business support planning and analysis and allow the planner to understand the aspects of the business that an organization, market product, business process, or domain contribute to. Understanding the contribution of individual business supports to operational aspects of the business sheds light on where future support is required in order to realize the enterprise's IT strategy. Operational aspects can be defined for operational business supports, tactical business supports, strategic business supports, and solution business supports. Potential gaps or redundancy in the support of operational aspects can be understood and analyzed in the context of business support maps.

Please note that the concept of operational aspects has no association with the concept of aspect evaluations, which are implemented to assess the performance of applications and components.

organization 

An organization describes an administrative or functional unit in the enterprise. Organizations form a self-referential hierarchy. An organization may have an unlimited number of subordinate organizations but only one ascendant organization. An organization may have primary responsibility for an object in the same way that an authorized user has or it may by defined as having an important role in relation to an object. Organizations are supported in their business activities through the business support and business services provided by applications.

Object class stereotypes may be configured for the class **Organization**.

prioritization scheme

A prioritization scheme is a weighted composite of a set of evaluation types. A prioritization scheme is typically used to determine a prioritized ranking for a set of objects, for instance during budgeting and prioritization of projects or to define the axis of a portfolio.

Prioritization schemes aim at aggregating evaluations along complementary evaluation dimensions. Whereas evaluation types are aggregates of indicators for semantically-related performance measurements of objects, prioritization schemes are typically aggregates of semantically-unrelated (or loosely related) performance measurements for objects. However, prioritization schemes are assigned to a portfolio usually describe performance along dependent dimensions.

project

A project is an activity undertaken in order to achieve a specified goal in the IT landscape. Typically, projects are based on configured project stereotypes. A project may have unlimited sub-projects (projects of the subordinate project stereotype) but only one ascendant project (a project of the ascendant project stereotype). Projects may be grouped into project groups as well as assigned to buckets for budgeting and prioritization purposes.

Depending on the configuration defined for a project stereotype, it may be possible to bundle demands to projects, document the as-is architecture that may be impacted by the project, plan the to-be architecture for the IT landscape, plan and assess the project's costs in a business case, capture the skill requests and resource requests needed for the project and manage the project resource plan, and monitor project target dates via milestones.

Furthermore, you can also work with project scenarios based on an existing project in order to plan alternative what-if scenarios in terms of costs, resources, scheduling, and to-be architectures. Finally, project baselines may be created for a project in order to understand the scope of changes that have occurred in a project.

project baseline

A project baseline captures the scope of its base project at the time that the project baseline is created. The project baseline provides a ReadOnly view of the base project's data and allows the deviation of the base project from its original scope to be tracked over time in terms of the total project costs and income, skill request and resource request costs, evaluations, scheduling and achievement of project milestones, and the changes made to the to-be architecture.

project solution

A project solution allows alternative projects to be drafted and considered. Multiple project solutions may be defined for a specific project, but only one project solution can be selected as the approved project solution. Once a project solution is selected, it is considered a project in its own right and replaces the original project that it was defined for in the project hierarchy. Project scenarios are an extension of the project solutions concept but provide much more flexibility and functionality in the planning and implementation of what-if scenarios.

project scenario

A project scenario is an alternative what-if scenario to be explored, planned, and potentially implemented for the project. The project scenario may target alternative to-be architectures, project costs and income, the scope of skill request and resource request required for the project, various evaluations of different aspects of the project, and different versions of project dependencies and time schedules.

A project scenario is an independent copy of the project and includes its start date and end date, custom properties as well as the base project's business case as-is architecture, to-be architecture, requested skills and resources, value nodes, contract deliverables, and input demands. All subordinate projects will also be copied.

Each project scenario can be further defined and modified without impacting the base project. ´The project scenario can be merged at any time to the base project in order to update the base project. The project scenario as well as the base project continue to exist and both can be further modified, if needed.

project stereotype

A project stereotype represents a level in the project management framework. The project hierarchy is made up of an arbitrary number of object class stereotypes defined for the class `Project`. An unlimited number of objects can be created in Alfabet for each project stereotype.

The project hierarchy can be configured as a linear hierarchy or as a branched tree structure that allows complex project structures and hierarchies of stereotypes to be captured. Each project stereotype may define multiple subordinate project stereotypes. The top-level project stereotype represents the most abstract level of a project and the lowest level project stereotype represents the most granular level of project tasks that must be acted upon in order to realize the project. For example, a typical linear project might be made up of the following project stereotypes: Level 1: Program, Level 2: Statement of Work, Level 3: Project, Level 4: Sub-Project, Level 5: Project Activity.

