software AG

API Integration with Third-Party Components

Alfabet Reference Manual

Documentation Version Alfabet 10.15

Table of Contents

Chapter 1:   Configuring Integration Solutions

Multiple interfaces are available in Alfabet that provide interoperability or integration with a variety of products and services. The implementation of an integration solution requires configuration of the interface. The basic configuration is carried out primarily in XML objects available in Alfabet Expand. Much of the interface will also depend on the configuration and execution of ADIF schemes for data exchange between the external service/product and the Alfabet database.

> ⚠ The data will be checked prior to import for all integration solutions that require the execution of an ADIF scheme, and characters that are not XML-compliant will be removed prior to import to avoid data storage issues in the Alfabet database.

The following information is available:

Chapter 2:   General Configurations Valid for Multiple Integration Solutions

The features described in this section apply to multiple or all integration solutions.

The following configurations are general configurations:

- Configuring Server Variables for Integration and Interoperability Solutions

- Configuring Use of Self-Signed Certificates for Integration Solutions

- Activating Interoperability with a Service Registry via the XML Object ServiceRegistryManager

Configuring Server Variables for Integration and Interoperability Solutions

The definition of server variables allows values for some XML attributes in the XML objects in the **Integration Solutions** folder in Alfabet Expand to be defined in the server alias configuration. Defining the values in the server alias configuration instead of directly defining them in the XML object eases the propagation of changes and enhances security. Values for server variables are stored encrypted in the server alias configuration. The setting of server variables can be done either directly in the server alias configuration editor or imported into the server variable in encrypted format using a command line utility.

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.

For an existing server alias configuration, a server variable can be changed or added via the command line tool `AlfaAdministratorConsole.exe` with the following command line:

```
AlfaAdministratorConsole.exe -msalias <alias name> -editvariables -variable
<variable name> -value <variable value>
```

For an existing server alias configuration, a server variable can be deleted via the command line tool `AlfaAdministratorConsole.exe` with the following command line:

```
AlfaAdministratorConsole.exe -msalias <alias name> -editvariables -variable
<variable name>
```

The table below displays the command line options:

Command Line Option	Mandatory/ Default	Explanation
`-editvariables`	Mandatory	To change the variable definitions in the server alias configuration, the command line must contain the parameter: `-editvariables`
`-msalias <alias name>`	Mandatory	Enter the server alias name as specified in the `AlfabetMS.xml` configuration file for access to the database.
`-msaliasesfile <Alfabet configuration file path>`	Optional	If the `AlfabetMS.xml` configuration file that contains the specification of the alias is not located in the same directory as the executable, the path to the `AlfabetMS.xml` file must be specified with this parameter.
`-variable <variable name>`	Mandatory	Specify the name of the server variable. If a server variable with the specified name does not exist in the server alias, a new server variable will be created. If a server variable with the specified name exists in the server alias, the value will be changed to the value defined with `-value`. To delete an existing server variable, specify the existing server variable name with `-variable` and do not add a `-value` definition to the command line. ⓘ The variable name can only contain letters (of the English alphabet), numbers and underscore.

Command Line Option	Mandatory/ Default	Explanation
`-value <variable value>`	Optional	Specify the value for the server variable. To delete an existing server variable, specify the existing server variable name with `-variable` and do not add a `-value` definition to the command line. ⓘ The following characters are not allowed in server variable values: " < > These characters can be written as HTML code in the server variable value: • `>` for > • `<` for < • `"` for "

Configuring Use of Self-Signed Certificates for Integration Solutions

For integration solutions based on web services, self-signed certificate validation can be used on HTTPS connections.

This requires the following configuration:

1) Copy the self-signed certificates from the third-party web service to a local folder that the Alfabet Web Application has access permissions to.

2) Open the Alfabet Administrator.

3) In the explorer, click **Alfabet Aliases**. The right pane displays a list of all available alias configurations.

4) In the table, select the alias configuration of the Alfabet Web Application.

5) In the toolbar, click the **Edit** button. You will see the editor in which you can edit the alias configuration.

6) Open the **Server Settings** > **Security** tab.

7) Enter the path to the folder containing the self-signed certificates in the field **Path for Self-Signed Public Certificate Files**.

8) Click **OK** to save your changes.

Chapter 3: Configuring Interfaces based on Open API Specifications

Alfabet 's generic API integration interface provides a flexible means to integrate with third-party components having a RESTful API described via the Open API swagger definition.

The import and export via the generic API integration interface is strictly customer configured. The interface does not provide any methodological pre-configuration. You configure the mapping of data from the third-party component to object classes and object class properties in Alfabet during setup of the interface.

For some third-party components, component specific interfaces are available which provide data import tailored to the specific component. For details about the component specific interfaces, see the respective section in this documentation.

Data import from or export to the third-party component is done in a batch process via the Alfabet data integration framework (ADIF). ADIF jobs can be scheduled on demand to import or export data in regular time intervals like once a day or once every hour.

> ⓘ   Software AG cannot guarantee that the implementation of the generic API integration will succeed with every third-party component having an Open API definition for its RESTful service interface. If data import or export via a generic API fails, please contact Software AG Support to evaluate whether the problem can be solved by changing the configuration or whether a connection to the third-party component cannot be supported. This also helps us to enhance the functionality supported by the generic API integration interface.

Prerequisites

Check the following to evaluate whether integration with a third-party component can be configured via the generic API integration interface:

- The RESTful service interface of the third-party component is described in a swagger file compliant with the Open API specification (OAS) notation.

- The Open API and OAuth authorization standard specifications apply to the OAS version 3 and OAuth standards.

- The software version specified in the swagger file matches the version of the third-party component you would like to integrate to.

- An HTTPS connection can be established to the RESTful services of the third-party component. The connection is For example, not blocked by a firewall. The interface will send POST, GET, and PULL requests in a stateless manner and receive the responses.

It is recommended to familiarize with the swagger definition prior to configuring an import. You can use a swagger editor, For example, `https://editor.swagger.io`, to view the content with enhanced usability. The following example shows part of a swagger file from GITHub® in a swagger editor. You can see the different types of data, like issues and projects, and the RESTful service endpoints for import or export of data for the data type. Endpoints using the method `GET` are defined for export of data while endpoints using the method `POST` are defined for import of data. The swagger definition provides detailed information about which attributes of a specific data type can be exported and which attributes must be provided to import data.

You will see the information about each endpoint with the path and the method. If you expand an endpoint, the parameters to be set in the request header as well as JSON body fields of the request and the response are listed.

Overview of Configuration Steps

The complete configuration covers

- the setup of the connection to the RESTful services of the third-party component,

- the definition of the structure required to exchange data via the RESTful service endpoints of the third-party component,

- the mapping of data between Alfabet and the third-party component, and

- the import and export mechanisms to trigger import and export via the Alfabet user interface.

The definitions require configuration steps in both Alfabet Expand and the Alfabet user interface. the configuration steps are described in the order of configuration required for defining the interface:

Defining the HTML and Authentication Parameters

Prerequisites: A valid access to the RESTful services of the third-party components is granted and the required configuration parameters are available for the configuration.

The configuration is done in Alfabet Expand in the XML element **GenericAPIIntegrationConfig**.

To access the XML element, click the **Presentation** tab and expand the explorer nodes **XML Objects** > **Integration Solutions**. Double click the **GenericAPIIntegrationConfig** node to open the XML definition.

The XML definition of a connection to a third-party component consists of three parts: the authentication definition, the connection definition, and an optional proxy definition.

You can define connection parameters for multiple third-party components. Each part of the definition is therefore wrapped in a header element.

```
<GenericAPIIntegrationConfig>

    <Proxy url="" user="" password="" domain="">

        <AdditionalProxies>

            <AdditionalProxy Name="" url="" user="" password="" domain="" />

        </AdditionalProxies>

    </Proxy>

    <Authentications>

        <Authentication Name="DevOpsCloudOApp" Type="OAuth2"
        Flow="AuthorizationCode" ClientId="" Scope="" ClientSecret=""
        CallBackURL="" AuthCode="">

            <Authorization
            Url="https://app.vssps.visualstudio.com/oauth2/authorize">

                <RequestParameters>

                    <Parameter Name="client_id" Value="{ClientId}"/>

                    <Parameter Name="response_type" Value="Assertion"/>

                    <Parameter Name="state" Value="{State}"/>

                    <Parameter Name="scope" Value="{Scope}"/>

                    <Parameter Name="callbackURL" Value="{CallBackURL}"/>

                </RequestParameters>

            </Authorization>

            <TokenRequest
            Url="https://app.vssps.visualstudio.com/oauth2/token">
```

```xml
            <RequestParameters>

                <Parameter Name="client_assertion_type"
                Value="urn:ietf:params:oauth:client-assertion-type:jwt-
                bearer"/>

                <Parameter Name="client_assertion" Value="{ClientSecret}"/>

                <Parameter Name="grant_type"
                Value="urn:ietf:params:oauth:grant-type:jwt-bearer"/>

                <Parameter Name="assertion" Value="{AuthCode}"/>

                <Parameter Name="redirect_uri" Value="{CallBackURL}"/>

            </RequestParameters>

        </TokenRequest>

        <RefreshTokenRequest
        Url="https://app.vssps.visualstudio.com/oauth2/token">

            <RequestParameters>

                <Parameter Name="client_assertion_type"
                Value="urn:ietf:params:oauth:client-assertion-type:jwt-
                bearer"/>

                <Parameter Name="client_assertion" Value="{ClientSecret}"/>

                <Parameter Name="grant_type" Value="refresh_token"/>

                <Parameter Name="assertion" Value="{RefreshToken}"/>

                <Parameter Name="redirect_uri" Value="{CallBackURL}"/>

            </RequestParameters>

        </RefreshTokenRequest>

    </Authentication>

</Authentications>

<GenericAPIIntegrationConnections>

    <GenericAPIIntegrationConnection Name="Onpremise1"
    InstanceURL="https://dev.azure.com" Proxy="None"
    Authentication="DevOpsCloudOApp" Timeout="60">

    </GenericAPIIntegrationConnection>

</GenericAPIIntegrationConnections>

</GenericAPIIntegrationConfig>
```

The parts of the configuration are described in the following sections:

- Specifying Authentication Parameters

  - HTTP Authentication

  - Authentication with API Key

  - OAuth Authentication

  - OAuth2 Authentication with Authorization

  - OAuth2 Authentication with Password Grant

- Specifying Connection Parameters

- [Specifying Routing of the Connection via a Proxy](#)

  - [Using one Proxy for all Connections](#)

  - [Using Multiple Proxies](#)

Specifying Authentication Parameters

Prerequisites: You have access to the authentication parameters required to access the RESTful services as provided by the third-party vendor.

Alfabet 's generic API integration interface supports HTTP, OAuth and OAuth2 authentication.

- [HTTP Authentication](#)

- [Authentication with API Key](#)

- [OAuth Authentication](#)

- [OAuth2 Authentication with Authorization](#)

- [OAuth2 Authentication with Password Grant](#)

HTTP Authentication

Set the following parameters in the `Authentication` XML element:

```
<Authentication

    Name="HTTPAuthenticationForMyApp"

    Type="Http"

    UserName="User1"

    Password="User1Password!"

>

</Authentication>
```

XML element / XML attribute	Value	Man-da-tory
Authentica-tion		
Name	The name of the authentication definition. The name is used to identify the authentication definition in other parts of the XML specification.	yes
Type	Http	yes

XML element / XML attribute	Value	Mandatory
UserName	A valid user name to access the RESTful service of the third-party component.	yes
Password	The password of the specified user to access the RESTful service of the third-party component. For security reasons you should specify at least the user password as a link to a server variable instead of using plain text. The server variable is either stored encrypted in the server alias of the Alfabet Server. The use of server variables is described in the section Configuring Server Variables for Integration and Interoperability Solutions.	yes

Authentication with API Key

Set the following parameters in the `Authentication` XML element:

```
<Authentication

    Name="KeyAuthenticationForMyApp"

    Type="APIKey"

>

    <APIKey Key="KeyField" Value="KeyValue">

</Authentication>
```

XML element / XML attribute	Value	Mandatory
Authentication		
Name	The name of the authentication definition. The name is used to identify the authentication definition in other parts of the XML specification.	yes
Type	APIKey	yes
APIKey		
Key	The RESTful service header field name for the key.	yes

XML element / XML attribute	Value	Mandatory
`Value`	the API key for access to the RESTful service. For security reasons you should specify at least the API key as a link to a server variable instead of using plain text. The server variable is stored encrypted in the server alias of the Alfabet Server. The use of server variables is described in the section Configuring Server Variables for Integration and Interoperability Solutions.	yes

OAuth Authentication

Prerequisites: The authentication tokens of the third-party component must match the standard authentication definitions. You can view the standard at https://www.oauth.com.

OAuth authentication is a dynamic authentication mode with a separate token request for each session.

Set the following parameters in the `Authentication` XML element:

```
<Authentication

    Name="OAuthAuthenticationForMyApp"

    Type="OAuth"

    Flow="Custom"

    Mode="Dynamic"

    ClientId="MyClientID"

    Scope="fullAccess"

    ClientSecret="MyClientSecret"

    CallBackURL="https://thiscomponent.com/api"

    AuthCode=""

>

    <TokenRequest Url="http://tokenrequestserver.com/api/auth/login">

        <RequestParameters>

            <Parameter Name="HeaderFieldName" Value="HeaderFieldValue"/>

        </RequestParameters>

        <ResponseParameters>

            <Parameter Name="AuthToken" Path="HeaderFieldName" From="Header"
            Value="">

        </ResponseParameters>

    </TokenRequest>

    <ResourceRequest>

        <RequestParameters>

            <Parameter Name="HeaderFieldName" Path='AuthToken'
            From="TokenRequest" Value=""/>
```

```
          </RequestParameters>

       </RefreshTokenRequest>

     </Authentication>
```

XML element / XML attribute	Value	Mandatory
Authentication		
Name	The name of the authentication definition. The name is used to identify the authentication definition in other parts of the XML specification.	yes
Type	OAuth	yes
Flow	Custom	yes
Method	Dynamic	yes
ClientId	The client ID provided by the third-party vendor for access to the RESTful services. For security reasons you should specify at least the user password as a link to a server variable instead of using plain text. The server variable is stored encrypted in the server alias of the Alfabet Server. The use of server variables is described in the section Configuring Server Variables for Integration and Interoperability Solutions.	depending on third-party vendoryes
ClientSecret	The client secret provided by the third-party vendor for access to the RESTful services.	depending on third-party vendorno
CallBackURL	The URL the request is redirected to after authentication to process the request. The callback URL is also referred to as redirect Uri and is provided by the third-party vendor for access to the RESTful services.	yes
Scope	If the third-party component limits access to functionality via Oauth2 via scopes, specify a scope defined for the client ID.	depending on third-party vendor
AuthCode	If the third-party component bases access on an authorization code provided to you, enter the code. Otherwise leave this field empty.	depending on third-

XML element / XML attribute	Value	Mandatory
		party vendor
TokenRequest		
URL	enter the URL for sending token requests as provided by the third-party component vendor.	yes
TokenRequest > RequestParameters > Parameter	Specify the RESTful service header fields which need to be included in the token request to the third-party component.	yes
Name and Value	Define the name and the value for each field in a separate XML element **Parameter**.	yes
TokenRequest > ResponseParameters > Parameter	The response parameter definition establishes a temporary storage of values returned by the token request call. The values are required for the subsequent resource request, that means for the call to the callback URL for data transmission. Usually, the token is the only parameter that needs to be defined.	yes
Name	Enter a name for the variable used to temporarily store the returned value.	yes
From	Header	yes
Path	The name of the request return header field containing the value.	yes
Value	Leave this XML attribute empty.	
ResourceRequest > RequestParameters > Parameter	Specify the authentication header fields required for authentication in the data transmission calls after token request. For each header field, an XML element **Parameter** must be defined, Usually, only the token needs to be defined.	yes
Name	The name of the resource request header field.	yes
From	TokenRequest	yes

XML element / XML attribute	Value	Mandatory
Path	The name of the variable defined for temporary storage of the value from the token response with the XML attribute `Name` in the XML element **TokenRequest > ResponseParameters > Parameter**.	yes
Value	Leave this XML attribute empty.	yes

OAuth2 Authentication with Authorization

Prerequisites: The authentication tokens of the third-party component must match the standard authentication definitions. You can view the standard at https://www.oauth.com.

OAuth2 authentication requests a new token in a configurable time interval.

Set the following parameters in the `Authentication` XML element:

```
<Authentication

    Name="OAuth2AuthenticationForMyApp"

    Type="OAuth2"

    Flow="AuthorizationCode"

    ClientId="MyClientID"

    Scope="fullAccess"

    ClientSecret="MyClientSecret"

    CallBackURL="https://thiscomponent.com/api"

    AuthCode=""

>

    <Authorization Url="https://thiscomponent.com/oauth2/authorize">

        <RequestParameters>

            <Parameter Name="client_id" Value="MyClientID"/>

            <Parameter Name="response_type" Value="Assertion"/>

            <Parameter Name="state" Value="State"/>

            <Parameter Name="scope" Value="ApplicableScope"/>

            <Parameter Name="callbackURL"
            Value="https://thiscomponent.com/api"/>

        </RequestParameters>

    </Authorization>

    <TokenRequest Url="https://app.vssps.visualstudio.com/oauth2/token">

        <RequestParameters>

            <Parameter Name="client_assertion_type"
            Value="urn:ietf:params:oauth:client-assertion-type:jwt-bearer"/>
```

```
                <Parameter Name="client_assertion" Value="{ClientSecret}"/>

                <Parameter Name="grant_type" Value="urn:ietf:params:oauth:grant-
                type:jwt-bearer"/>

                <Parameter Name="assertion" Value="{AuthCode}"/>

                <Parameter Name="redirect_uri" Value="{CallBackURL}"/>

            </RequestParameters>

        </TokenRequest>

        <RefreshTokenRequest
        Url="https://app.vssps.visualstudio.com/oauth2/token">

            <RequestParameters>

                <Parameter Name="client_assertion_type"
                Value="urn:ietf:params:oauth:client-assertion-type:jwt-bearer"/>

                <Parameter Name="client_assertion" Value="{ClientSecret}"/>

                <Parameter Name="grant_type" Value="refresh_token"/>

                <Parameter Name="assertion" Value="{RefreshToken}"/>

                <Parameter Name="redirect_uri" Value="{CallBackURL}"/>

            </RequestParameters>

        </RefreshTokenRequest>

    </Authentication>
```

XML element / XML attribute	Value	Manda-tory
Authentication		
Name	The name of the authentication definition. The name is used to identify the authentication definition in other parts of the XML specification.	yes
Type	OAuth2	yes
Flow	AuthorizationCode	yes
ClientId	The client ID provided by the third-party vendor for access to the RESTful services. For security reasons you should specify at least the user password as a link to a server variable instead of using plain text. The server variable is stored encrypted in the server alias of the Alfabet Server. The use of server variables is described in the section Configuring Server Variables for Integration and Interoperability Solutions.	yes
ClientSecret	The client secret provided by the third-party vendor for access to the RESTful services.	yes

XML element / XML attribute	Value	Manda-tory
CallBackURL	The URL the request is redirected to after authentication to process the request. The callback URL is also referred to as redirect Uri and is provided by the third-party vendor for access to the RESTful services.	yes
Scope	If the third-party component limits access to functionality via Oauth2 via scopes, specify a scope defined for the client ID.	depending on third-party vendor
AuthCode	If the third-party component bases access on an authorization code provided to you, enter the code. This	depending on third-party vendor
Authorization		
URL	Enter the URL for authorization requests to the RESTful service of the third-party component.	yes
Authorization > RequestParameters > Parameter	Specify the RESTful service header fields which need to be included in the authorization call to the third-party component for requesting authorization parameters. Specify one XML element **Parameter** for each header field.	
Name and Value	Define the name and the value for the header field. Usually, the values for the header fields are identical to the values you defined in the XML element **Authentication**. You can take these values over by specifying the respective XML attribute name in curly brackets. For example, defining Value={ClientID} will set the value to the client id defined in the XML element **Authentication**.	yes
TokenRequest		
Url	Enter the URL for token requests to the RESTful service of the third-party component.	yes
TokenRequest > RequestParameters > Parameter	Specify the RESTful service header fields which need to be included in the token request to the third-party component. Specify one XML element **Parameter** for each header field.	

XML element / XML attribute	Value	Manda-tory
`Name` and `Value`	Define the name and the value for the header field. Usually, the values for the header fields are identical to the values you defined in the XML element **Authentication**. You can take these values over by specifying the respective XML attribute name in curly brackets. For example, defining `Value={ClientID}` will set the value to the client id defined in the XML element **Authentication**.	yes
RefreshToken-Request		
`URL`	Enter the URL for token refresh requests to the RESTful service of the third-party component.	yes
RefreshToken-Request > RequestParameters > Parameter	Specify the RESTful service header fields which need to be included in the token refresh request to the third-party component. Specify one XML element **Parameter** for each header field.	
`Name` and `Value`	Define the name and the value for the header field. Usually, the values for the header fields are identical to the values you defined in the XML element **Authentication**. You can take these values over by specifying the respective XML attribute name in curly brackets. For example, defining `Value={ClientID}` will set the value to the client id defined in the XML element **Authentication**.	yes

OAuth2 Authentication with Password Grant

Prerequisites: The authentication tokens of the third-party component must match the standard authentication definitions. You can view the standard at https://www.oauth.com.

⚠ According to the OAuth standard specification password grant is deprecated and should not be used for connections between components for security reasons. OAuth2 authentication with authorization is the preferred method.

OAuth2 authentication requests a new token in a configurable time interval.

Set the following parameters in the `Authentication` XML element:

```
<Authentication

    Name="OAuth2AuthenticationForMyApp"

    Type="OAuth2"

    Flow="PasswordGrant"

    ClientId="MyClientID"
```

```
            Scope="fullAccess"

            ClientSecret="MyClientSecret"

            CallBackURL="https://myComponent.com/api"

            AuthCode=""

        >

        <TokenRequest Url="https://myComponent.com/oauth2/token">

            <RequestParameters>

                <Parameter Name="client_assertion_type"
                Value="urn:ietf:params:oauth:client-assertion-type:jwt-bearer"/>

                <Parameter Name="client_assertion" Value="{ClientSecret}"/>

                <Parameter Name="grant_type" Value="urn:ietf:params:oauth:grant-
                type:jwt-bearer"/>

                <Parameter Name="assertion" Value="{AuthCode}"/>

                <Parameter Name="redirect_uri" Value="{CallBackURL}"/>

            </RequestParameters>

        </TokenRequest>

        <RefreshTokenRequest Url="https://myComponent.com/oauth2/token">

            <RequestParameters>

                <Parameter Name="client_assertion_type"
                Value="urn:ietf:params:oauth:client-assertion-type:jwt-bearer"/>

                <Parameter Name="client_assertion" Value="{ClientSecret}"/>

                <Parameter Name="grant_type" Value="refresh_token"/>

                <Parameter Name="assertion" Value="{RefreshToken}"/>

                <Parameter Name="redirect_uri" Value="{CallBackURL}"/>

            </RequestParameters>

        </RefreshTokenRequest>

    </Authentication>
```

XML element / XML attribute	Value	Manda-tory
Authentication		
Name	The name of the authentication definition. The name is used to identify the authentication definition in other parts of the XML specification.	yes
Type	OAuth2	yes
Flow	PasswordGrant	yes

XML element / XML attribute	Value	Manda-tory
ClientId	The client ID provided by the third-party vendor for access to the REST-ful services. For security reasons you should specify at least the user password as a link to a server variable instead of using plain text. The server variable is stored encrypted in the server alias of the Alfabet Server. The use of server variables is described in the section Configuring Server Variables for Integration and Interoperability Solutions.	yes
ClientSecret	The client secret provided by the third-party vendor for access to the RESTful services.	yes
CallBackURL	The URL the request is redirected to after authentication to process the request. The callback URL is also referred to as redirect Uri and is pro-vided by the third-party vendor for access to the RESTful services.	yes
Scope	If the third-party component limits access to functionality via Oauth2 via scopes, specify a scope defined for the client ID.	depending on third-party ven-dor
AuthCode	If the third-party component bases access on an authorization code pro-vided to you, enter the code. This	depending on third-party ven-dor
TokenRequest		
Url	Enter the URL for token requests to the RESTful service of the third-party component.	yes
TokenRequest > RequestParame-ters > Parameter	Specify the RESTful service header fields which need to be included in the token request to the third-party component. Specify one XML ele-ment **Parameter** for each header field. User name and password must be defined in the token request.	
Name and Value	Define the name and the value for the header field. If a value for a header field is identical to a value you defined in the XML element **Au-thentication**, you can take the value over by specifying the respective XML attribute name in curly brackets. For example, defining Value={ClientID} will set the value to the client id defined in the XML element **Authentication**.	yes

XML element / XML attribute	Value	Manda-tory
RefreshToken-Request		
URL	Enter the URL for token refresh requests to the RESTful service of the third-party component.	yes
RefreshToken-Request > Re-questParameters > Parameter	Specify the RESTful service header fields which need to be included in the token refresh request to the third-party component. Specify one XML element **Parameter** for each header field. User name and password must be defined in the token request.	
Name and Value	Define the name and the value for the header field. If a value for a header field is identical to a value you defined in the XML element **Au-thentication**, you can take the value over by specifying the respective XML attribute name in curly brackets. For example, defining Value={ClientID} will set the value to the client id defined in the XML element **Authentication**.	yes

Specifying Connection Parameters

Prerequisites: You have already specified the authentication with the third-party RESTful service in the XML object **GenericAPIIntegrationConfig**.

Set the following parameters in the GenericAPIIntegrationConnections XML element:

```
<GenericAPIIntegrationConnection

    Name="MyExternalInterface"

    InstanceURL="https://thiscomponent.com"

    Proxy="None"

    Authentication="OAuth2AuthenticationForMyApp"

    Timeout="60">

</GenericAPIIntegrationConnection>
```

XML element / XML attribute	Value	Man-datory
Authentication		
Name	The name of the connection definition. The name is used to identify the connection definition in other parts of the configuration of the generic API integration interface.	yes
Authentication	The name of an XML definition of an authentication specified in this XML element. For information about authentication definitions, see XXX.	yes
Proxy	The setting depends on the use of proxies to route the connection to the third-party components connected via generic API integration: • No proxy is used: Set the attribute to None. • One proxy is used for all connections and this connection is routed via a proxy: Do not set the attribute. • Multiple proxies are used and this connection is routed via a proxy: Enter the name of the proxy definition	no
Timeout	The HTTP request timeout in seconds. Only change the timeout value if problems with the default of 60 seconds occurs.	no

Specifying Routing of the Connetion via a Proxy

Preconditions: The connection to the third-party component is routed via a proxy.

This definition is optional.

The definition depends on the number of proxies used:

• [Using one Proxy for all Connections](#)

• [Using Multiple Proxies](#)

Using one Proxy for all Connections

The defined proxy will be used either by all or by a subset of the specified generic API integration connection definitions.

• Add an XML element **Proxy** to the XML object **GenericAPIIntegrationConfig**:

```
<GenericAPIIntegrationConfig>
```

```
        <Proxy url="124.24.78.90:5567" user="ConnectionAdmin"
        password="ConnectionAdminPW" domain="mycompany">

    </GenericAPIIntegrationConfig>
```

- Set the XML attributes for the XML object:

  - `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.

- If a connection shall not be routed via the proxy, add an XML attribute `Proxy="None"` to the XML element **GenericAPIIntegrationConnection** specifying the connection parameters. All other connections will be routed automatically via the defined proxy.

Using Multiple Proxies

The defined proxy will be used either by all or by a subset of the specified generic API integration connection definitions.

- Add an XML element **Proxy** to the XML object **GenericAPIIntegrationConfig**:

```
    <GenericAPIIntegrationConfig>

        <Proxy>

            <AdditionalProxies>

                <AdditionalProxy Name="Proxy1" url="" user="" password=""
                domain=""url="124.24.78.90:5567" user="ConnectionAdmin1"
                password="ConnectionAdminPW1" domain="mycompany" />

                <AdditionalProxy Name="Proxy2"url="124.24.78.95:4567"
                user="ConnectionAdmin2" password="ConnectionAdminPW2"
                domain="mycompany" />

            </AdditionalProxies>

    </GenericAPIIntegrationConfig>
```

- Set the XML attributes for the XML object:

  - `Name`: Enter a name for the proxy definition. The name is used to map the proxy definition to the generic API integration connection definitions.

  - `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.

- Add an XML attribute `Proxy` to each XML element **GenericAPIIntegrationConnection** specifying a connection. Set the XML attribute `Proxy` to the name of the relevant proxy definition or to `None` if the connection is not routed via a proxy.

Defining the OAS Based RESTful Service Endpoints

Prerequisite: The OAS compliant swagger definition file for the RESTful service of the third-party component is available on the local file system.

All calls to the RESTful service of a third-party component are defined in the same configuration object called **Resource Bundle**.

1) In Alfabet Expand, go to the **Reusable Elements** tab.

2) In the explorer, right-click the **Resource Bundles** node and select the **Create New Bundle** option.

3) Click the new **Bundle** child node of the **Resource Bundles** node.

4) Set the following attributes in the attribute window:

   - **Assistant**: Select `GenericAPIIntegration_OASAssistant`.

   - **Bundle Type**: Select `OpenAPISpecification`.

   - **Name**: Enter a name for the resource bundle. The name is used in the explorer and to identify the resource bundle in other parts of the generic API integration interface configuration.

   - **Description**: Optionally enter a description to explain the purpose of the configuration to other users.

   - **Group**: If you would like to group the configurations in folders beneath the **Resource Bundles** explorer node, enter a folder name to create a folder or to add the resource bundle to an existing folder with the specified name.

5) Right-click the resource bundle and select the **Create using OAS Assistant**. The resource bundle editor opens.

6) Define a name for the API connection definition in the **OAS Connection Name**. This name is used to identify the configuration in later configuration steps.

7) Click the arrow on the right of the **Import OAS File** field and select the OAS compliant swagger definition file of the third-party component for import.

8) Repeat the following steps in the given order for each connection to a RESTful end point of the third-party component.

   In this configuration step, the endpoints for either data import to or data export from the third-party component are selected and parameters are defined, if applicable. In the swagger definition, the following information is relevant for this configuration step:
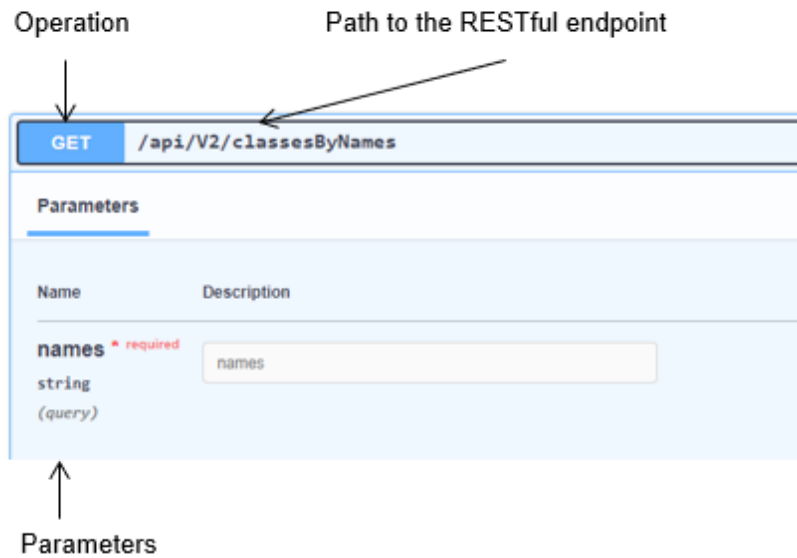
1) Select an endpoint in the **Path to REST API Endpoint** attribute.

2) Select the method in the **Operation** attribute. Use `POST` for import of data to Alfabet and `GET` for import of data to the third-party component. You can check in the swagger file if the method is available for the selected endpoint.

3) **Tag**: This field is currently not in use.

4) Click the **Save OAS API Connector** button.

Parameters which need to or may be added to the RESTful service call are listed after selecting the path and the operation. This is for informational purposes only.

5) Go to the **Map Response** tab.

In this tab, the mapping of data between Alfabet and the RESTful service call response third-party component is prepared. The actual operational execution of data import or export will be performed via ADIF. ADIF is using temporary tables as intermediate step to prepare Alfabet data for export or data coming in from the third-party component for import to Alfabet.

The third-party RESTful service is returning the data in the format that is defined in the swagger editor in the section **Response** of each endpoint description.

For a `GET` request the data will be added to a temporary table structure that can then be used in the ADIF definition to map import data to standard Alfabet object class database tables.

For `POST` requests, the return value can be useful to re-import information to Alfabet. For example, if a new object is created in the third-party component for each Alfabet object of a defined object class and the return value includes the unique ID of the objects in the third-party component, the ID can be mapped to a custom property of the Alfabet object class to store the information which object in the third-party component corresponds to which object in Alfabet.

6) Provide a parameter mapping for all endpoints that you defined as an OAS API connector:

  1) Click the **New** button.

  2) Select the operation in the **Operation** field.

  3) Enter a name for the temporary table that will be created for the ADIF scheme to store the incoming data from the third-party component or the Alfabet data gathered for export to the third-party component.

  4) Set a checkmark in the column **Include** in the table listing all the body fields returned for the endpoint by the third-party component. If many fields are returned, you can use the Search options to search for fields and the **Show All Included** button to return to all results.

     The table shows the fields with the correct data type extracted from the swagger specification.

Defining the Connection to the Third-Party Component

Prerequisites: A connection parameter definition and an authentication definition for connection to the third-party component is available in the XML object **GenericAPIIntegrationConfig** and a resource bundle is defined in Alfabet Expand.

Specify the connection to the third-party component in the **Integration Solutions Configuration** functionality in the Alfabet user interface.

1) In the explorer, click the **Generic API Integration Database Connection** node.

2) In the toolbar of the view that opens, click **New** > **Create Generic API Integration Database Connection**.

3) Specify the connection parameters:

   Basic Data tab:

   - **Name**: Define a name for the connection. The name is used to identify the connection in the following configuration steps.

   - **Description**: Optionally enter a description of the purpose of this connection.

   - **Release Status**: If your company supports release status definitions for API connections, you can select a release status to specify the approval state of the connection definition.

   Authorized Access tab: Define the authorized user responsible for maintaining the connection definition and the user groups whose members shall have edit rights for the connection definition.

   Connection tab:

   - **Generic API Connection**: Select the name of the relevant generic API integration connection definition in the XML object **GenericAPIIntegrationConfig**.

   - **OAS Bundle**: Select the name of the relevant resource bundle.

   - **OAS Item**: Select the name of the OAS connection which you defined in the resource bundle's assistant in the attribute **OAS Connection Name**.

   If you are using OAUth2 authentication for the connection to the third-party component, go to the **OAuth2 Configuration** tab and enter the OAUth authorization code provided by the third-party vendor in the **authorization Code** field and click **Get Access Token**. If the connection succeeds, the access token will be displayed behind the text **Access Token**. Otherwise, an error message is displayed in the field **Response**.

4) Click **Next** if you would like to define fixed parameters handed over to the third-party's RESTful service during each call. If all mandatory parameters are defined in the connection definition, ADIF schemes configured to use this connection definition will not require a parameter mapping for the call. If you do not want to specify fixed parameters, you can directly exit the editor.

5) In the **Connection Details** view, click **New** > **Create New Connection Property** to define a parameter to be handed over to the connection. A wizard opens. Define the name, data type and value for the parameter. If parameters are handed over in a hierarchy, you can define the subordinate connection properties in the next step of the wizard.

Defining a Report Category for the Generic API Integration Interface

The ADIF import and export schemes read data from configured reports which need to be assigned to the correct use case category for generic API integration.

Use case categories are defined in Alfabet Expand.

1) Go to the **Presentation** tab and expand the **XML Objects** node.

2) Double-click the XML object **UseCaseCategories** and, if not yet specified, add a new XML element **UseCaseInfo** to the XML root element **UseCaseCategories** with the following settings:

```
<UseCaseInfo UseCase="GenericOASIntegration">
```

```
<ScopeInfo Scope="Report" Categories="CategoryName" />

</UseCaseInfo>
```

The category name can be any string. Once defined, it can be selected in the drop-down list of the **Category** attribute of configured reports.

3)   Click the **Save**   button in the toolbar to save the definition.

Defining Import of Data to Alfabet

Prerequisites:

- A connection parameter definition and an authentication definition for connection to the third-party component is available in the XML object **GenericAPIIntegrationConfig**,

- A resource bundle is defined in Alfabet Expand.

- A **Generic API Integration Database Connection** is defined in the Alfabet user interface.

- A use case category is defined for the use case `GenericAPIIntegration` in the XML object **UseCaseCategories**.

Data import from a third-party component may require that data is provided to the third-party's RESTful endpoint in a defined format either as parameter in the header or in the body of the request in a JSON object. The ADIF import scheme will read the data from a configured tabular report. The complete import configuration therefore includes both the definition of the configured report and of the ADIF import scheme.

- [Defining a Configured Report to Set Import Call Parameters](#)

- [Defining the ADIF Import Scheme](#)

Defining a Configured Report to Set Import Call Parameters

Configuration of the configured report is done in Alfabet Expand. For information about how to create a tabular configured report, see the chapter *Configuring Reports* of the reference manual *Configuring Alfabet with Alfabet Expand*.

Note the following required settings for the configured report:

- The **Category** attribute must be set to the use case category for the use case `GenericAPIIntegration`.

- The report result is a simple table which includes one column for each parameters to be sent in the header of the call and one column for each field to be set in the JSON body of the call. the report can include multipe columns. Data will then be requested from the third-party components for each combination of parameters in a row.

- The columns of the report dataset can have a name differing from the parameter names and JSON body field names. Mapping of the report data to the correct parameters and JSON fields is done in the ADIF configuration.

For example, issue in a GITHub® repository shall be imported to Alfabet. The swagger file for the endpoint informs about two mandatory parameters to identify the owner and the repository the data shall be created in.

The report for the import of issues returns two rows for the same owner and different repositories. The owner and repository specification are the same for both rows.

	OWNER	REPO	ORG
1	AlfabetPLM	TradeNet6.3.1	AlfabetPLM
2	AlfabetPLM	CRMCSS3.2	AlfabetPLM

Defining the ADIF Import Scheme

Definition of the ADIF import scheme is done in Alfabet Expand.

1) Go to the **ADIF** tab.

2) Right-click the root node and select **Create Import Scheme**.

3) Set at least the following attributes of the new ADIF import scheme:

- **Name**: Define a unique name for the ADIF import scheme.

- **Assistant**: Select `GenericAPIIntegration_Assistant`.

- **Category**: If you would like to execute the ADIF import via the **Job Scheduler** functionality, you must select the correct category for the user case `JobScheduler`. For an overview of the configuration required to execute an ADIF scheme via the **Job Scheduler** functionality, see *Activating the Job Schedule Functionality* in the reference manual *Configuring Alfabet with Alfabet Expand*.

  For a description of the other optional parameters of the ADIF import scheme, see *Configuring Data Export with ADIF* in the reference manual *Alfabet Data Integration Framework*.

4) Right-click the ADIF import scheme in the explorer and select the **Create ADIF Scheme Details Using the Generic API Integration Assistant**.

5) In the assistant, define the basic attributes:

- **Name**: Define the name of the ADIF import entry that will be created within the ADIF import scheme by the assistant. this ADIF import entry will trigger the generation of the temporary table the imported data will be stored in during the import process.

- **Generic API Integration Database Connection**: Select the Generic API integration database connection which you defined in the configuration step Defining the Connection to the Third-Party Component.

6) Click the **New** > **Add Resource Filter** in the toolbar in the attribute field. A new editor opens.

7) Define the following attributes in the given order:

- **Operations** attribute, select a `GET` endpoint from the third-party RESTful endpoints configured for the Generic API integration database connection.

- **Primary Report**: Select the report providing the parameter values to be sent to the third-party's RESTful service endpoint for the import call. The report was created in the configuration step Defining a Configured Report to Set Import Call Parameters.

- **Request Mappings**: The table lists all parameters and JSON body fields of the selected endpoint. In the **Report Column** column, select the column of the dataset in the configured report providing the data for the parameter or JSON body field. You need to map at least the mandatory parameters and JSON body fields.

  > For the example given in the section Defining a Configured Report to Set Import Call Parameters the parameter mapping would be:

8) Click **OK** to close the mapping definition.

9) Click **OK** to close the assistant.

10) In the **ADIF** explorer, right-click the ADIF import scheme and select the **Reread from Database** option to view the changes made in the ADIF scheme by the assistant.

11) Configure the ADIF scheme to change the object data in the Alfabet database according to demand. The required configuration as well as available test and execution methods are described in the reference manual *Alfabet Data Integration Framework*.

Defining Export of Data from Alfabet

Prerequisites:

- A connection parameter definition and an authentication definition for connection to the third-party component is available in the XML object **GenericAPIIntegrationConfig**,

- A resource bundle is defined in Alfabet Expand.

- A **Generic API Integration Database Connection** is defined in the Alfabet user interface.

- A use case category is defined for the use case `GenericAPIIntegration` in the XML object **UseCaseCategories**.

Data export requires that data is provided to the third-party's RESTful endpoint in a defined format either as parameter in the header or in the body of the request in a JSON object. The ADIF export scheme will read the data from a configured tabular report. The complete export configuration therefore includes both the definition of the configured report and of the ADIF export scheme.

- Defining a Configured Report providing Export Parameters

- Defining the ADIF Export Scheme

Defining a Configured Report providing Export Parameters

Configuration of the configured report is done in Alfabet Expand. For information about how to create a tabular configured report, see the chapter *Configuring Reports* of the reference manual *Configuring Alfabet with Alfabet Expand*.

Note the following required settings for the configured report:

- The **Category** attribute must be set to the use case category for the use case `GenericAPIIntegration`.

- The report result is a simple table which includes both the parameters for the report call and the JSON data to be set in the body of the call. The parameters must be identical for all rows of the report.

- The columns of the report dataset can have a name differing from the parameter names and JSON body field names. Mapping of the report data to the correct parameters and JSON fields is done in the ADIF configuration.

For example, projects defined in Alfabet shall be created in a GITHub® repository. The swagger file for the connection informs about two mandatory parameters to identify the repository the data shall be created in and a JSON body containing the data for the project to be created.

The report for the Export of two projects returns two rows. One for each project. The owner and repository specification are the same for both rows.

NAME	DESCRIPTION
Consolidate Trading Applicatio...	There are currently much too many applications in use which support
Retire GL Applications	This project is focused on retiring the obsolete applications "Bysis GL"

Defining the ADIF Export Scheme

Definition of the ADIF export scheme is done in Alfabet Expand.

1) Go to the **ADIF** tab.

2) Right-click the root node and select **Create Export Scheme**.

3) Set at least the following attributes of the new ADIF export scheme:

- **Name**: Define a unique name for the ADIF Export scheme.

- **Assistant**: Select `GenericAPIIntegration_Assistant`.

- **Category**: If you would like to execute the ADIF export via the **Job Scheduler** functionality, you must select the correct category for the user case `JobScheduler`. For an overview of the configuration required to execute an ADIF scheme via the **Job Scheduler** functionality, see *Activating the Job Schedule Functionality* in the reference manual *Configuring Alfabet with Alfabet Expand*.

  For a description of the other optional parameters of the ADIF export scheme, see *Configuring Data Export with ADIF* in the reference manual *Alfabet Data Integration Framework*.

4) Right-click the ADIF export scheme in the explorer and select the **Create ADIF Scheme Details Using the Generic API Integration Assistant**.

5) In the assistant, define the basic attributes:

- **Name**: Define the name of the ADIF export entry that will be created within the ADIF export scheme by the assistant.

- **Generic API Integration Database Connection**: Select the Generic API integration database connection which you defined in the configuration step Defining the Connection to the Third-Party Component.

6) Click the **New** > **Add Resource Filter** in the toolbar in the attribute field. A new editor opens.

7) Define the following attributes in the given order:

- **Operations** attribute, select a `POST` endpoint from the third-party RESTful endpoints configured for the Generic API integration database connection.

- **Primary Report**: Select the report providing the data for the export which you created in the configuration step Defining a Configured Report providing Export Parameters.

- **Request Mappings**: The table lists all parameters and JSON body fields of the selected endpoint. In the **Report Column** column, select the column of the dataset in the configured report providing the data for the parameter or JSON body field. You need to map at least the mandatory parameters and JSON body fields.

👁 For the example given in the section Defining a Configured Report providing Export Parameters the parameter mapping would be:

	Parameter Caption	Parameter Type	Mandatory	Default Value	
1	owner	String	✔		Organization
2	repo	String	✔		Repository
3	User-Agent	String		Alfabet	*Select*
4 ▼	Content	object			
5	name	String	✔		Project
6	**body**	**String**			Description

8) Optionally, you can map attributes of the response of the RESTful service call to object class properties of the object class the configured report returns data for. In preparation of this step, it might be required to create a custom object class property for the object class to store the data. For information about creating object class properties, see *Configuring Custom Properties for Protected or Public Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand*. If response values are mapped in the ADIF assistant, the respective object class properties will be automatically filled with the returned values on execution of the export.

To map response values to object class properties in Alfabet:

1) In the **Response Class Mapping** field, click **New** > **Add**.

2) Select the object class in the **Class to Map the Response** field.

3) The return values mapped to the generic API integration connection via the resource bundle defined in the configuration step Defining the OAS Based RESTful Service Endpoints are displayed in a table. In the **Class Property** column, select the object class properties the return values shall be mapped to.

4) Click **OK** to save the mapping.

👁 For the example given in the section Defining a Configured Report providing Export Parameters the object class Project has two custom object class properties `SC_GitHub_ID` and `SC_GitHub_Name` to store the id and name of the GitHub Project which is created on basis of the Alfabet project during the call:

Map Response to Class (GAI_Export_ResponseMap)

Class to Map the Response*

Project ▼

	Response Parameter	Parameter Type	Class Property	Cla
1	id	Integer	SC_GitHub_ID ▼	Inte
2	name	String	SC_GitHub_Name ▼	Stri

The mapping to the object class properties can also be used to restrict the projects listed in the configured reports to projects which do not have these properties set and are therefore not yet created in GitHub.

5) Click **OK** to close the mapping definition.

6) Click **OK** to close the assistant.

7) In the **ADIF** explorer, right-click the ADIF export scheme and select the **Reread from Database** option to view the changes made in the ADIF scheme by the assistant. No other configurations are required in the ADIF scheme. You can now test and execute the ADIF scheme. Available test and execution methods are described in the reference manual *Alfabet Data Integration Framework*.

Chapter 4:   Configuring Interoperability with Microsoft Teams

Communication about Alfabet objects can be enabled via integration with Microsoft® Teams®. The integration with Microsoft Teams allows communication about Alfabet objects such as applications, business capabilities, or strategies, for example, to concur synchronously in Microsoft Teams and in Alfabet. If integration with Microsoft Teams is configured, the buttons in the **Collaboration Panel** and **My Collaboration Topics** functionality will display icons for Microsoft Teams-based collaboration.

The Collaboration capability in Alfabet has been revised and communication about Alfabet objects can be enabled via integration with Microsoft Teams. The integration with Microsoft Teams offers a new collaboration platform that allows communication about Alfabet objects such as applications, business capabilities, or strategies, for example, to concur synchronously in Microsoft Teams and in Alfabet. If integration with Microsoft Teams is configured, the buttons in the **Collaboration Panel** and **My Collaboration Topics** functionality will display icons for Microsoft Teams-based collaboration.

Depending on the configuration, an Alfabet user can open a predefined Microsoft Teams channel or create a new Microsoft Teams channel for an Alfabet object in the **Collaboration Panel** in the object profile of the relevant object and specify the initial set of users that may collaborate about the object. Additional users with a Microsoft Teams ID can be added to the discussion at any time with the Microsoft Teams client. Posts about the object that are made in Alfabet will be displayed in Microsoft Teams and likewise posts made in Microsoft Teams about the Alfabet object will be displayed in the **Collaboration Panel** in the object profile of the relevant object in. In this way, users can reply to each other's posts from both Microsoft Teams and the Alfabet user interface.

Furthermore, the online status of an Alfabet user with a Microsoft Teams ID will be displayed next to the user's name in the **Attributes** section of previews, object profiles, and object cockpits. Users can click the online status icon and open the Microsoft Teams chat and call the person. When configured, the Microsoft Teams configuration will supersede any existing interoperability with a Skype © for Business Server.

For information about working with the **Collaboration Panel** and **My Collaboration Topics** functionality if the interoperability with Microsoft Teams has been implemented, see *Communicating with Your Colleagues via the Alfabet Internal Collaboration Functionality* in the reference manual *Getting Started with Alfabet*.

The following configuration is required to implement interoperability with Microsoft Teams:

- Registering an App with Microsoft Azure

- Configuring Access to Microsoft Teams in Alfabet

- Enabling Snapshots of Alfabet Views to be Visible in Microsoft Teams

Registering an App with Microsoft Azure

An app must be registered with Microsoft Azure to allow access of Alfabet to Microsoft Teams. For information about how to register an app, see the Microsoft help available a t https://docs.microsoft.com/en-us/azure/active-directory/develop/quickstart-register-app

After having registered the app, the app must be configured in Microsoft Azure as follows:

1) Click the app in the list of apps to open the app details view.

2) Go to **Certificates & Secrets** and create a new secret. Store the secret for later use in Alfabet configuration and the secret will later not be visible any longer in Microsoft Azure.

3) Go to **API Permissions** and select **Microsoft Graph**.

4) Assign at least the following permissions to the app. Most permissions can either be granted as Application Permissions or Delegated Permissions. By default, Alfabet expects these permissions to be

Delegated Permissions. Granting Application Permissions requires a configuration on Alfabet side. This is described below in the section Configuring Access to Microsoft Teams in Alfabet. In exceptional cases, the use of either Application Permissions or Delegated Permissions is mandatory. These permissions are listed separately in the following.

Application Permissions or Delegated Permissions:

- `Team.Create`

- `Team.ReadBasic.All`

- `Channel.ReadBasic.All`

- `Channel.Create`

- `TeamsTab.Create`

- `Channel.ReadBasic.All`

- `ChannelMember.Read.All`

- `ChannelMember.ReadWrite.All`

- `TeamMember.Read.All`

- `Team.ReadBaisc.All`

- `TeamMember.ReadWrite.All`

- `TeamMember.Read.All`

- `Calendars.Read`

- `Calendars.ReadWrite`

- `Files.Read.All`

- `User.Read.All`

> ⓘ If your company's security policy does not allow to set this permission, you can alternatively read the user GUIDs for all users via other mechanisms from Microsoft Teams and write them in the object class property `TeamsUserInternalId` of the object class `Person`. In the next configuration step described in the following, the XML attribute `UserReadAllEnabled` in the XML object **MicrosoftTeamsIntegrationConfig** must then be set to `false` to disable direct reading of the value from Microsoft Teams and switch to reading the value from the Alfabet database.

Delegated Permissions:

- `Calendars.Read.Shared`

- `ChannelMessage.Read.All`

- `ChannelMessage.Send`

- `Presence.Read.All`

- `User.Read`

- `Offline_access`

Application Permissions:

- `Calendars.Read` (This permissions is required for the synchronization of Microsoft Teams meetings via ADIF jobs. This functionality cannot work with Delegated Permissions. Independent from that, the synchronization of Microsoft Teams meetings via the user interface uses `Calendars.Read` Delegated Permissions by default and can optionally be changed to use Application Permissions.)

5) Go to **Authentication** and add a **Web** platform with a **Redirect URI** pointing back to your Alfabet installation. The **Redirect URI** must end with `/Home.aspx`.

> For example, if the Alfabet server URL is `https://CompanyHost/Alfabet` the **Redirect URI** must be defined as `https://CompanyHost/Alfabet/Home.aspx`.

6) In the **Implicit grant** section, set the checkmarks for the **Access tokens** and **ID tokens** options.

For the configuration on Alfabet side you will need the information about the secret and the **Essentials** displayed in the **Overview** page.

Configuring Access to Microsoft Teams in Alfabet

Open the configuration tool Alfabet Expand and do the following:

1) In the **Presentations** tab, expand the nodes **XML Objects** > `IntegrationSolutions`.

2) Right-click the **MicrosoftTeamsIntegrationConfig** node and select **Edit XML**.

3) Enter the following XML code or alter the existing code. The meaning of the XML attributes and XML elements is described below the code example:

> ℹ️ Instead of directly defining sensible information like Azure access data directly in the XML object, you can use server variables for the definition of XML attributes. Server variables are substituted with values defined for each variable in the server alias of the Alfabet Web Application. Server variables are stored encrypted in the server alias configuration. They can be set via a command line tool, enabling an Azure administrator to set the values without providing the information to the administrator of the Alfabet components. For more information about server variables, see Configuring Server Variables for Integration and Interoperability Solutions.

```
<MicrosoftTeamsIntegrationConfig IsActive="true"
AllowedClasses="Application,Domain,Project" MaxMessagesReadInRequest="50"
MaxRepliesReadInRequest="7" MaxNumPagesReadInRequest="1" >

    <Proxy url="" user="" psw="" domain=""/>

    <MicrosoftTeamsIntegrationInfo

        Name="TeamsConfig"

        IsActive="true"

        TeamsCollaborationEnabled="true"

        TeamsPresenceEnabled="true"

        PresenceRefreshSeconds="30"

        Timeout="90"

        AdminUserId="$TeamsAdminUser"
```

```
          TenantID="$TeamsTenantID"

          ClientID="$TeamsClientID"

          ClientSecret="$TeamsClientSecret">

          <SelectableTeam ID="$TeamsDemoTeamGUID" />

          <ChannelSettings CreateTab="true"/>

          <GroupSettings CanCreateGroup="false"/>

          <CalendarSettings RemoveExpiredSyncInstances="true"
          NumMonthsSync="12"/>

          <EndpointSettings Name="CreateTeam" PermissionType="Application"
          />

      </MicrosoftTeamsIntegrationInfo>

  </MicrosoftTeamsIntegrationConfig>
```

XML attributes of the XML element **MicrosoftTeamsIntegrationConfig**:

- `IsActive`: Set to `true` to activate Microsoft Teams integration.

- `AllowedClasses`: Enter a comma-separated list of Alfabet object classes for that a discussion about object via Microsoft Teams shall be available.

- `MaxMessagesReadInRequest`: Enter the maximum number of messages in a Microsoft Teams Team Channel that will be displayed directly in Alfabet in the **Collaboration** capability. If more messages are available in the Microsoft Teams Team Channel, only the most recent messages up to the defined maximum will be displayed.

- `MaxRepliesReadInRequest`: Enter the maximum number of replies to a message that shall be displayed directly in Alfabet in the **Collaboration** capability. If more replies are available for a message, only the most recent replies up to the defined maximum will be displayed.

- `MaxNumPagesReadInRequest`: Enter an integer between 1 and 10 to define the maximum number of pages read into the selector that opens if a user uses the **Add Web Link Based on MS Teams File Link** option in the **Attachments** page view of an object assigned to a Microsoft Teams team channel. The selector shows the folders and files available in the **Files** tab of the assigned Microsoft Teams team channel. The default value for this XML attribute is 1.

XML attributes of the XML element **MicrosoftTeamsIntegrationInfo**:

- `Name`: Enter an Alfabet internal name for the connection to Microsoft Teams. The name must not include special characters or whitespaces.

- `IsActive`: Set to `true` to activate the connection. Multiple XML elements `MicrosoftTeamsIntegrationInfo` can be added to the XML element `MicrosoftTeamsIntegrationConfig`, For example, to define the connections to different instances of Microsoft Teams in a test environment and in a production environment. But only one of the XML elements `MicrosoftTeamsIntegrationInfo` can be set to activated at a time.

- `IsMultitenant`: Apps registered with Microsoft Azure can either be single-tenant and accept only sign-ins from users in the Azure Active Directory (Azure AD) tenant the app is created for, or multi-tenant and accept sign-ins from users in any Azure AD tenant. If you have configured your registered Alfabet app to be multi-tenant, you must set this XML attribute to `true`.

- `TeamsCollaborationEnabled`: Set to `true` to enable integration of Microsoft Teams for the **Collaboration** capability. If set to `false`, the Alfabet internal **Collaboration** capability will be available to the users.

- `TeamsPresenceEnabled` Set to `true` to enable display of the Microsoft Teams availability status in Alfabet everywhere the user information is displayed. The availability status information is displayed in the same way as the Skype® availability status that is displayed if integration with Skype is implemented. It is therefore not possible to integrate with both Skype and Microsoft Teams. If both integration solutions are configured and activated, the Skype integration will be ignored.

- `PresenceRefreshSeconds`: Define the minimum number of seconds between refresh of the Microsoft Teams user availability status information on the Alfabet user interface.

- `UserReadAllEnabled`: This attribute is by default set to `true` and the Microsoft internal user GUID is read from Microsoft Teams via the Microsoft Graph API. This requires the `User.Read.All` application permissions for the registered Alfabet app. The user GUID is required to read the user's status and to add users to a Teams Channel. Only set the attribute to `false` if you have not granted the `User.Read.All` access permissions for the registered app. You must then read the user GUIDs for all users via other mechanisms from Microsoft Teams and write them in the object class property `TeamsUserInternalId` of the object class `Person`.

- `Timeout`: Enter the maximum time in seconds that the Alfabet Web Application will wait for an answer when sending a request to the Microsoft Graph engine handling the incoming requests on Microsoft side.

- `AdminUserId`: Enter the Microsoft Teams user ID of a user with administrative rights in Microsoft Teams. This user will be the owner of all newly created Microsoft Teams Teams.

- `TenantID`: Enter the Microsoft Azure Tenant ID (also called Directory ID) of your registered application.

  > ℹ️ In Microsoft Azure, you can find the Tenant ID in the **Overview** tab for your registered application.

- `ClientID`: Enter the Microsoft Azure Client ID (also called Application ID) of your registered application.

  > ℹ️ In Microsoft Azure, you can find the Client ID in the **Overview** tab for your registered application.

- `ClientSecret`: Enter the Microsoft Azure Client Secret of your registered application.

  > ℹ️ The generation of the client secret is part of the configuration of the registered application in Microsoft Azure described in the section Registering an App with Microsoft Azure.

XML attributes of the XML element **SelectableTeam**: You can predefine Teams in Microsoft Teams that shall be used for the communications about objects in Alfabet. The user starting a new conversation in the collaboration panel on the Alfabet user interface can select one of the Teams via a drop-down list. In addition, the user might see an option to create new Teams. This option can be activated or deactivated with the XML element `GroupSettings`. For each Team that a user shall be able to start a conversation about an Alfabet object in, add an XML element **SelectableTeam** to the XML element **MicrosoftTeamsIntegrationInfo** and define the following XML attribute for it:

- `ID`: Enter the ID of the Team.

  > ℹ️ To read the ID of the Team in Microsoft Teams, click on the ellipsis to the right of the Team name and select **Get link to team**. Copy the link from the window that opens

and past it into a text editor. In the link, the ID of the Team is the string between `groupId=` and `&tenantId`.

XML attributes of the XML element **GroupSettings**

- `CanCreateGroup`: Set to `true` if users shall be able to create new Microsoft Teams Teams when working in the collaboration panel on the Alfabet user interface. Set to `false` if the user shall only be able to select a Team defined in a `SelectableTeam` XML element.

XML attributes of the XML element **ChannelSettings**:

- `CreateTab`: Set to `true` if for each newly created Team Channel a tab with the name of the object shall also be generated in the channel.
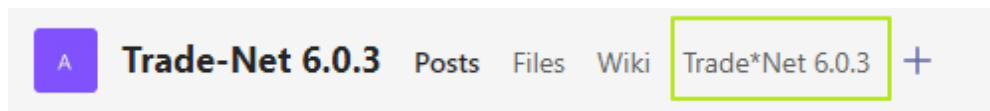
Users opening the tab will see the view about the object in Alfabet with up-to-date information.

> (i) Please note that access for anonymous users must be granted by selecting the **Allow Anonymous User** checkbox in the **Server Settings** > **Security** tab of the server alias configuration of the Alfabet Web Application.

XML attributes of the XML element **CalendarSettings**:

- `RemoveExpiredSyncInstances`: Set to `true` to remove the information about already performed synchronization activity if this activity was performed prior to the current synchronization period during synchronization of Microsoft teams meetings imported to Alfabet with the corresponding calendar in Microsoft Teams via the standard private ADIF job `SynchronizeCalendarEvents`. The stored information about synchronization activity is used to perform a delta synchronization if a synchronization was already done for a time period lying in the current synchronization period.

- `NumMonthsSync`: Enter an integer between 1 and 24 to specify how many months in the past starting from the current date will be scanned in Microsoft Teams calendars to synchronize Microsoft Teams meetings imported to Alfabet with the corresponding calendar in Microsoft Teams via the standard private ADIF job `SynchronizeCalendarEvents`. the default value is 12. Please note that this value is only used as default value and can be overwritten during scheduling of the ADIF job.

Some functions of the Alfabet app can use Microsoft Azure Application Permissions or Delegated Permissions. By default, Delegated Permissions are expected at Microsoft Azure side. If an access permission is granted as Application Permission instead, an XML element **EndpointSettings** must be added for each function of the Alfabet app using the permission with the following attributes:

> (i) In example code for the XML object **MicrosoftTeamsIntegrationConfig**, a comment section is available with XML elements **EndpointSettings** for the relevant access permissions. To change the use of Microsoft Graph access permission to Application Permission for a permission type, copy the relevant XML element **EndpointSettings** and paste it in the XML element **MicrosoftTeamsIntegrationInfo** as a child element.

- `Name`: The name of the function of the Alfabet app based on the access permission. The following functions rely on a Microsoft Azure access permission:

Name of Alfabet function	Scope = Microsoft Azure access permission
ReadUser	`user.readbasic.all` (Delegated) `user.read.all` (Application)
ReadUsers	`user.readbasic.all` (Delegated) `user.read.all` (Application)
CreateTeam	team.create
GetTeamMembers	TeamMember.Read.All
AddTeamMember	TeamMember.ReadWrite.All
GetTeam	Team.ReadBasic.All
GetChannels	Channel.ReadBasic.All
GetChannel	Channel.ReadBasic.All
GetLinkToChannel	Channel.ReadBasic.All
CreateChannel	Channel.Create
CreateChannelTab	TeamsTab.Create
GetChannelMembers	ChannelMember.Read.All
AddChannelMember	ChannelMember.ReadWrite.All
ListChannelMessages	ChannelMessage.Read.All
GetChannelMessage	ChannelMessage.Read.All
GetChannelMessageReplies	ChannelMessage.Read.All

Name of Alfabet function	Scope = Microsoft Azure access permission
GetChannelFilesFolder	Files.Read.All
GetDriveContents	Files.Read.All
MeCalendarEventsRead	Calendars.Read
MeCalendarView	Calendars.Read
MeCalendarEventsCreate	Calendars.ReadWrite
MeCalendarEventsUpdate	Calendars.ReadWrite
MeCalendarEventsChangeState	Calendars.ReadWrite

- `Scope`: This attribute is optional and only relevant if the access permission in Microsoft Azure is different for Application Permissions and Delegated Permissions. The `Scope` attribute must then be set to the name of the access permission in Microsoft Azure. This is currently only relevant for the `ReadUser` and `ReadUsers` functions of the Alfabet app. The required access permission is `user.read.all` for Application Permissions and `user.readbasic.all` for Delegated Permissions.

- `PermissionType`: Set to `Application` to use Application Permissions, to `Delegated` to use Delegated Permissions, or to `None` if you have not granted the permissions in Microsoft Azure. The default value is `Delegated`.

Enabling Snapshots of Alfabet Views to be Visible in Microsoft Teams

Snapshots of the view the user is working in when opening the collaboration panel can be included into conversations and replies. In Microsoft Teams, snapshots are included into Team Channels including a link that opens the view the snapshot has been taken from in Alfabet.

This feature is executed via the Alfabet RESTful services. Authentication for the RESTful service call will be performed using the user configured for execution of self-reflective events. RESTful service permission **Has View Snapshot Access** must be activated for both the Alfabet Web Application and the user for self-reflective events.

All configuration steps required for activation of the RESTful services as well as the configuration of the user for self-reflective events is described in detail in the reference manual *Alfabet RESTful API*.

Chapter 5: Configuring Interoperability with Confluence

Snapshots of Alfabet views can be integrated into Confluence™ pages. Snapshots of Alfabet views are included without the masthead and are updated in regular intervals with up-to-date information from Alfabet.

Users can click on a snapshot in Confluence to open the respective view in the Alfabet user interface. Access permissions to Alfabet for Confluence users are the same as for users opening Alfabet express views.

Users can integrate snapshots into Confluence once the setup of the integration is done.

> ⚠ This feature is only available for on premise Confluence servers and is not available for Confluence Cloud.

The following information is available:

- Setting up Confluence Integration

- Remove Alfabet Snapshots from Confluence

    - Activating the Module for Manual Snapshot Removal

    - Running a Snapshot Deletion Job

- Adding Alfabet Views with a Link to Alfabet to Confluence Pages

Setting up Confluence Integration

The Confluence integration is based on an Alfabet Connector app that must be implemented in Confluence. When a user enters an express view link to an Alfabet view in a Confluence page, the Alfabet Connector app requests the snapshot from Alfabet via the Alfabet RESTful services. The snapshot of the Alfabet view is then stored in Confluence and displayed to users instead of the link text. Users can click the snapshot image to navigate to Alfabet via the express view link. The Alfabet Connector app will request a new snapshot for each existing express view links in Confluence pages via the Alfabet RESTful services once a day. This ensures that users will see up-to-date information directly in Confluence.

The following prerequisites are required to set up Confluence integration:

- The Alfabet RESTful services must be enabled and an Alfabet user must be granted access permissions for the RESTful services as described in the reference manual *Alfabet RESTful API*. Please note that for both the user as well as the Alfabet Web Application the RESTful service endpoint access permission **Has View Snapshot Access** must be granted.

    The information about the user name and password must be provided to Confluence during the setup.

- The access permissions for the access to Alfabet from express views is also used for opening views from Confluence. The settings of these access permissions should be checked for suitability for the Confluence integration. For information about the configuration of access permissions for express views, see *Configuring the Setup To Open the Alfabet Interface via Links in Email Notifications* in the reference manual *System Administration*.

The setup of the connection is done in Confluence:

> The following is a short overview of a configuration done in a   third-party product. For detailed information about managing apps in Confluence, see the Confluence documentation at https://support.atlassian.com/confluence-cloud/docs/manage-your-apps/.

1) A JAR file is delivered with the Alfabet release in the folder **ConfluencePlugIn** in the installation directory of the Alfabet components. Store the JAR file in a location accessible when working in Confluence.

2) Log in to Confluence.

3) In the Confluence header, click on [icon] > **Manage Apps**.

4) Click the **Upload App** link at the top right side of the **Manage Apps** view.

5) Click the **Browse** button and select the JAR file delivered with Alfabet in the file explorer.

6) Click the **Upload** button. The Alfabet Connector app will be added to the list of available apps.

7) Expand the section about the Alfabet Connector app in the list and click the **Configure** button. A new view opens.

8) Enter the following information:

   - **Alfabet REST API Url**: Enter the URL of the Alfabet Web Application.

   - **Alfabet User Name**: The name of the Alfabet user that shall be used for authentication of calls to the Alfabet RESTful API.

   - **Alfabet User API Password**: The API password of the Alfabet user that shall be used for authentication of calls to the Alfabet RESTful API.

   - **Hours to snapshot renewal**: Renewal of Alfabet snapshots in confluence is done once a day at a full hour. Enter the full hour the renewal shall take place in the 24-hour time notation. Allowed values are integers from 0 (midnight) to 23.

   - **Open Alfabet in New Tab**: Select the checkbox to open Alfabet in a new tab when a user clicks a link to Alfabet in Confluence.

   - **Snapshot sizes**: You can define multiple sizes for display of Alfabet snapshots in Confluence. Users adding a link to a page in Confluence can select one of the defined sizes. To define a screenshot size, click **Define another size** and enter the width and the height in pixels in the respective fields of the editor that opens. You can remove existing size definitions by clicking on the **x** in the column **Delete** in the row for the size.

     > It is recommended to provide different size definitions for small page view snapshots and large snapshots like For example, object cockpit snapshots and to test the sizes with example links to avoid scrollbars in the display of snapshots.

Remove Alfabet Snapshots from Confluence

Confluence retains old snapshots that have been substituted with refreshed versions in the repository unless they are actively deleted. The Alfabet Connector app runs a job to delete snapshots from the repository once a day prior to requesting the updated snapshot versions via the Alfabet RESTful services. This is done via the **Cleanup snapshot storage** module of the Alfabet Connector app.

If Alfabet view snapshots shall be removed manually at a specific time, the **Wipe snapshot storage** module can optionally be activated and started via the **Scheduled Jobs** functionality in Confluence. Please note

however that this action will remove all Alfabet snapshots without providing new snapshots via the Alfabet REST-ful services. Users will no longer be able to view the snapshots in the Confluence pages they are embedded in. The following is required to delete all Alfabet snapshots:

- Activating the Module for Manual Snapshot Removal

- Running a Snapshot Deletion Job

Activating the Module for Manual Snapshot Removal

To activate the **Wipe snapshot storage** module:

- Log in to Confluence.

- In the Confluence header, click on ⚙ ˅ > **Manage Apps**.

- Expand the section about the Alfabet Connector app in the list and click the **9 of 10 modules are enabled** link. A list of modules is displayed.

- Move the cursor over the line of the **Wipe snapshot storage** module and click the **Enable** button that appears on the right.

  ⚠ Do not disable any of the other modules. The functionality of the Alfabet Connector app depends on the full functionality of all nine modules which are by default enabled.

Running a Snapshot Deletion Job

Once the **Wipe snapshot storage** module is enabled, you can run the job managed by the module any time:

- Log in to Confluence.

- In the Confluence header, click on ⚙ ˅ > **Manage Apps**.

- In the Confluence menu on the left, select the **Scheduled Job** page.

- In the line displaying the **Alfabet Delete All Snapshots** job, click **Run**.

Adding Alfabet Views with a Link to Alfabet to Confluence Pages

To add an Alfabet view snapshot with a link to Alfabet to a Confluence page, an express view link must be generated in Alfabet and pasted in the Confluence page.

Alfabet authorization rules are applied for access to Alfabet:

- The user access permissions to the view in Alfabet is handled according to the enterprise's configuration of express views. For information about the access permissions and the configuration of the express view functionality, see *Configuring the Express View (Email) Capability* in the reference manual *System Administration*.

- Express views cannot be created for some explorers and views. For example, explorer root nodes as well as all functionalities used for administrative purposes that are available via an administrative user

profile as well as the **Configuration** functionalities in the Alfabet user interface that are not associated with objects governed by access permissions do not allow express views to be created. In this case, the functionality to create an express view will be disabled.

To add an Alfabet snapshot to a Confluence page:

1) In Alfabet, open the view for which you want to create an express view.

2) Click **Bookmark** > **Mail Express View** in the main toolbar. A dialog box opens.

3) Copy the URL.

4) In Confluence, open the page that shall contain the Alfabet snapshot and select **Edit** in the menu on the upper right.

5) Click the location on the Confluence page where the Alfabet snapshot with the link to Alfabet shall be included.

6) In the Confluence menu, select **Insert Content** (the **+** sign) > **Other macros**.

7) In the window that opens, select the **Alfabet Connector** ✖ **ALFABET**   . A new window opens.

8) Define the following:

   • **Target URL**: Enter the URL that you copied from the express view dialog box in Alfabet in step 3.

   • **snapshot-size**: Select the size for the Alfabet snapshot on the Confluence page from the list of configured snapshot sizes.

9) Click **Preview** to see a preview of the Alfabet snapshot.

10) Click **Insert**. You will see the link as a link text in the edit view. The snapshot will only be visible after you have published the changes.

Chapter 6:   Configuring Interoperability with Technopedia

Software AG provides an interface to the Technopedia® repository of software and hardware products. The Technopedia capability allows vendor products to be captured in a standardized manner and the technology information to be aligned across the enterprise. The catalog of structured information about the IT infrastructure ensures a unified language and discipline to manage and plan the technology portfolio. With consistent naming and standardized data, the complexity of the IT landscape can be reduced, overlapping technologies eliminated, and existing and known technologies reused. If interoperability with Technopedia is supported by your enterprise, Alfabet users will be able to create vendor products in Alfabet based on Technopedia software products and/or hardware products.

Technopedia® has a two-level taxonomy comprised of product categories and sub-categories that contain the Technopedia products. Software AG does not own the Technopedia® taxonomy and using the Technopedia® taxonomy may result in a structure that is different from your enterprise's technology domain structure. Typically, the entire repository of Technopedia® product categories would be imported to Alfabet. Existing vendor product categories already defined in Alfabet will remain in the Alfabet database but will not be displayed in the Technopedia selector available in the Alfabet user interface that allows users to import Technopedia hardware and software products. The top level of the Technopedia® product categories will be imported to the top level of the vendor product hierarchy and subordinate categories will be displayed below their parent vendor product category.

The imported vendor product categories will initially have no vendor products assigned to them. In other words, the vendor products must be explicitly created based on the software/hardware product in the Technopedia repository. The software and hardware products in the Technopedia® repository can be selected on a category-by-category basis in order to create vendor products in Alfabet. The vendor products that are created based on Technopedia products are saved to the Alfabet database and can be further defined as needed. When the new vendor product is created, the vendor defined for that Technopedia product will be automatically assigned to the new vendor product in Alfabet. If the vendor does not already exist in the Alfabet database, it will be created in Alfabet along with the new vendor product. If an existing vendor product is already mapped to a Technopedia software or hardware product and the user selects the same Technopedia product and its `id` property matches the `TP_ID` property of the existing vendor product, then the existing vendor product will be updated and a new vendor product will not be created.

This documentation primarily describes the creation of vendor products based on Technopedia software and hardware products. However, some enterprises may prefer to implement the Technopedia capability to create components rather than vendor products. This will depend on the methodology implemented in your enterprise. Please note the following regarding using components instead of vendor products:

- The Technopedia® product categories will be mapped to component categories and the hardware and software products will be mapped to components.

- The object class `Component` must be specified in the XML attribute `ClassMapping` in the XML object **TechnopediaConfig**.

- For each component created in Alfabet based on a Technopedia software of hardware product, an object in the object class `VendorProduct` will first be created and then an object in the object class `Component` will be created. For more information, see the section [Understanding the Mapping of Technopedia Products to Vendor Products in Alfabet](#).

- The relevant page views must be made available to the user community as described in the section [Making the Technopedia Capability Available to the User Community](#).

- The predefined ADIF import scheme `ALFABET_TECHNOPEDIA_UPDATE` is only relevant for the update of Alfabet vendor products and ICT objects. You must configure an ADIF import

scheme in order to synchronize Alfabet components with the current information in Technopedia.

A subscription concept can be implemented for ICT objects that are based on Technopedia products in order to simplify the maintenance of the enterprise's technology catalog. If the ICT object is subscribed to the Technopedia product that it is based on, vendor products can be automatically created and maintained for all relevant release versions of the Technopedia product that the ICT object is based on. To implement the subscription concept for an ICT object, the **Is Subscribed** checkbox must be selected in the **Technopedia** tab of the **ICT Object** editor. The **Is Subscribed** checkbox specifies that the ICT object has a subscription to the Technopedia product that it is based on. The release version level (**All Releases**, **Major Releases**, **Minor Releases**) of the Technopedia product that shall be used to create the vendor products must be specified in the **Subscription Level** field. The ADIF job `ALFABET_TECHNOPEDIA_UPDATE` must be executed by a user with an administrative user profile in order to create the vendor products. Vendor products will be created for all relevant release versions of the Technopedia product that the ICT object is based on. The vendor products will be added to the **Vendor Products** page views for the ICT object they are associated with.

Vendors can also be created in Alfabet based on Technopedia manufacturers.

> The configuration of the Alfabet solution to interface with the Technopedia API as well as a subscription to Technopedia services is required in order to access the Technopedia repository and create vendor products based on the software products and hardware products stored in Technopedia.

The XML object **TechnopediaConfig** allows you to activate the Technopedia® capability as well as to configure the login information for the Technopedia service. Furthermore, you can configure information about the selectors in Alfabet used to find Technopedia products, how to map the lifecycle information of Technopedia products to the object class `VendorProduct`, and which Technopedia attributes to map to custom object class properties in Alfabet configured for the object classes `VendorProduct`, `Component`, `ICT Object`, and `Vendor`.

Data is requested from Technopedia via a RESTful service call sending a `GET` request to the following Technopedia APIs:

- For software: `/api/v1/software_extended/`

- For hardware: `/api/v1/hardware_extended/`

> If your enterprise plans to base new vendor products on both hardware products and software products from Technopedia, then it is recommended that object class stereotypes are configured for the object class `VendorProduct`. For more information about configuring object class stereotypes, see the section *Configuring Object Class Stereotypes for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand*.

> A predefined ADIF import scheme `ALFABET_TECHNOPEDIA_UPDATE` is available to update the vendor products, vendor product categories, ICT objects, and vendors that are based on Technopedia products with the current data in the Technopedia repository. When executed, the ADIF import scheme reads the configuration specified in the XML object **TechnopediaConfig**. For more information about updating the vendor products and vendors based on Technopedia products as well as the criteria that must be met for the update, see the section Updating Technopedia Products in Alfabet via the ADIF Import Scheme ALFABET_TECHNOPEDIA_UPDATE.