The name, number, and hierarchical ordering of the project stereotypes are configurable. A project stereotype is similar to an object class. Like an object class, the projects based on a project stereotype are instances of the project stereotype. For example, the project stereotype Project Activity will have objects called project activity. Each project stereotype may have only one ascendant project stereotype and only one subordinate project stereotype. However, projects of a specific stereotype do not have to be to be associated with a project of the ascendant project stereotype nor with projects of the subordinate project stereotype.

The definitions that can be made for each project stereotype must be configured. Therefore the views that are available may differ for each project stereotype in the hierarchy. For example, the configuration will determine for which project stereotype users can bundle demands to projects, document the as-is architecture, plan a to-be architecture, assess alternative project scenarios, track costs via cost centers, and monitor projects via milestones.

query

See alfabet query.

release status

A release status describes the state of approval for or agreement about an object in the enterprise. Typical status values could be, for example, `Draft`, `Described`, `Reviewed`, `Approved`,`Rejected`, and `Retired`. However the release statuses will largely depend on the object class that they describe. For example, the release statuses for an architecture element such as an application, component, or standard platform is typically used to express agreement to the state of the documented information whereas, the release statuses for a planning artifact like a demand or project reflects an approval status. The release status for an assignment describes the state of progress or completion of the task that the assignment is about.

An object's release status can be changed by a user with Read/Write access permissions. For all release statuses that are configured as non-editable release statuses, users will not be able to edit the attributes in the object's editor and page views. Users will see the  icon at the top of the object profile of an object that may not be edited due to its current release status definition.

Release status definitions can be customized per object class.

report

A report is a configured display of data from the Alfabet database. Alfabet provides a number of standard reports, however customized reports can also be configured by a solution designer.

A configured report can either be generated by mechanisms internal to alfabet via alfabet queries or native SQL queries or by means of an external reporting tool that searches the Alfabet database for current results using SQL query language. Alfabet internal reports are based on preconfigured design types such as tabular reports, treemap reports or layered diagram reports. Once a report has been configured, it can be made available to the user community in the Alfabet interface.

resource

A resource is any entity that is needed as a source of support for another entity in the enterprise. A resource may be provided by many different object classes such as an organization, application, device, deployment, etc. A resource is typically requested by a project, service product, or organization.

role

A role defines the functional relationship or responsibility that a user or organization has to an object. A role is based upon a configured role type that is configured for an object class. Roles are defined for informational purposes only and provide detail about users or organizations that may have information about or a stake in the object. The definition of a role for an object does not impact access permissions.

role type

A role type is configured in order to allow a functional role to be defined for objects in a specified object class. Role types can be configured to be explicitly available for a user or organization or both.

rule-based access permission

A rule-based access permission is a rule based on an alfabet query or native SQL query that specifies access permissions for a found set of objects or object class.

service level agreement (SLA)

An SLA is a service level agreement that is defined for a service product. The SLA captures information such as a volume base that is relevant for the service product (for example, Service Desk Support Hours, Target Resolution Time, Defect Backlog Size Limit, Maximum Number of Test Failures, etc). Only one SLA may be assigned to a service product. The service level agreement automatically inherits the authorized user definition of the service product that it is assigned to.

Object class stereotypes may be configured for the object class **Service Level Agreement**. This allows for different attributes to be used for different types of service level agreements.

service product

A service product is a service that is owned by an organization and made available to other entities. A service product consists of one or more service product items, which constitute the service product. Each service product item is associated with an object such as an application, component, ICT object, deployment, device, standard platform, organization, or other service product, which provides the service product item to the service product.

Each service product may have one SLA (service level agreement) assigned to it.

Service products are organized according to the organization that owns them. Each organization and the service products that it owns constitute a service product catalog. Additionally, service products may be structured in one or more service product groups independent of the organization that provides the service products.

A service product may be defined as a contract deliverable for a contract, whereby the service product is consumed by the contract. A service product may be assigned to a market product in order to indicate that the service product is made available via the market product.

service product item

A service product item is an entity that supports a service product. The service product may be constituted by one or more service product items. Each service product item is associated with an object from a permissible object class (such as an application, component, ICT object, deployment, device, standard platform, organization, or other service product) that provides the service product item to the service product.