The following information is available:

- Understanding the Mapping of Technopedia Products to Vendor Products in Alfabet

- Configuring the XML Object TechnopediaConfig

- Making the Technopedia Capability Available to the User Community

- [Updating Technopedia Products in Alfabet via the ADIF Import Scheme ALFABET_TECHNOPEDIA_UPDATE](#)

Understanding the Mapping of Technopedia Products to Vendor Products in Alfabet

This section provides details about how Technopedia classes are mapped to Alfabet object classes and how the attributes of these Technopedia classes are mapped to the standard object class properties for the classes `VendorProduct`, `Component`, `ICT Object`, and `Vendor`. Custom properties are mapped as specified for the relevant object class in the XML element `ClassMapping` in the XML object **TechnopediaConfig**.

The Technopedia classes are mapped as follows to Alfabet object classes:

Technopedia Class	Alfabet Object Class
software_extended	VendorProduct, Component
hardware_extended	VendorProduct, Component
manufacturer	Vendor
taxonomy2012	VendorProductCategory, ComponentCategory, ICTObjectCategory
sw_product	ICTObject
hw_product	ICTObject

The following information describes how the attributes of these Technopedia classes are mapped to the standard object class properties for the classes `VendorProduct`, `Component`, `ICT Object`, and `Vendor`.

(i) The information below only describes mapping to the Alfabet object classes `VendorProduct`, `Component`, `ICT Object`, and `Vendor`. Please note the following if your enterprise maps Technopedia software of hardware products to the object class `Component`:

- For each component created in Alfabet based on a Technopedia software of hardware product, an object in the object class `VendorProduct` will first be created and then an object in the object class `Component` will be created. In other words, whenever a component is created based on a Technopedia product, two objects (a vendor product and a component) will be added to the Alfabet database.

- The vendor product is the base object for the new component and the component will have a reference to the vendor product. The following object class properties will be copied from the vendor product to the component: `Name`, `Version`, `StartDate`, `EndDate`, `Vendor`, and `TP_CATEGORY`. If the object class property `TP_CATEGORY` is specified for the vendor

product, then the system will try to find a component category object with the same `ID` and set this as the component category of the component.

- The predefined ADIF import scheme `ALFABET_TECHNOPEDIA_UPDATE` will not update components. You must configure an ADIF import scheme in order to synchronize components in Alfabet with the current information in Technopedia.

⚠ Only properties of the type `String` are supported when mapping Technopedia properties to Alfabet properties.

Technope-dia Class	Technopedia Attribute	Alfabet Object Class	Alfabet Object Class Property
soft-ware_ex-tended		Vendor-Product	
	id		`TP_ID` The technopedia `id` is an integer that is stored as a string in the `TP_ID`object class property. Mapping of objects is done via the id and therefore integration will fail if the content of the object class property is changed by any other than the Alfabet in-build mapping mechanism. this also applies to changes to the number format. Please note that if a class key specifies the `TP_ID` property for the object class `VendorProduct` and vendor products are imported with duplicate values for the Technopedia IDs (Technopedia `id` attribute), a class key violation will occur. If this occurs, the error will be written to the log file and the duplicate vendor product will be skipped and will not be imported to the Alfabet database.
	product_name		Name
	version		Version
	product_alias		Alias
	suit_name		Suite
	edition		Edition

Technope-dia Class	Technopedia Attribute	Alfabet Object Class	Alfabet Object Class Property
	`manufacturer`		`Vendor`
	`sub_category`		`TP_SubCategory`
	`cat_taxon-omy2012_id`		`TP_CATEGORY`
	`general_availabil-ity_date`		`StartDate`
	`end_of_life_date` and `obsolete_date`		`EndDate`, whereby the end date is the maximum of either `end_of_life_date` or `obsolete_date`. In order to avoid issues with the display of lifecycle charts in Alfabet, date fields will remain empty if the Technopedia products have the dates 1.1.1900 and 31.12.2999.
`hard-ware_ex-tended`		`Vendor-Product`	
	`id`		`TP_ID` Please note that Technopedia constructs the `id` based on a concatenation of `model_id+product_id`.
	`product`		`Name`
	`model`		`Version`
	`manufacturer`		`Vendor` If the predefined ADIF scheme `ALFABET_TECHNOPE-DIA_UPDATE` is executed, a new vendor will be automatically created in the Alfabet database for any vendor product that is updated based on a Technopedia hardware product that does not have a value specified for the `Manufacturer` property.
	`cat_taxon-omy2012_id`		`TP_CATEGORY`

Technope-dia Class	Technopedia Attribute	Alfabet Object Class	Alfabet Object Class Property
	`general_availabil-ity_date`		`StartDate`
	`last_availabil-ity_date` and `obso-lete_date`		`EndDate`, whereby the end date is the maximum of either `last_availability_date` or `obso-lete_date`. In order to avoid issues with the display of lifecycle charts in Alfabet, date fields will remain empty if the Technopedia products have the dates 1.1.1900 and 31.12.2999.
`manufac-turer`		`Vendor`	
	`manufacturer`		`Name`
	`symbol`		`ShortName`
	`description`		`Description`
	`country`		`Country`
	`city`		`City`
	`street`		`Street`
	`zip`		`Zip`
	`email`		`Email`
	`fax`		`Fax`
	`phone`		`Phone`
	`website`		`Website`

Technope-dia Class	Technopedia Attribute	Alfabet Object Class	Alfabet Object Class Property
sw_prod-uct		IC-TObject	
	cat_sw_product_id		TP_ID
	product_name		Name
	manufacturer		Vendor If the predefined ADIF scheme `ALFABET_TECHNOPE-DIA_UPDATE` is executed, a new vendor will be automatically created in the Alfabet database for any vendor product that is updated based on a Technopedia hardware product that does not have a value specified for the `Manufacturer` property.
	cat_taxon-omy2012_id		TP_CATEGORY
	create_date		StartDate
hw_prod-uct		IC-TObject	
	cat_hw_product_id		TP_ID
	product_name		Name
	manufacturer		Vendor If the predefined ADIF scheme `ALFABET_TECHNOPE-DIA_UPDATE` is executed, a new vendor will be automatically created in the Alfabet database for any vendor product that is updated based on a Technopedia hardware product that does not have a value specified for the `Manufacturer` property.
	cat_taxon-omy2012_id		TP_CATEGORY

Technope-dia Class	Technopedia Attribute	Alfabet Object Class	Alfabet Object Class Property
	create_date		StartDate

Configuring the XML Object TechnopediaConfig

The XML object **TechnopediaConfig** allows you to activate the Technopedia® capability as well as to configure the login information for the Technopedia service. Furthermore, you can specify the selectors in Alfabet used to find Technopedia products, how to map the lifecycle information of Technopedia products to the object class VendorProduct (or its object class stereotypes), and which Technopedia attributes to map to custom object class properties in Alfabet configured for the object classes VendorProduct and Vendor.

⚠️ Please note that only the mapping of properties of the type String is supporting when mapping Technopedia properties to Alfabet properties.

ⓘ Please note that server variables can be used in the XML object **TechnopediaConfig** to read the value of the XML attribute at runtime from the server alias configuration of the Alfabet Web Application when a connection to Technopedia is established. For information about server variables, see Configuring Server Variables for Integration and Interoperability Solutions.

To edit the XML object **TechnopediaConfig**:

1) Go to the **Presentation** tab, expand the **XML Objects** folder, and expand by the **Integration Solutions** folder.

2) Right-click **TechnopediaConfig** and select **Edit XML...**. The XML object **TechnopediaConfig** opens.

   ⓘ 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 **TechnopediaConfig**. 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.

3) Specify the following XML attributes below the root XML element TechnopediaConfig:

   - active: Enter "true" to activate interoperability with Technopedia. If the attribute active is set to "true", the menu options to import Technopedia categories and create new vendor products based on Technopedia products will be available in the relevant views. For more information about which views are relevant for the Technopedia capability, see the section Making the Technopedia Capability Available to the User Community.

   - service: Enter the scheme, host, and port of the URL targeting the Technopedia server used as the end point to retrieve information from Technopedia. This is usually http://api.technopedia.com.

   - api: Specify the URL path for the end point to retrieve information from Technopedia. This typically targets the Technopedia server used as the end point to retrieve information from Technopedia. This is usually /api/v1/.

- `authorization`: Enter the string used to authorize the RESTful API call to the Technopedia server. You must obtain a user name and key from Technopedia. The string should appear as follows: `apikey xxxlicence:0123456890abcdef`.

- `data_portion`: Enter an integer between 10-1000 to be used as a parameter in RESTful API calls to restrict the number of returned records. The recommended value is 50. Please note that an excessively high number will impact performance.

- `search_limit`: Enter an integer between 10-1000 to be used to determine the maximum number of records returned in the selector used to find Technopedia products in the Alfabet interface. The recommended value is 300. Please note that an excessively high number will impact performance.

- `category_source`: Specify whether the product categories in the **Technopedia (Sub-) Category** field in the selector are populated with the Technopedia categories from the Alfabet database or are populated via a separate service call with the Technopedia categories in the Technopedia repository. Enter either "`ALFABET`" to populate the **Technopedia (Sub-) Category** field in the selector with vendor product categories/component categories from the Alfabet database or enter "`TECHNOPEDIA`" to populate the product categories via a separate service call to the Technopedia repository.

  ⓘ Please note the following:

  - The value `TECHNOPEDIA` must be specified to initially import the taxonomy of Technopedia product categories. Users can only import vendor products/components from the Technopedia repository to a vendor product category/component category/ICT object in Alfabet that has the same name as the product category that it is assigned to in the Technopedia repository.

  - Populating the selector with the Technopedia product categories in the Alfabet database will provide for better performance but there may potentially be a discrepancy between the information available in the Alfabet database and that in the Technopedia repository.

  - While it is technically possible to create vendor product categories in Alfabet and implement categories from Technopedia®, it is recommended that your enterprise use only one source of vendor product categories. If you enter "`TECHNOPEDIA`" in the XML attribute `category_source`, users will not be able to view the vendor product categories in the **Vendor Products** explorer that have been created in Alfabet nor select these vendor product categories in the Technopedia selectors.

  - If the value "`ALFABET`" is specified for the XML attribute `category_source` and the subscription concept is implemented and ICT objects are created based on Technopedia products, vendor product categories will be used to populate the **Technopedia (Sub-) Category** field

- `timeout`: Specify the timeout (in seconds) to be applied to the HTTP request send to the Technopedia API endpoint.

- `certificate_path`: Specify the path for self-signed certificates provided by Technopedia.

- `default_gadate_period`: Specify an integer to be used to calculate a default start date if the `general_availability_date` property is empty for a Technopedia software of hardware product. The integer shall represent the number of years to subtract from the `end_of_life_date` property for the Technopedia product. The object class property

`TP_ARTIFICIAL_GADATE` will be automatically set to `True` for the vendor product if the start date is calculated based on the XML element `default_gadate_period`.

- `tp_sync_scope`: Specify `VendorProductCategory`, `ICTObjectCategory`, and/or `ComponentCategory` in a comma-separated list to allow the respective categories to be created or updated via the predefined ADIF import scheme `ALFABET_TECHNOPEDIA_UPDATE`. For example, if a XML element `ClassMapping` shall be specified for the class `VendorProduct`, add `VendorProductCategory` to the XML element `tp_sync_scope`. If a XML element `ClassMapping` shall be specified for the class `ICTObject`, add `ICTObjectCategory` to the XML element `tp_sync_scope`. For each class specified, the **Import All Technopedia Categories** button will be available in the relevant view. If this XML element is not defined, these classes will not be imported or synchronized via the ADIF import scheme `ALFABET_TECHNOPEDIA_UPDATE`.

4) The XML element `ClassMapping` allows you to specify the relevant mapping for the following classes: `VendorProduct`, `Component`, `ICTObject`, `Vendor`, `VendorProductCategory`, `ICTObjectCategory`, and `ComponentCategory`. Add a child XML element `ClassMapping` to the root XML element `TechnopediaConfig` for each relevant object class and specify the following XML attributes:

> If class keys for which the **Unique** attribute has been set to `True` have been defined for a class entered in the XML attribute `class`, you should ensure that the **Strict Null Handling** attribute for the class key for the class `VendorProduct` (or `Component`) is set to `False` so that 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. For more information about configuring unique keys, see the section *Configuring Class Keys for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand*.

- **To configure the mapping for the class VendorProduct (or Component)**:

  - `class`: Enter `VendorProduct` (or `Component`).

  - `sw_stereotype` and `hw_stereotype`:These are optional attributes that allow you to map object class stereotypes specified for the object class specified in the XML attribute `class`. In this way, you can implement different vendor product stereotypes to represent different kinds of hardware products and software products. Define the XML attributes `sw_stereotype` and `hw_stereotype` as follows:

    - `sw_stereotype`: Enter the name of the object class stereotype specified for the object class specified in the XML attribute `class` that may be created based on Technopedia software products.

    - `hw_stereotype`: Enter the name of the object class stereotype specified for the object class specified in the XML attribute `class` that may be created based on Technopedia hardware products.

  - `lc_phase1` and `lc_phase2`: These are optional attributes that allow you to map the lifecycle phases specified for the object class specified in the XML attribute `class`. Some Technopedia products will have dates defined for the start of the product's lifecycle, the end of the product's lifecycle, and the obsolete date of the product. Lifecycle phases will be created if all dates are defined in the Technopedia® product and the XML attributes `lc_phase1` and `lc_phase2` are defined. The dates of the first lifecycle phase (`lc_phase1`) will be based on the date range from Date 1 to min(Date 2, Date 3) and the dates of the second lifecycle phase (`lc_phase2`) will be based on the date range from min(Date 2,Date 3) to max(Date 2, Date 3). The lifecycle attributes for Technopedia software products are:

- Date 1 = `general_availability_date`

- Date 2 = `end_of_life_date`

- Date 3 = `obsolete_date`

Please note that the Technopedia lifecycle attributes `general_availability_date`, `end_of_life_date`, and `obsolete_date` describe the lifecycle of Technopedia software products. The lifecycle attributes for Technopedia hardware products are:

- Date 1 = `general_availability_date`

- Date 2 = `last_availability_date`

- Date 3 = `obsolete_date`

The XML attributes `lc_phase1` and `lc_phase2` should be mapped to two lifecycle phases defined for the object class `VendorProduct` in the XML object **ObjectLifecycleManager**. Please note that these names must be correctly spelled as written in the XML object **Object-LifecycleManager** for the object class `VendorProduct`. Define the XML attributes `lc_phase1` and `lc_phase2` as follows:

- `lc_phase1`: Enter the name of the lifecycle phase defined for the object class `VendorProduct` in the XML object **ObjectLifecycleManager** that you want to map to the period represented by the date range Date 1 (`general_availability_date` – the minimum value of either Date 2 (`end_of_life_date`/`last_availability_date`) or Date 3 (`obsolete_date`). The date for `end_of_life_date`/`last_availability_date` will be used if no `obsolete_date` has been defined.

- `lc_phase2`: Enter the name of the lifecycle phase defined for the object class `VendorProduct` in the XML object **ObjectLifecycleManager** that you want to map to the period represented by the date range minimum of either Date 2 (`end_of_life_date`/`last_availability_date`) or Date 3 (`obsolete_date`) – maximum of either Date 2 (`end_of_life_date`/`last_availability_date`) or Date 3 (`obsolete_date`). The date for `general_availability_date` will be used if no `obsolete_date` has been defined.

- `update_mode`: If a vendor product is associated with a Technopedia product and the Technopedia project is associated with a Technopedia product version/release, typically an ICT object will be created in Alfabet based on the referenced Technopedia product version/release when the ADIF scheme `ALFABET_TECHNOPEDIA_UPDATE` is executed. The XML attribute `update_mode` allows you to specify what should happen if the vendor product exists in Alfabet but has no reference to the relevant ICT object. Specify one of the following for the class `VendorProduct` in order to create and merge vendor products for the ICT objects added to Alfabet with missing references to vendor products:

- `Merge`: The system tries to find an existing ICT object that matches values for specified Alfabet properties that have been mapped to Technopedia attributes,

- `CreateOrMerge`: The system tries to find an existing ICT object and creates a new ICT object if none is found

- `None`: No attempt is made to merge or create

- merge_props: Specify the Alfabet properties to use when matching Alfabet objects with Technopedia objects. The default merge properties for the class `VendorProduct` are `Name`, `Version`, `Alias`, `Suite`, `Edition`, and `Stereotype`. An XML element `AttributeMapping` must be configured for each Alfabet property specified in the XML attribute `merge_props`.

- For each XML element `ClassMapping`, create one or more XML elements `AttributeMapping` in order to map the attributes in Technopedia to custom object class properties configured for the object class specified in the XML attribute `class`. Specify the following XML attributes for each XML element `AttributeMapping`:

  - tp_attr: Enter the names of the attribute in Technopedia to map to the Alfabet custom object class property specified in the attribute `alfa_attr`.

  - alfa_attr: Enter the names of the Alfabet custom object class property to map to the Technopedia® attribute specified in the attribute `tp_attr`.

- **To configure the mapping for the class ICTObject**. This configuration is only relevant if the subscription concept is implemented, whereby Technopedia products associated with an ICT object can be automatically added to Alfabet and regularly updated. In this case, ICT objects are created in Alfabet based on software and hardware products in Technopedia. The ICT objects based on Technopedia products are versionless and thus may serve as a subscription to a Technopedia product. The versions of the Technopedia product that the ICT object is based on can be imported to Alfabet as vendor products when the ICT object is updated. To implement the subscription concept, the **Is Subscribed** checkbox must be selected in the **Technopedia** tab of the **ICT Object** editor (`ICTO_TP_Editor`). If selected, vendor products will be created for any new Technopedia product versions that are associated with the Technopedia product that the ICT object is based. The vendor products will be added to the **Vendor Products** page views when the ADIF job `ALFABET_TECHNOPEDIA_UPDATE` is executed.

  - class: Enter `ICTObject`.

  - sw_stereotype: Enter the name of the object class stereotype specified for the object class `ICTObject` that may be created based on Technopedia software products.

  - hw_stereotype: Enter the name of the object class stereotype specified for the object class `ICTObject` that may be created based on Technopedia hardware products.

  - update_mode: If a vendor product is associated with a Technopedia product and the Technopedia project is associated with a Technopedia product version/release, typically an ICT object will be created in Alfabet based on the referenced Technopedia product version/release when the ADIF scheme `ALFABET_TECHNOPEDIA_UPDATE` is executed. The XML attribute `update_mode` allows you to specify what should happen if the relevant ICT object exists in Alfabet but has no reference to the vendor product. Specify one of the following for the class `ICTObject` in order to create and merge ICT objects for the vendor products added to Alfabet with missing references to ICT objects:

    - Merge: The system tries to find an existing ICT object that matches values for specified Alfabet properties that have been mapped to Technopedia attributes.

    - Create: The system creates a new ICT object. The value `Create` is the default value.

    - CreateOrMerge: The system tries to find an existing ICT object and creates a new ICT object if none is found

    - None: No attempt is made to merge or create

- `merge_props`: Specify the Alfabet properties to use when matching Alfabet objects with Technopedia objects. The default merge properties for the class `ICTObject` are `Name` and `Stereotype`. An XML element `AttributeMapping` must be configured for each Alfabet property specified in the XML attribute `merge_props`.

- Create one or more XML elements `AttributeMapping` as a child of the XML element `ClassMapping` in order to map the attributes in Technopedia to custom object class properties configured for the object class specified in the XML attribute `class`. Specify the following XML attributes for each XML element `AttributeMapping`:

  - `tp_attr`: Enter the names of the attribute in Technopedia to map to the Alfabet custom object class property specified in the attribute `alfa_attr`.

  - `alfa_attr`: Enter the names of the Alfabet custom object class property to map to the Technopedia® attribute specified in the attribute `tp_attr`.

- **To configure the mapping for the class Vendor**:

  - `class`: Enter `Vendor`.

  - For each XML element `ClassMapping`, create one or more XML elements `AttributeMapping` in order to map the attributes in Technopedia to custom object class properties configured for the object class specified in the XML attribute `class`. Specify the following XML attributes for each XML element `AttributeMapping`:

    - `tp_attr`: Enter the names of the attribute in Technopedia to map to the Alfabet custom object class property specified in the attribute `alfa_attr`.

    - `alfa_attr`: Enter the names of the Alfabet custom object class property to map to the Technopedia® attribute specified in the attribute `tp_attr`.

- **To add the description information for Technopedia Categories and Sub-Categories to the associated *Category objects in Alfabet (VendorProductCategory, ComponentCategory, and ICTObjectCategory)**:

  - Ensure that the category class (`VendorProductCategory`, `ICTObjectCategory`, and/or `ComponentCategory`) is specified in the XML element `tp_sync_scope`.

  - Add a child XML element `ClassMapping` to the root XML element `TechnopediaConfig` for each relevant category class specified in the XML element `tp_sync_scope`. Specify the following XML attributes:

    - class: Enter the relevant category class (`VendorProductCategory`, `ICTObjectCategory`, or `ComponentCategory`).

    - Add a child XML element `AttributeMapping` to the XML element `ClassMapping`.

      `tp_attr`: Enter `DESCRIPTION`

      `alfa_attr`: Enter `Description`

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

Making the Technopedia Capability Available to the User Community

You must ensure that the relevant page views supporting the Technopedia capability are available to the user community. The page views should be available as needed in the object views that are associated with the user profiles of the users who are responsible for defining and maintaining vendor products (or components). For more information, see the chapters *Configuring Object Views* and *Configuring User Profiles for the User Community* in the reference manual *Configuring Alfabet with Alfabet Expand*.

If your enterprise has implemented object class stereotypes for the class `VendorProduct`, then you can hide the menu options that are not relevant for the vendor product stereotype in the custom object view. Or, if your enterprise does not support the used of both hardware and software products, it is advised that you also hide the menu option that is not relevant. For more information about how to do this, see the section *Hiding Functionalities in a Page View or Configured Report* in the reference manual *Configuring Alfabet with Alfabet Expand*.

The following Alfabet views support the creation of vendor products based on Technopedia software and hardware products. The views you need to include in a user profile will depend on the methodology implemented in your enterprise as well as the configuration of the XML object **TechnopediaConfig**:

- The following views allow vendor products to be created based on Technopedia software and hardware products. A different selector is implemented to search for either Technopedia software products or Technopedia hardware products:

  - **Vendor Products** page view (`VPC_Products`) in the Vendor Product Category object profile

  - **Vendor Products** page view (`VDR_VendorProducts`) in the Vendor object profile

  - **Vendor Products** page view (`DOM_VendorProducts`) in the Domain object profile

  - **Vendor Products** page view (`ICTO_VendorProducts`) in the ICT Object object profile

  - **Vendor Products** page view (`COM_VendorProduct`) in the Component object profile

  - **Capture Vendor Products** (`VP_CaptureVendorProduct`and `COM_CaptureCVendorProduct_Ex`)

  - **Document Vendor Products** (`Document_VendorProduct`) functionalities

  - **Vendor Product** object profile (`VP_ObjectView`): This view allows you to merge existing vendor products with vendor products in the Technopedia repository.

  - **Root Vendor Product Categories** page view (`VPC_RootCategories`) available on the root node of the **Vendor Products** explorer. This view allows you to import the entire repository of Technopedia® product categories as vendor product categories.

- A subscription concept can also be implemented whereby Technopedia products associated with an ICT object can be automatically added to Alfabet and regularly updated. In this case, ICT objects are created in Alfabet based on software and hardware products in Technopedia. The ICT objects based on Technopedia products are versionless and thus may serve as a subscription to a Technopedia product release. The release versions of the Technopedia product that the ICT object is based on can be imported to Alfabet as vendor products when the ICT object is updated.

  The following views allow ICT objects to be created based on Technopedia software and hardware products to be created. A different selector is implemented to search for either Technopedia software products or Technopedia hardware products:

  - **ICT Objects** page view (`ICTOC_ICTObjects`) in the ICT Object Category object profile

  - **ICT Objects** page view (`ICTOG_ICTObjects`) in the ICT Object Group object profile

- **ICT Objects** page view (`DOM_ICTObjects`) in the Domain object profile

- **ICT Objects** page view (`VD_ICTObjects`) in the Vendor object profile

- **Capture ICT Objects** functionality (`ICTO_CaptureICTObjects` and `ICTO_CaptureICTObjects_Ex`)

- **Document ICT Objects** functionality (`Document_ICTObjects`)

- Please note that the **ICT Object** editor (`ICTO_TP_Editor`) is required to implement the subscription concept and must be assigned to the relevant class settings for the class ICT Object. This editor includes an **Is Subscribed** checkbox in the **Technopedia** tab that if selected specifies that the ICT object has a subscription to the Technopedia product that it was based on. A **Subscription Level** field allows the release version level (**All Releases**, **Major Releases**, **Minor Releases**) of the Technopedia product to be specified that shall be used to create the vendor products. Vendor products will be created for all relevant release versions of the Technopedia product that the ICT object is based on. Once the ICT object has been created in Alfabet, the ADIF job `ALFABET_TECHNOPEDIA_UPDATE` must be executed to create the vendor products. The vendor products will be added to the **Vendor Products** page views for the ICT object they are associated with.

- **ICT Object** object profile (`ICTO_ObjectView`): This view allows you to merge existing ICT objects with ICT objects in the Technopedia repository.

- **Root ICT Object Categories** page view (`ICTOC_Roots`) available on the root node of the **ICT Objects by Category** explorer. This view allows you to import the entire repository of Technopedia product categories as ICT object categories.

- Vendors can be created based on a Technopedia manufacturer in the following views:

  - **Vendors Explorer** (`VDR_Explorer`)

  - **Vendors** page view (`VDRG_Vendors`) in the Vendor Group object profile

  - **Capture Vendors** functionality (`COM_CaptureVendor` and `COM_CaptureVendor_Ex`)

> ⓘ If your enterprise implements the Technopedia capability to create components rather than vendor products, the following Alfabet views support the creation of components based on Technopedia software and hardware products:

  - **Root Categories** page view (`COMC_RootCategories`) available on the root node of the **Components** explorer. This view allows you to import the entire repository of Technopedia® product categories as component categories.

  - The following views allow Technopedia software and hardware products to be created. A different selector is implemented to search for either Technopedia software products or Technopedia hardware products:

    - **Components** page view (`COMC_Components`) in the Component Category object profile

    - **Components** page view (`COMG_Components`) in the Component Group object profile

    - **Components** page view (`DOM_Components`) in the Domain object profile

    - **Capture Components** functionality (`COM_CaptureComponent` and `COM_CaptureComponent_Ex`)

- **Document Components** functionality (`Document_Component`) functionalities

Updating Technopedia Products in Alfabet via the ADIF Import Scheme ALFABET_TECHNOPEDIA_UPDATE

A predefined ADIF import scheme is available to update the vendor products, vendor product categories, ICT objects, and vendors based on Technopedia software and hardware products as well as their vendors with the current data in the Technopedia repository. This ADIF import scheme `ALFABET_TECHNOPEDIA_UPDATE` is a protected ADIF import scheme and cannot be edited. When executed, the ADIF import scheme reads the configuration specified in the XML object **TechnopediaConfig**. Changes that have occurred in the Technopedia repository will be updated in the Alfabet database. If no changes have been made to the relevant products in the Technopedia repository, a message will be displayed stating that no changes were made.

The Technopedia products can be synchronized by a user with an administrative user profile in the *ADIF Jobs Administration Functionality*. For more information, see the section *Executing and Controlling ADIF Jobs* in the reference manual *User and Solution Administration*.

The following describes information relevant to the update of data in Alfabet via the predefined ADIF import scheme `ALFABET_TECHNOPEDIA_UPDATE`. The update will align with data in Alfabet with the current information in the Technopedia repository. For an overview of the attribute mapping, see the section Understanding the Mapping of Technopedia Products to Vendor Products in Alfabet .

Please note the following:

- **Vendor Products**:

  - Each vendor product in the Alfabet database will be updated if the following criteria is fulfilled:

    - The `TP_ID` property of the vendor product is defined (which constitutes the identifier of the `product_name` value in Technopedia that the vendor product is based on).

    - The `TP_UPDATE` value is earlier than the `last_modified_date` attribute of the product in the Technopedia repository. The `TP_UPDATE` value constitutes the timestamp of the most recent update of the vendor product with the corresponding product in the Technopedia repository.

    - All other vendor products that do not fill this criteria will not be changed.

  - The following object class properties will be updated for software products via the ADIF import scheme `ALFABET_TECHNOPEDIA_UPDATE`:

    - `Name`

    - `Version`

    - `Vendor`

    - `Alias`

    - `Suite`

    - `Edition`

    - `StartDate`: The `StartDate` will be updated if the `general_availability_date` is defined.

- EndDate: The EndDate will be updated if a specified date is defined for one of the following dates in the following sequence: obsolete_date, end_of_life_date, general_availability_date.

- Lifecycle phases will be created if all relevant dates are defined in the Technopedia® product and the XML attributes lc_phase1 and lc_phase2 are specified in the XML object **TechnopediaConfig**. Please note the following:

  - The first lifecycle phase will consist of <general_availability_date minus the minimum value of either end_of_life_date or obsolete_date>.

  - The last lifecycle phase will consist of <minimum of end_of_life_date/last_availability_date or obsolete_date minus the maximum of end_of_life_date or obsolete_date>.

- All other properties mapped via the XML attribute AttributeMapping for the class VendorProduct in the XML object **TechnopediaConfig**.

- The following object class properties will be updated for hardware products via the ADIF import scheme ALFABET_TECHNOPEDIA_UPDATE:

  - Name

  - Version

  - Vendor

  - Alias

  - StartDate: The StartDate will be updated if the general_availability_date is defined.

  - EndDate: The EndDate will be updated if a specified date is defined for one of the following dates in the following sequence: obsolete_date, last_availability_date, general_availability_date.

  - Lifecycle phases will be created if all relevant dates are defined in the Technopedia® product and the XML attributes lc_phase1 and lc_phase2 are specified in the XML object **TechnopediaConfig**. Please note the following:

    - The first lifecycle phase will consist of <general_availability_date minus the minimum value of either last_availability_date or obsolete_date>.

    - The last lifecycle phase will consist of <minimum of end_of_life_date/last_availability_date or obsolete_date minus the maximum of last_availability_date or obsolete_date>.

  - All other properties mapped via the XML attribute AttributeMapping for the class VendorProduct in the XML object **TechnopediaConfig**.

- **Vendors**: Please note the following

  - Each vendor in the Alfabet database will be updated if the following criteria is fulfilled:

    - The TP_ID property of the vendor is defined (which constitutes the identifier of the manufacturer in Technopedia that the vendor is based on).

    - The TP_UPDATE value is earlier than the last_modified_date attribute of the product in the Technopedia repository. The TP_UPDATE value constitutes the timestamp of the most

recent update of the vendor product with the corresponding product in the Technopedia repository

- A new vendor will be automatically created in the Alfabet database for any vendor product that is updated based on a Technopedia hardware product that does not have a value specified for the `manufacturer` property.

- The following object class properties will be updated via the ADIF import scheme `ALFABET_TECHNOPEDIA_UPDATE`:

  - `Name`

  - `ShortName`

  - `Description`

  - `Country`

  - `City`

  - `Street`

  - `Zip`

  - `Email`

  - `Fax`

  - `Phone`

  - `Website`

  - All other properties mapped via the XML attribute `AttributeMapping` for the class `Vendor` in the XML object **TechnopediaConfig**.

- **Components**: Components will not be updated via the predefined ADIF import scheme `ALFABET_TECHNOPEDIA_UPDATE`. You must configure an ADIF import scheme in order to synchronize Alfabet components with the current information in Technopedia.

- **ICT Objects**: Please note the following:

  - If the `TP_Subscribed` property of the ICT object is set to `True`, the execution of the predefined ADIF import scheme `ALFABET_TECHNOPEDIA_UPDATE` will create new vendor products as specified via the `TP_SUBSCRIPTIONLEVEL` property of the ICT object.

  - The reference of a vendor product to an ICT object will be updated via the execution of the ADIF import scheme `ALFABET_TECHNOPEDIA_UPDATE` if both the vendor product as well as the ICT object originated from Technopedia and the Technopedia product version/release associated with the vendor product references a Technopedia product that is different from the Technopedia product referenced by the ICT object. The specification of the XML attribute `update_mode` determines what should happen if the relevant ICT object exists in Alfabet but has no reference to the vendor product.

  - The `StartDate` and `EndDate` as well as lifecycle information will not be copied from the Technopedia product to the ICT object.

  - Properties mapped via the XML attribute `AttributeMapping` for the class `ICTObject` in the XML object **TechnopediaConfig** can be updated via the ADIF import scheme `ALFABET_TECHNOPEDIA_UPDATE`.

- **Vendor Product Categories**: Vendor product categories will be updated via the predefined ADIF import scheme `ALFABET_TECHNOPEDIA_UPDATE` if the value `VendorProductCategory` is specified for the XML element `tp_sync_scope` in the XML object **TechnopediaConfig**. If a vendor product is created based on a Technopedia product and the associated category does not exist in Alfabet, then the vendor product category will be created in Alfabet when the ADIF import scheme `ALFABET_TECHNOPEDIA_UPDATE` is executed.

- **ICT Object Categories**: ICT object categories will be updated via the predefined ADIF import scheme `ALFABET_TECHNOPEDIA_UPDATE` if the value `ICTObjectCategory` is specified for the XML element `tp_sync_scope` in the XML object in the XML object **TechnopediaConfig**. If an ICT object is created based on a Technopedia product and the associated category does not exist in Alfabet, then the ICT object category will be created in Alfabet when the ADIF import scheme `ALFABET_TECHNOPEDIA_UPDATE` is executed.

- **Component Categories**: Component categories will be updated via the predefined ADIF import scheme `ALFABET_TECHNOPEDIA_UPDATE` if the value `ComponentCategory` is specified for the XML element `tp_sync_scope` in the XML object in the XML object **TechnopediaConfig**. If a component is created based on a Technopedia product and the associated category does not exist in Alfabet, then the component category will be created in Alfabet when the ADIF import scheme `ALFABET_TECHNOPEDIA_UPDATE` is executed.

Chapter 7:   Configuring Interoperability with webMethods API Gateway

Software AG provides interoperability between Alfabet and webMethods® API Gateway so that your enterprise can administer and monitor APIs as operational technical services in the Alfabet user interface. The webMethods API Gateway product provides a policy framework to manage and secure APIs. You can configure interoperability between Alfabet and the webMethods API Gateway and thus provide integrated functions to import APIs to Alfabet as operational technical services, document and describe those technical services, and export the modified technical service definition back to the API gateway.

APIs may be imported in the Alfabet user interface via the **Technical Services Registry Services** view (`COMSR_Services`) or the **Technical Services Registry Services - Filtered** (`COMSR_ServicesExt`) view. Once the technical services have a specified release status, they can be exported from Alfabet to API Gateway in the user interface via the **Technical Services** view (`COM_TechServices`) view. By the click of a button, users can navigate from Alfabet to webMethods API Gateway and view the API definition directly in the API gateway interface.

APIs may also be imported to Alfabet by means of an ADIF import scheme. Technical services cannot be exported via an ADIF scheme to webMethods API Gateway. If the import of APIs via ADIF is specified in the XML object **APIGatewayConfig**, then an ADIF scheme must be executed to trigger the import. The data from webMethods API Gateway will be imported to temporary tables in ADIF. Further configuration is required in order to update the Alfabet database tables with the data in the temporary database tables. For more information about configuring and executing an ADIF scheme, see the reference manual *Alfabet Data Integration Framework*.

APIs and their references are mapped to predefined object classes in Alfabet. When an API is imported to Alfabet via the views in the Alfabet user interface, a technical service will be created in Alfabet. If any `Operations`, `Resources`, `Methods`, and `Method Parameters` are specified for the API, the correspondent technical service operation, business data, technical service operation method, and technical service operation method parameter will be created in Alfabet. The mapping is described in the table below:

API Gateway Object	Alfabet Object Class
API	**Technical Service** (`Service`) Please note that additional specification of the mapping of `APIs` to technical services must be configured in the XML object **APIGatewayConfig**.
Operation	**Technical Service Operation** (`ServiceOperation`)
Resource	**Business Data** (`BusinessData`) Please note that additional specification of the mapping of `Resources` to business data must be configured in the XML object **ServiceResourceMapping**.
Method	**Technical Service Operation Method** (`ServiceOperationMethod`)
Method Parameter	**Technical Service Operation Method Parameter** (`ServiceOperationMethodParameter`)

The following information is available to configure interoperability with webMethods API Gateway:

- [Overview of the Configuration Required for webMethods API Gateway](#)

- [Configuring the Alfabet Class Model for Interoperability with webMethods API Gateway](#)

- [Configuring Connections and API Asset Mapping for webMethods API Gateway](#)

- [Configuring the Mapping of API Gateway Resources to Business Data](#)

Overview of the Configuration Required for webMethods API Gateway

The following is an overview of all configuration steps required for interoperability with webMethods® API Gateway:

1) Configure object class stereotypes for the object classes `Service`, `ServiceOperation`, and `ServiceOperationMethod`. You will later map the object class stereotypes of the class `Service` to the API assets available in the API gateway. This is described in more detail in the section [Configuring the Alfabet Class Model for Interoperability with webMethods API Gateway](#).

2) Map the object class stereotypes for the object classes `Service`, `ServiceOperation`, and `ServiceOperationMethod` 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. For more information, see the section *Configuring Object Class Stereotypes for Technical Services* in the chapter *Configuring Alfabet Functionalities Implemented in the Solution Environment* in the reference manual *Configuring Alfabet with Alfabet Expand*.

3) Configure release statuses for the object class `Service` as well as the object class stereotypes in the XML object **ReleaseStatusDefs**. These release statuses should correspond to the statuses used in webMethods API Gateway. Please note that the release status `HandoverStatus` must be specified in order to be able to export the technical services in Alfabet to webMethods API Gateway. This is described in more detail in the section [Configuring the Alfabet Class Model for Interoperability with webMethods API Gateway](#).

4) Specify one or more connections to webMethods API Gateway in the XML object **APIGatewayConfig**. Here you also specify whether import occurs via the Alfabet user interface or via an ADIF import scheme, how API assets are mapped to the object class stereotypes configured for the object class `Service`, and the HandoverStatus required for the technical services in order to export them to webMethods API Gateway. This is described in more detail in the section [Configuring Connections and API Asset Mapping for webMethods API Gateway](#).

5) If server variables are used to configure the connection in the XML object **APIGatewayConfig**, the server variables must be specified for the server alias in the Alfabet Administrator. For more information about configuring server variables, see the section [Configuring Server Variables for Integration and Interoperability Solutions](#).

6) Specify the mapping of data associated with `Resources` to the objects of the class **Business Data**. This is specified in the XML object **ServiceResourceMapping**. This is described in more detail in the section [Configuring the Mapping of API Gateway Resources to Business Data](#).

7) The attributes **Assembly** and **Assembly Class** available for the **ServiceRegistryManager** solution manager must be specified to ensure interoperability. The XML object **ServiceRegistryManager** is available in Alfabet Expand.The following values must be specified:

- **Assembly**: `ITPlan`

- **Assembly Class**: `ITPlanSolution.GenericServiceRegistryManager`

8) Create API gateway connection objects in the **API Gateway Database Connection** view in the **Integration Solutions Configurations** functionality in the Alfabet user interface. An API gateway connection object should be created for each API gateway connection configured in the XML object **APIGatewayConfig**. The API gateway connection objects allow a user to specify the connection to the relevant API gateway instance to use when exporting/synchronizing a technical service in the **Technical Services** page view (`COM_TechServices`). You must also specify whether Alfabet should automatically synchronize the technical services when the **Technical Services** page view is loaded. The creation of API gateway connection objects is described in the section *Configuring Semantic Connections for Integration Solutions* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.

9) The relevant user profiles must be configured to include the **Technical Services Registry Services - Filtered** page view (`COMSR_ServicesExt`) available on the root node of the **Components** explorer (`COM_Explorer`) and **Technical Services** page view (`COM_TechServices`) available in the **Components** object view. (This view is only available for components of the type `Service`). The **Technical Services Registry Services** page view (`COMSR_Services`) available on the root node of the **Components** explorer (`COM_Explorer`) should not be visible for the user profiles accessing webMethods API Gateway. For more information about configuring user profiles, see the chapter *Configuring User Profiles for the User Community* in the reference manual *Configuring Alfabet with Alfabet Expand*.

10) If APIs are to be imported to Alfabet by means of an ADIF import scheme, you must create an ADIF scheme via the **API GAteway Assistant** in order to import the APIs to temporary database tables. Your enterprise must configure the update of the Alfabet database tables with the data in the temporary database tables. This is described in more detail in the reference manual *Alfabet Data Integration Framework*.

Configuring the Alfabet Class Model for Interoperability with webMethods API Gateway

The following configuration is required in order to provide interoperability with webMethods® API Gateway:

- Configure object class stereotypes for the object class `Service`. Each API asset type that should be displayed in Alfabet must be mapped to an object class stereotype of the class `Service` in the XML element **AssetTypeMappings** in the XML object **APIGatewayConfig**. Therefore, before the XML object **APIGatewayConfig** can be specified, you must first configure the relevant object class stereotypes for the class `Service`. For detailed information about how to configure object class stereotypes, see the section *Configuring Object Class Stereotypes for Object Classes* in the chapter *Configuring the Class Model* in the reference manual *Configuring Alfabet with Alfabet Expand*.

    For example, the object class stereotypes for the object class `Service` could be specified as follows:

```
<ClassStereotypes>
  <Stereotype Name="SOAPService" Caption="SOAP Service"
        CaptionPlural="SOAP Services" Comments="" HasMandates="false" />
  <Stereotype Name="DataService" Caption="DataService"
        CaptionPlural="Data Services" Comments="" HasMandates="false" />
  <Stereotype Name="RESTService" Caption="REST Service"
        CaptionPlural="REST Services" Comments="" HasMandates="false" />
  <Stereotype Name="XMLService" Caption="XML Service"
        CaptionPlural="XML Services" Comments="" HasMandates="false" />
</ClassStereotypes>
```

- Configure object class stereotypes for the object class `ServiceOperation`. Each `Operation` associated with an imported API will be created as a technical service operation in Alfabet.

- Configure object class stereotypes for the object class `ServiceOperationMethod`. Each `Method` associated with an imported API will be created as a technical service operation method in Alfabet.

- Configure the mapping of technical service stereotypes (based on class `Service`), technical service operation stereotypes (based on class `ServiceOperation`), and technical service operation method stereotypes (based on class `ServiceOperationMethod`) 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. For more information about mapping the object class stereotypes defined for technical services and technical service operations, see the section *Configuring Object Class Stereotypes for Technical Services* in the reference manual *Configuring Alfabet with Alfabet Expand*.

- Configure release statuses for the object class `Service` as well as all relevant object class stereotypes for the object class `Service`. Each asset type in API Gateway will have a status. Therefore, each object class stereotype should have corresponding release statuses that are aligned with the statuses of the asset type that it will be mapped to. Please keep the following in mind:

  - The release status set for each relevant object class stereotype must be configured in the XML object **ReleaseStatusDefs**. A release status definition must be created for the object class `Service` as a whole as well as each object class stereotype. The release status definition for the object class must include the complete set of release statuses configured for its object class stereotypes. 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 more information about configuring release statuses, see the section *Configuring Release Status Definitions for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand*.

  - The release statuses in Alfabet should be specified in the primary language associated with the base culture definition of the default culture setting. An error message will be displayed if a user attempts to synchronize data in Alfabet with API Gateway assets and one or more of those API Gateway assets have statuses defined in a language that is not the primary language in Alfabet.

  - For each status available for an asset type, a corresponding release status must be defined for the relevant object stereotype. The name of the release status must be the same as the name of the API gateway status.

  - The set of release statuses available for the object class stereotype may include release statuses that are relevant to managing the technical service in Alfabet. Such release statuses are considered Alfabet -owned, as they will be defined in Alfabet only.

  - The statuses defined in and owned by API Gateway should have the same names as the release statuses defined in Alfabet.

  - One of the release statuses configured in the release status set will be specified in the XML attribute `HandoverStatus` in the XML object **APIGatewayConfig** as the release status that the technical service must have in order to be updated to the API Gateway repository. If necessary, a different handover status may be specified for each object class stereotype. This is described in the section Configuring Connections and API Asset Mapping for webMethods API Gateway.

Configuring Connections and API Asset Mapping for webMethods API Gateway

Multiple connections to webMethods® API Gateway can be defined in order to address the needs of federated organizations. The XML object **APIGatewayConfig** allows you to configure the connections to webMethods API Gateway instances as well as how the APIs should be imported to Alfabet. You can configure the information necessary so that users can import and export APIs to Alfabet in the Alfabet user interface and/or how to import the APIs via an ADIF import scheme. Please note the following:

- APIs may be imported via the **Technical Services Registry Services** view (`COMSR_Services`) or the **Technical Services Registry Services - Filtered** (`COMSR_ServicesExt`) view. If the import of APIs should occur via the Alfabet user interface, you must map the API assets to the relevant object class stereotypes configured for the class `Service` in the XML object **APIGatewayConfig**. You must also specify the relevant `HandoverStatus` release status that a technical service must have in order to be exported from Alfabet to the API Gateway.

- APIs may also be imported to Alfabet by means of an ADIF import scheme. Technical services cannot be exported via an ADIF scheme to webMethods API Gateway. If the import of APIs via ADIF is specified in the XML object **APIGatewayConfig**, then the ADIF scheme must be executed to trigger the import. The data from webMethods API Gateway will be imported to temporary tables in ADIF. Further configuration is required in order to update the Alfabet database tables with the data in the temporary database tables. For more information about configuring an ADIF scheme, see the reference manual *Alfabet Data Integration Framework*.

⚠️ If you want to enter a string that contains special characters in the XML object **APIGatewayConfig**, you must replace the special characters with respective XML compliant code, for example:

- > for >

- < for <

- " for "

- [for [

-] for]

The configuration of the connections is carried out in the XML object **APIGatewayConfig** by means of the root XML element **APIGatewayConfig** and its child XML element **APIGatewayConnections**. Once the configuration of the XML object **APIGatewayConfig** is complete, connection objects must also be created in the **API Gateway Database Connections** view available in the **Integration Solutions Configurations** functionality in the Alfabet interface.

The root XML element **APIGatewayConfig** has two child elements:

- **Proxy**: This XML element allows you to configure how to send requests to webMethods API Gateway instances via a proxy server. The configuration of the XML element **Proxy** is described in the section Configuring Server Variables for Integration and Interoperability Solutions.

- **APIGatewayConnections**: This XML element allows you to configure multiple connections to webMethods API Gateway instances. A child XML element **APIGatewayConnection** should be configured for each connection to a webMethods API Gateway instance. Each XML element has two child elements. These child elements are **AssetTypeMappings**, which allows you to specify the mapping of APIs from the API gateway to Alfabet when imported via the Alfabet user interface, and **DataConnectivityInfo**, which allows you to specify the import of APIs from the API gateway to Alfabet when imported via an ADIF import scheme.

The XML object usually includes an example definition. In addition, a template is available via the attribute **XML Template** of the XML object. The template can be copied to the XML object to avoid having to write the configuration manually. The following information describes a configuration from scratch. With a sample configuration, you must edit the existing XML elements rather than add them to the XML configuration.

You can configure multiple connections from Alfabet to multiple webMethods API Gateway instances. To define a connection to a webMethods API Gateway instance.

1) In Alfabet Expand, go to the **Presentation** tab and expand the explorer nodes **XML Objects** > **IntegrationSolutions**.

2) Right-click the XML object **APIGatewayConfig** and select **Edit XML...**. The XML object opens.

> The XML element **Proxy** allows you to configure how to send requests to webMethods API Gateway instances via a proxy server. This is optional. If you have configured multiple connections and each connection shall use a different proxy, you can add additional proxies to your proxy configuration and specify the relevant proxy definition in the API Gateway connection configuration. The configuration of the XML element **Proxy** is described in the section Configuring Server Variables for Integration and Interoperability Solutions.

3) Add a child XML element **APIGatewayConnections** to the root XML element **APIGatewayConfig**.

4) Add a child XML element **APIGatewayConnection** to the XML element **APIGatewayConnections** for each database connection instance that you will define.

5) Configure a connection to each API gateway instance by setting the following XML attributes for the XML element **APIGatewayConnection**:

- `Name`: Enter a unique name for the connection to the API gateway.

- `ServerURL`: Enter the URL to access the API gateway or enter the relevant server variable name.

  > In some of the XML attributes, server variables can be used to read the value of the attribute at runtime from the server alias configuration of the Alfabet Web Application when a connection to API Gateway is established. For more information about using server variables, see the section Configuring Server Variables for Integration and Interoperability Solutions.

- `UserName`: Enter your enterprise's user name to access the API gateway or enter the relevant server variable name.

- `Password`: Enter your enterprise's password to access the API gateway or enter the relevant server variable name.

- `ServiceViewURL`: Enter the prefix for the complete link when navigating to the API gateway from Alfabet when the user clicks the **Open Service's View in Service Registry** button in the **Technical Services Registry Services - Filtered** page view. Enter the URL to access the service view in API Gateway or enter the relevant server variable name. For example: `http://<APIGatewayServer>:<Port>/#default/apiDetails`)

- `TransactionHistoryPeriod`: Enter the number of days for which the transactions are fetched. This is only necessary if a bulk import is being executed via ADIF.

- `Timeout`: Enter the number of minutes of user inactivity after which the connection to the API gateway should be terminated.

6) If APIs shall be imported to Alfabet via the Alfabet user interface, add an XML element `AssetTypeMappings` to each XML element **APIGatewayConnection**. For each asset type that you want to import to Alfabet, specify a child XML element `AssetTypeMapping`.

7) Configure each asset type mapping by setting the following XML attributes for the XML element `AssetTypeMapping`:

   • `APIGatewayAssetType`: Enter the name of the webMethods API Gateway asset type that should be mapped to the object class stereotype specified in the XML element `AlfabetStereotype`.

   • `HandoverStatus`: Specify the release status that the technical services based on the object class stereotype in the XML attribute `AlfabetStereotype` must have in Alfabet in order to be updated to the webMethods API Gateway instance. For more information about configuring release statuses for webMethods API Gateway interoperability, see the section *Configuring Release Status Definitions for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand*.

   • `AlfabetStereotype`: Enter the name of the object class stereotype of the class `Service` in Alfabet that should be mapped to the API gateway asset type specified in the XML element `APIGatewayAssetType`.

8) If APIs shall be imported to Alfabet via an ADIF import scheme, add an XML element `DataImport` with a child XML element `DataConnectivityInfo` to each XML element **APIGatewayConnection**.

9) For each import definition that you want to configure, specify a child XML element `DataConnection` and an XML attribute `Name`. Enter the name of the data connection.

10) For each import type in webMethods API Gateway that you want to import via ADIF, enter an XML element `Import` with an XML attribute `Type`. Enter the import type that may be imported via the ADIF import scheme. Permissible import types are: `API`, `POLICY`, `APPLICATION`, `TRANSATION`, `TRANSACTIONDETAIL`, `PACKAGE`, and `PLAN`:

11) In the toolbar of Alfabet Expand, click the **Save** 🖫 button to save your changes.

Configuring the Mapping of API Gateway Resources to Business Data

An API that is imported from the webMethods API Gateway to Alfabet will be created as a technical service based on the stereotype configured in the XML object **APIGatewayConfig**. When the API is imported, the associated `Resources` may be imported as well. In this case, an instance of the object class `BusinessData` may be updated or created for each `Resource`. For this to happen, you must map the logic to use to find a matching business data in Alfabet and update that business data with the information about the `Resource` or, if necessary, to create a new business data and write the information about the `Resource` to it. Several mapping logics may be configured to find or create business data.

ⓘ If `Methods` and `Parameters` are defined for a `Resource`, they will also be added to the business data in Alfabet. For each `Method` specified for a `Resource`, a technical service operation method (instance of class `ServiceOperationMethod`) will be created and for each `Parameter`, a technical service operation method parameter (instance of class `ServiceOperationMethodParameter`) will be created. Please note that the technical service operation method (instance of class `ServiceOperationMethod`) will be created based on an object class stereotype that is specified in the XML object **TechServiceManager**. For more information, see the section *Configuring Object Class Stereotypes for Technical Services* in the chapter *Configuring Alfabet Functionalities Implemented in the Solution Environment*.

The following must be configured in the XML object **ServiceResourceMapping**.

- Specify one or more `ResourceMapping` definitions in order to find existing business data or find existing business objects that new business data is created for. The mapping logic is highly configurable.

  For example, you could specify the following mapping logics:

  - Priority 1: Find an existing business data that matches all properties specified in the resource mapping and update it. For example, find a business data that has the same the **Name** property as the `Resource`.

  - Priority 2: Find an existing business data that matches some of the properties specified in the resource mapping and update it. For example, find an existing business data that has a part of the **Name** property as the `Resource`.

  - Priority 3: Create a new business data for a business object that matches all properties specified in the resource mapping. For example, find an existing business object that has the same **Name** property as the `Resource` and create a business data for that business object.

  - Priority 4: Create a new business data for a business object that matches some of the properties specified in the resource mapping. For example, find an existing business object that has a part of the **Name** property as the `Resource` and create a business data for that business object.

- Specify how the data types (such as `string`, `integer`, etc.) for `Resources` are mapped to the property types for the object class property types available for the class `BusinessData`. In the Alfabet class model, each object class property has a **Property Type** attribute that has a value such as `String`, `StringArray`, `Integer`, `Real`, `Text`, etc. For example, the **Name** and **Version** properties both have the **Property Type** = `String`.) You must specify this mapping for the business data created/updated in the context of each object class stereotype of the class `Service` used in the API portal integration.

- Specify how the data type types (such as `string`, `integer`, etc.) for `Resources` mapped to the business data attribute types relevant for Alfabet business data. Business data attributes are attributes that describe the business data. Business data attributes are captured the `Attributes` property of the class `BusinessData`. The `Type` attribute of the class `BusinessDataAttribute` is configured by your solution designer for your solution configuration via the protected enumeration **BusinessDataAttributeType**. Each enumeration item defined for the protected enumeration **BusinessDataAttributeType** constitutes a type of business data attribute.

To specify the XML object **ServiceResourceMapping** for integration with webMethods API Gateway:

1) In Alfabet Expand, go to the **Presentation** tab and expand the explorer nodes **XML Objects** > **IntegrationSolutions**.

2) Right-click the XML object **ServiceResourceMapping** and select **Edit XML...**. The XML object opens.

3) The names of the `Resource` in webMethods API Gateway may have one or more prefixes that you may want to exclude from the mapping. To specify which prefixes to ignore, add a child XML element **PrefixesToIgnoreOnResourceMapping** to the root XML element **ServiceResourceMapping**.

4) For each prefix that you want to specify, add a child XML element **Value** to the XML element **PrefixesToIgnoreOnResourceMapping**. In the XML element **Value**, enter the prefix that should be ignored.

> 👁 For example, a `Resource` named `Employee` may have a name that constitutes the path in the API gateway where the `Resource` is located. For example, the name of the looks like a resource path such as `<StoreName>/<APIName>/Employee`. In this case, you may want to add the `ResourceEmployee` as a business data to A but ignore the prefixes `Store` and `API` when. In this case, you would specify:

```
...

<PrefixesToIgnoreOnResourceMapping>

    <Value>store</Value>

    <Value>api</Value>

</PrefixesToIgnoreOnResourceMapping>
```

5) For each rule to map the `Resource` to business data, create an XML element **ResourceMapping**. Configure the following XML attributes for the XML element **ResourceMapping**:

   - `Class`: Enter `BusinessData` to search for existing business data to map to. Enter `BusinessObject` to search for existing business objects for which a new business data should be created. New business objects cannot be created in the context of this configuration.

   - `Priority`: Enter an integer to specify the priority of this resource mapping. The resource mapping definition with `Priorty="1"` will be searched for first, `Priorty="2"` will be searched second, etc.

   - `Properties`: Specify one or more object class properties of the business data/business object that must be searched to find a match. The name of the `Resource` typically uses the following syntax: `{Name}\{Property1}\{Property2}`. You can specify multiple object classes to facilitate finding a match. For example, `Properties="{Name}/{Version}"`.

   - `MatchType`: Enter `FullString` if all values in the XML attribute `Properties` must be met in order to find the business data or business object. Enter `SubString` if only some of the values in the XML attribute `Properties` must be met in order to find the business data or business object. If no match is found, the `Resource` will be ignored and business data will not be updated/created.

6) To map the data property information for business data, create an XML element **DataTypeMappings**.

7) For each object class stereotype of the class `Service` specified in the XML element `AlfabetStereotype` XML object **APIGatewayConfig**, create an XML element **DataTypeMapping** that is a child of the XML element **DataTypeMappings**.

8) In the XML element **DataTypeMapping**, create an XML attribute `AlfabetStereotype` and enter the name of the object class stereotype configured for the class `Service` that you want to specify.

9) In the XML element **DataTypeMapping**, create an XML element **DataType** for each property type that is relevant for the object class stereotype specified in the XML attribute `AlfabetStereotype`. Alfabet property types are `String`, `Boolean`, `Date`, `Integer`, `Text`, etc. The property type is specified in the **Property Type** attribute for each relevant property of the object class `Service`. (For example, the **Name** and **Version** properties for the class `BusinessData` both have the **Property Type** = `String`.) Configure the following XML attributes for the XML element **DataType**:

   - `AlfabetPropertyType`: Enter the type for the Alfabet object class property (`String`, `Boolean`, `Date`, `Integer`, `Text`, etc.) to map to the data type of the API specified in the XML attribute `ExternalPropertyType`.

   - `ExternalPropertyType`: Enter the data type for the API to map to the Alfabet object class property type defined in the XML attribute `AlfabetPropertyType`.

10) In the XML element **DataTypeMapping**, create an XML element **AttributeDataType** for each type of business data attribute that is relevant for Alfabet business data. Business data attributes are attributes that describe the business data. Business data attributes are captured via the `Attributes` property of the class `BusinessData`. The `Type` attribute of the class `BusinessDataAttribute` is configured by your solution designer for your solution configuration via the protected enumeration **BusinessDataAttributeType**. Each enumeration item defined for the protected enumeration **BusinessDataAttributeType** constitutes a type of business data attribute. Configure the following XML attributes for the XML element **AttributeDataType**:

- `AlfabetAttributeType`: Enter the enumeration item specified for the protected enumeration **BusinessDataAttributeType** to map to the `Resource` data type specified int the XML attribute `ExternalPropertyType`.

- `ExternalPropertyType`: Enter the `Resource` data type to map to the business data attribute type specified in the XML attribute `AlfabetPropertyType`.

11) In the toolbar of Alfabet Expand, click the **Save** button to save your changes.

Chapter 8:   Configuring Interoperability with Google's Apigee API Platform Tools

Software AG offers interoperability betweeen the Alfabet product and Google's Apigee™ product so that your enterprise can review and test APIs as planned technical services in the Alfabet user interface. The Apigee product serves as a platform to design, monitor, and publish APIs. You can configure interoperability between Alfabet and Apigee and thus provide integrated functions to import APIs to Alfabet as planned technical services, document and describe those planned technical services, and export the modified technical service definition back to Apigee.

APIs may be imported in the Alfabet user interface via the **Technical Services Registry Services** view (`COMSR_Services`) or the **Technical Services Registry Services - Filtered** (`COMSR_ServicesExt`) view. Once the technical services have a specified release status, they can be exported from Alfabet to Apigee in the user interface via the **Technical Services** view (`COM_TechServices`) view. By the click of a button, users can navigate from Alfabet to Apigee and view the API definition directly in the Apigee interface.

> APIs may also be imported to Alfabet by means of an ADIF import scheme. Technical services cannot be exported via an ADIF scheme to Apigee. If APIs shall be imported via ADIF, then the ADIF scheme must be created via an assistant. The assistant defines the correct import of data from Apigee to temporary tables in ADIF. Further configuration is required in order to update the Alfabet database tables with the data in the temporary database tables. The import may include mappings that are not included in standard mapping via the functionalities on the user interface. For more information, see [Creating an ADIF Import Scheme for Import from Apigee](#) below.

Apigee API proxies and their references are mapped to predefined object classes in Alfabet. When an API is imported to Alfabet via the views in the Alfabet user interface, a technical service will be created in Alfabet. If any endpoints or methods are specified for the API, objects of the object classes specified in the table below will be created in Alfabet:

Apigee Object	Alfabet Object Class
API Proxy	**Technical Service** (`Service`) Please note that additional specification of the mapping of `APIs` to technical services must be configured in the XML object **APIRepositoryConfig**.
Endpoint	**Technical Service Operation** (`ServiceOperation`)
Target Endpoint	**Business Data** (`BusinessData`) Please note that additional specification of the mapping of target endpoints to business data must be configured in the XML object **ServiceResourceMapping**.
Method	**Technical Service Operation Method** (`ServiceOperationMethod`)
Method Parameter	**Technical Service Operation Method Parameter** (`ServiceOperationMethodParameter`)

The following information is available to configure interoperability with Apigee:

- [Overview of the Configuration Required for Interoperability with Apigee](#)

- [Configuring the Class Model for Interoperability with Apigee](#)

- [Configuring Connections and API Proxy Mapping for Apigee Integration](#)

  - [Sending Requests to Apigee via a Proxy Server](#)

- [Creating Apigee Data Connections](#)

- [Creating an ADIF Import Scheme for Import from Apigee](#)

Overview of the Configuration Required for Interoperability with Apigee

The following is an overview of all configuration steps required for interoperability with Apigee:

1) Configure object class stereotypes for the object classes `Service`, `ServiceOperation`, and `ServiceOperationMethod`. You will later map the object class stereotypes of the class `Service` to the API proxies available in Apigee. This is described in more detail in the section [Configuring the Class Model for Interoperability with Apigee](#).

2) Map the object class stereotypes for the object classes `Service`, `ServiceOperation`, and `ServiceOperationMethod` 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. For more information, see the section *Configuring Object Class Stereotypes for Technical Services* in the chapter *Configuring Alfabet Functionalities Implemented in the Solution Environment* in the reference manual *Configuring Alfabet with Alfabet Expand*.

3) Configure release statuses for the object class `Service` as well as the object class stereotypes in the XML object **ReleaseStatusDefs**. Please note that the release status `HandoverStatus` must be specified in order to be able to export the technical services in Alfabet to Apigee. This is described in more detail in the section [Configuring the Class Model for Interoperability with Apigee](#).

4) Specify one or more connections to Apigee in the XML object **APIRepositoryConfig**. Here you also specify how API proxies are mapped to the object class stereotypes configured for the object class `Service`, and the `HandoverStatus` required for the technical services in order to export them to Apigee. This is described in more detail in the section [Configuring Connections and API Proxy Mapping for Apigee Integration](#).

5) If server variables are used to configure the connection in the XML object **APIRepositoryConfig**, the server variables must be specified for the server alias in the Alfabet Administrator. For more information about configuring server variables, see the section [Configuring Server Variables for Integration and Interoperability Solutions](#).

6) Specify the mapping of data associated with target endpoints to the objects of the class **Business Data**. This is specified in the XML object **ServiceResourceMapping**. This is described in more detail in the section [Configuring Connections and API Proxy Mapping for Apigee Integration](#).

7) The attributes **Assembly** and **Assembly Class** available for the **ServiceRegistryManager** solution manager must be specified to ensure interoperability. The XML object **ServiceRegistryManager** is available in Alfabet Expand.The following values must be specified:

   - **Assembly**: `ITPlan`

   - **Assembly Class**: `ITPlanSolution.GenericServiceRegistryManager`

8) Create Apigee data connection objects in the **Apigee Data Connection** view in the **Integration Solutions Configuration** functionality in the Alfabet user interface. An Apigee data connection object should be created for each Apigee connection configured in the XML object **APIRepositoryConfig**. The Apigee data connection objects allow a user to specify the connection to the relevant Apigee instance to use when exporting/synchronizing a technical service in the **Technical Services** page view (`COM_TechServices`). The creation of Apigee data connection objects is described in the section Creating Apigee Data Connections.

9) .The relevant user profiles must be configured to include the **Technical Services Registry Services - Filtered** page view (`COMSR_ServicesExt`) available on the root node of the **Components** explorer (`COM_Explorer`) and **Technical Services** page view (`COM_TechServices`) available in the **Components** object view. (This view is only available for components of the type `Service`). The **Technical Services Registry Services** page view (`COMSR_Services`) available on the root node of the **Components** explorer (`COM_Explorer`) should not be visible for the user profiles accessing webMethods API Portal. For more information about configuring user profiles, see the chapter *Configuring User Profiles for the User Community* in the reference manual *Configuring Alfabet with Alfabet Expand*.

10) If APIs are to be imported to Alfabet by means of ADIF import, you must create an ADIF import scheme using the `ApigeeImport_Assistant` in order to import the APIs to temporary database tables. Your enterprise must configure the update of the Alfabet database tables with the data in the temporary database tables. The ADIF import must then be executed via one of the available import methods. For more information about creating the ADIF import scheme via the assistant, see Creating an ADIF Import Scheme for Import from Apigee. The configuration of ADIF import schemes and the execution of ADIF imports are described in detail in the reference manual *Alfabet Data Integration Framework*.

Configuring the Class Model for Interoperability with Apigee

The following configuration is required in order to provide interoperability with Apigee:

- Configure object class stereotypes for the object class `Service`. Each API proxy type that should be displayed in Alfabet must be mapped to an object class stereotype of the class `Service` in the XML element **AssetTypeMappings** in the XML object **APIRepositoryConfig**. Therefore, before the XML object **APIRepositoryConfig** can be specified, you must first configure the relevant object class stereotypes for the class `Service`. For detailed information about how to configure object class stereotypes, see the section *Configuring Object Class Stereotypes for Object Classes* in the chapter *Configuring the Class Model* in the reference manual *Configuring Alfabet with Alfabet Expand*.

  For example, the object class stereotypes for the object class `Service` could be specified as follows:

  ```
  <ClassStereotypes>
    <Stereotype Name="SOAPService" Caption="SOAP Service"
          CaptionPlural="SOAP Services" Comments="" HasMandates="false" />
    <Stereotype Name="DataService" Caption="DataService"
          CaptionPlural="Data Services" Comments="" HasMandates="false" />
    <Stereotype Name="RESTService" Caption="REST Service"
          CaptionPlural="REST Services" Comments="" HasMandates="false" />
    <Stereotype Name="XMLService" Caption="XML Service"
          CaptionPlural="XML Services" Comments="" HasMandates="false" />
  </ClassStereotypes>
  ```

- Configure object class stereotypes for the object class `ServiceOperation`. Each endpoint associated with an imported API will be created as a technical service operation in Alfabet.

- Configure object class stereotypes for the object class `ServiceOperationMethod`. Each method associated with an imported API will be created as a technical service operation method in Alfabet.

- Configure the mapping of technical service stereotypes (based on class `Service`), technical service operation stereotypes (based on class `ServiceOperation`), and technical service operation method stereotypes (based on class `ServiceOperationMethod`) 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. For more information about mapping the object class stereotypes defined for technical services and technical service operations, see the section *Configuring Object Class Stereotypes for Technical Services* in the reference manual *Configuring Alfabet with Alfabet Expand*.

- Configure release statuses for the object class `Service` as well as all relevant object class stereotypes for the object class `Service`. Please note that the release status `HandoverStatus` must be specified in order to be able to export the technical services in Alfabet to Apigee. Please keep the following in mind:

  - The release status set for each relevant object class stereotype must be configured in the XML object **ReleaseStatusDefs**. A release status definition must be created for the object class `Service` as a whole as well as each object class stereotype. The release status definition for the object class must include the complete set of release statuses configured for its object class stereotypes. 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 more information about configuring release statuses, see the section *Configuring Release Status Definitions for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand*.

Configuring Connections and API Proxy Mapping for Apigee Integration

Multiple connections to Apigee can be defined in order to address the needs of federated organizations. The XML object **APIRepositoryConfig** allows you to configure the connections to Apigee instances as well as how the APIs should be imported to Alfabet. You can configure the information necessary so that users can import and export APIs to Alfabet in the Alfabet user interface and/or how to import the APIs via an ADIF import scheme. Please note the following:

- Apigee API proxies may be imported via the **Technical Services Registry Services** view (`COMSR_Services`) or the **Technical Services Registry Services - Filtered** (`COMSR_ServicesExt`) view. If the import of APIs should occur via the Alfabet user interface, you must map the API proxy types to the relevant object class stereotypes configured for the class `Service` in the XML object **APIRepositoryConfig**. You must also specify the relevant `HandoverStatus` release status that a technical service must have in order to be exported from Alfabet to Apigee.

⚠ If you want to enter a string that contains special characters in the XML object **APIRepositoryConfig**, you must replace the special characters with respective XML compliant code, for example:

  - > for >

  - < for <

  - " for "

  - [for [

-] for]

The configuration of the connections is carried out in the XML object **APIRepositoryConfig** by means of the root XML element **APIRepositoryConfig** and its child XML element **RepositoryConnections**. Once the configuration of the XML object **APIRepositoryConfig** is complete, connection objects must also be created in the **Apigee Data Connections** view available in the **Integration Solutions Configuration** functionality in the Alfabet interface.

The root XML element **APIRepositoryConfig** has two child elements:

- **Proxy**: This XML element allows you to configure how to send requests to Apigee instances via a proxy server. The configuration of the XML element **Proxy** is described in the section.

- **RepositoryConnections**: This XML element allows you to configure multiple connections to Apigee repositories. A child XML element **RepositoryConnection** should be configured for each connection to an Apigee repository. Each XML element **RepositoryConnection** has attributes for the definition of the connection data to connect to the Apigee repository and a child element **AssetTypeMappings**, which allows you to specify the mapping of API Proxie types from Apigee to Alfabet when imported via the Alfabet user interface.

> The XML object usually includes an example definition. In addition, a template is available via the attribute **XML Template** of the XML object. The template can be copied to the XML object to avoid having to write the configuration manually. The following information describes a configuration from scratch. With a sample configuration, you must edit the existing XML elements rather than add them to the XML configuration.

You can configure multiple connections from Alfabet to multiple Apigee repositories. To define a connection to an Apigee repository:

1) In Alfabet Expand, go to the **Presentation** tab and expand the explorer nodes **XML Objects** > **IntegrationSolutions**.

2) Right-click the XML object **APIRepositoryConfig** and select **Edit XML...**. The XML object opens.

> The XML element **Proxy** allows you to configure how to send requests to Apigee instances via a proxy server. This is optional. If you have configured multiple connections and each connection shall use a different proxy, you can add additional proxies to your proxy configuration and specify the relevant proxy definition in the Apigee connection configuration. The configuration of the XML element **Proxy** is described in the section Sending Requests to Apigee via a Proxy Server.

3) Add a child XML element **RepositoryConnections** to the root XML element **APIRepositoryConfig**.

4) Add a child XML element **RepositoryConnection** to the XML element **RepositoryConnections** for each repository connection that you will define.

5) Configure a connection to each Apigee repository by setting the following XML attributes for the XML element **RepositoryConnection**:

   - `Name`: Enter a unique name for the connection to the Apigee repository.

   - `Repository`: Enter the name of the Apigee repository on the Apigee instance that you would like to access with this connection.

   - `ServerURL`: Enter the URL of the Apigee Edge RESTful management API server without the endpoint specification or enter the relevant server variable name.

> (i) In some of the XML attributes, server variables can be used to read the value of the at-
> tribute at runtime from the server alias configuration of the Alfabet Web Application
> when a connection to API Portal is established. For more information about using server
> variables, see the section Configuring Server Variables for Integration and Interoperabil-
> ity Solutions.

- `UserName`: Enter your enterprise's email address for access to the Apigee Edge RESTful management API or enter the relevant server variable name.

- `Password`: Enter your enterprise's password for access to the Apigee Edge RESTful management API or enter the relevant server variable name.

- `ServiceViewURL`: Enter the prefix for the complete link when navigating to the Apigee user interface from Alfabet when the user clicks the **Open Service's View in Service Registry** button in the **Technical Services Registry Services - Filtered** page view. Enter the URL to access the service view in Apigee or enter the relevant server variable name. For example: `https://apigee.com/platform`)

- `MaxThreadCount`: Enter the maximum number of parallel threads for calls to Apigee. This specification is optional. The default settings usually provide optimal import behavior. The setting should only be changed if problems are encountered during integration.

- `PageSize`: The maximum number of API Proxy data imports via one thread. This specification is optional. The default settings usually provide optimal import behavior. The setting should only be changed if problems are encountered during integration.

- `Timeout`: Enter the number of minutes of user inactivity after which the connection to the API portal should be terminated.

6) If API Proxies shall be imported to Alfabet via the Alfabet user interface, add an XML element `AssetTypeMappings` to each XML element **RepositoryConnection**. For each asset type that you want to import to Alfabet, specify a child XML element element `AssetTypeMapping`.

7) Configure each asset type mapping by setting the following XML attributes for the XML element `AssetTypeMapping`:

- `RepositoryAssetType`: Enter the name of the Apigee API Proxy type that should be mapped to the object class stereotype specified in the XML element `AlfabetStereotype`.

- `HandoverStatus`: Specify the release status that the technical services based on the object class stereotype in the XML attribute `AlfabetStereotype` must have in Alfabet in order to be updated to the Apigee repository. For more information about configuring release statuses for Apigee interoperability, see the section *Configuring Release Status Definitions for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand*.

- `AlfabetStereotype`: Enter the name of the object class stereotype of the class `Service` in Alfabet that should be mapped to the API portal asset type specified in the XML element `RepositoryAssetType`.

8) In the toolbar of Alfabet Expand, click the **Save** button to save your changes.

Sending Requests to Apigee via a Proxy Server

Optionally, you can configure the Apigee integration interface to connect to Apigee via a proxy server. This requires the following additional configuration in the XML object **APIRepositoryConfig** for the repository connection configuration described above:

1) In Alfabet Expand, go to the **Presentation** tab and expand the explorer nodes **XML Objects** > **IntegrationSolutions**.

2) Right-click **APIRepositoryConfig** and select **Edit XML...**. The XML object with your Apigee data import definition opens.

3) Add a child XML element `Proxy` to the XML element `APIRepositoryConfig`.

4) 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.

5) In the toolbar of Alfabet Expand, click the **Save** button to save your changes.

If you have configured multiple repository connections and each repository connection shall use a different proxy, you can add additional proxies to your proxy configuration and refer to one of the proxies in the repository connection configuration. The proxy definition above will be used as default if no proxy is assigned to a data connection. The use of an additional proxy requires the following configuration:

1) In Alfabet Expand, go to the **Presentation** tab and expand the explorer nodes **XML Objects** > **IntegrationSolutions**.

2) Right-click **APIRepositoryConfig** and select **Edit XML...**. The XML object with the yourApigee repository connection definition opens.

3) Add a child XML element `AdditionalProxies` to the XML element `Proxy`.

4) For each additional proxy you want to define, add a child XML element `AdditionalProxy` to the XML element `AdditionalProxies` and define the following XML attributes for the XML element `AdditionalProxy`:

- `Name`: Define a unique name for the additional proxy. This name is used to refer to the proxy in the data connection configuration.

- `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.

5) Add an XML attribute `Proxy` to each **RepositoryConnection** XML element that shall use one of the additional proxies. The value of the XML attribute`Proxy` must be identical to the value of the `Name` XML attribute of the XML element **AdditionalProxy**.

6) In the toolbar of Alfabet Expand, click the **Save** button to save your changes.

Creating Apigee Data Connections

The **Apigee Data Connections** view allows you to create semantic definitions for all relevant Apigee connections that have been configured in the XML object **APIRepositoryConfig**. The Apigee data connection definitions are necessary to define which environments in an organization managed in Apigee the connection will target. Users can specify which configured connection to use to synchronize Alfabet technical services with Apigee assets. At least one Apigee data connection definition should be created for each connection configured in the XML object **APIRepositoryConfig**.

Each time the **Technical Services from Repositories** page views (`COMSR_ServicesExt` or `COMSR_Services`) is loaded, each configured connection in the XML object **APIPortalConfig** will be established and the technical services based on Apigee assets will be synchronized with the data in the Apigee repository. If the user wants to synchronize a selected technical service in the **Technical Services** page view (`COM_TechServices`) with the corresponding Apigee asset, the relevant connection definition to use for the synchronization must be selected in the **Master Repository Connection** field in the **Operational Repositories** tab of the **Technical Service** editor. If Alfabet should automatically synchronize the technical services based on APIs from the Apigee repository when the **Technical Services** page view (`COM_TechServices`) is loaded, you must select the **Automatically Update Technical Services** checkbox in the **API Portal Database Connection** editor. If the checkbox is not selected, the user must explicitly update a selected technical service.

To configure an Apigee data connection to display in the **Connection** field in the **Service Registry** tab of the **Technical Service** editor:

1) Go to the **Integration Solutions Configuration** functionality and click the **Apigee Data Connection** node in the **Integration Solutions Configuration** explorer.

2) In the view, click **New** > **Create Apigee Data 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** > **Creat Apigee Data 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 **Apigee Data Connection** editor, define the following fields as needed.

   Basic Data tab:

   • **ID**: Alfabet assigns a unique identification number to each Apigee data connection. This number cannot be edited.

   • **Name**: Enter a unique name for the Apigee data connection. The name should help the user synchronize the technical service to identify the Apigee repository that will be targeted by the connection.

   • **Release Status**: Select the Apigee data connection's current release status.

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

- **Background Color**: Specify the background color to be used for technical services that are associated with this Apigee data connection. In the **Technical Services from Repositories** views, the colored rows represent assets for which technical services have been created in Alfabet. In the **Technical Services** view, the colored rows represent technical services that exist in Alfabet and have been exported to the operational repository.