A name may be defined for the service product item to capture the purpose of the object associated with the service product item. For example, the service product User Support Service might consist of the service product items Application User Support Services which is provided by applications, Deployment User Support Services which is provided by deployments, and Device User Support Services which is provided by devices.

skill request template

A skill request template allows skills to be specified in order to plan resources for a project. A skill request template specifies the head count or man days required for the skill and the expected cost required by the

skill in the context of a particular kind of project. Users making a skill request thus have predefined information that helps to estimate the resources and costs involved if the skill is implemented in the project. Skill request templates can be assigned to a project template in order to streamline the process of project planning.

solution object

A solution object is any architecture element that is proposed in the context of a to-be architecture for a project. Solution objects may be based on objects assigned to the project's as-is architecture or may be created from scratch in the context of the to-be architecture. Solution objects include solution applications, solution components, solution devices, solution information flows, solution local components, solution peripherals, solution standard platforms, solution technical services, and solution business supports. A solution object exists outside the inventory until the architectural changes proposed in the to-be architecture are checked in to the inventory. After check-in, the solution object becomes a real inventory object and will have the object state **Plan**. The object state must be manually changed to **Active** when operation begins.

strategic business support

A strategic business support is a targeted business support defined for an unspecified timeframe in the context of the long-term target architecture. A strategic business support is typically provided by an existing or planned ICT object. However, strategic business support may also be configured to be provided by a virtual ICT object to define an ad-hoc ICT object that does not yet exist in the inventory, a solution building block to describe functional planning as endorsed by TOGAF, or an organization to describe non-IT support such as internal or external services.

technical service

A technical service is a service provided by a component in order to fulfill technical needs that are necessary to support business service requests. A technical service may define technical service operations that support the technical realization of business functions.

Technical services can be defined for components and local components for which the **Component Type** attribute is set to `Service`. When a local component of the type `Service` is embedded into the architecture of an application or another component (of the type `Infrastructure` or `Business`, for example), the technical service operations of the embedded component can be mapped to the business services provided by the application or the local component.

Object class stereotypes may be configured for the class **Technical Service**.

technical service operation

A technical service operation is a detailed description about how a technical service is to be provided by a component in order to support a component in providing a business service. An operation is defined for the specific technical service that it helps to realize. The operation defined for a technical service can be

assigned to the business services that are provided by the component that the component supports. A technical service can have multiple technical service operations defined for it.

Object class stereotypes may be configured for the class **Technical Service Operation**.

text template

A text template is the predefined text in an email that is automatically generated as the result of an action triggered by a user. Text templates for email messages are available for the following contexts:

The text template defines the email text in English, the relevant references to objects and object class properties in Alfabet, and hyperlinks to the relevant objects in Alfabet. The content displayed in the English-language text template can be edited to suit the needs of your enterprise. Locale text templates can be created in order to translate the content of the text template to the non-English languages implemented in your solution interface.

to-be architecture

The to-be architecture is a medium-term plan for the enterprise's architecture and defines a clearly approved strategic guideline. The planned consolidated to-be architecture is captured in one or more master plans, which define the planned tactical business support. The realization of the to-be architecture is addressed in the context of projects.

A project's to-be architecture typically includes inventory objects from the as-is architecture that may be affected by proposed architectural changes as well as new solutions objects such as solution applications, solution components, solution devices, solution information flows, solution local components, solution peripherals, solution standard platforms, solution technical services, and solution business supports created in the context of the to-be architecture. The solution objects are considered to be "shadow" objects — copies of the base applications, information flows, etc. that they were derived from. The solution objects exist only in the context of the to-be architecture definition and may be modified within the context of the to-be architecture without affecting the inventory objects they are based on. The architectural changes defined in the context of the proposed to-be architecture can be checked in to the inventory, whereby the solution objects that are based on inventory objects overwrite their corresponding inventory object. All architecture elements added to the inventory via the proposed to-be architecture will have an object state indicating that the object is planned. The object state must be manually changed to an active object state when operational use of the to-be architecture element begins, which is typically upon completion of the project's implementation.

user

A user is a person that is known to the Alfabet system. Users have access permissions and are associated with one or more user profiles, which control accessibility to the functionalities. A user may or may not be an authorized user for an object. Authentication and authorization are also controlled by means of the user definition.

user group 

A user group bundles a set of users. Once a user group is assigned to an object, it is referred to as the object's authorized user group.

user profile

User profiles are the basis of user administration in Alfabet and serve as the entry point when accessing Alfabet. Every user must log in with a user profile that has been assigned to him/her by a user administrator. Therefore, all users accessing Alfabet must be assigned at least one user profile. However, users may possess multiple user profiles in accordance with their responsibilities in the user community and in the enterprise as a whole. A user can switch to another permissible user profile at any point during a user session.