- **Foreground Color**: Specify the foreground color to be used in conjunction with the background color.

Authorized Access tab:

- **Authorized User**: Click the **Search** icon to assign an authorized user to the selected Apigee 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:

- **Apigee Connection**: Select the relevant connection configured in the XML element **RepositoryConnection** in the XML object **APIRepositoryConfig** that will be established when the user selects this Apigee data connection in the **Master Repository Connection** field in the **Operational Repositories** tab of the **Technical Service** editor.

- **Organization**: Enter the name of the organization in Apigee to that the connection shall be established.

- **Deployed Environments**: After having selected the organization in the **Organization** field, click the **Get Environments** button to view all implemented Apigee environments in the Apigee organization. Select the relevant environment containing the API Proxies to be integrated with technical services in Alfabet via the Apigee data connection.

- **Consider Revision**: Select the checkbox if you would like to import any other than the latest revision of API Proxies. If the checkbox is selected, a user can choose between all revisions in the selected environments for integration. Export to Apigee will create an API Proxy with revision 1. If the checkbox is not selected, the latest revision of an API Proxy in the selected environments will be used for integration.

- **Proxy Name Template**: Technical services created in Alfabet on basis of API Proxies in Apigee will be named according to the naming convention defined with this field. The definition can contain the following variables:

  - `{Name}` will be substituted with the API Proxy name.

  - `{Version}` will be substituted with the API Proxy version.

  By default, the naming convention without considering revisions is: `{Name}_V{Version}`. If revisions are considered, the specified name will be amended with "`r.RevisionNumber`".

4) Repeat this for all configured **Connection** elements in the XML object **APIPortalConfig** that users should be able to select in the **Connection** field in the **Operational Repositories** tab of the **Technical Service** editor.

Creating an ADIF Import Scheme for Import from Apigee

Data from Apigee can be imported via ADIF import jobs to import all data in batch. ADIF import jobs can be used to integrate additional data from Apigee, like Policies or Statistics or integrate the Apigee data in a way that differs from the standard integration provided by the standard functionalities for integration of technical service data.

Part of the ADIF import scheme must be created via an assistant as described below. Prior to creating the ADIF import scheme, the following configuration steps need to be performed:

- Configuring Connections and API Proxy Mapping for Apigee Integration

- Creating Apigee Data Connections

To create an ADIF import job via the assistant:

1) In the **ADIF** tab in Alfabet Expand, right-click the **ADIF Schemes** root node in the explorer or any sub-folder and select **Create Import Scheme**. The new import scheme is added to the explorer. The attribute window of the new import scheme is displayed on the right.

2) In the attribute window, set the following attributes for the ADIF import scheme:

- **Name**: Enter a unique name. The name is used to identify the ADIF import scheme in technical processes. It must be unique and should not contain white spaces or special characters.

- **Assistant**: Select `ApigeeImport_Assistant` from the drop-down list.

- **Import File Required**: Ensure that the value `False` is set.

- **Commit After Run**: If set to `True`, the result of the data import is written persistently to the Alfabet database. If set to `False`, the import process will be rolled back after execution and no changes will be written to the database. Configuration of the automatic start of workflows during import is ignored when **Commit After Run** is set to `False`. It is recommended that you set **Commit After Run** to `False` for a new import scheme to allow debugging without the risk of corrupting the database. After the successful testing of the data import and verification that the resulting changes to the Alfabet database are as expected, you can reset the **Commit After Run** attribute to `True` to perform regular data import.

  ⚠️ Please note the following:

  - Setting the **Commit After Run** attribute rolls back all changes to data records in existing tables caused by DML statements. The creation or deletion of tables is not included in the roll back. For example, if you test an ADIF scheme that is configured to persistently write temporary tables to the database, these temporary tables will be created persistently even if **Commit After Run** is set to `False`. SQL commands of the type **OnActivate** are also excluded from roll back.

  - When new objects are created during an ADIF import job, the data bind mechanism assigns `REFSTR` values for the new objects. When **Commit After Run** is set to `False`, the objects are not created in the database, but nevertheless the `REFSTR` values are regarded as in use and will not be used for data bind in the next ADIF run unless the Alfabet Server or Alfabet Expand application used to process the ADIF jobs is restarted.

  - Changes triggered by **OnActivate** commands are not rolled back if the option **Commit After Run** is set to `False` for the import scheme.

- **Drop Temp Tables**: If set to `True`, all temporary tables are dropped after import. Only the changes to the Alfabet database are stored persistently. If set to `False`, the temporary tables are kept in the database after import is finished. Storing temporary tables persistently is only required for special import/export cycles designed for data manipulation that require input from the temporary tables of a previously set import. In most cases, setting this attribute to `True` is recommended to clean the database of data that is not part of the Alfabet meta-model.

3) In the explorer, right-click the node of the new ADIF import scheme and select **Create ADIF Scheme Details Using the Apigee Import Assistant**. The assistant opens in your standard Web browser.

4) In the **Import Entry** tab, specify the following:

- **Import Entry Name**: Enter a unique name for the import entry that will be created in the ADIF scheme for import of the data from Apigee.

- **Apigee Connection**: Select the Apigee data connection that shall be used for data import from the drop-down list of available Apigee data connections.

- **Data to Import**: Select the data that you would like to import from Apigee. You can import Api Proxies, Application, Statistics, Policies, API Products, Developers and Companies.

5) If you have selected Statistics to be imported, you must further define the way statistics are imported in the **Statistics Filter** tab:

- **Environment**: Select the relevant Apigee environment for generation of statistic data.

- **Dimensions**: Select the dimensions that shall be included in the data.

- **Metrics**: Select the metrics that shall be calculated for the selected dimensions.

- **Functions**: Select an aggregate function to calculate the data. You can return the average (avg), the maximum (max), minimum (min), or sum of all data.

- **Time Unit**: Select the interval that shall be used for calculation of times and dates.

- **Transaction History Period**: Enter the number of days in the past for that the statistics will be imported.

- **Filter Query**: If applicable, enter an Apigee filter query to filter for API proxies to be included into the statistics.

6) Go to the **Import Entry** tab, click **Save Import Entry**. The import entry is listed in the field **Select Import Entry to Edit/Remove**.

> (i) If you have added data in the **Statistics** Filter tab, you do not have to go back to the **Import Entry** tab. You can save the ADIF import entry resulting from your definition directly via the **Save Import Entry** button in the **Statistics Filter** tab.

7) You can now do any of the following:

- **Create a new import entry**: Specify the data as described above in the respective fields with a different name in the **Import Entry Name** field and click **Save Import Entry** to save the next import entry.

- **Edit the existing import entry**: Select the checkbox next to the name of the import entry. Alter the data in the respective fields of the editor and click **Save Import Entry** to save the changes for the existing import entry.

- **Delete the existing import entry**: Select the checkbox next to the name of the import entry. Click **Remove Selected Import Entry**.

8) Click the **OK** button to trigger the ADIF scheme to fetch the specified content from Apigee and close the **Apigee Import Assistant**. The temporary tables will be generated and displayed below the import scheme node in the explorer. Or click **Cancel** to close the editor without triggering the execution of the import scheme. If you click **Cancel**, your settings in the **Apigee Import Assistant** will not be saved.

9) Close the browser window and execute **Meta-Model** > **Reread from Database** in Alfabet Expand. You will then see all automatically generated ADIF elements in the ADIF scheme. Each ADIF import entry will have data import to temporary tables defined via the attribute elements in the **Attributes** folder. The import to the standard Alfabet database tables is not included in the configuration.

10) To complete the integration of the data to the Alfabet database, additional configuration of the import scheme is required to map the imported Apigee data in the temporary tables to temporary tables that will be created for the relevant Alfabet object classes and objects. The configuration required for this last step of the import will depend on how your enterprise chooses to implement the Apigee data in Alfabet. For information about the configuration options available via ADIF import schemes, see the reference manual *Alfabet Data Integration Framework*.

After having defined the ADIF scheme, you must execute it using any of the methods described in the chapter *Configuring ADIF Schemes* of the reference manual *Alfabet Data Integration Framework*.

Chapter 9: Configuring the Creation/Export of Technical Services Based on WSDL and OpenAPI Specification Files

Technical services may be created based on WSDL files as well as OpenAPI Specification Swagger files. The **Technical Services** view (`COM_TechServices`) includes the options **Create Technical Service(s) from WSDL** and **Create Technical Service(s) from OpenAPI Specification**. Technical services in Alfabet can also be exported to WSDL and JSON formats based on the options **Export to WSDL** and **Export to OpenAPI Specification**.

⚠️ Alfabet supports only Swagger 2.0 JSON to create technical services. Earlier versions of Swagger are not supported.

ℹ️ For more information regarding the configuration of technical services, see the section *Configuring Object Class Stereotypes for Technical Services* in the chapter *Configuring Alfabet Functionalities Implemented in the Solution Environment* in the reference manual *Configuring Alfabet with Alfabet Expand*.

Please note the following configuration requirements to make this possible

- The XML object **TechServiceManager** must be configured as described below: in order to map the relevant stereotypes for the class Service has been extended in order to configure the creation of technical services based on WSDL files and OpenAPI Specification-compliant Swagger files. A new XML element `FileExtension` has been added to the XML element `Stereotype` available to specify the object class stereotype for the class `Service`. The value "`.wsdl`" should be entered in the XML element `FileExtensions` specified for a relevant object class stereotype to enable the **Create Technical Service(s) from WSDL** and **Export to WSDL** options in the **Technical Services** view (`COM_TechServices`). The value "`.json`" should be specified to enable t the **Create Technical Service(s) from OpenAPI Specification** and **Export to OpenAPI Specification** options in the **Technical Services** view.

  - For each technical service stereotype, specify which technical service operation stereotypes are permissible.

  - Below the root XML element **TechServiceManager**, create a child XML element `Stereotype` and specify the following XML attributes:

    👁 The following example displays the mapping for the stereotype `SOAPService` configured for the object class `Service`:

```
<TechServiceManager>

  <Stereotype ClassName="Service" Name="SOAPService"
  FileExtension=".wsdl">

    <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>
```

- 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" for a relevant object class stereotype to enable the **Create Technical Service(s) from WSDL** and **Export to WSDL** options in the **Technical Services** view (COM_TechServices). ENE ".json" for a relevant object class stereotype to enable the **Create Technical Service(s) from OpenAPI Specification** and **Export to OpenAPI Specification** options in the **Technical Services** view.

- 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 file extensions.wsdl and.json must not be in the black list specified in the XML object **FileExtensions**. If a white list of file extensions is enabled, the file extensions must be included in the white list. The download of a ZIP file containing multiple WSDL or JSON files is not possible. For more information about configuring the permissibility of file extensions, see the section *Configuring the Permissibility of Files and Web Links in Alfabet* in the reference manual *Configuring Alfabet with Alfabet Expand*.

- A protected or custom wizard must be specified in the **Wizard** attribute in the relevant class setting for the class Service in order to create a technical service based on a WSDL or JSON file. (Note that the definition should NOT be made for the object class stereotype for the class Service.) The specification of the wizard is required in order to ensure that a pre-wizard step is implemented that allows the relevant file to be imported to Alfabet in order to create the technical service.

- A protected or custom wizard may be specified in the **Wizard** attribute in the relevant class setting for the class Service in order to create a technical service based on WSDL or an Open API specification.

Chapter 10: Configuring Integration of Data from Amazon Web Services

Alfabet 10.15 provides an interface to Amazon Web Services that can fetch data about the application and services footprint your company is running in the Amazon Web Services infrastructure (for example, the components that are provided by Amazon Web Services to your company) and create and update objects in the Alfabet database based on this data. How the data is integrated in the Alfabet database is not preconfigured. Customers specify the import logic as needed. The interface to Amazon Web Services (AWS interface) requires the following configuration:

- The connection to Amazon Web Services must be configured in the XML object **AmazonWebServicesConfig** in Alfabet Expand.

- The data to import must be further specified in the XML object **AmazonWebServicesConfig** in Alfabet Expand.

- Import of data from Amazon Web Services is managed via an ADIF import scheme created via an assistant that implements the required basic configuration. The assistant creates a number of ADIF import sets in the ADIF import scheme. The import sets are configured to import data from Amazon Web Services to temporary database tables in the Alfabet database. This ADIF import scheme must be amended with the customer-defined import logic to update the database tables of the Alfabet object class model with the data in the temporary tables.

To execute data import from Amazon Web Services, you must start the ADIF import triggered by the ADIF import scheme. The ADIF import then executes both data retrieval from Amazon Web Services and data integration into the Alfabet database. The ADIF import should be executed in regular intervals to ensure data consistency between Amazon Web Services and Alfabet.

> ⓘ Information about the execution of ADIF schemes is provided in the reference manual *Alfabet Data Integration Framework*.

The following sections describe the configuration steps for the Amazon Web Services interface in the required order of execution:

- Configuring the Connection to Amazon Web Services
  - Sending Request to Amazon Web Services via a Proxy Server
- Configuring Integration of Amazon Web Services Data into the Alfabet database

Configuring the Connection to Amazon Web Services

The connection to the Amazon Web Services must be defined in the XML object **AmazonWebServicesConfig**. Multiple connections to Amazon Web Services can be configured. Data will be imported from all defined Amazon Web Services via the same ADIF import scheme.

The general structure of the XML in the XML object **AmazonWebServicesConfig** is the following:

```
<AmazonWebServicesConfig>
<DataConnections>
    <DataConnection
    Name="Account1"
```

```
        IsActive="true"

        AWSAccessKey="$AWSAccessKey"

        AWSSecretKey="$AWSSecretKey"

        AWSAccount="$AWSAccountNumber">

            <Regions>

                <Region>eu-west-1</Region>

                <Region>us-west-1</Region>

                <Region>ap-southeast-1</Region>

            </Regions>

        </DataConnection>

    </DataConnections>

</AmazonWebServicesConfig>
```

The root XML element `AmazonWebServicesConfig` has a child XML element **DataConnections**. The XML element **DataConnections** can contain multiple child XML element **DataConnection**, each defining a connection to a different access to the Amazon Web Services.

To define connection to the Amazon Web Services:

1) In Alfabet Expand, go to the **Presentation** tab and expand the explorer nodes **XML Objects** > **IntegrationSolutions**.

2) Right-click **AmazonWebServicesConfig** and select **Edit XML...**. The XML object opens.

> The XML object usually includes an example definition. In addition, a template is available via the attribute `XML Template` of the XML object. The template can be copied to the XML object to avoid having to write the configuration manually. The following description describes a configuration from scratch. With a sample configuration, you have to edit rather than add the XML elements described in the following.

3) Add a child XML element `DataConnections` to the root XML element `AmazonWebServicesConfig`.

4) Add a child XML element `DataConnection` to the XML element `DataConnections` and define the following XML attributes for the `DataConnection` element:

- `Name`: The name of the connection to the Amazon Web Services that is used to identify the connection in the ADIF scheme configuration.

- `IsActive`: Set to `true` if data should be imported from the Amazon Web Services instance. Set to `false` if data should currently not be imported from the Amazon Web Services instance.

- `AWSAccessKey`: The access key ID of the access key for connection to the Amazon Web Services.

- `AWSSecretKey`: The secret access key for connection to the Amazon Web Services.

- `AWSAccount`: Enter the ID of your Amazon Web Services account.

> Please note the following:
>
> - In the XML elements for the definition of the Amazon Web Services connection server variables can be used to read the value of the XML attribute at runtime from the server alias configuration of the Alfabet Web Application when a connection to

Amazon Web Services is established. For information about server variables, see Configuring Server Variables for Integration and Interoperability Solutions.

- Amazon recommends to change the access key in regular intervals. For security reasons, you should change your keys and update the configuration of the `AWSSecretKey` and `AWSAccessKey` XML attributes accordingly.

5) Optionally, add a child XML element `Regions` to the XML element `DataConnection` to define the region where the Amazon Web Services is located. For each region, add a child XML element `Region` to the XML element `Regions` and enter the Amazon name of the region as text into this XML element.

6) In the toolbar of Alfabet Expand, click the **Save** button to save your changes.

Sending Request to Amazon Web Services via a Proxy Server

Optionally, you can configure the Amazon Web Services integration interface to send requests to the Amazon Web Services via a proxy server. This requires the following additional configuration in the XML object **Amazon-WebServicesConfig** for the data import configuration described above:

1) In Alfabet Expand, go to the **Presentation** tab and expand the explorer nodes **XML Objects** > **IntegrationSolutions**.

2) Right-click **AmazonWebServicesConfig** and select **Edit XML...**. The XML object with your Amazon Web Services data import definition opens.

3) Add a child XML element `Proxy` to the XML element `AmazonWebServicesConfig`.

4) Define the following XML attributes for the XML element `Proxy`:

- `Url`: Define the URL of the proxy server.

- `UserName`: 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.

5) In the toolbar of Alfabet Expand, click the **Save** button to save your changes.

If you have configured multiple data connections and each data connection shall use a different proxy, you can add additional proxies to your proxy configuration and refer to one of the proxies in the data connection configuration. The proxy definition above will be used as default if no proxy is assigned to a data connection. The use of an additional proxy requires the following configuration:

1) In Alfabet Expand, go to the **Presentation** tab and expand the explorer nodes **XML Objects** > **IntegrationSolutions**.

2) Right-click **AmazonWebServicesConfig** and select **Edit XML...**. The XML object with your Amazon Web Services data import definition opens.

3) Add a child XML element `AdditionalProxies` to the XML element `Proxy`.

4) For each additional proxy you want to define, add a child XML element `AdditionalProxy` to the XML element `AdditionalProxies` and define the following XML attributes for the XML element `AdditionalProxy`:

- `Name`: Define a unique name for the additional proxy. This name is used to refer to the proxy in the data connection configuration.

- `Url`: Define the URL of the proxy server.

- `UserName`: 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.

5) Add an XML attribute `Proxy` to each **DataConnection** XML element that shall use one of the additional proxies. The value of the XML attribute`Proxy` must be identical to the value of the `Name` XML attribute of the XML element **AdditionalProxy**.

6) In the toolbar of Alfabet Expand, click the **Save** ![save icon] button to save your changes.

Configuring Integration of Amazon Web Services Data into the Alfabet database

Data import from Amazon Web Services is performed by running an ADIF import scheme that triggers the data transmission from Amazon Web Services on basis of the configured connection details, stores the incoming data into temporary tables in a predefined way, and integrates the data into the standard object class tables of the Alfabet database as configured in the ADIF import scheme by the customer.

In the ADIF tab of Alfabet Expand, the import scheme can be configured using an assistant. When you open the assistant, you can select one or multiple of the data connections defined in the XML object **AmazonWeb-ServicesConfig**. When closing the assistant, the ADIF scheme automatically filled with the required entries to read standard data from Amazon Web Services into temporary database tables. To define integration of the data into the Alfabet database, the entries resulting from the semi-automatic creation of the ADIF import scheme elements must be amended with the information about how data from the temporary tables in integrated into the Alfabet database. The configuration required for this last step of the import highly depends on your demands.

(i) This section only describes the handling of the Amazon Web Services assistant. Information about how to configure an ADIF import scheme to write data from the temporary database tables to the Alfabet database is given in detail in the reference manual *Alfabet Data Integration Framework* and is not repeated here.

Using the Amazon Web Services assistant does not only provide semi-automatic configuration of the ADIF import scheme. It also establishes the mechanisms required to open the connection to Amazon Web Services and import the data when the ADIF import on basis of the ADIF import scheme is executed.

After having defined a valid configuration for data import in the XML object `AmazonWebServicesConfig`, you must define a new ADIF import scheme to configure data integration using the Amazon Web Services import assistant:

1) In the **ADIF** tab in Alfabet Expand, right-click the **ADIF Schemes** root node in the explorer or any sub-folder and select **Create Import Scheme**. The new import scheme is added to the explorer. The attribute window of the new import scheme is displayed on the right.

2) In the attribute window, set the following attributes for the ADIF import scheme:

- **Name**: Enter a unique name. The name is used to identify the ADIF import scheme in technical processes. It must be unique and should not contain white spaces or special characters.

- **Assistant**: Select `AWSImport_Assistant` from the drop-down list.

- **Import File Required**: Select `False`.

- **Commit After Run**: If set to `True`, the result of the data import is written persistently to the Alfabet database. If set to `False`, the import process will be rolled back after execution and no changes will be written to the database. Configuration of the automatic start of workflows during import is ignored when **Commit After Run** is set to `False`. It is recommended that you set **Commit After Run** to `False` for a new import scheme to allow debugging without the risk of corrupting the database. After the successful testing of the data import and verification that the resulting changes to the Alfabet database are as expected, you can reset the **Commit After Run** attribute to `True` to perform regular data import.

  ⚠ Please note the following:

  - Setting the **Commit After Run** attribute rolls back all changes to data records in existing tables caused by DML statements. The creation or deletion of tables is not included in the roll back. For example, if you test an ADIF scheme that is configured to persistently write temporary tables to the database, these temporary tables will be created persistently even if **Commit After Run** is set to `False`. SQL commands of the type **OnActivate** are also excluded from roll back.

  - When new objects are created during an ADIF import job, the data bind mechanism assigns `REFSTR` values for the new objects. When **Commit After Run** is set to `False`, the objects are not created in the database, but nevertheless the `REFSTR` values are regarded as in use and will not be used for data bind in the next ADIF run unless the Alfabet Server or Alfabet Expand application used to process the ADIF jobs is restarted.

  - Changes triggered by **OnActivate** commands are not rolled back if the option **Commit After Run** is set to `False` for the import scheme.

- **Drop Temp Tables**: If set to `True`, all temporary tables are dropped after import. Only the changes to the Alfabet database are stored persistently. If set to `False`, the temporary tables are kept in the database after import is finished. Storing temporary tables persistently is only required for special import/export cycles designed for data manipulation that require input from the temporary tables of a previously set import. In most cases, setting this attribute to `True` is recommended to clean the database of data that is not part of the Alfabet meta-model.

3) In the explorer, right-click the node of the new ADIF import scheme and select **Create ADIF Scheme Details Using the AWS Import Assistant**. A warning dialog is displayed.

  ⚠ Using the assistant with an existing, already configured ADIF import scheme will overwrite all automatically generated parts of the ADIF scheme. If changes have been made to these import sets/entries, these changes will get lost.

4) Click **Yes**. The assistant opens in your standard Web browser.

5) In the field **Select Data Connections to Execute**, all XML elements `DataConnection` in your `AmazonWebServicesConfig` XML object will be displayed. Click on each data connection for that data shall be integrated with this ADIF import scheme.

6) After having selected all relevant `DataConnection` XML elements, click the button **OK** below the field. A new page is displayed that reminds you to rescan the ADIF tree in Alfabet Expand.

7) Close the browser window and return to the **ADIF** tab of Alfabet Expand.

8) Right-click the ADIF import scheme and select **Rescan Tree** in the context menu. You will then see all automatically generated ADIF elements in the ADIF scheme. Each ADIF import entry will have data import to temporary tables defined via the attribute elements in the **Attributes** folder. The import to the standard Alfabet database tables is not included in the configuration.

9) Configure the data integration to the standard Alfabet database tables according to your demands. For information about the configuration options available via ADIF import schemes see the reference manual *Alfabet Data Integration Framework*.

Chapter 11:   Configuring Integration of Data Between ServiceNow and Alfabet

Alfabet provides an interface with ServiceNow to import data available in ServiceNow to Alfabet and vice versa. This functionality allows you to import data about ServiceNow hosted services to the Alfabet database in order to include them in your IT management and planning processes and to import data about planned and approved services to ServiceNow.

Data transmission is performed via RESTful API calls that are triggered by Alfabet ADIF jobs. While the import and export is triggered from Alfabet side, ServiceNow must still be configured to provide the required access rights and, in case of import of data from ServiceNow into Alfabet, to provide the data in the required format. Please note that the configuration of the ServiceNow product is not included in this documentation. The documentation provided by Alfabet addresses the configuration required in Alfabet to integrate data between ServiceNow and Alfabet. For details about configuring the ServiceNow product, please consult the documentation delivered with the ServiceNow product.

> ⚠️   Integration of Alfabet with ServiceNow will not work if authentication requests from Alfabet are redirected to any SSO platform.

The following information is available:

- Configuring Integration of Data from ServiceNow

    - Configuring Data Transmission from ServiceNow to Alfabet

    - Configuring Integration of ServiceNow Data into the Alfabet Database

    - Changing an Existing Configuration for ServiceNow Integration

- Configuring Integration of Alfabet Data Into ServiceNow

    - Defining Data to Export from the Alfabet database Via a Configured Report

    - Configuring Data Transmission from Alfabet to ServiceNow

    - Configuring the ADIF Export Scheme for Data Export to ServiceNow

    - Changing an Existing Configuration for ServiceNow Integration

- Sending Requests to ServiceNow via a Proxy Server

- Sending Requests to ServiceNow via an API Gateway

Configuring Integration of Data from ServiceNow

If you are using ServiceNow®, you can import data about your ServiceNow hosted services to the Alfabet database to include them in your IT management and planning processes. There is no predefined object class for storing the incoming data. Which object classes and object class attributes are relevant for storing the information is completely customer configured via the integration interface.

Data about ServiceNow hosted services can be imported from ServiceNow database tables, ServiceNow database views and ServiceNow reports. For integration of data from ServiceNow to the Alfabet database, the existing interface must be configured with the following information:

- Access information for establishing the connection to ServiceNow. Data transfer is done via RESTful web services.

- Specification about the source and kind of data to be imported and HTTP transmission details.

- Mapping between ServiceNow data types and Alfabet data types.

- The object classes and object class properties that are the target of the import in the Alfabet database and how object data is integrated into the existing object data in the Alfabet database.

The access to ServiceNow and the specification of data import is configured in the XML object `ServiceNowImportConfig`.

This configuration is required as prerequisite for the second configuration step that defines the creation and update of object data in the Alfabet database. This data integration is done via an ADIF import and one or multiple ADIF import schemes must be created and configured according to the requirements.

To execute data import from ServiceNow, you must start the ADIF import triggered by the ADIF import scheme. The ADIF import executes both data retrieval from ServiceNow and data integration into the Alfabet database. The ADIF import should be executed in regular intervals to ensure data consistency between ServiceNow and Alfabet.

> (i) Information about the execution of ADIF schemes is provided in the reference manual *Alfabet Data Integration Framework*.

The following sections describe the configuration steps for the ServiceNow interface in the required order of execution:

- [Configuring Data Transmission from ServiceNow to Alfabet](#)

- [Configuring Integration of ServiceNow Data into the Alfabet Database](#)

- [Changing an Existing Configuration for ServiceNow Integration](#)

Configuring Data Transmission from ServiceNow to Alfabet

Configuration of data import is done in the XML object **ServiceNowImportConfig**. The XML object allows connections to multiple ServiceNow sources to be defined. The following displays the general structure of the XML in the XML object **ServiceNowImportConfig**:

```
<ServiceNowConfig>

    <DataTransferMappings>

        <DataTransferMapping>

            <ServiceAccessInfo>...</ServiceAccessInfo>

            <DataConnectivityInfo>...</DataConnectivityInfo>

        </DataTransferMapping>

    </DataTransferMappings>

    <DataTypeMappings>...</DataTypeMappings>
```

The root XML element `ServiceNowConfig` of the XML objects contains two child elements:

- `DataTransferMappings`: The XML element `DataTransferMappings` can contain one or more child elements `DataTransferMapping`. Each child element `DataTransferMapping` contains the definition of a connection to a different ServiceNow server. The definition includes the specification of the connection to the REST API of the web services of the ServiceNow server in a child element `ServiceAccessInfo`, and the specification of data to be transferred in a child element `DataConnectivityInfo`. Different data transfer conditions can be defined for different data types in the `DataConnectivityInfo` element.

- `DataTypeMappings`: The XML element `DataTypeMappings` contains the mapping of imported data types to the data types that are valid in the Alfabet database. The data is converted to the specified data types during data import. The data type definitions are valid for all data connections on all data transfer mappings.

To define the data import from a ServiceNow server:

1) In Alfabet Expand, go to the **Presentation** tab and expand the explorer nodes **XML Objects** > **IntegrationSolutions**.

2) Right-click **ServiceNowImportConfig** and select **Edit XML...**. The XML object opens.

   (i) The XML object usually includes an example definition. In addition, a template is available via the **XML Template** attribute of the XML object **ServiceNowImportConfig**. The template can be copied to the XML object to avoid needing to write the configuration manually. The following describes a configuration from scratch. With you begin with a sample configuration, you must edit rather than add the XML elements.

3) Add a child XML element `DataTransferMappings` to the root XML element `ServiceNowConfig`.

4) Add a child XML element `DataTransferMapping` to the XML element `DataTransferMappings` and define the following XML attribute for the `DataTransferMapping` element:

   - `Name`: The name of the ServiceNow server connection that is used to identify the connection in the ADIF scheme configuration.

   - `IsActive`: Set to `true` if data should be imported from the ServiceNow instance. Set to `false` if data should currently not be imported from the ServiceNow instance.

5) Add a child XML element `ServiceAccessInfo` to the XML element `DataTransferMapping` to define the information required to establish a connection to the ServiceNow server.

6) Add an XML attribute `AutomaticSwitchToBasic` to the XML element `ServiceAccessInfo` and set it to `true` to connect via basic authentication if OAuth authentication fails.

7) The information required to establish the connection to the REST API of the web services of the ServiceNow server must be added as strings to the following child XML elements, which have to be added to the `ServiceAccessInfo` XML element:

   - `service`: The URL of the ServiceNow instance.

   - `apipath`: The path to the REST API at the ServiceNow instance. Enter `/api/now/v1/`.

   - `oauthpath`: `OAuth` is required for authentication at ServiceNow. You must specify the path to the `OAuth` endpoint with this XML element.

   - `username`: The user name for access to ServiceNow.

     ⚠ The user must have the required access permissions to perform the following api calls:

     - Authentication call: https://<BaseAddress>/oauth_token.do

- Reading of tables: https://<BaseAddress>/<tableName>.do?SCHEMA

- Reading of database views:
  https://<BaseAddress>/<databaseviewname>.do?CSV&sysparm_record_count=
  1

- Reading of reports:
  https://<BaseAddress>/sys_report_template.do?CSV&sysparm_record_count=1
  &jvar_report_id=<reportid>

- Reading of reports:
  https://<BaseAddress>/api/now/v1/table/sys_report/<reportID>

Sufficient access would For example, be granted with the following access rights:

- `rest_service` for Read access to data via a RESTful API call.

- `soap_query` for Read access to schema information of ServiceNow database tables or database views. This permission is not required for import from reports.

- `itil` for Read access to the relationship tables `cmdb_rel_ci` and `cmdb_rel_type`.

- `itil_admin`

- `password`: The password for access to ServiceNow.

- `client_id`: Enter the Client ID for the `OAuth` endpoint at ServiceNow.

- `client_secret`: Enter the Client Secret for the `OAuth` endpoint at ServiceNow into this XML element.

  (i) Server variables can be used to define the ServiceNow server connection. The server variables read the value of the XML element at runtime from the server alias configuration of the Alfabet Web Application when a connection to ServiceNow is established. For information about server variables, see Configuring Server Variables for Integration and Interoperability Solutions.

8) Add a child XML element `DataConnectivityInfo` to the XML element `DataTransferMapping` to define the information required to request data from the ServiceNow server. The XML element `DataConnectivityInfo`. Define the default data transmission parameters for the data requests by setting the following XML attributes of the XML element `DataConnectivityInfo`:

- `DefaultPageSize`: You can define a limit for data to be transmitted simultaneously on the data connection for the import from database tables and database views containing many records. Enter the number of records that shall be transmitted simultaneously. This value will be valid for all data connections that do not have a `PageSize` attribute defined. If a data limitation is not defined for the data connection nor via the XML attribute `DefaultPageSize`, the value 100 will be set per default. The XML attribute `DefaultPageSize` is not supported for import from reports.

- `DefaultTimeout`: Define the default HTTP request timeout that shall be used for all data connection that do not include a timeout definition via the XML attribute `Timeout` of the XML element `DataConnection`. If a timeout is not defined for the data connection nor via the XML attribute `DefaultTimeout`, the default value defined for the Alfabet Web Application will be used.

- `DefaultMaxRecordCount`: Define the maximum number of records that shall be transmitted for a request to ServiceNow on a data connection if the data is read from a ServiceNow report. The

value defined here is used as the default for all data connections that do not have a maximum number of records to be transmitted defined via the XML attribute `MaxRecordCount` of the XML element `DataConnection`. If the maximum number of records is not defined for the data connection nor via the XML attribute `DefaultMaxRecordCount`, the value 1000 will be set per default. The XML attribute `DefaultMaxRecordCount` is not supported for the import from database tables and database views.

9) Add one or multiple child elements `DataConnection` to the XML element `DataConnectivityInfo`. Each XML element `DataConnection` bundles data to be imported via one request to the ServiceNow server. Define the data transmission parameters for the data request by setting the following XML attributes of the XML element `DataConnection`:

- `DataConnectionName`: Enter a unique name for the data connection. This name is used to identify the data connection in the configuration of the ADIF scheme.

- `FetchType`: Enter the type of data source used to retrieve data via the connection. Allowed values are:

    - `Table`: To read data from a ServiceNow database table.

    - `DatabaseView`: To read data from a ServiceNow database view.

    - `Report`: To read data from a ServiceNow report.

- `UseJson`: The RESTful services API of ServiceNow can provide all data in comma-separated format. In addition, the RESTful services API V1 of ServiceNow can provide data from tables in JSON format and the RESTful services API V2 of ServiceNow can provide data from tables and database views in JSON format. Set this XML attribute to `true` if ServiceNow data shall be requested in JSON format and the XML attribute `FetchType` is set to a value supporting JSON data in the ServiceNow RESTful services.

- `Timeout`: Define the HTTP request timeout for the data connection. If no timeout is defined, the default timeout defined in the XML element `DataConnectivityInfo` is used.

- `PageSize`: Define the maximum number of records to be transmitted simultaneously on the data connection. This attribute is not supported for import from reports and database views.

    i   It is recommended that the XML attribute `PageSize` is set to a value lower than 500. An excessively high number of records may lead to transmission problems.

- `MaxRecordCount`: Define the maximum number of records that shall be transmitted for a request to ServiceNow for data import from reports. If the XML attribute `PageSize` is not set or not supported, the value for the XML attribute `MaxRecordCount` should not exceed 10.000. This attribute is not supported for import from database tables and database views.

- `IncludeDisplayValue`: Set the XML attribute `IncludeDisplayValue` to `true` if you want to include the defined display value for integers in ServiceNow, like For example, for the priority of change requests, in addition to the integer. The display value as string is written into a separate column of the temporary table created for data import. The integer value and the display value will be imported into two different columns of the temporary table with the extensions "_IV" (integer value) and "_DV" (display value) to the column name. If this XML attribute is not set or set to `false`, only the integer value will be imported.

- `Prefix`: Data import is triggered by an ADIF import scheme that is created semi-automatically via an assistant. The assistant generates the names of temporary tables and columns of temporary tables from the string defined in this attribute followed by the column names imported from

ServiceNow. It is recommended to set the XML attribute `Prefix` to make sure that the column name is not identical with a reserved SQL keyword (for example: Order).

10) For each XML element `DataConnection`, define the source from which the data shall be imported by adding an XML element `Entry` as child XML element of the XML element `DataConnection`. An XML element `DataConnection` can have multiple child XML elements `Entry`. Each XML element `Entry` corresponds to one ADIF Entry in the ADIF import scheme configured to integrate the data into the Alfabet database. Define the data source with the following XML attributes of the XML element `Entry`:

- `Name`: Enter the ServiceNow `sys_id` of the report or the name of the database table/database view that is the source of the data import in ServiceNow.

  > (i) To retrieve the `sys_id` of a report, you can use the **Copy sys_id** command in the context menu available for the entries of list view records in ServiceNow.

- `Id`: Enter a name that shall be used in Alfabet in the ADIF import scheme configuration.

11) Add an XML element `DataTypeMappings` to the root XML element `ServiceNowConfig`. Add the XML attribute `UnknownServiceNowTypeAsString`. Enter `true` to import all data types that are not explicitly defined as data types in the child elements of the XML element as string. If you enter `false` and an imported data type is missing in the specification, import will fail.

12) For each data type that should not be imported as string, add an XML element `DataType` as child element to the XML element `DataTypeMappings` and set the following XML attributes for the XML element `DataType`:

- `ServiceNowType`: Enter the ServiceNow data type to be converted to a specific Alfabet data type during import.

- `ADIFType`: Enter the Alfabet specific data type that shall be used to write the data to the temporary tables created during import via the ADIF import scheme.

13) In the toolbar of Alfabet Expand, click the **Save** button to save your changes.

Configuring Integration of ServiceNow Data into the Alfabet Database

Data import from ServiceNow is performed by running an ADIF import scheme. In ADIF, schemes can be configured using a ServiceNow assistant. When you open the assistant, a connection to the ServiceNow instance that you want to define data for is opened and you can see which data is available for import in the database table, database view or report. You can select the data in the assistant and when closing the assistant, the ADIF scheme is automatically filled with the required entries to read the data into temporary database tables. To define integration of the data into the Alfabet database, the entries resulting from the semi-automatic creation of the ADIF import scheme elements must be amended with the information about how data from the temporary tables in integrated into the Alfabet database. The configuration required for this last step of the import highly depends on your demands.

> (i) This section only describes the handling of the ServiceNow assistant. Information about how to configure an ADIF import scheme to write data from the temporary database tables to the Alfabet database is given in detail in the reference manual *Alfabet Data Integration Framework* and is not repeated here.

Using the ServiceNow assistant does not only provide semi-automatic configuration of the ADIF import scheme. It also establishes the mechanisms required to open the connection to ServiceNow and import the data when the ADIF import on basis of the ADIF import scheme is executed.

Within the ADIF import scheme, an XML element `DataConnection` in the XML object `ServiceNowImportConfig` corresponds to an ADIF import entry if the import is performed for reports in CSV format or in an ADIF import set if the import is performed from database tables or database views and therefore XML is used as data transmission format. Each ADIF entry can be deactivated or activated individually. Therefore, you can limit data import via ADIF to subsets of the data configured in the XML object `ServiceNowImportConfig` without changing the XML object configuration.

After having defined a valid configuration for data import in the XML object `ServiceNowImportConfig`, you must define a new ADIF import scheme to configure data integration using the ServiceNow import assistant:

1) In the **ADIF** tab in Alfabet Expand, right-click the **ADIF Schemes** root node in the explorer and select **Create Import Scheme**. The new import scheme is added to the explorer. The attribute window of the new import scheme is displayed on the right.

2) In the attribute window, set the following attributes for the ADIF import scheme:

- **Name**: Enter a unique name. The name is used to identify the ADIF import scheme in technical processes. It must be unique and should not contain white spaces or special characters.

- **Assistant**: Select `ServiceNowImport_Assistant` from the drop-down list.

- **Import File Required**: Select `False`.

- **Commit After Run**: If set to `True`, the result of the data import is written persistently to the Alfabet database. If set to `False`, the import process will be rolled back after execution and no changes will be written to the database. Configuration of the automatic start of workflows during import is ignored when **Commit After Run** is set to `False`. It is recommended that you set **Commit After Run** to `False` for a new import scheme to allow debugging without the risk of corrupting the database. After the successful testing of the data import and verification that the resulting changes to the Alfabet database are as expected, you can reset the **Commit After Run** attribute to `True` to perform regular data import.

  ⚠ Please note the following:

  - Setting the **Commit After Run** attribute rolls back all changes to data records in existing tables caused by DML statements. The creation or deletion of tables is not included in the roll back. For example, if you test an ADIF scheme that is configured to persistently write temporary tables to the database, these temporary tables will be created persistently even if **Commit After Run** is set to `False`. SQL commands of the type **OnActivate** are also excluded from roll back.

  - When new objects are created during an ADIF import job, the data bind mechanism assigns `REFSTR` values for the new objects. When **Commit After Run** is set to `False`, the objects are not created in the database, but nevertheless the `REFSTR` values are regarded as in use and will not be used for data bind in the next ADIF run unless the Alfabet Server or Alfabet Expand application used to process the ADIF jobs is restarted.

  - Changes triggered by **OnActivate** commands are not rolled back if the option **Commit After Run** is set to `False` for the import scheme.

- **Drop Temp Tables**: If set to `True`, all temporary tables are dropped after import. Only the changes to the Alfabet database are stored persistently. If set to `False`, the temporary tables are kept in the database after import is finished. Storing temporary tables persistently is only required for special import/export cycles designed for data manipulation that require input from the temporary tables of a previously set import. In most cases, setting this attribute to `True` is recommended to clean the database of data that is not part of the Alfabet meta-model.

3) In the explorer, right-click the node of the new ADIF import scheme and select **Create ADIF Scheme Details Using the ServiceNow Import Assistant**. The assistant opens in your standard Web browser.

4) In the field **(1) Create ADIF Entries for Data Connection Entries**, all XML elements `Entry` defined for all `DataConnection` XML elements included in your `ServiceNowImportConfig` XML object will be displayed. For each XML element `Entry`, the name of the `Entry` followed by the name of the `DataConnection` in parentheses is displayed. Click on each data connection for which data shall be integrated with this ADIF import scheme.

5) After having selected all relevant `Entry` XML elements, click the button **Create Entries** below the field to trigger automatic creation of the ADIF entries / ADIF import sets in your ADIF import scheme. After the action is completed, you will see the message **Create Entries completed** in the lower left corner of the assistant window.

6) Select one of the `Entry` XML elements for which you want to edit the ADIF configuration in the drop-down list of the field **(2) Select an Entry to Edit**. A number of new fields are displayed.

7) Optionally, you can define the import to include only data from a subset of the columns in a database table, database view or report. By default, **Select all columns to import** is selected in the field **(3) Select ServiceNow Columns to Import**. If you would like to exclude data from single columns from import, you can select **Select columns manually to import** instead.

⚠️ Exclusion of columns from import can be used if the data volume for import becomes too high. There is a restriction though that should be taken into account: when manually defining which columns to import, the request sent to ServiceNow to import the data must include all defined column names instead of requesting all data from a specified table. Depending on the length of the column name, the size restriction for API calls could be exceeded.

8) The field **(4) Edit Selected Entry** contains a table listing all data that might be imported. For each column in a database table, database view or report a row is displayed in the table. The first column **Column Name** displays the name of the database column, database view column or column header in the report. If applicable, change the default import from ServiceNow by editing the data in the table:

- **Include**: By default, ADIF import entries / ADIF import sets are defined for all data available in a ServiceNow database table, database view or report. You can exclude data from import by clearing the checkbox in the respective table row. The respective ADIF import entry / ADIF import set will be removed from the ADIF import scheme. It will also be removed from this table because it does not list the `Entry` XML elements of the XML object **ServiceNowImportConfig** but the available ADIF import entry configuration in the ADIF import scheme. This column is only editable if the field **(3) Select ServiceNow Columns** is set to **Select columns manually to import**.

- **Type**: Check whether the data type for integration of the data into the object class database tables of the Alfabet database is correct. It should be identical with the data type of the object class property the data will be stored in. Redefine the definition when necessary.

- **Size**: If `String` is selected in the **Type** field, check whether the defined string length displayed in this column matches the size specification of the target object class property in the Alfabet database. For technical reasons, the **Size** value must be set to <size in the target database table>+1. For example, if you want to import a value with a permissible size of 16, you must define the **Size** attribute as 17.

9) Optionally, you can configure the import to be limited to objects that have been changed since the last data import with the same ADIF import scheme. The drop-down list of the field **(5) Select Netchange ServiceNow Columns** lists all columns for the entry that return dates. Select one or multiple columns to perform net change import on basis of these dates. The dates in the selected columns are compared to the data of the last ADIF import and the data is only imported if one of the dates lies after the date of the last ADIF import. Net change import is performed on the level of dates. Import time is not relevant for the comparison.

> If you set the Type of the property to Date in the field **(4) Edit Selected Entry**, you will only see the property in the drop-down list of the filed **(5) Select Netchange ServiceNow Columns** after having clicked the button **Update Entries**.

10) Click the button **Update Entries** to apply your changes to the ADIF import entries/ ADIF import sets in the ADIF import scheme. After the action is completed, you will see a message informing you that the update has been completed. The message is displayed in the lower left corner of the assistant window.

11) Repeat step 6) - 11) for all data connection entries.

12) In the field **(6) Select to activate/deactivate Data Connection**, you can optionally deactivate execution of single ADIF import entries/ADIF import sets in your ADIF import scheme. By default, all ADIF elements are activated and are displayed with a checkmark in the list. Activation or deactivation is done by clicking a name in the drop-down list to change the checkmark setting.

13) Click **OK** to write the changes into the ADIF import scheme.

14) Close the browser window to close the assistant and return to the ADIF tab of Alfabet Expand.

15) Right-click the ADIF import scheme and select **Rescan Tree** in the context menu. You will then see all automatically generated ADIF elements in the ADIF scheme. Each ADIF import entry will have data import to temporary tables defined via the attribute elements in the **Attributes** folder. The import to the standard Alfabet database tables is not included in the configuration.

> The assistant generates the names of temporary tables and columns of temporary tables from the string defined in the attribute `Prefix` of the `DataConnection` XML element in your `ServiceNowImportConfig` XML object followed by the column names imported from ServiceNow. Names are truncated after 30 characters.

16) Configure the data integration to the standard Alfabet database tables according to your demands. For information about the configuration options available via ADIF import schemes see the reference manual *Alfabet Data Integration Framework*.

Changing an Existing Configuration for ServiceNow Integration

During execution of ADIF data import from ServiceNow, the configuration in the XML object `ServiceNowImportConfig` is used to establish the connection and find the data to be imported. Changes in the XML object that are having an effect on the way data is imported require that the ADIF scheme is changed to reflect the changes. For example, when importing integer values with display values, two database columns are required in the temporary tables created during ADIF import to store the data. When creating the ADIF scheme via the ServiceNow assistant this is taken into account. If you later change the import options to import integer values only, the temporary database table still has the columns with "_IV" and "_DV" amended for import of both integer and display value while the import mechanism is trying to write the data into a database column without the suffixes.

It is therefore required to adapt the ADIF scheme to changes made to the configuration in the XML object `ServiceNowImportConfig`. This must be performed via the ServiceNow assistant to make sure that the import mechanism is correctly adapted to the changes.

Changes in the ServiceNow data are displayed in the ServiceNow import assistant. To check whether your configuration is still in accordance with Service Now data, select the `Entry` XML element in the drop-down list of the field **(2) Select an Entry to Edit** and check the icons that are displayed in the field **(3) Edit Selected Entry**:

- ☑ : Nothing has changed for this data.

- ⊗ : The data is no longer available in ServiceNow.

- ⊕ : The data has been added in ServiceNow.

ⓘ If you click into the mapping table, a floating toolbar is displayed that provides access to a legend explaining the icons.

Updates can be performed per `Entry` XML element in a `DataConnection` XML element in the XML object `ServiceNowImportConfig`. You can do one of the following:

- To add a new ADIF import set / ADIF import entry for a new `Entry` XML element without changing the rest of the configuration, select the new `Entry` only in the field **(1) Create ADIF Entries for Data Connection Entries**, click the button **Create Entries** and click **OK** to apply your changes to the ADIF scheme. The ADIF import set or ADIF import entry will be created without changes to the existing import configuration for other `Entry` XML elements.

- To update an existing ADIF import set / ADIF import entry with new columns for a changed `Entry` XML element without changing the rest of the configuration, select the `Entry` XML element in the drop-down list of the field **(2) Select an Entry to Edit**, edit the data in the field **(3) Edit Selected Entry** according to demands, click the button **Update Entries** and click **OK** to apply your changes to the ADIF scheme.

  Please note that the ADIF import set / ADIF import entry is completely overwritten by this action. Any configuration performed manually in the ADIF scheme is removed.

- To edit an existing ADIF entry configuration for an existing and unchanged `Entry` XML element, select the `Entry` XML element in the drop-down list of the field **(2) Select an Entry to Edit**, edit the data in the field **(3) Edit Selected Entry**, click the button **Update Entries** and click **OK** to apply your changes to the ADIF scheme. The changes will be selectively applied without changing any other configurations.

- To delete an ADIF import set / ADIF import entry because the corresponding `Entry` XML element has been removed, you must manually remove ADIF import set / ADIF import entry from the ADIF scheme in the **ADIF Schemes** explorer in the **ADIF** tab from Alfabet Expand. This action is not supported by the ServiceNow assistant.

Please note that any changes are only applied after clicking the button **OK** of the assistant. The changes are not automatically visible in the ADIF Schemes explorer in Alfabet Expand. Right-click the ADIF import scheme that you have edited via the ServiceNow assistant and select **Rescan Tree** to make the changes visible in the ADIF import scheme.

Configuring Integration of Alfabet Data Into ServiceNow

If you are using ServiceNow, you can import data about the services planned and approved in Alfabet from the Alfabet database into ServiceNow to re-use the data for management of hosted services. There is no predefined way to export the data. The Alfabet and ServiceNow data that is involved is completely customer configured.

The data from various tables from the Alfabet database is first gathered into a configured report representing a single tabular dataset. The data from this configured report must match an Import Set Table defined at Service-Now. During export, the data from the Alfabet configured report is written into the corresponding columns of the Import Set Table at ServiceNow. It is then imported to ServiceNow database tables by a customer configured transformation process.
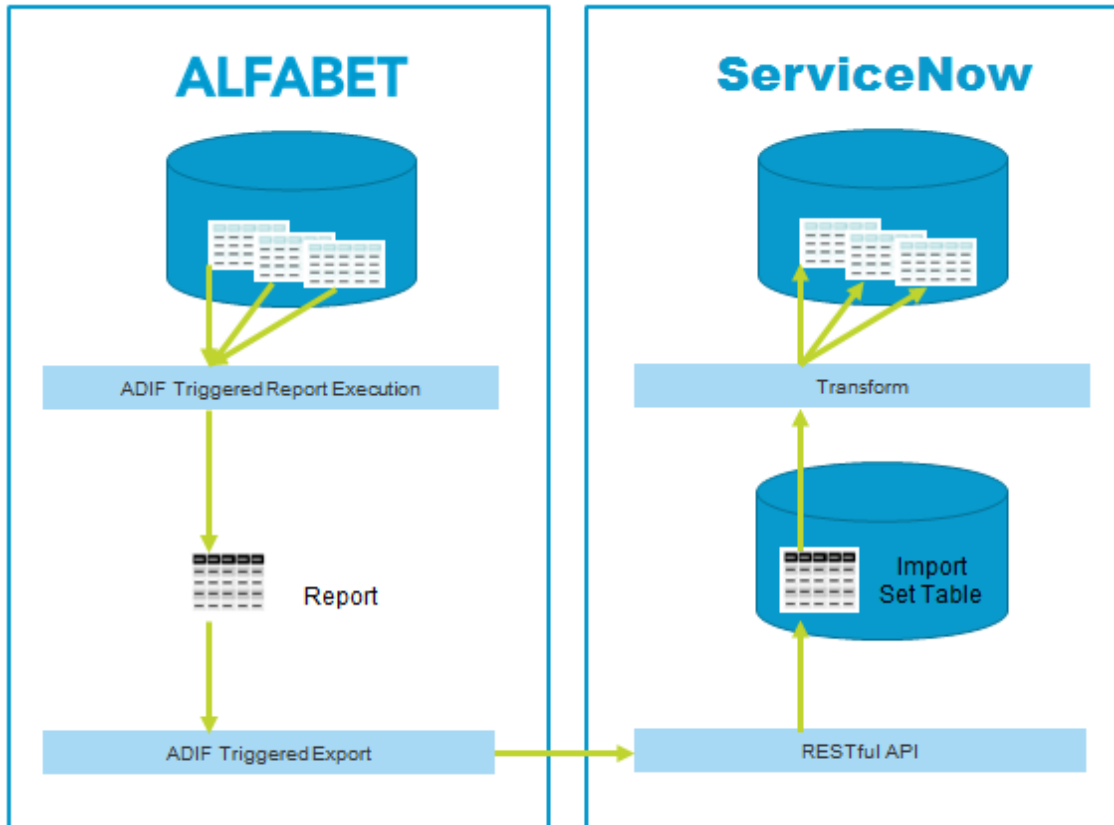
FIGURE: Overview over the data collection from Alfabet to ServiceNow

The following configuration steps are required to export data from the Alfabet database to ServiceNow:

- In Alfabet, a configured report of the **Type** `Query` or `NativeSQL` must be defined that gathers the data from the database tables of the Alfabet database into a simple dataset for export to ServiceNow. Filters can be defined for the report that can be filled with values during export via the command line of the ADIF export scheme that triggers data export. The attribute **Category** of the configured report must be set to a string that is then defined in the XML object **ServiceNowExportConfig** for the data connection exporting the data from the report.

- In Alfabet, the connection to ServiceNow and the mapping of data types must be defined in the XML object **ServiceNowExportConfig**.

- In Alfabet, an ADIF export scheme must be created with the **Assistant** attribute set to `ServiceNowExport_Assistant`. The definition of the export scheme is completely performed via the assistant and does not require manual adaptations. Mapping of data from the result dataset of the configured report to the columns of the staging table of the ServiceNow export is performed in the assistant. A net change export can also be configured.

- In ServiceNow, a service must be created with the import method HTTP and with an Import Set Table that will be filled during import with the data from Alfabet. For information about this configuration step please refer to the documentation of ServiceNow. The import from the Import Set Table to the tables of ServiceNow must be configured in ServiceNow via a Transform Map. Please note that changes to data

tables in ServiceNow can invoke business rules which might lead to performance issues or errors during imports. The amount of data to import per request should be carefully considered to avoid such problems. For information about this configuration step please refer to the documentation of ServiceNow.

- In ServiceNow, the plugin insertMultiple must be activated.

To execute data export from Alfabet to ServiceNow, you must start the ADIF export triggered by the ADIF export scheme. The ADIF export executes both data retrieval from the configured report and data transmission to the ServiceNow service. The ADIF export should be executed in regular intervals to ensure data consistency between ServiceNow and Alfabet.

> (i) Information about the execution of ADIF schemes is provided in the reference manual *Alfabet Data Integration Framework*.

The following sections describe the configuration steps for the ServiceNow interface on Alfabet side in the required order of execution:

- [Defining Data to Export from the Alfabet database Via a Configured Report](#)

- [Configuring Data Transmission from Alfabet to ServiceNow](#)

- [Configuring the ADIF Export Scheme for Data Export to ServiceNow](#)

- [Changing an Existing Configuration for ServiceNow Integration](#)

Defining Data to Export from the Alfabet database Via a Configured Report

All data that shall be exported must be collected in a configured report that is based on an Alfabet query or a native SQL query and returns a simple tabular dataset.

The configured report is created in the configuration tool Alfabet Expand:

1) In the **Reports** tab of Alfabet Expand, 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, 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.

- **Type**: Select `Query` to define a report based on an Alfabet query or `NativeSQL` to define a report based on a native SQL query.

- **Category**: Enter a unique string that will be used to identify the report in the data connection definition when configuring the data transmission to ServiceNow. You can define multiple reports having the same **Category** setting. If you configure an ADIF Export scheme for data export on the data connection referring to the report category, you can select the configured report to be used for the export from all configured reports having the defined **Category** setting.

  > ⚠ The **Category** strings must be completely different from each other, because for technical reasons, the drop-down list for the reports in the ADIF Export scheme assistant is not only finding exact matches of the string defined in the data connection configuration, but also reports with **Category** settings containing the defined string. For example, if the data connection is configured to work with configured reports of the **Category** `ExportApplications`, configured reports with a category like `ExportApplicationGroups` are also found.

- **Alfabet Query / Native SQL Query**: Click the **Browse** 🔳 button to open the **Alfabet Query Builder** or the text editor for definition of a native SQL query. Define a query which returns the information to be transmitted to the ServiceNow service in a simple tabular dataset. Note the following for the configuration of the query:

  - Filter parameters can be defined in the query for the configured report. The parameters will lead to the generation of parameter elements in the ADIF scheme. The values for the parameters are set via the command line when executing the ADIF export.

    > ⓘ For a description of how to define a query with filter parameters, see *Defining a WHERE Clause that Causes the Generation of a Filter Field in the Report* in the section *Defining Filters for Configured Reports and Selectors* in the reference manual *Configuring Alfabet with Alfabet Expand*.
    >
    > For a description of the setting of parameter values of ADIF export schemes, see *Configuring Import Dependent on Parameters* in the reference manual *Alfabet Data Integration Framework*.

  - If you want to define net change import, the data set shall include at least one column that returns a date that can be used to perform net change data export by exporting only data that has a date after the last identical export was executed.

- If indicators and roles are added to an Alfabet query via **Show Properties** of the type `RoleType` or `Indicator`, they are processed as string. Settings for **Show Indicator As Icon** are ignored.

- **Apply to Class**: Optionally define an object class. 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. The parameter `BASE` must then be filled with the REFSTR of an object of the selected class via the command line of the ADIF job executing the configured report.

> (i) For information about Alfabet parameters, see *Referring to the Current Alfabet Context in a WHERE Condition* and *Using Alfabet Parameters* in the chapter *Defining Queries* in the reference manual *Configuring Alfabet with Alfabet Expand*.

3) In the toolbar, click the **Save** 💾 button to save your changes.

4) 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 query until the output of the configured report meets your expectations.

5) Right-click the configured report in the explorer and select **Set State to Active** from the context menu.

Configuring Data Transmission from Alfabet to ServiceNow

Configuration of data export is done in the XML object **ServiceNowExportConfig**. The XML object allows connections to multiple ServiceNow sources to be defined. The general structure of the XML in the XML object is the following:

```
<ServiceNowConfig>
    <DataTransferMappings>
        <DataTransferMapping>
            <ServiceAccessInfo>...</ServiceAccessInfo>
            <DataConnectivityInfo>...</DataConnectivityInfo>
        </DataTransferMapping>
    </DataTransferMappings>
    <DataTypeMappings>...</DataTypeMappings>
```

The root XML element `ServiceNowConfig` of the XML objects contains two child elements:

- `DataTransferMappings`: This XML element can contain one or more child elements `DataTransferMapping`, each containing the definition of a connection to a different ServiceNow server. The definition includes the specification of the connection to the REST API of the web services of the ServiceNow server in a child element `ServiceAccessInfo` and the specification of data to be transferred in a child element `DataConnectivityInfo`. Within the `DataConnectivityInfo` element, different data transfer conditions can be defined for different data types.

- `DataTypeMappings`: This XML element contains the mapping of data types that are exported to data types that are valid in ServiceNow. The data is converted to the specified data types during data import. The data type definitions are valid for all data connections on all data transfer mappings.

To define the data import from a ServiceNow server:

1) In Alfabet Expand, go to the **Presentation** tab and expand the explorer nodes **XML Objects** > **IntegrationSolutions**.

2) Right-click **ServiceNowExportConfig** and select **Edit XML...**. The XML object opens.

> ⓘ The XML object usually includes an example definition. In addition, a template is available via the XML attribute `XML Template` of the XML object. The template can be copied to the XML object to avoid having to write the configuration manually. The following description describes a configuration from scratch. With a sample configuration, you have to edit rather than add the XML elements described in the following.

3) Add a child XML element `DataTransferMappings` to the root XML element `ServiceNowConfig`.

4) Add a child XML element `DataTransferMapping` to the XML element `DataTransferMappings` and define the following XML attribute for the XML element `DataTransferMapping`:

- `Name`: The name of the ServiceNow server connection that is used to identify the connection in the ADIF scheme configuration.

- `IsActive`: Set to `true` if data should be exported to the ServiceNow instance. Set to `false` if data should currently not be exported to the ServiceNow instance.

5) Add a child XML element `ServiceAccessInfo` to the XML element `DataTransferMapping` to define the information required to establish a connection to the ServiceNow server.

6) Add an XML attribute `AutomaticSwitchToBasic` to the XML element `ServiceAccessInfo` and set it to `true` to connect via basic authentication if OAuth authentication fails.

7) The information required to establish the connection to the REST API of the web services of the ServiceNow server must be added as strings to the following child XML elements, which have to be added to the XML element `ServiceAccessInfo`:

- `service`: The URL of the ServiceNow instance.

- `apipath`: The path to the REST API at the ServiceNow instance. Enter `/api/now/v1/`.

- `oauthpath`: `OAuth` is required for authentication. Specify the path to the `OAuth` endpoint with the XML element `oauthpath`.

- `username`: The user name for access to ServiceNow.

> ⚠ The user must have the required access permissions to perform the following REST api service calls to ServiceNow:
>
> - https://<BaseAddress>/oauth_token.do
>
> - https://<BaseAddress>/<databaseviewname>.do?WSDL
>
> - https://<BaseAddress>/api/now/v1/table/sys_dictionary?sysparm_query=GOTO name=<tableName>
>
> - https://<BaseAddress>/api/now/v1/table/sys_transform_map?source_table=<tableName>
>
> - https://<BaseAddress>/api/now/v1/table/sys_transform_entry?map=<mapid>
>
> - https://<BaseAddress>/api/now/v1/table/sys_db_object?name=<tableName>
>
> For example, the following permissions would allow to execute all required service calls:

- `import_admin` to perform the required changes to import set tables.

- `soap` to perform SOAP operations

- `personalize_dictionary` for access to `sys_dictionary` and `sys_db_object` tables that is required during import into ServiceNow to get additional details for each table and the definition for every column on each table.

- `Rest_service` for access rights to required database tables via REST API calls.

- `itil`, if CMDB CI tables re used in ServiceNow import.

- `password`: The password for access to ServiceNow.

- `client_id`: Enter the Client ID for the `OAuth` endpoint at ServiceNow.

- `client_secret`: Enter the Client Secret for the `OAuth` endpoint at ServiceNow.

> (i) In the XML elements for the definition of the ServiceNow server connection server variables can be used to read the value of the XML element at runtime from the server alias configuration of the Alfabet Web Application when a connection to ServiceNow is established. For information about server variables, see Configuring Server Variables for Integration and Interoperability Solutions.

8) Add a child XML element `DataConnectivityInfo` to the XML element `DataTransferMapping` to define the information required to transfer data the ServiceNow server. The XML element `DataConnectivityInfo`. Define the default data transmission parameters for the data exports by setting the following XML attributes of the XML element `DataConnectivityInfo`:

- `DefaultPageSize`: You can define a limit for data to be transmitted simultaneously on the data connection. Enter the number of records that shall be transmitted simultaneously. This value will be valid for all data connections that do not have an attribute `PageSize` defined. If it is neither defined for the data connection nor with this XML attribute, a default of 100 will be set.

- `DefaultTimeout`: Define the default HTTP request timeout that shall be used for all data connections that do not include a timeout definition via the XML attribute `Timeout` of the `DataConnection` XML element. If a timeout is neither defined for the data connection nor with this XML attribute, the default value defined for the Alfabet Web Application is set.

- `DefaultMaxThreadCount`: Define the maximum number of parallel threads for data transmission to ServiceNow on a data connection. The value defined here is used as default for all data connections that do not have a maximum number of records to be transmitted defined via the XML attribute `MaxRecordCount` of the `DataConnection` XML element.

9) Add one or multiple child elements `DataConnection` to the `DataConnectivityInfo` XML element. Each XML element `DataConnection` bundle data to be exported via one transmission to the ServiceNow server. Each XML element `DataConnection` corresponds to one service of ServiceNow for data import. That means that all information to be exported for an XML element `DataConnection` must have the same export format and data structure. Define the data transmission parameters for the data request by setting the following XML attributes of the XML element `DataConnection`:

- `DataConnectionName`: Enter a unique name for the data connection. This name is used to identify the data connection in the configuration of the ADIF scheme.

- `ReportCategory`: Enter the string that is defined in the attribute **Category** of the configured report used for data export via that connection.

- `Timeout`: Define the HTTP request timeout for the data connection. If no timeout is defined, the default timeout defined in the XML element `DataConnectivityInfo` is used.

- `PageSize`: Define the maximum number of records to be transmitted simultaneously on the data connection.

  > It is recommended that the XML attribute `PageSize` is set to a value lower than 500. An excessively high number of records may lead to transmission problems.

- `MaxthreadCount`: Define the maximum number of parallel threads for data transmission to ServiceNow.

10) For each XML element `DataConnection`, define the source from that the data shall be exported by adding an XML element `Entry` as child XML element of the XML element `DataConnection`. An XML element `DataConnection` can have multiple child XML elements `DataConnectionEntry`. Please note that the data structure exported via the defined sources must be consistent within one XML element `DataConnection`. Define the data source with the following XML attributes of the XML element `Entry`:

- `Name`: Enter a unique name to identify the entry in the ADIF Export scheme configuration.

- `Id`: Enter the identifier of the ServiceNow service that is the target of the export.

11) Add an XML element `DataTypeMappings` to the root XML element `ServiceNowConfig`. Add the XML attribute `UnknownServiceNowTypeAsString`. Enter `true` to export all data types that are not explicitly defined as data types in the child elements of the XML element as string. If you enter `false` and an exported data type is missing in the specification, export will fail.

12) For each data type that should not be exported as string, add an XML element `DataType` as child element to the XML element `DataTypeMappings` and set the following XML attributes for the XML element `DataType`:

- `ServiceNowType`: Enter the ServiceNow data type that will be transmitted to ServiceNow.

- `ADIFType`: Enter the Alfabet specific data type that is converted to the ServiceNow data type defined with the XML attribute `ServiceNowType`.

13) In the toolbar of Alfabet Expand, click the **Save** button to save your changes.

Configuring the ADIF Export Scheme for Data Export to ServiceNow

Data export to ServiceNow is performed by running an ADIF export scheme. In ADIF, export schemes can be configured using a ServiceNow assistant. When you open the assistant, a connection to the ServiceNow instance that you want to export data to is opened and you can directly perform data mapping. You can select the data in the assistant and when closing the assistant, the ADIF scheme is automatically filled with the required entries to export the data.

Using the ServiceNow assistant does not only provide automatic configuration of the ADIF export scheme. It also establishes the mechanisms required to open the connection to ServiceNow, execute the configured Alfabet report and export the data when the ADIF export on basis of the ADIF export scheme is executed.

Within the ADIF export scheme, an XML element **Entry** in an XML element `DataConnection` in the XML object `ServiceNowImportConfig` corresponds to an ADIF export entry. Each ADIF entry can be deactivated or

activated individually. Therefore you can limit data export via ADIF to subsets of the data configured in the XML object `ServiceNowImporExportConfig` without changing the XML object configuration.

After having defined a valid configuration for data export in the XML object `ServiceNowExportConfig`, you must define a new ADIF export scheme to configure data integration using the ServiceNow import assistant:

1) In the **ADIF** tab in Alfabet Expand, right-click the **ADIF Schemes** root node in the explorer and select **Create Export Scheme**. The new ADIF export scheme is added to the explorer. The attribute window of the new ADIF export scheme is displayed on the right.

2) In the attribute window, set the following attributes for the ADIF Export scheme:

   - **Name**: Enter a unique name. The name is used to identify the ADIF export scheme in technical processes. It must be unique and should not contain white spaces or special characters.

   - **Assistant**: Select `ServiceNowExport_Assistant` from the drop-down list.

   - **Commit After Run**: If set to `True`, the result of the data export is written persistently to the external files or the external target database. If set to `False`, the export process is rolled back after execution and no changes are written to the external database or files. It is recommended to set **Commit After Run** to `False` for a new export scheme for export to an external database to allow debugging without the risk to corrupt the external database. After successful testing of the data export and verification that the resulting changes to the external database are as expected, **Commit After Run** can be set to `True` to perform regular data export.

     ⚠️ The following restrictions apply to setting **Commit After Run** to `False`:

       - **Commit After Run** only affects database transactions. If you export data to a file, the export file is created and data is added to the file also if **Commit After Run** is set to `False`.

       - Setting **Commit After Run** to `False` rolls back all changes caused by DML statements (changes to data records in existing tables). Creating or deleting tables is not included in the roll back. That means, for example, if you test an ADIF scheme that is configured to write temporary tables to the database persistently, these temporary tables will be created persistently even if **Commit After Run** is set to `False`. SQL Commands of the type `OnActivate` are also excluded from roll back.

3) In the explorer, right-click the node of the new ADIF export scheme and select **Create ADIF Scheme Details Using the Service Now Import Assistant**.

4) Confirm the warning. The assistant opens in your standard Web browser.

5) In the field **(1) Select a Data Connection**, select a data connection configured in the XML object **ServiceNowExportConfig** from the drop-down list. The Entry elements for the DataConnection element are listed in the field **(2) DataConnection Entries**. After each Entry element, the information (Entry not created) is displayed, because the entries are not yet included into the ADIF export scheme.

6) In the field **(3) Select an Entry to Edit/Update**, select one of the available entries to create an ADIF export entry for it.

7) In the field **(4) Map Alfabet Report with Selected Entry**, select one of the configured reports that are mapped to the data connection via their **Category** setting, to be executed for export via this **Entry**.

8) Optionally, you can define in the field **(5) Select Display Option for Alfabet Report Properties**. Select one of the following:

- **Sort Report Columns as Defined in Report**: The column names in the configured report are displayed instead of the object class property names. The sort order is identical to the order of columns in the configured report.

- **Sort Report Columns with Lexicographic Sorting**: The column names in the configured report are displayed instead of the object class property names. The sort order is alphanumeric.

- **Sort Property Columns as Defined in Report**: The object class properties are displayed as <ObjectClassName>.<PropertyName>. The sort order is identical to the order of columns in the configured report.

- **Sort Property Columns with Lexicographic Sorting**: The object class properties are displayed as <ObjectClassName>.<PropertyName>. The sort order is alphanumeric.

9) Optionally, you can configure the export to be limited to objects that have been changed since the last data export with the same ADIF export scheme and the same command line parameters. The drop-down list of the field **(6) Select Netchange Report Parameter** lists all columns of the selected report that return dates. Select one or multiple columns to perform net change export on basis of these dates. The dates in the selected column are compared to the date of the last ADIF export execution and the data is only exported if the date lies after the date of the last ADIF export. Net change export is performed on the level of dates. Export time is not relevant for the comparison.

10) Click the button **Create Attributes**. After the action is completed, you will see a message informing you that the attribute creation has been completed in the lower left corner of the assistant window and a table is displayed in the field **(7) Map Alfabet Class Properties to ServiceNow Web Service Fields**.

11) In the table of the field **(7) Map Alfabet Class Properties to ServiceNow Web Service Fields** all fields from the ServiceNow Import Set Table are listed in the column ServiceNow Web Service Field. For each row, select the matching column of the Alfabet configured report from the drop-down list. The additional columns of the table inform you about the target data in the standard ServiceNow table to that the data is imported. The ServiceNow property name and type is given as well as information about important characteristics:

- *C = The property is the unique key to map the data to the ServiceNow data during import.

- *R = The property is set as display value in ServiceNow, which means that the value is displayed in the user interface of ServiceNow if the record is referenced.

12) Click **Create Entry**.

13) Repeat step 6) - 11) for all data connection entries.

14) Click **OK** to write the changes into the ADIF import scheme.

15) Close the browser window to close the assistant and return to Alfabet Expand.

16) In the menu of Alfabet Expand, select **Meta-Model** > **Re-Read Meta-Model**. The ADIF scheme is updated with the settings performed in the assistant. the ADIF export entries that were created on basis of the settings will have data export defined via the attribute elements in the **Attributes** folder.

Changing an Existing Configuration for ServiceNow Integration

You can change an existing configuration, For example, by altering the export report to integrate different data or to integrate new properties that have been added at the ServiceNow side. Changes to the configuration always require a change in the ADIF export scheme via the **ServiceNow Export Assistant**.

If you start the **ServiceNow Export Assistant** of an existing ADIF scheme already configured to export data to ServiceNow, and select a changed data connection entry for update, the export assistant compares the current

configuration in Alfabet and ServiceNow with the existing configuration in the ADIF scheme. In the field for mapping of properties of the **ServiceNow Export Assistant**, the changes are marked with icons to help you finding your changes in short time. The icons provide the following information:

- [icon] : the configuration has not changed for this mapping.

- [icon] : the property has been added.

- [icon] : the property has been removed.

- [icon] : the property has been changed in ServiceNow.

- [icon] : the property has been changed in Alfabet.

> (i) If you click into the mapping table, a floating toolbar is displayed that provides access to a legend explaining the icons.

You can do one of the following:

- To apply changes in the property mapping of an existing entry, change the settings in the window **(7) Map Alfabet Class Properties to Service New Web Service Fields** and click **Update Entry**.

- To base the import on a completely new report, change the exiting setting in the field **(4) Map Alfabet Report with Selected Entry**. Alter the properties in the window **(7) Map Alfabet Class Properties to Service New Web Service Fields** and click **Create Entry**.

Sending Requests to ServiceNow via a Proxy Server

Optionally, you can configure the ServiceNow integration interface to send requests to a ServiceNow instance via a proxy server. This requires the following additional configuration in the XML objects **ServiceNowExportConfig** and **ServiceNowImportConfig** for the data export and data import configurations described above:

1) In Alfabet Expand, go to the **Presentation** tab and expand the explorer nodes **XML Objects** > **IntegrationSolutions**.

2) Right-click the node **ServiceNowExportConfig** or **ServiceNowImportConfig** respectively and select **Edit XML...**. The XML object opens.

3) Add a child XML element `Proxy` to the XML element `ServiceAccessInfo` of the XML element `DataTransferMappings` for which you want to define data transmission via a proxy.

4) 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.

5) In the toolbar of Alfabet Expand, click the **Save** button to save your changes.

If you have configured multiple data connections and each data connection shall use a different proxy, you can add additional proxies to your proxy configuration and refer to one of the proxies in the data connection configuration. The proxy definition above will be used as default if no proxy is assigned to a data connection. The use of an additional proxy requires the following configuration:

1) In Alfabet Expand, go to the **Presentation** tab and expand the explorer nodes **XML Objects** > **IntegrationSolutions**.

2) Right-click **ServiceNowExportConfig** and select **Edit XML...**. The XML object with your Amazon Web Services data import definition opens.

3) Add a child XML element `AdditionalProxies` to the XML element `Proxy`.

4) For each additional proxy you want to define, add a child XML element `AdditionalProxy` to the XML element `AdditionalProxies` and define the following XML attributes for the XML element `AdditionalProxy`:

- `Name`: Define a unique name for the additional proxy. This name is used to refer to the proxy in the data connection configuration.

- `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.

5) Add an XML attribute `Proxy` to each **DataConnection** XML element that shall use one of the additional proxies. The value of the XML attribute `Proxy` must be identical to the value of the `Name` XML attribute of the XML element **AdditionalProxy**.

6) In the toolbar of Alfabet Expand, click the **Save** button to save your changes.

Sending Requests to ServiceNow via an API Gateway

Requests to ServiceNow can optionally be sent via an API gateway. The following configurations are required in addition to the configurations described above in the sections Configuring Integration of Data from ServiceNow and Configuring Integration of Alfabet Data Into ServiceNow:

- In the configuration of the ServiceNow integration in the XML objects **ServiceNowImportConfig** and **ServiceNowExportConfig**, the XML element **service** has to be set to the URL of the API gateway.

- The API gateway must be configured to route requests to ServiceNow. A list of resources that need to be accessed at ServiceNow is available in the appendix.

Chapter 12: Configuring Integration with Jira

Alfabet supports data import from Jira® to Alfabet as well as the export of Alfabet data to Jira. The capability to import Jira data to Alfabet allows the enterprise to import issues, projects, categories, and boards documented in Jira to specified entities in Alfabet such as applications, deployments, standard platforms, projects, etc., thus providing a bridge between agile decisions on the enterprise level with operational decisions taken by SCRUM teams, project teams, etc.

The capability to export Alfabet data to Jira supports the planning and prioritization of operative development work and provides support for bimodal IT and agile projects in the context of portfolio evaluation. The goal of the Jira export is to create new issues in Jira based on issues that have been captured via demands, features, issues, value nodes, etc. in Alfabet.

> For details about the entities in Jira such as projects, issues, boards, and sprints as well as the overall Jira product, please consult the documentation delivered with the Jira product.

The Jira content that can be integrated will depend on the Jira version used by your enterprise.

- For Jira versions prior to Jira 5, no integration is available as Jira 5 is the first version supporting the REST API used to establish the integration.

- For Jira versions 5-7, only the Jira Server API is available. It can be used to fetch issues, projects, and sprints assigned to issues.

- For Jira version 7, both the Jira Server API and the Jira Agile API are available. Therefore, in addition to the issues and projects provided by the Jira Server API, the Jira Agile API ensures that scrum boards and all sprints are fetched. Please note that there may be certain functionalities that may only be available with various minor releases of Jira. If a fetch (for example, of a status category) is unsuccessful, it will be written to the verbose ADIF log and the import will continue with the remaining fetches.

> OAuth is used by Jira for authentication. However, because the OAuth version implemented by JIRA is OAuth 1.0a 3-Legged (3LO) and involves user interaction, the integration will use basic authentication to authenticate the Jira Server API and the Jira Agile API.