A user profile specifies the Alfabet functionalities available to a user, the visibility and editability of object classes and object class attributes, as well as the availability of associated capabilities including, for example, wizards and workflows.

Each user profile must have one view scheme assigned to it that bundles various class settings. The user profile may have only one class setting defined per object class.

value node

A value node represents a strategic intention at a specific level of abstraction. The level of abstraction is determined by the value node stereotype that the value node belongs to. Value nodes assigned to a top-level value node stereotype represent a highly abstract strategic intention (for example, Become an Integrated Financial Service Provider) whereas the leaf-level value nodes typically describe the action needed to realize the ascendant strategic intention. Thus the value node stereotype at the lowest level usually represents strategic initiatives (for example, Establish New Credit & Loans Management Solution).

value node stereotype

A strategy network is made up of an arbitrary number of levels called value node stereotypes. The top-level value node stereotype represents the most abstract level of strategic intention and the lowest leaf-level value node stereotype represents those strategic initiatives that must be acted upon in order to realize the abstract strategic intentions. For example, a typical strategy network might be made up of the following value node stereotypes: Level 1: Vision, Level 2; External Trends, Level 3: Business Drivers, Level 4: Business Requirements, Level 5: Architectural Requirements, Level 6: Initiatives. The name, number, and hierarchical ordering of value node stereotypes is configurable.

A value node is thus defined for a specific value node stereotype. Any pair of value nodes that are related to one another in the strategy network must belong to different value node stereotypes that are adjacent to each other in the hierarchy. In other words, value nodes are related via a parent-child relationship. The relationship between two value nodes is represented by value node arcs, which allow the relative ranking of the strategic intentions to be computed.

vendor product 

A vendor product is a good or service that is provided by a vendor to the company. A vendor product may be associated with one or more technical components. For example, the vendor product Oracle RDBMS may be split into the components Oracle 11i Server, Oracle 11i Client, and Oracle 11i OCL. Further, a component may reference a vendor product, thus indicating that it has been derived from the vendor product.

Vendor products may be created directly in Alfabet or, depending on the configuration of your Alfabet solution, imported from the Technopedia® repository. Vendor products may only be imported from Technopedia® product categories that are on the leaf level of the product category hierarchy.

Object class stereotypes may be configured for the object class **Vendor Product**.

view scheme

A view scheme is associated with a user profile and groups a set of class settings that describe the visibility of objects in the associated object classes. Only one class setting per object class may be assigned to a view scheme.

The view scheme also contains the definition about the functionalities available in the object views accessible via the view scheme. For each standard and configured object view available to the view scheme, workspaces, page views, and toolbar buttons that allow object data to be created, edited, and deleted, for example, can be hidden.

virtual ICT object 

A virtual ICT object represents a visionary ICT object that may not yet be defined in the inventory but is needed to represent a strategic business support or tactical business support defined in an IT strategy map or a master plan map. A virtual ICT object is for planning purposes only and is not visible anywhere outside of the areas of strategy planning or master planning.

After a virtual ICT object is defined in a business support map, it can later be replaced with a real ICT object or application. In this case, the real ICT object replaces all business supports associated with that virtual ICT object.

vocabulary

A vocabulary constitutes the set of original and translated strings that are displayed in Alfabet. The translatable strings are divided into separate terminology sets based on their source of origin in the software. These include `METAMODEL`, `ITPlan`, `Platform`, and `GUIDEPAGES`.

wizard

A wizard is an assistant that consists of a configured set of standard editors, custom editors, standard views, and configured reports. The wizard will typically guide a user through a linear multi-step process to capture data for an object that the user has access permissions to.

Standard data capture wizards are available for commodity classes in which a large number of objects are typically documented. An unlimited number of custom wizards may be configured for an object class in order to allow different users working in different contexts to capture the object data that they are responsible for. However, only one wizard may be available for an object class per user profile.

wizard step

A wizard step is an editor, view, or report in a wizard. The wizard step constitutes one aspect of data entry or review that is required in the data collection process. A wizard step may be reused for multiple wizards for the same object class.

Each wizard step may have instructions configured that are displayed in the header of the wizard step. The current step that the user is currently processing vs. the total number of wizard steps are automatically generated and displayed in the header of the wizard step.

A wizard step may consist of any of the following:

An editor, view, or report can be reused in the configuration of multiple wizard steps in the same wizard, thus enabling the user to enter the data in multiple steps rather than having to complete the data entry process all at once. Furthermore, each wizard step can be configured so that only the relevant standard and custom object class properties are displayed for a wizard step.