The following image displays the meta-model used in Jira. You should familiarize yourself with this information in order to configure integration with Jira and to map Jira objects with Alfabet objects.
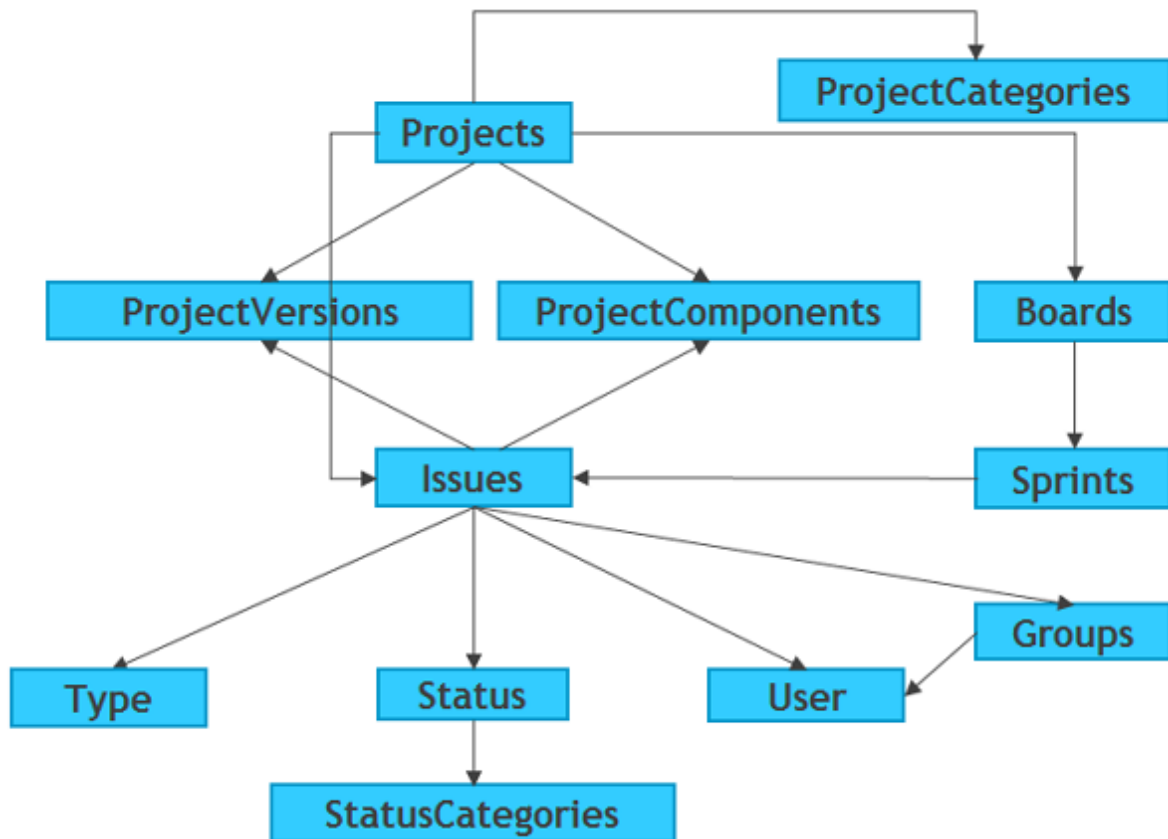
The following information is available:

- [Importing Jira Data to Alfabet](#)

  - [Configuring Connections to Import Jira Data to Alfabet](#)

  - [Importing Jira Data to Alfabet via ADIF Schemes](#)

- [Exporting Alfabet Data to Jira](#)

  - [Configuring Connections and Mapping to Export Data from Alfabet to Jira](#)

  - [Configuring the Primary and Secondary Reports](#)

  - [Configuring Object Filter Reports](#)

  - [Configuring the ADIF Export Assistant to Export Alfabet Data to Jira](#)

  - [Configuring the Event Templates to Trigger the ADIF Export Schemes for Synchronization](#)

- [Configuring Semantic Connections to Link and Synchronize Jira Projects with Alfabet Objects](#)

  - [Creating a Jira Connection for Project-Based Integration](#)

  - [Creating a Jira Connection for Architecture-Based Integration](#)

- [Linking to and Synchronizing the Jira Project](#)

Importing Jira Data to Alfabet

Depending on your Jira® version, your enterprise may import Jira projects (including project categories and project components, for example), issues, boards, and sprints and map them to relevant Alfabet object classes. You can select different issue fields to be imported and optionally choose to import the data for `Sprints` or `Sprints & Boards`. All other data such as `Projects`, `Project Categories`, etc. will be imported per default with the issues. The Jira objects can be mapped to and to relevant Alfabet object classes and imported as applications, components, deployments, standard platforms, and projects, thus providing a bridge between portfolio- and priortization-related decisions at the enterprise level and the operational decisions taken by SCRUM teams, project teams, etc.

For example, the IT department of a banking enterprise documents bugs and change requests for their trading software in Jira and captures their operational plans in Alfabet. In this case, for example, the Jira issues could be imported to Alfabet as features that are planned for applications that are owned and fiscally managed via ICT objects. The following mapping schema provides an example of how Jira projects could be mapped to an ICT object structure in Alfabet:

Jira Class	Alfabet Class
ProjectCategory	ICT Object Group
Project	ICT Object
ProjectVersion	Application
Issue	Feature
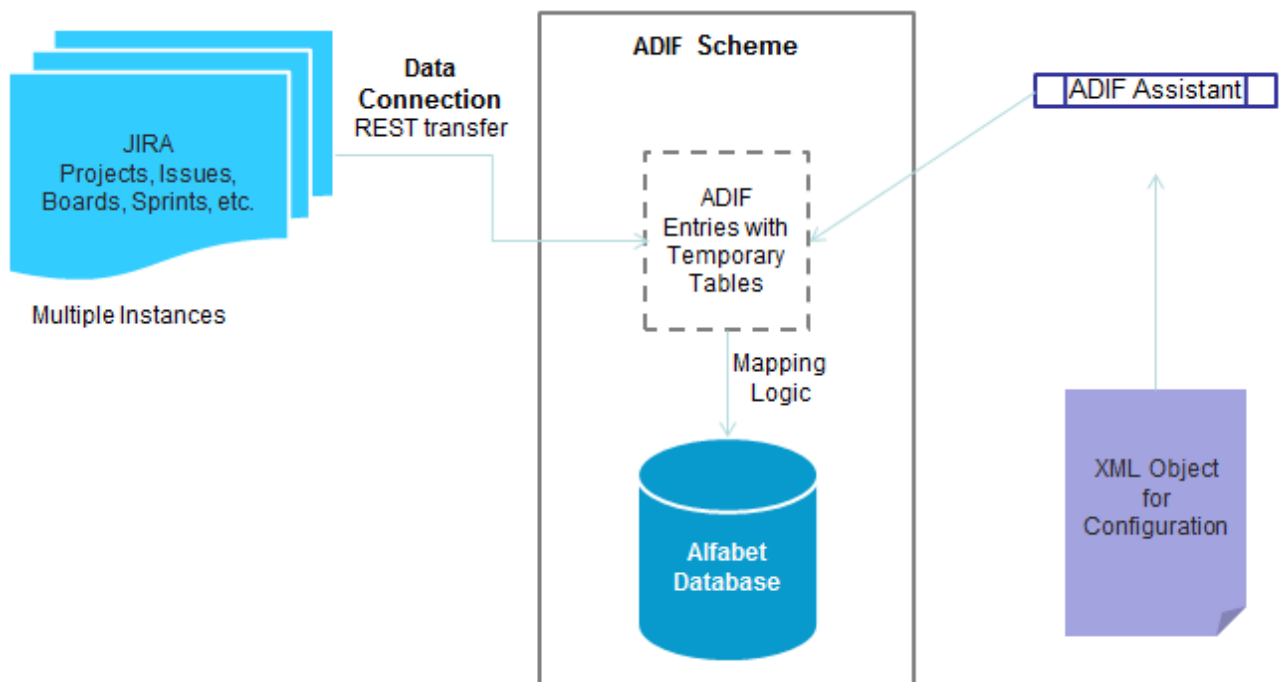
FIGURE: Overview of import from Jira

The following configuration is required to import data from Jira to Alfabet:

1) The connection to Jira must be configured in the XML object **JIRAIntegrationConfig** in Alfabet Expand.

2) The data to be imported must be configured for an ADIF import scheme via the **JIRA Import Assistant**. In the **JIRA Import Assistant** you configure which connection to use for the import as well as the projects, issue types, and issue statuses, and standard and custom issue fields to import. Upon execution of the import scheme, temporary tables will be created for Jira issues and the relevant Jira issue attributes. Depending on the Jira version you are working with, projects, springs, and boards may also be imported depending on the Jira version and configuration of the **JIRA Import Assistant**.

3) Additional configuration of the import scheme is required to map the imported issue attributes from Jira to temporary tables that will be created for the relevant Alfabet object classes. Information about how to configure the ADIF import scheme to write data from the temporary database tables to the Alfabet database is described in the reference manual *Alfabet Data Integration Framework*.

4) The semantic connections must be configured that specify the integration pattern to use for the database connection configured in the XML object **JIRAIntegrationConfig**. A semantic definition should be created for each data connection configured in the XML object **JIRAIntegrationConfig**. You can configure multiple semantic data connections for each data connection configured in the XML element **DataConnection** in the XML object **JIRAIntegrationConfig**. For each semantic connection you create, you must specify the integration pattern to map objects in the Alfabet object hierarchy to the Jira project, project version, or project component. You can choose an architecture-based integration pattern mapping or a project-based integration pattern mapping. You may specify any level in the Jira project hierarchy to begin the integration with. For example, you could choose to map Alfabet applications to Jira project versions.

The following integration patterns may be specified for a project-based approach:

Integration Type	Jira Project	Jira Project Version	Jira Project Component
Jira Projects, Project Versions, and Project Components	**Project** Stereotype 1	**Project** Stereotype 2	**Project** Stereotype 3
Jira Projects and Project Versions	**Project** Stereotype 1	**Project** Stereotype 2	
Jira Project and Project Components	**Project** Stereotype 1		**Project** Stereotype 3
Jira Projects	**Project** Stereotype 1		
Jira Project Versions		**Project** Stereotype 2	

Integration Type	Jira Project	Jira Project Version	Jira Project Component
Jira Project Versions and Project Components		**Project** Stereotype 2	**Project** Stereotype 3
Jira Project Components			**Project** Stereotype 3

The following integration patterns may be specified for an architecture-based approach:

Integration Type	Jira Project	Jira Project Version	Jira Project Component
Jira Projects, Project Versions, and Project Components	ICT Object (or stereotype)	Application (or stereotype)	Local Component (or stereotype)
	ICT Object (or stereotype)	Component (or stereotype)	Local Component (or stereotype)
	ICT Object (or stereotype)	Standard Platform (or stereotype)	Platform Element (or stereotype)
Jira Projects and Project Versions	ICT Object (or stereotype)	Application (or stereotype)	
	ICT Object (or stereotype)	Component (or stereotype)	
	ICT Object (or stereotype)	Standard Platform (or stereotype)	
Jira Projects	ICT Object (or stereotype)		
Project Versions		Application (or stereotype)	
		Component (or stereotype)	

Integration Type	Jira Project	Jira Project Version	Jira Project Component
		Standard Platform (or stereotype)	
Project Versions and Project Components	ICT Object (or stereotype)	Application (or stereotype)	Local Component (or stereotype)
	ICT Object (or stereotype)	Component (or stereotype)	Local Component (or stereotype)
	ICT Object (or stereotype)	Standard Platform (or stereotype)	Platform Element (or stereotype)
Jira Projects, Project Versions, and Project Components		Application (or stereotype)	Local Component (or stereotype)
		Component (or stereotype)	Local Component (or stereotype)
		Standard Platform (or stereotype)	Platform Element (or stereotype)
Project Components		Component (or stereotype)	Local Component (or stereotype)
		Standard Platform (or stereotype)	Platform Element (or stereotype)

The following information is available:

- [Configuring Connections to Import Jira Data to Alfabet](#)

- [Importing Jira Data to Alfabet via ADIF Schemes](#)

Configuring Connections to Import Jira Data to Alfabet

Multiple connections to Jira® can be configured. Data will be imported from all defined Jira instances via the same ADIF import scheme. The connections to Jira must be defined in the XML object **JIRAIntegrationConfig**.

The general structure of the XML in the XML object **JIRAIntegrationConfig** is the following:

```
<JIRAIntegrationConfig>
```

```
<Proxy Url="" UserName="" Password="" Domain="" />

<DataConnections>

    <DataConnection

    Name="JiraInstance1"

    MajorVersion="7"

    ServerUrl="$JiraServer"

    UserName="$JiraUser"

    Password="$JiraPassword"

    PageSize="-1">

        <IssueRank Enabled="true" RankFieldId="10005"/>

    </DataConnection>

</DataConnections>

</JIRAIntegrationConfig>
```

The root XML element `JIRAIntegrationConfig` has a child XML element `DataConnections`.

> ℹ️ Please note that server variables can be used in the XML object **JIRAIntegrationConfig** to read the value of the XML attribute at runtime from the server alias configuration of the Alfabet Web Application when a connection to Jira is established. For information about server variables, see Configuring Server Variables for Integration and Interoperability Solutions.

To define a connection to Jira:

1) In Alfabet Expand, go to the **Presentation** tab and expand the explorer nodes **XML Objects** > **IntegrationSolutions**.

2) Right-click **JIRAIntegrationConfig** and select **Edit XML...**. The XML object **JIRAIntegrationConfig** opens.

> ℹ️ 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 **JIRAIntegrationConfig**. 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.

3) Add a child XML element `DataConnections` to the root XML element `JIRAIntegrationConfig`.

4) Add a child XML element `DataConnection` to the XML element `DataConnections`. The XML element `DataConnections` may contain multiple child XML elements `DataConnection`, each defining a connection to a different Jira instance.

5) Specify the following XML attributes for each XML element `DataConnection`:

- `Name`: The name of the connection to Jira that is used to identify the connection in the ADIF scheme configuration. A unique name must be specified for the connection.

- `MajorVersion`: Specify the major Jira version (5, 6, 7) in order to call the relevant Jira API (for Jira v. 5 and 6, the Jira Server API; for Jira v. 7, Jira Server API and Jira Agile API). You should only specify the major version (5, 6, 7) and not the minor version number of the Jira instance. (For example, specify `MajorVersion` = 5, not `MajorVersion` = 5.3.2.).

- `ServerURL`: Specify the URL to the Jira instance for the connection.

- `UserName`: Specify the user name of the Jira user for the connection.

- `Password`: Specify the password of the Jira user for the connection.

- `MaxThreadCount`: The number of parallel threads that may be used for simultaneous data transfer is set per default to -1 in the XML attribute `MaxThreadCount`. This default value can be modified as needed. The number of CPU cores is typically be a good starting point for the `MaxThreadCount` value.

    > ⓘ Please note the following regarding multi-threading in the context of Jira integration:
    >
    > - If the XML attribute `MaxThreadCount` is not defined the default behavior is as follows:
    >
    >   - If the option `Import Data Using JIRA Query Language Definition` is selected in the **JIRA Import Assistant**, no multi-threading will be performed.
    >
    >   - If the option `Import Data Using List Box` is selected in the **JIRA Import Assistant**, the number of threads will be based on the maximum number of options that have been selected in the **JIRA Projects**, **JIRA Issue Types**, or **JIRA Issue Statuses** fields. For example, if the user selects 5 projects, 3 issue types and 2 issue statuses, then there will be 5 threads created (one for each project). If the option `Import Data Using List Box` is selected in the **JIRA Import Assistant** and no options are selected in the **JIRA Projects**, **JIRA Issue Types**, or **JIRA Issue Statuses** fields, then multi-threading is based on the field having the most options. Thus, if 20 projects, 8 issue types and 5 issues statuses are available in the fields, there will be 20 threads (one for each project)
    >
    > - The specification of the XML attribute `MaxThreadCount` will override the default behavior.

- `PageSize`: Specifies the response page size and is set per default to -1. This default value takes the default page size value from the Jira instance but can be modified as needed. Please refer to the API documentation on the pagination capabilities of the different APIs in JIRA before modifying the default value of -1.

- `HTTPTimeout`: Specify a timeout value in seconds if required. Otherwise, the default HTTP timeout value will be used.

6) Optionally, you can add a child XML element `IssueRank` to the XML element `DataConnection` to specify whether issue ranking should be imported with issues. This is only relevant for active sprints. To populate issues with their ranking values, add the following XML attributes to the XML element `IssueRank`:

- `Enabled`: Set to `True`.

- `IssueRankID`: Enter the ID of the field used for ranking issues in the Jira sprint. The ranking of issues will then be populated in the RANK attribute of the ISSUE_SPRINT mapping/temporary table in the ADIF scheme for all the active sprints.

7) In the toolbar of Alfabet Expand, click the **Save** 💾 button to save your changes.

Importing Jira Data to Alfabet via ADIF Schemes

Data import from Jira is performed by running an ADIF import scheme that triggers the data transmission from Jira based on the configured connection details and stores the incoming data into temporary tables in a predefined way. In the ADIF tab of Alfabet Expand, the import scheme can be configured using the **JIRA Import Assistant**. The **JIRA Import Assistant** allows you to select which data connection defined in the XML object **JIRAIntegrationConfig** to use as well as to specify the data to import.

Once the ADIF scheme to fetch appropriate content has been configured using the **JIRA Import Assistant**, the integration must be completed with subsequent entries to map the imported artifacts in the temporary tables to the relevant Alfabet object classes and objects. Therefore, to complete the integration of the data to the Alfabet database, additional configuration of the import scheme is required to map the imported artifacts in the temporary tables to the relevant Alfabet object classes and objects. The configuration required for this last step of the import will depend on how your enterprise chooses to implement the Jira data in Alfabet. Finally, event templates must be configured to trigger the configured ADIF import schemes for the synchronization functionalities.

> (i) This section only describes the handling of the **JIRA Import Assistant**. Information about how to configure the ADIF import scheme to write data from the temporary database tables to the Alfabet database is provided in detail in the reference manual *Alfabet Data Integration Framework* and is not repeated here. For information about the configuration of event templates, see the chapter *Configuring Events* in the reference manual *Configuring Alfabet with Alfabet Expand*. A user with an administrative user profile may review the success of the triggered events in the *Event Administration Functionality*. For more information, see the chapter *Managing Events* in the reference manual *User and Solution Administration*.

The following table displays which Jira objects will be imported to Alfabet and any special settings required in the **Jira Import Assistant** to import the Jira object:

Jira Object	Content of Import	Context of Import
`Issues`	Options selected in the **JIRA Standard Issue Fields to Import** field.	Always imported
`IssueCustomFields`	Options selected in the **JIRA Custom Issue Fields to Import** field.	Always imported
`IssueComponents`	`ProjectComponents` associated with the imported `Issues`.	Imported if `Components` is selected in the **JIRA Custom Issue Fields to Import** field.
`IssueSprints`	`Sprints` associated with the imported `Issues`.	Imported if `Sprints` or `Sprints and Boards` is selected in the **JIRA Sprint Import Option** field. Relevant only for Jira versions 5-7.
`IssueFixVersions`	`ProjectVersions` for which the associated `Issues` have been fixed.	Imported if `Fix Versions` is selected in the **JIRA Standard Issue Fields to Import** field

Jira Object	Content of Import	Context of Import
IssueAffectsVersions	`ProjectVersions` that the associated `Issues` impact.	Imported if `Affects Versions` is selected in the **JIRA Standard Issue Fields to Import** field
IssueLinks	`IssueLinks` associated with the imported `Issues`.	Imported if `Linked Versions` is selected in the **JIRA Standard Issue Fields to Import** field
Projects	`Projects` in Jira. Please note that the selection of a project in the **JIRA Projects** field only restricts import of issues to issues belonging to the selected project and does not have any effect on import of projects.	Always imported
ProjectVersions	`ProjectVersions` in Jira.	Always imported
ProjectComponents	`ProjectComponents` in Jira.	Always imported
ProjectCategories	`ProjectCategories` in Jira.	May depend on Jira version.
StatusCategories	Categorization of `Statuses` associated with `Issues`.	May depend on Jira version.
Sprints	`Sprints` in Jira.	Imported if `Sprints` or `Sprints and Boards` is selected in the **JIRA Sprint Import Option** field. Relevant only for Jira versions 5-7. If the Jira version is below version 7, then only `Sprints` associated with `Issues` will be imported.
GroupMembers	`Groups` and the associated persons.	May depend on Jira version.
Boards	Scrum `Boards` in Jira.	Imported if `Sprints and Boards` is selected in the **JIRA Sprint Import Option** field. Relevant only for Jira version 7.

Jira Object	Content of Import	Context of Import
Board-Sprints	`Sprints` associated with the imported `Boards`.	Imported if `Sprints and Boards` is selected in the **JIRA Sprint Import Option** field. Relevant only for Jira version 7.
Board-Projects	`Projects` associated with the imported `Boards`.	Imported if `Sprints and Boards` is selected in the **JIRA Sprint Import Option** field. Relevant only for Jira version 7.

⚠ The JIRA user who will execute the Jira import should have his / her language set to Alfabet primary language (English) in JIRA. In addition, the user should have the necessary permissions to import the following data.

- Issues (using the Search Issues API)
- Projects
- Project Categories
- Project Components
- Project Versions
- Status Categories
- Group Members

- Fields
- Issue Types
- Issue Statuses
- Boards
- Sprints per Board
- Projects per Board

⚠ Errors due to characters that are non-compliant for XML or JSON processing may occur when importing Jira content to Alfabet. For example, this could occur when importing JIRA bug tickets that have descriptions in which error dumps or code snippets, etc. have been pasted. Therefore, it is highly recommended that you consider the meaningfulness of the data that will be imported from Jira to Alfabet. For example, in the case of Jira issues for bug fixes, the number of bug fixes for applications may be relevant in portfolio management whereas the detailed descriptions of the bug fixes typically will not be relevant. Please consider this when configuring the respective JIRA import scheme using the **JIRA Import Assistant**.

After the XML object **JIRAIntegrationConfig** has been defined, you must define a new ADIF import scheme to configure data integration using the **JIRA Import Assistant**:

1) In the **ADIF** tab in Alfabet Expand, right-click the **ADIF Schemes** root node in the explorer or any subfolder and select **Create Import Scheme**. The new import scheme is added to the explorer. The attribute window of the new import scheme is displayed on the right.

2) In the attribute window, set the following attributes for the ADIF import scheme:

- **Name**: Enter a unique name. The name is used to identify the ADIF import scheme in technical processes. It must be unique and should not contain white spaces or special characters.

- **Assistant**: Select `JiraImport_Assistant` from the drop-down list.

- **Import File Required**: Ensure that the value `False` is set.

- **Commit After Run**: If set to `True`, the result of the data import is written persistently to the Alfabet database. If set to `False`, the import process will be rolled back after execution and no changes will be written to the database. Configuration of the automatic start of workflows during import is ignored when **Commit After Run** is set to `False`. It is recommended that you set **Commit After Run** to `False` for a new import scheme to allow debugging without the risk of corrupting the database. After the successful testing of the data import and verification that the resulting changes to the Alfabet database are as expected, you can reset the **Commit After Run** attribute to `True` to perform regular data import.

    ⚠ Please note the following:

    - Setting the **Commit After Run** attribute rolls back all changes to data records in existing tables caused by DML statements. The creation or deletion of tables is not included in the roll back. For example, if you test an ADIF scheme that is configured to persistently write temporary tables to the database, these temporary tables will be created persistently even if **Commit After Run** is set to `False`. SQL commands of the type **OnActivate** are also excluded from roll back.

    - When new objects are created during an ADIF import job, the data bind mechanism assigns `REFSTR` values for the new objects. When **Commit After Run** is set to `False`, the objects are not created in the database, but nevertheless the `REFSTR` values are regarded as in use and will not be used for data bind in the next ADIF run unless the Alfabet Server or Alfabet Expand application used to process the ADIF jobs is restarted.

    - Changes triggered by **OnActivate** commands are not rolled back if the option **Commit After Run** is set to `False` for the import scheme.

- **Drop Temp Tables**: If set to `True`, all temporary tables are dropped after import. Only the changes to the Alfabet database are stored persistently. If set to `False`, the temporary tables are kept in the database after import is finished. Storing temporary tables persistently is only required for special import/export cycles designed for data manipulation that require input from the temporary tables of a previously set import. In most cases, setting this attribute to `True` is recommended to clean the database of data that is not part of the Alfabet meta-model.

3) In the explorer, right-click the node of the new ADIF import scheme and select **Create ADIF Scheme Details Using the JIRA Import Assistant**. A warning dialog is displayed.

    ⚠ Using the assistant to modify an existing, already configured ADIF import scheme will over-write all automatically generated parts of the ADIF scheme. If changes have been made to these import sets/entries, these changes will get lost.

4) Click **Yes**. The assistant opens in your standard Web browser.

5) In the field **Data Connection**, all XML elements `DataConnection` in your **JIRAIntegrationConfig** XML object will be displayed. Click the data connection to import the Jira data by means of the selected ADIF import scheme and click the button **OK** below the field. A message is displayed that reminds you to reread the meta-model in Alfabet Expand.

6) To do so, click the **Meta-Model** menu in the toolbar of Alfabet Expand and select **Reread Meta-Model**.

7) Return to the import scheme you have created for the Jira import and right-click the node and select **Modify ADIF Scheme Details Using the Jira Import Assistant**.

8) The **JIRA Import Assistant** opens. By default, if no filter criteria are provided and no explicit fields are selected, all issues along with all their standard and custom fields will be imported from the selected Jira connection. Define the following fields in the **JIRA Import Assistant**:

9) Select the **Net Change** checkbox if only Jira issues should be imported that have been changed since the last update of the ADIF scheme.

10) In the **Import Filter Type** drop-down list, define the type of import you want to implement. Specify one of the following:

- `Import Data Using List Box` if the issues to be imported may be filtered via certain predefined filter criteria. These filter criteria are either selected project/s and/or an issue type and/or issue status filters. Define the following fields:

    ⚠ Please note that the following filter criteria settings are cumulative. This means that if some projects are selected and the user searches for and selects another project, the previously defined filter criteria will remain selected unless they are explicitly unchecked. For example, if you select a project and define the **JIRA Issue Types** and **JIRA Issue Statuses** fields and then sect a second project in the **JIRA Projects** field, the settings defined in the **JIRA Issue Types** and **JIRA Issue Statuses** fields for the first project will remain selected unless they are explicitly unchecked.

    - **JIRA Projects**: Select one or more Jira projects for which issues shall be imported. You can enter a name or part of a name and click the **Search** button to find projects. Please note that this filter only applies to the import of issues. Independent from the setting of this field all Jira projects will be imported.

    - **JIRA Issue Types**: Select one or more Jira issue types to import.

    - **JIRA Issue Statuses**: Select one or more Jira issue statuses to import.

- `Import Data Using JIRA Query Language Definition` to specify to filter the Jira issues to import. The query must be written in Jira Query Language and entered in the **Enter JQL Query** field. For more information about Jira query language, please consult the documentation delivered with the Jira product.

11) In the **JIRA Sprint Import Option** field, specify whether sprints only or sprints and boards should be imported. The availability of these options will depend on the Jira version specified for the connection.

    ⚠ Please note that the issue selection settings are cumulative. This means that if some issue fields are selected and the user searches for and selects another project, the previously selected issue fields will remain selected unless they are explicitly unchecked.

12) You can specify which information should be imported for the projects (and, if relevant sprints and/or boards):

- In the **JIRA Standard Issue Fields to Import** field, specify one or more standard issue fields to import. If no standard issue field is selected, all standard issue fields will be imported. The temporary tables in ADIF will be automatically created for standard issues.

- In the **JIRA Custom Issue Fields to Import**, specify one or more custom issue fields to import. You can filter the custom issue fields displayed in the combo-box field by selecting a custom field type (string, number, etc.) in the drop-down list. If no custom issue field is selected, all custom issue fields will be imported.

    ⓘ Please note that the custom issues fields will be imported to a temporary table `Is-sueCustomFields`. Custom issue fields may have scalar values (such as `FieldName`, `Value`, and `FieldID`) or referenced values (such as `Group`). For referenced values, the

REFID will be added to the temporary table. If no REFID is available for the custom is-
sue field, the REFNAME will be added to the temporary field.

⚠️ Please note that the definition of the custom field type (String, Integer, etc.) set-
tings is cumulative. This means that if some custom field type (String, Integer, etc.)
are selected and the user searches for and selects another project, the previously se-
lected custom field types (String, Integer, etc.) will remain selected unless they are
explicitly unchecked.

13) Click the **OK** button to trigger the ADIF scheme to fetch the specified content from Jira and close the
JIRA Import Assistant. The temporary tables will be generated and displayed below the import
scheme node in the explorer. Or click **Cancel** to close the editor without triggering the execution of the
import scheme. If you click **Cancel**, your settings in the **JIRA Import Assistant** will not be saved.

14) Close the browser window and execute **Meta-Model** > **Reread from Database** in Alfabet Expand.
You will then see all automatically generated ADIF elements in the ADIF scheme. Each ADIF import
entry will have data import to temporary tables defined via the attribute elements in the **Attributes**
folder. The import to the standard Alfabet database tables is not included into the configuration.

15) To complete the integration of the data to the Alfabet database, additional configuration of the import
scheme is required to map the imported Jira data in the temporary tables to temporary tables that will
be created for the relevant Alfabet object classes and objects. The configuration required for this last
step of the import will depend on how your enterprise chooses to implement the Jira data in Alfabet.
For information about the configuration options available via ADIF import schemes, see the reference
manual *Alfabet Data Integration Framework*.

Exporting Alfabet Data to Jira

The export of Alfabet data to Jira® is available to support the planning and prioritization of operative develop-
ment work and provides support for bimodal IT and agile projects in the context of portfolio evaluation. The goal
of the Jira export is to create new issues in Jira based on issues that have been captured via demands, features,
issues, value nodes, etc. in Alfabet. In Jira, issues are typically assigned to a project and projects will typically
have a ProjectVersion and possibly ProjectComponent defined.

👁️ For example, a banking enterprise captures demands in order to plan the modification of the bank's
trading capabilities. The demands are assigned to projects in Alfabet. The Project stereotype "Pro-
gram" in Alfabet could be mapped to Projects in the Jira instance. The demands will be exported
as issues to Jira in order to implement the projects and operationalize the needed changes. The sub-
ordinate project stereotype Project could be mapped to the Jira ProjectVersion and the project
stereotype Project Step would be mapped to the Jira ProjectComponent. The following mapping
schema provides an example of how Jira projects could be mapped to a project structure in Alfabet:

Alfabet Class	Jira Class
Project Stereotype 1	Project
Project Stereotype 2	ProjectVersion
Project Stereotype 3	ProjectComponent

Demand	`Issue`

> ℹ An overview of all Jira objects that can be imported based on the import of Alfabet data to Jira is provided in the section Importing Jira Data to Alfabet.

Because issues in Jira are typically assigned to Jira projects, your enterprise must first consider the following questions in order to conceptualize the mapping of your enterprise's data in Alfabet to Jira:

- Which Jira projects will the exported issues be mapped to. For example, a Jira project may target a product such as FD Trading. Projects in Jira can either be found by specifying the project key of the projects or by finding the relevant projects based on key identifiers.

- Which Jira issue type will the information from Alfabet be mapped to. For example, the Jira issue types Bug or Change Requests.

- Which objects in Alfabet will be mapped as issues to Jira. Typically, this would be a demand, issue, feature, or value node, but other classes could also be mapped. For example, the demand Integrate Risk Management could be exported to Jira as a change request issue.

- Will existing issues in Jira be updated with data from the Alfabet objects or will new issues be created in Jira?

- Once it is understood which Alfabet objects will be exported to which Jira projects, the integration pattern must be specified to map the relevant Alfabet objects to Jira projects, project versions, and project components. You may specify any level in the Jira project hierarchy to begin the integration with. It is possible to implement one or more predefined integration patterns. For example, one of the integration patterns for ICT objects allows you to map a selected ICT Object Stereotype to the Jira project, Application (or stereotype) to the Jira project's version, and Local Component (or stereotype) to the Jira project's component. Whether your enterprise chooses an integration pattern for ICT objects or projects will depend on your methodology of using Jira. If your company associates demands with products in Alfabet, the ICT object integration pattern may be more relevant, but if demands are associated with projects, the project hierarchy integration pattern may be more suitable.

- Which Jira instance will you connect to in order to map the Alfabet objects to issues for the specific projects. Alfabet 's integration with Jira supports multi-instance scenarios as typical for large enterprises.
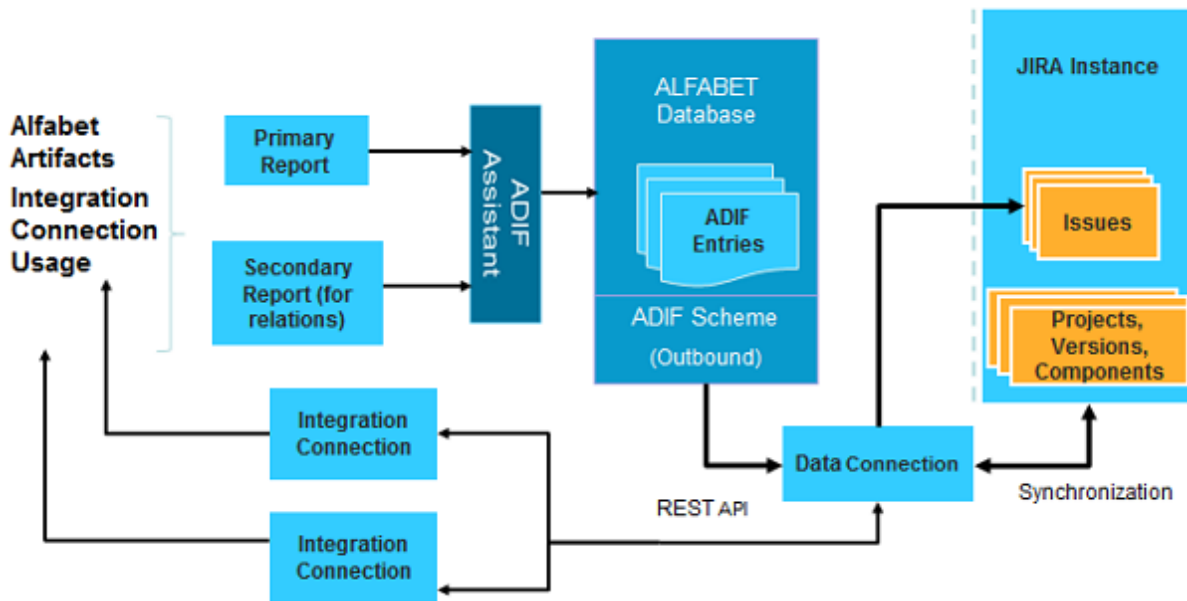
FIGURE: Overview of export to Jira

The following describes an overview of the steps required to configure the export to Jira.

1) Configure custom properties of type `String` in order to populate values in Jira for projects, project versions, and project components. Custom properties will be needed to populate the Jira identifiers such as `Name`, `ID`, or `Key`. Please note the following character restrictions for the Jira user interface:

Values in Jira	Jira Projects	Jira Project Versions/Project Components	Jira Issues
`Name`	80	255	not relevant
`ID`	18	18	18
`Key`	10	not relevant	10
`Summary`	not relevant	not relevant	255

2) Configure the XML object **JIRAIntegrationConfig**. The same XML object **JIRAIntegrationConfig** used to import data from Jira to Alfabet is used to configure the connection to export Alfabet data to Jira. The following should be defined in the XML object **JIRAIntegrationConfig**:

- XML element `DataConnection`: Specify data connections to one or more Jira instances.

- XML element `ExportSettings`: Specify the Alfabet report categories that will be used by the ADIF assistant to list the primary and secondary configured reports that find the objects and references in Alfabet that will be exported as issues to Jira.

- XML element `JIRADataTypeIdentifierTypes`: Specify the identifiers needed to search for projects and issues in Jira. This includes an identifier to search for projects (such as `Key`) and identifier to search for issue types (such as `Name`).

- XML element `JIRAProjectFilters`: Specify filters used when searching for projects while mapping them with Alfabet artifacts. For example, the filter could specify that only projects with a project type equal to "Software" or "Business" are returned.

- XML element `AlfabetClassMappingStandardPropertySettings`: Specify the standard object class properties of the type `String` to populate the fields in the **Jira Connection for Project-Based Integration** and **Jira Connection for Architecture-Based Integration** editors available in the **Jira Connection** view in the **Integration Solutions Configuration** functionality.

3) Configure the reports that will be used to find the Alfabet objects that will be exported as issues. You may configure two configured reports: A primary report finds the Alfabet objects such as demands, issues, features, or value nodes based on scalar attributes. The secondary report is used to complement the issue information with reference arrays, string arrays, etc. (n:n relationships).

4) Configure a report to find the Alfabet object's that are allowed to integrate with Jira projects. The objects are typically found via the stereotype specification in the **Stereotype Filter** field in the **Jira Connection for Project-Based Integration** and **Jira Connection for Architecture-Based Integration** editors available in the **Jira Connection** view in the **Integration Solutions Configuration** functionality. The configured report can be selected in the **Object Filter Report** field in the editors as an additional mechanism to find objects. The report must provides a list of Alfabet REFSTRs of the appropriate projects that are allowed to integrate with Jira projects.The **Link to Jira Project** editor option will be available in the **Action** menu in the object profiles of the projects found via the definition of the **Stereotype Filter** field or the **Object Filter Report** field.

5) Define which information from Alfabet shall be used to constitute Jira issues by defining the ADIF export scheme by means of the **Jira Export Assistant**. The **Jira Export Assistant** focuses on two Alfabet reports configured to find the Alfabet objects to export (primary report) and the references to export (secondary report). The following must be specified:

> ⚠ Jira export is based on the principle of consistency between the Jira user interface and Jira API. However, the Jira user interface and Jira API may have different formats, which is the case for date fields for example. Please note that users configuring the ADIF export scheme should not rely solely on the filter fields in the Jira user interface but instead should understand the format that the Jira API expects and configure the ADIF export scheme accordingly. If the formats are inconsistent, the results of the export from Alfabet to Jira may not be correct.

- In the **Map Primary Alfabet Report Columns to Jira Issue Fields** tab, select the data connection to use.

- Select a project and an issue type that have at least one issue defined and have a Jira issue structure that is representative for the targeted export. The mapping table will be pre-populated with the fields specified for the create and update actions of the selected project and issue type. Any fields that are not returned based on the schema retrieval can be individually added. The Jira issue fields will be mapped to the relevant columns of the primary report.

- Select the primary report to find the Alfabet objects.

- For each field displayed in the dataset, map the relevant Alfabet project in the **Report Column Name**. The dataset indicates which Jira fields are mandatory to create an issue as well as to update an issue.

- In the **Map Secondary Alfabet Report Columns to Jira Issue Fields** tab, select one or more secondary reports to specify the mapping for multi-valued attributes of Jira issues such as the assignment of Jira project versions and Jira project components.