A wizard step may have an unlimited number of subordinate wizard steps that can constitute a branch in the wizard process. Any wizard step or subordinate wizard steps may be controlled by pre-conditions that determine whether the wizard step should be entered or skipped. The pre-conditions are configured as Alfabet queries or SQL queries that determine the criteria that must be fulfilled in order to enter a specified step. Multiple post-conditions may be defined per wizard step in order to prompt users to provide data for specified standard or custom object class properties before proceeding to the next wizard step.

workflow

A workflow is a collaborative process made up of workflow steps that are typically carried out by one or more users. A workflow is based on a configured workflow template that determines the sequence of workflow steps that are to be performed on a specific object and its references by specified user(s). Workflow steps may have specific pre- and post-conditions that determine different paths to take in the workflow depending on whether the conditions are or are not met.

Typically, the workflow owner is the user who initiates and is responsible for maintaining the workflow. When a workflow is initiated by the workflow owner and when a workflow advances to the next workflow step, relevant users may be informed via automatically-generated emails of their impending responsibility for the workflow step. This functionality ensures that all relevant users are informed and reminded of their responsibilities in the collaborative workflow. The option to refuse, delegate, and pause workflow steps, remind users of an impending target date for a workflow step as well as redirect a workflow that has

encountered an error enables workflow owners and workflow administrators to keep track, coordinate, and manage the completion of each workflow step and the workflow as a whole.

workflow post-condition

A post-condition is based on a query that checks whether a workflow step has been completed. Each workflow step may have multiple post-conditions defined. All post-conditions must be fulfilled in order to confirm and complete a workflow step and advance to the subsequent workflow step.

workflow pre-condition

A pre-condition is based on an Alfabet query or an SQL query that checks whether a workflow step may or may not be executed. Each workflow step may have multiple pre-conditions defined. All pre-conditions must be fulfilled before the workflow step may be started. Pre-conditions are required, for example, when a workflow step has multiple next workflow steps defined. Each next workflow step must have one or more pre-conditions defined in order to determine which of the potential next workflow steps should be the subsequent next workflow step.

workflow step

A workflow step is an activity or task in a workflow that must be performed. The workflow template may have as many workflow steps as needed to complete the relevant task. Standard and custom editors, wizards, page views, configured reports, and object profiles may be implemented in the workflow step so that users can complete the task. A workflow step may entail entering, modifying or reviewing data, or a workflow step may be configured to be automatically executed by the Alfabet system. If the workflow step is not automatically executed by the system, one or more users may be defined as responsible to perform the workflow step. A user may delegate a workflow step to another user or refuse a workflow step. Depending on the configuration of the workflow step, a user may be required to confirm a workflow step before the workflow can advance to the next workflow steps, or the confirmation of the workflow step may occur automatically. A workflow step may also have one or more associated workflow step actions which allow various operations to be performed when the workflow step is entered, exited, refused, or expired. Additionally, each workflow step may have one or more pre-conditions or post-conditions that must be fulfilled in order to enter or exit the workflow step.

workflow step action

A workflow step action is a configured operation that is automatically executed when a workflow step is entered, exited, refused, or expired. A workflow step action may consist of a property being updated for the targeted object, an e-mail notification being set to users, or a sub-workflow being triggered.

Additional workflow step actions are available for administrative purposes such as stopping a running workflow or closing steps for a sub-workflow that has stagnated.

workflow template

A workflow template is a customer-defined blueprint for one or more workflows. The template specifies what object class is the point-of-departure in the workflow, which user groups and/or user profiles may initiate and administrate the workflow, which workflow steps comprise the workflow as well as their sequence, any possible per- and post-conditions or update actions associated with a workflow step, and what kinds of workflow notifications should be sent to collaborating users in which contexts. The user who creates the workflow template is the workflow template owner.

A workflow template must have the attribute **Workflow State** attribute set to `Plan` to be configured and validated. Once the workflow template is completed and approved, the attribute **Workflow State** must be switched to `Active` in order to make it available to the user community. Once the workflow template is available in Alfabet, a permitted user may initiate a workflow based on the workflow template. Multiple workflows may be simultaneously initiated and running for a workflow template.

XML object

An XML object is a configuration file made up of XML elements that allows definitions to be specified for diverse objects and functionalities in the solution environment. In Alfabet Expand, you will find a variety of existing preconfigured XML objects  that serve as templates to configure the XML objects relevant to your enterprise.

INDEX