6) Once the mapping has been completed, event templates must be configured to trigger the ADIF export schemes when the users execute the **Synchronize with Project Data** functionality in the relevant

object profiles. For the export from Alfabet, the event template should return a set of objects that can be mapped to Jira Issues. The set of objects returned by the event template must be a subset of the objects returned by the primary report of the ADIF job referenced by the event template. Only the returned objects that are common to both the event template and the primary report will be exported to Jira. For additional information about the configuration of event templates for export from Alfabet, see the section Configuring the Event Templates to Trigger the ADIF Export Schemes for Synchronization.

7) The semantic connections must be configured that specify the integration pattern to use for the database connection configured in the XML object **JIRAIntegrationConfig**. A semantic definition should be created for each data connection configured in the XML object **JIRAIntegrationConfig**. You can configure multiple semantic data connections for each data connection configured in the XML element **DataConnection** in the XML object **JIRAIntegrationConfig**. For each semantic connection you create, you must specify the integration pattern to map objects in the Alfabet object hierarchy to the Jira project, project version, or project component. You can choose an architecture-based integration pattern mapping or a project-based integration pattern mapping. You may specify any level in the Jira project hierarchy to begin the integration with. For example, you could choose to map Alfabet applications to Jira project versions.

The following integration patterns may be specified for a project-based approach:

Integration Type	Jira Project	Jira Project Version	Jira Project Component
Jira Projects, Project Versions, and Project Components	**Project** Stereotype 1	**Project** Stereotype 2	**Project** Stereotype 3
Jira Projects and Project Versions	**Project** Stereotype 1	**Project** Stereotype 2	
Jira Project and Project Components	**Project** Stereotype 1		**Project** Stereotype 3
Jira Projects	**Project** Stereotype 1		
Jira Project Versions		**Project** Stereotype 2	
Jira Project Versions and Project Components		**Project** Stereotype 2	**Project** Stereotype 3
Jira Project Components			**Project** Stereotype 3

Jira issues will be mapped to the relevant Alfabet object class (for example, **Demand**, **Feature**, **Issue**, **Value Node**, etc.) specified in the relevant ADIF scheme associated with the Jira connection.

The following integration patterns may be specified for an architecture-based approach:

Integration Type	Jira Project	Jira Project Version	Jira Project Component
Jira Projects, Project Versions, and Project Components	ICT Object (or stereotype)	Application (or stereotype)	Local Component (or stereotype)
	ICT Object (or stereotype)	Component (or stereotype)	Local Component (or stereotype)
	ICT Object (or stereotype)	Standard Platform (or stereotype)	Platform Element (or stereotype)
Jira Projects and Project Versions	ICT Object (or stereotype)	Application (or stereotype)	
	ICT Object (or stereotype)	Component (or stereotype)	
	ICT Object (or stereotype)	Standard Platform (or stereotype)	
Jira Projects	ICT Object (or stereotype)		
Project Versions		Application (or stereotype)	
		Component (or stereotype)	
		Standard Platform (or stereotype)	
Project Versions and Project Components	ICT Object (or stereotype)	Application (or stereotype)	Local Component (or stereotype)
	ICT Object (or stereotype)	Component (or stereotype)	Local Component (or stereotype)
	ICT Object (or stereotype)	Standard Platform (or stereotype)	Platform Element (or stereotype)

Integration Type	Jira Project	Jira Project Version	Jira Project Component
Jira Projects, Project Versions, and Project Components		Application (or stereotype)	Local Component (or stereotype)
		Component (or stereotype)	Local Component (or stereotype)
		Standard Platform (or stereotype)	Platform Element (or stereotype)
Project Components		Component (or stereotype)	Local Component (or stereotype)
		Standard Platform (or stereotype)	Platform Element (or stereotype)

Jira issues will be mapped to the relevant Alfabet object class (for example, **Demand**, **Feature**, **Issue**, **Value Node**, etc.) specified in the relevant ADIF scheme associated with the Jira connection.

The following information is available:

- [Configuring Connections and Mapping to Export Data from Alfabet to Jira](#)

- [Configuring the Primary and Secondary Reports](#)

- [Configuring Object Filter Reports](#)

- [Configuring the ADIF Export Assistant to Export Alfabet Data to Jira](#)

- [Configuring the Event Templates to Trigger the ADIF Export Schemes for Synchronization](#)

Configuring Connections and Mapping to Export Data from Alfabet to Jira

The following must be configured in the XML object **JIRAIntegrationConfig**:

- Specify data connections to one or more Jira instances in the XML element `DataConnection`.

- Specify the Alfabet report categories that will be used by the ADIF assistant to list the primary and secondary configured reports that find the objects and references in Alfabet that will be exported as issues to Jira. This is done via the XML element `ExportSettings`.

- Specify the identifiers needed to search for projects and issues in Jira via the XML element `JIRADataTypeIdentifierTypes`. This includes an identifier to search for projects (such as `Key`) and identifier to search for issue types (such as `Name`).

- Specify filters to limit the Jira projects, project versions, and project components available in the **Link to Jira Project** editor when users link an Alfabet object to Jira. This is done via the XML element

`JIRAProjectFilters`. For example, the filter could specify that only projects with a project type equal to "Software" or "Business" are returned.

- Specify the standard object class properties of the type `String` to populate the Jira property fields in the **Jira Connection for Project-Based Integration** and **Jira Connection for Architecture-Based Integration** editors available in the **Jira Connection** view in the **Integration Solutions Configuration** functionality. This is done via the XML element `AlfabetClassMappingStandardPropertySettings`.

ⓘ You can refer to the **XML Template** attribute for an example of the specification of the XML object **JIRAIntegrationConfig**.

The connection to Jira® must be defined in the XML object **JIRAIntegrationConfig**. Multiple connections to Jira can be configured. Data will be imported from all defined Jira instances via the same ADIF import scheme.

ⓘ Please note that server variables can be used in the XML object **JIRAIntegrationConfig** to read the value of an XML element at runtime from the server alias configuration of the Alfabet Web Application when a connection to Jira is established. For information about server variables, see Configuring Server Variables for Integration and Interoperability Solutions.

To define a connection to Jira:

1) In Alfabet Expand, go to the **Presentation** tab and expand the explorer nodes **XML Objects** > **IntegrationSolutions**.

2) Right-click **JIRAIntegrationConfig** and select **Edit XML...**. The XML object **JIRAIntegrationConfig** opens.

ⓘ 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 **JIRAIntegrationConfig**. 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.

3) Add a child XML element `DataConnections` to the root XML element `JIRAIntegrationConfig`.

4) Add a child XML element `DataConnection` to the XML element `DataConnections`. The XML element `DataConnections` may contain multiple child XML elements `DataConnection`, each defining a connection to a different Jira instance.

5) Specify the following XML attributes for each XML element `DataConnection`:

- `Name`: The name of the connection to Jira that is used to identify the connection in the ADIF scheme configuration. A unique name must be specified for the connection. The name of the data connection can be assigned to a semantic Jira connection in the **Data Connection** field in the he **Jira Connection for Project-Based Integration** and **Jira Connection for Architecture-Based Integration** editors.

- `MajorVersion`: Specify the major Jira version (5, 6, 7) in order to call the relevant Jira API (for Jira v. 5 and 6, the Jira Server API; for Jira v. 7, Jira Server API and Jira Agile API). You should only specify the major version (5, 6, 7) and not the minor version number of the Jira instance. (For example, specify `MajorVersion = 5`, not `MajorVersion = 5.3.2`.).

- `ServerURL`: Specify the URL to the Jira instance for the connection.

- `UserName`: Specify the user name of the Jira user for the connection.

- `Password`: Specify the password of the Jira user for the connection.

- `MaxThreadCount`: The number of parallel threads that may be used for simultaneous data transfer is set per default to -1 in the XML attribute `MaxThreadCount`. This default value can be modified as needed. The number of CPU cores is typically a good starting point for the `MaxThreadCount` value. The specification of the XML attribute `MaxThreadCount` will override the default behavior.

- `HTTPTimeout`: Specify a timeout value in seconds if required. Otherwise, the default HTTP timeout value will be used.

6) Add a child XML element `ExportSettings` to the root XML element `JIRAIntegrationConfig`:

- Specify the Alfabet report categories containing the configured reports that shall be available in the **ADIF Export Assistant**. The report categories should contain the relevant primary and secondary configured reports that find the objects and references in Alfabet that will be exported as issues. The same report categories may be specified for both the primary and secondary reports:

  - `PrimaryAlfabetReportCategory`: Specify one or more relevant report categories of the configured reports that will find the Alfabet objects to export. Multiple report categories should be listed as comma-separated values. All configured reports assigned to the specified category value will be listed in the **Primary Alfabet Report** field in the **Map Primary Alfabet Report Columns to JIRA Issue Fields** tab in the **ADIF Export Assistant**.

  - `SecondaryAlfabetReportCategory`: Specify one or more relevant report categories of the configured reports to specify the mapping for multi-valued attributes of Jira issues such as the assignment of Jira project versions and Jira project components. Multiple report categories should be listed as comma-separated values. All configured reports assigned to the specified category value will be listed in the **Select Secondary Report to Define Mapping** field in the **Map Secondary Alfabet Report Columns to JIRA Issue Fields** tab in the **ADIF Export Assistant**.

  - `UpdateJIRAObjectsOnStructureSync`: Enter `True` if Jira projects should be updated when synchronizing the Alfabet architecture/project structure. Enter `False` if Jira projects should not be updated when synchronizing the Alfabet architecture/project structure. The XML attribute `UpdateJIRAObjectsOnStructureSync` is set to `True` per default.

- Specify the following in order to enhance the performance when importing a large set of Jira issues to Alfabet:

  - `BulkCreateMode`: Set to `True` for large scale synchronization with single Jira Instances. This is not relevant if a large number of issues is synchronized across a considerable number of Jira instances. The default value is `False`.

  - `BulkCreateBatchSize`: Specify a batch size for the creation of issues in the Jira system that is being connected. The default value is 200.

  - `JIRAFieldIDToMapAlfabetObject`: Specify the custom field in Jira that stores object ID of the Alfabet object on the issue in Jira. It is recommended that this is a ReadOnly field in Jira. Please note that the custom field should be present on all Jira instances used to create and update/edit issues in Jira so that the field is available on the associated Jira API. This field must be represented by the fully-qualified identifier (such as customfield_10202 where the five digit integer is the variable).

  - `BulkCreateMaxThreadCount`: Specify the number of parallel threads to be spawned when creating issues in Jira. This number is a function of various aspects of the Jira installation (such as machine size and configuration of Apache web server used as application server for Jira and the size, infrastructure, and configuration of the Jira database (in particular size of the connection pool available). If not specified, the `MaxThreadCount` property value will be used.

7) Specify the identifiers (`ID`, `Key`, or `Name`) needed to find Jira projects when synchronizing the Alfabet structure (project hierarchy or ICT object structure) to the Jira project hierarchy or to export issues to Jira. Jira identifiers include `ID`, `Key`, or `Name` for projects and `Key` and `Name` for issues. To do so, add a child XML element `JIRADataTypeIdentifierTypes` to the root XML element `JIRAIntegrationConfig` and specify the following XML attributes for the XML element `DefaultDataTypeIdentifierTypes`:

- `DefaultDataTypeIdentifierType`: The default identifier if no matches are found via the other specified data types. This is typically `ID`.

- `IssueDataTypeIdentifierType`: Specify an identifier to find Issues in Jira. Jira issue identifiers include `ID` or `Key`.

- `ProjectDataTypeIdentifierType`: Specify an identifier to find for projects in Jira. Jira project identifiers include `ID`, `Key`, or `Name`.

- `IssueTypeDataTypeIdentifierType`: Specify an identifier to find issues in Jira via the `IssueType` field values.

- `PriorityDataTypeIdentifierType`: Specify an identifier to search for projects in Jira via the `Priority` field values.

- `ComponentDataTypeIdentifierType`: Specify an identifier to search for projects in Jira via the `Component` field values.

- `VersionDataTypeIdentifierType`: Specify an identifier to search for projects in Jira via the `Version` field values.

8) Specify filters to limit the number of projects displayed in the **Link to JIRA Project** editor available in the object profile of a relevant Alfabet object. You can specify multiple conditions for each filter. The filter can be assigned to a semantic Jira connection in the **Project Filter** field in the **Jira Connection for Project-Based Integration** and **Jira Connection for Architecture-Based Integration** editors available in the **Integration Solutions Configuration** functionality available in the Alfabet user interface. To do so, add a child XML element `JIRAProjectFilters` to the root XML element `JIRAIntegrationConfig` and add a child XML element `Filter` for each filter definition required.

- Specify the following XML attributes for the XML element `Filter`.

  - `Name`: Enter a name for the filter.

  - `Desc`: Enter text providing information about the filter criteria required. This will be displayed in the **Project Filter Description** field in the **Jira Connection for Project-Based Integration** editor and **Jira Connection for Architecture-Based Integration** editor available in the **Jira Connection** view in the I **Integration Solutions Configuration** functionality.

- Add one or more XML elements `Condition` as a child element of the XML element `Filter` and specify the following XML attributes for the XML element `Filter`.

  - `PropertyName`: Specify the property of the Jira project that should be used as filter criteria. You may specify any of the following: `Key`, `Name`, `Type`

  - `Operation`: Specify the operation for the property. You may specify any of the following operations: `StartsWith`, `Contains`, `Equals`.

  - `Value`: Specify the value that should be used to fulfill the search criteria.

9) Specify the standard object class properties of the type `String` to populate the relevant fields to map Jira properties in the **Jira Connection for Project-Based Integration** editor and **Jira Connection for Architecture-Based Integration** editors available in the **Jira Connection** view in the **Integration Solutions Configuration** functionality. Custom properties of the type `String` will automatically be added to the editors. To do so, add a child XML element `AlfabetClassMappingStandardPropertySettings` to the root XML element `JIRAIntegrationConfig` for each class definition required. Please note the following:

- If your enterprise will use a project-based integration, specify the following XML attributes for the XML element `AlfabetClassMappingStandardPropertySettings`.

  - `Class`: Enter `Project`.

  - `PermittedStandardProperties`: Specify the standard properties that should be available in the various editor fields in the **Jira Connection for Project-Based Integration** editor. The field allows you to select the Alfabet property that is used to store specific attributes for projects in Jira.

- If your enterprise will use an architecture-based integration, specify the following XML attributes for the XML element `AlfabetClassMappingStandardPropertySettings`. The following classes are relevant depending on the integration pattern specified:

  - `Class`: Enter the name of the class relevant for the architecture-based integration. For example, if you select the integration pattern **ICT Object** > `Application` > `Local Component`, you must create a `Class` entry for each class other than **ICT Object**.

    (i) An entry cannot be created for the class **ICT Object** because the Jira `Project` values such as `ID`, `Key` and `Name` are stored in custom properties only.

  - `PermittedStandardProperties`: Specify the standard properties that should be available in the various editor fields in the **Jira Connection for Architecture-Based Integration** editor. The field allows you to select the Alfabet property that is used to store specific attributes for projects in Jira.

10) In the toolbar of Alfabet Expand, click the **Save** 🖫 button to save your changes.

Configuring the Primary and Secondary Reports

For each entry created in the ADIF export scheme, a primary configured report and possibly a secondary configured report should be created to find the Alfabet objects that will be exported as issues. The primary and secondary reports must be of the type `Query` or `NativeSQL`. Please note that only the primary configured report is mandatory. The secondary report is only required if other issue fields of n:n relationships or that have sub-properties must be exported.

The primary report finds the Alfabet objects such as demands, issues, features, or value nodes based on scalar attributes. The primary configured report must be defined in order to populate the Jira issue fields that require only a single scalar value. Typical values would be `Summary`, `Description`, `Project`, `Issue Type`, etc. The secondary report is used to complement the issue information with reference arrays, string arrays, etc. (n:n relationships) and thus should be defined to populate the JIRA issue fields that may require more than one value (arrays) or that have sub-properties such as `Issue Link` or `Attachment`. The fields that might be filled with multiple values include, for example, `Fix Versions`, `Affects Versions`, `Components`, etc. The secondary report is also required to provide values of fields that have sub-properties. For example, the field `Issue Link` requires the values for parent issue and the type of relation.

> For more information about defining configured reports, see the chapter *Configuring Reports* in the reference manual *Configuring Alfabet with Alfabet Expand*.

> All issues are created in single record-by-record export. Please note that due to limitations of Jira, a rollback is not possible.

Please note the following information regarding the report configuration and execution:

- The identifiers that are relevant for the primary and secondary reports must be specified in the XML element `JIRADataTypeIdentifierTypes` in the XML object **JIRAIntegrationConfig**.

- Multiple primary and secondary reports may be defined. The **Category** attribute of the reports must be defined. The specification of the XML element `PrimaryAlfabetReportCategory` in the XML object **JIRAIntegrationConfig** will determine which reports will be available for a selected data connection in the **Primary Alfabet Report** field in the **Map Primary Alfabet Report Columns to JIRA Issue Fields** tab in the **ADIF Export Assistant**. Likewise, the XML element `SecondaryAlfabetReportCategory` will determine the secondary reports available in the **Map Secondary Alfabet Report Columns to JIRA Issue Fields** tab.

- At least one primary configured report must be defined as the source of the export data. This report must find the Alfabet objects to export.

  > The following SQL query is an example of the query specified for a primary report:

  ```
  SELECT j.REFSTR, j.REFSTR AS 'ConnID', (SELECT EXTERNALID
  FROM INTEGRATIONCONNECTIONUSAGE WHERE A_OBJECT = ftr.REFSTR)
  AS 'ID', ftr.REFSTR AS 'Object', ftr.DESCRIPTION, ftr.STATUS
  'Status', 'New Feature' 'IssueType', ftr.NAME, 'Customer' AS
  'User', icto.SC_JIRAKEY 'Project'

  FROM FEATURE ftr, APPLICATION app, ICTOBJECT icto,
  JIRA_DBCONNECTION j, INTEGRATIONCONNECTIONUSAGE ictou

  WHERE ftr.OBJECT = app.REFST

      AND app.ICTOBJECT = icto.REFSTR

      AND j.NAME = 'ICT-App-LCom'

      AND ictou.CONNECTION = j.REFSTR

      AND ictou.A_OBJECT = icto.REFSTR
  ```

  The output of the report shall populate Jira issue fields that require only a single scalar value such as `Summary`, `Description`, `Project`, `Issue Type`, etc. The primary configured report must return the following columns:

- Alfabet Integration Connection ID: The `REFSTR` of the Jira database connection used for the export.

- Alfabet Object ID: The `REFSTR` value of the Alfabet object that shall be exported and mapped as the Jira issue.

- Jira `Project ID` or `Key`: The `ID` or `Key` of the JIRA project must be included in the report. You can either provide it as a string or capture it in a custom property of an Alfabet object class that is mapped to JIRA projects and read it from this object class when defining the report.

> Whether `ID` or `Key` is specified will depend on the specification in the XML element `JI-RADataTypeIdentifierTypes` in the XML object **JIRAIntegrationConfig**. This applies to the columns `Project ID` or `Key`, `Issue ID` or `Key`, and `Jira Issue Type ID` or `Key`.

- Jira `Issue ID` or `Key`: Please note that this column is optional. If a value is provided for `Issue ID` or `Key`, the corresponding Jira issue will be updated with the new set of values. If a value for `Issue ID` or `Key` is not provided, a new issue will be created in Jira for that record. For issues that have already been exported from Alfabet, the Jira `Issue ID` can be read from the object class property `ExternalID` of the object class `IntegrationConnectionUsage`.

- `Jira Issue Type ID` or `Key`

- Other relevant single-value issue fields in Jira such as `Summary`, `Description`, `Assignee`, etc.

- A secondary configure report should be defined to find the references and populate the Jira issue fields that are non-scalar and that may require more than one value (arrays) or that have sub-properties such as `Issue Link` or `Attachment`. This includes, for example, `Fix Versions`, `Affects Versions`, `Components`, etc.

  - The names of the secondary report columns relevant for the Jira `Issue Field` columns must be specified in the XML element `JIRADataTypeIdentifierTypes` in the XML object **JIRAIntegrationConfig**. The export will only populate these columns and all other columns will be ignored. Please note the following:

    - All non-scalar fields (that have sub-properties) and fields with arrays that are part of the issue schema can be exported through the secondary report.

    - To export issue fields of type `Array` the users need to provide their values through 3 columns in the report: a column for `Issue Field ID`, a column for `Issue Field Name`, and a column for `Issue Field Value`.

    - To export Issue fields that have sub-properties like Issue link and Attachment, users need to map report columns for each individual sub-property to the corresponding `Issue Field` column in order to export them correctly.

  - A unique combination of the Alfabet Integration Connection ID, Jira `Project ID`, and Alfabet Object ID is required if the Jira `Issue ID` is not given or known by the primary or secondary reports. If multiple issues have same `Issue ID` and Alfabet Integration Connection ID or same combination of Alfabet Integration Connection ID, Jira `Project ID`, and Alfabet Object ID, the last `ID` value exported will override the `ID` other values and be written in the `IntegrationConnectionUsage` database table. However, only the attributes specified in the primary report will be overwritten and the update may therefore be cumulative. Please note the following regarding the secondary report:

    - If the combination of Alfabet Integration Connection ID, Jira `Project ID`, and Alfabet Object ID is same for multiple rows (and `Issue ID`/`Key` is empty), then all rows apply to an issue that will be created.

    - If the combination of Alfabet Integration Connection ID, Jira `Project ID`, and Alfabet Object ID are the same for multiple rows (and `Issue ID`/`Key` is not empty), then all the rows apply for the same existing issue that will be updated.

- An Alfabet property of type `StringArray` can be mapped to a Jira `Issue Field` column if a multi-select value should be set. Each value of the string array must be available as an individual option in the corresponding Jira `Issue Field` column.

The secondary configured report should return the following columns:

- Alfabet Object ID: The `REFSTR` value of the Alfabet object that shall be exported and mapped as the Jira issue.

- Jira `Issue field ID` or `Issue Field Name`: For custom fields, the `Issue field ID` should be specified since names may not be unique for custom fields. Standard fields can be identified via either `Issue Field ID` or `Issue Field Name`.

- Jira `Issue Field Value`: A new row will be created for any field that has sub-properties such as `Issue Link` or `Attachment`. For an `Issue Link` field, for example, the columns `Issue Link Outward Issue` and `Issue Link Relation` will be added to the standard list of issue field columns.

- Please note the following additional configuration requirements regarding the export of Web links and documents.

  - To export Alfabet Web links to Jira.

    1) In Jira, ensure a custom field of type `Text Field (multi-line)` is available. If this is not available, it is necessary to create a custom field of type `Text Field (multi-line)`.

    2) Navigate to the field configuration for the custom field.

    3) Navigate to the field and click `Renderers`.

    4) Select `Wiki Style Renderer`.

    5) In the Alfabet report, provide the value for the Web links using the format [new link|http://example.com].

  - To export documents to Jira.

    1) 23540Jira is unable to provide unique identifiers for attached documents. Therefore, if the same attachment is sent again for an issue, the issue will contain the same document multiple times. This is a limitation of Jira and it is advised that documents are only exported when required. The configured report should specify that documents are only added to the result data set if the `CREATION_DATE` and/or `CHANGE_DATE` properties of the class `ALFA_IDOCUMENT` return a date later than the last execution of the Jira issue synchronization job.

    2) In the configured report, a dataset instruction `RetrieveIDOCPath` must be defined to trigger the export of attachments to the runtime folder of the Alfabet Web Application The configuration of the dataset instruction is described in the section *Exporting Attachments to a Runtime Folder During Report Execution* in the chapter *Defining Queries* in the reference manual *Configuring Alfabet with Alfabet Expand*.

- If issues are created in Jira based on an Alfabet export, then the `Issue ID` or `Issue Key` used for the export will be updated in the `IntegrationConnectionUsage` database table. If the Jira `Issue ID` is not given or known by the primary or secondary reports, a unique combination of the Alfabet Integration Connection ID, Jira `Project ID`, and Alfabet Object ID is required. If multiple issues have same Issue ID and Alfabet Integration Connection ID or same combination of Alfabet Integration Connection ID, Jira `Project ID`, and Alfabet Object ID, the last `ID` value exported will override the `ID`

other values. However, only the attributes specified in the primary report will be overwritten and the update may therefore be cumulative.

- If the combination of Alfabet Integration Connection ID, Jira `Project ID`, and Alfabet Object ID is same for multiple rows (and `Issue ID/Key` is empty), then all rows apply to an issue that will be created.

- If the combination of Alfabet Integration Connection ID, Jira `Project ID`, and Alfabet Object ID are the same for multiple rows (and `Issue ID/Key` is not empty), then all the rows apply for the same existing issue that will be updated.

- The report definitions are stored internally as part of the ADIF scheme. If an update is made to either the primary or secondary configured report, ADIF scheme must be updated. To do so, the ADIF scheme needs to be opened for modification via the **ADIF Export Assistant**, select the relevant entry in the **Select Export Entry to Edit** field, click the **Edit** button and click the **Save Temporary Entry** button. Go to the second tab, select a relevant report in the **Select Secondary Report to Define Mapping** field, click the **Edit** button and click the **Save Temporary Entry** button. Repeat this for each secondary report that needs to be saved. When finished, click the **OK** button to save changes to the Alfabet database and close the **ADIF Export Assistant**.

Configuring Object Filter Reports

Optionally, you can configure a report to find the Alfabet objects that are allowed to integrate with Jira projects. The objects are typically found via the stereotype specification in the **Stereotype Filter** field in the **Jira Connection for Project-Based Integration** and **Jira Connection for Architecture-Based Integration** editors available in the **Jira Connection** view in the **Integration Solutions Configuration** functionality. However configured reports are an additional mechanism to find objects to integrate with Jira projects, Jira project versions, and Jira project components.

The configured reports can be assigned to a semantic Jira connection in the **Object Filter Report** field in the **Jira Connection for Project-Based Integration** and **Jira Connection for Architecture-Based Integration** editors. The **Link to Jira Project** editor option will be available in the **Action** menu in the object profiles of the objects found via the definition of the **Stereotype Filter** field or the **Object Filter Report** field.

Please note the following about the configured report:

- The report must return a list of Alfabet REFSTR in the first column of the appropriate projects that are allowed to integrate with Jira projects.

- The REFSTR must be for the same class that the **Object Filter Report** field is defined for.

  ⓘ When an Alfabet object is synchronized with Jira, the objects will only be synchronized if they are in the return set of the associated report defined in the **Object Filter Report** field or in the returned set of object stereotype specified in the **Stereotype Filter** field. For example, a report configured as an object filter for the class Application has been configured to return only applications with the status set to Approved. Thus, a new application with the status set to Draft shall not be used to create a new version in the associated Jira project. In this case, the **Link to Jira Project** editor option will be available in the **Action** menu in the object profile of the new application with the status set to Draft.

- The **Category** attribute pf the configured report must be set to `JIRAIntegration`. Note that only the configured reports for which the **Category** attribute has been set to `JIRAIntegration` will be displayed in the **Object Filter Report** field.

ⓘ For more information about defining configured reports, see the chapter *Configuring Reports* in the reference manual *Configuring Alfabet with Alfabet Expand*.

Configuring the ADIF Export Assistant to Export Alfabet Data to Jira

Once the Alfabet structure and Jira project structure have been mapped and linked, you must configure the export of issues to Jira based on the primary and secondary configured reports and the configuration of a corresponding ADIF scheme. To do so, you must map the fields in Jira used to capture information about issues to the relevant columns in the configured reports. The primary configure report allows you to capture scalar Jira issues fields with a single value and the secondary report allows you to capture fields that require more than one value (arrays) or that have that sub-properties such as `Issue Link` or `Attachment`. Once the mapping has been completed, the ADIF export scheme can be executed via a batch job and the relevant Alfabet objects such as demands, features, etc. will be exported as issues to Jira.

When you define the **JIRA Export Assistant**, the dataset in the assistant must first be filled with the relevant issue fields that are used to capture information about the issues in Jira. The various issue fields must be mapped to the columns of the configured report. To do so, you must define a schema retrieval that populates the dataset with the issue fields. If not all issue fields are retrieved by the schema, you can manually add the remaining issue fields. The schema retrieval requires the following definition:

- A data connection to use to retrieve the information from the Jira instance.

- A Jira project that has existing issues defined. Please note that this does not necessarily have to the actual project that the issues will be assigned to.

- The issue type for which you want to map the issue fields.

ⓘ The target Jira instance should have at least one issue of every issue type in the selected project in order to retrieve the schema information. If an issue type is selected for which no issue exists in the selected project, the issue fields cannot be retrieved and the dataset will not be populated.

⚠ Jira export is based on the principle of consistency between the Jira user interface and Jira API. However, the Jira user interface and Jira API may have different formats, which is the case for date fields for example. Please note that users configuring the ADIF export scheme should not rely solely on the filter fields in the Jira user interface but instead should understand the format that the Jira API expects and configure the ADIF export scheme accordingly. If the formats are inconsistent, the results of the export from Alfabet to Jira may not be correct.

Every issue is associated with one project. Alfabet may have objects (demands) that are associated with multiple projects.

1) Create an ADIF export scheme with at least one entry. Please note that data will be exported to all defined Jira instances via the same ADIF export scheme. For more information about how to create the ADIF export scheme and entry, see the reference manual *Alfabet Data Integration Framework*.

2) Set the **Alfabet User Interface Behavior** attribute of the export scheme to `VisibleExecutable` and click the **Save** button.

3) Right-click the ADIF export scheme and select **Create ADIF Scheme Details Using the Jira Export Assistant**. The **Jira Export Assistant** opens. The **Jira Export Assistant** has two tabs. The **Map Primary Alfabet Report Columns to Jira Issue Fields** tab must first be defined in order to trigger the primary report and fill the dataset in the editor with the scalar Jira issue fields that require one single value. Then the **Map Secondary Alfabet Report Columns to Jira Issue Fields** tab can be

defined to specify the JIRA issue fields that may require more than one value (arrays). For details about configuring the required reports, see the section Configuring the Primary and Secondary Reports.

4) In the **Map Primary Alfabet Report Columns to Jira Issue Fields** tab, specify the following:

- **Enter Export Entry Name**: To create a new export entry, enter the name of the entry defined for the ADIF export scheme.

  > Alternatively, to edit an existing export entry, select the export entry in the **Select Export Entry to Edit** field that you want to edit. The assistant fields will be updated with the existing definition. Click the **Edit** button to edit the fields described below.

- **Data Connection**: Select the relevant data connection to use to connect to the Jira instance.

- **JIRA Project for Schema Retrieval**: Select a representative project to retrieve issue schema. This project does not have to be the project that the issue is to be sent to. All the projects available in the Jira instance targeted by the connection will be displayed.

- **JIRA Issue Type for Schema Retrieval**: Select a representative issue type to retrieve issues.

  > ⚠ You must select a project and an issue type for which at least one issue can be retrieved and that has a Jira issue structure that is representative for the targeted export. The mapping table will be pre-populated with the fields available for the found issue(s). Any fields that are not returned based on the schema retrieval can be individually added. If an issue type is selected for which no issue exists in the selected project, the issue fields cannot be retrieved and the dataset will not be populated. An error message will be displayed in this case.

- **Primary Alfabet Report**: Specify the primary configured report to capture scalar Jira issues fields with a single value. All reports assigned to the report category defined for the selected data connection in the XML object **JIRAIntegrationConfig** will be displayed in the drop-down list.

- **Display Option for Alfabet Report Properties**: Select one of the following to determine the sorting of the data displayed in the dataset:

  - **Sort Report Columns as Defined in Report**: The column names in the configured report are displayed instead of the object class property names. The sort order is identical to the order of columns in the configured report.

  - **Sort Report Columns with Lexicographic Sorting**: The column names in the configured report are displayed instead of the object class property names. The sort order is alphanumeric.

  - **Sort Property Columns as Defined in Report**: The object class properties are displayed as <ObjectClassName>.<PropertyName>. The sort order is identical to the order of columns in the configured report.

  - **Sort Property Columns with Lexicographic Sorting**: The object class properties are displayed as <ObjectClassName>.<PropertyName>. The sort order is alphanumeric.

5) Click the **Save Temporary Entry** button. The **Map Jira Issue Fields to Report Columns** section displays all Jira issue fields found via the export entry definition. To add additional Jira issue fields, click the **Add More Fields** button and select the relevant fields in the **JIRA Fields** selector. Click **OK** to add the fields to the dataset.

- **Jira Issue Field ID**: Displays the ID of the Jira issue field.

- **Jira Issue Field Name**: Displays the name of the Jira issue field. The dataset indicates which Jira fields are mandatory to create an issue as well as to update an issue. All fields that are mandatory are displayed in red

- **Jira Issue Field Type**: Displays the type of the Jira issue field.

- ***C**: Displays an M for mandatory the export if the **Report Column Name** field must be mapped in order to create an issue in Jira.

- ***U**: Displays an M for mandatory the export if the **Report Column Name** field must be mapped in order to update an issue in Jira.

- **Report Column Name**: Select the Alfabet property that the Jira issue field shall be mapped to. Some rows can only be specified after the primary report has been executed.

> ⚠️ Please note the following regarding the Status issue field in Jira:
>
> - Jira issue fields may not be updated if the Status value cannot be correctly set for the Jira issue.
>
> - In order to create an issue in Jira based on an Alfabet object, the new issue must either be assigned the initial status permissible for the issue type or the status that follows the initial status. Please note the following:
>
>   - If the Alfabet status is not the same as the initial Jira status, the system will assess whether a status transition is possible between the two status values.
>
>   - If the status transition is not possible, the new issue will be deleted.
>
>   - If the status transition is possible, the issue will be created and the issues fields that may be updated in the current status of the Alfabet object before the status transitions.
>
>   - The system then checks whether the current Jira status has a transition to itself. If the status transition is possible, the transition will be performed and the issues fields that may be updated in the current Jira issue will be updated.
>
> - In order to update an issue in Jira based on an Alfabet object, the system checks whether the Alfabet status is the same as the Jira status. Please note the following:
>
>   - If the Alfabet status is not the same as the Jira status, the system will assess whether a status transition is possible between the two status values.
>
>   - If the status transition is not possible, an error message will be displayed and the update of the issue will be skipped.
>
>   - If the status transition is possible, the update will be performed for the issues fields that may be updated in the current status of the Alfabet object before the status transitions.
>
>   - The system then checks whether the current Jira status has a transition to itself. If the status transition is possible, the transition will be performed and the issues fields that may be updated in the current Jira issue will be updated.

6) Click the **Save Temporary Entry** button. A message will be displayed if the export entry has been saved.

7) Go to the **Map Secondary Alfabet Report Columns to Jira Issues Fields** tab, specify the following to specify the mapping for multi-valued attributes that require more than one value (arrays):

- **Enter Export Entry Name**: Displays the name of the export entry selected in the tab or the name entered for a new entry.

- **Secondary Alfabet Reports Master Set**: Specify the secondary configured reports to capture multi-value attributes. All reports assigned to the report category defined for the selected data connection in the XML object **JIRAIntegrationConfig** will be displayed in the drop-down list.

  > Alternatively, to edit the existing definition based on one or more secondary reports, select the secondary reports in the **Select Secondary Report to Define Mapping** field that you want to edit. The assistant fields will be updated with the existing definition. Click the **Edit** button to edit the fields described below.

- **Secondary Report to Defined Mapping**: If necessary, select additional secondary reports to execute to find multi-valued attributes.

- **Display Option for Alfabet Report Properties**: Select one of the following to determine the sorting of the data displayed in the dataset:

  - **Sort Report Columns as Defined in Report**: The column names in the configured report are displayed instead of the object class property names. The sort order is identical to the order of columns in the configured report.

  - **Sort Report Columns with Lexicographic Sorting**: The column names in the configured report are displayed instead of the object class property names. The sort order is alphanumeric.

  - **Sort Property Columns as Defined in Report**: The object class properties are displayed as <ObjectClassName>.<PropertyName>. The sort order is identical to the order of columns in the configured report.

  - **Sort Property Columns with Lexicographic Sorting**: The object class properties are displayed as <ObjectClassName>.<PropertyName>. The sort order is alphanumeric.

8) Click the **Save Temporary Entry** button. The **Map Secondary Alfabet Report Columns to JIRA Issue Fields** section displays all Jira issue fields found via the secondary reports.

- **Jira Issue Field Name**: Displays the name of the Jira issue field. The dataset indicates which Jira fields are mandatory to create an issue as well as to update an issue. All fields that are mandatory are displayed in red.

- **Jira Issue Field Type**: Displays the type of the Jira issue field.

- ***C**: Displays an M for mandatory the export if the **Report Column Name** field must be mapped in order to create an issue in Jira.

- ***U**: Displays an M for mandatory the export if the **Report Column Name** field must be mapped in order to update an issue in Jira.

- **Report Column Name**: Select the Alfabet property that the Jira issue field shall be mapped to. Some rows can only be specified after the secondary report has been executed.

9) Click the **Save Temporary Entry** button.

10) Click **OK** to close the editor. The ADIF export scheme can be executed via a batch job and the relevant Alfabet objects such as demands, features, etc. will be exported as issues to Jira.

Configuring the Event Templates to Trigger the ADIF Export Schemes for Synchronization

Once the mapping has been completed, event templates must be configured to trigger the ADIF export schemes associated with the relevant configured reports. The configured report associated with the ADIF export scheme should return a set of objects that can be mapped to Jira Issues. The set of objects returned by the event template must be a subset of the objects returned by the primary report of the ADIF job referenced by the event template. Only the returned objects that are common to both the event template and the primary report will be exported to Jira. The execution of the event template may be parameterized as described below in order to limit the set of returned objects.

The event templates can be assigned to a semantic Jira connection in the **Event Templates for Synchroniza-tion** field in the **Jira Connection for Project-Based Integration** and **Jira Connection for Architecture-Based Integration** editors. The event template will be triggered when the users execute the **Synchronize with Project Data** functionality in the relevant object profiles.

Please note the following regarding the configuration of event templates for the ADIF export schemes configured for Jira integration:

- A parameter is not required by the reports specified for the ADIF export scheme. If a parameter is specified, it will be ignored.

- The following attributes must be defined for the event template:

  - **ADIF Scheme**: Select the ADIF export scheme that the event template shall trigger.

  - **Values for Variables via Query**: Specify a query to limit the objects returned by the report. For example, if the user is synchronizing an application to a Jira project version, only the features relevant to the application that is being synchronized will be exported to Jira. Please note that the REFSTR must have the name ObjectRef as shown in the example query:

        SELECT NULL; REFSTRA AS 'ObjectRef'

        FROM FEATURE

        WHERE OBJECT = @BASE

ⓘ For the import from Jira, the event template should return a set of objects that are associated with the relevant Jira structure objects.

For information about the configuration of event templates, see the chapter *Configuring Events* reference manual *Configuring Alfabet with Alfabet Expand*. A user with an administrative user profile may review the success of the triggered events in the *Event Administration Functionality*. For more information, see the chapter *Managing Events* in the reference manual *User and Solution Administration*.

Configuring Semantic Connections to Link and Synchronize Jira Projects with Alfabet Objects

The **Jira Connection** view in the **Integration Solutions Configuration** functionality in the Alfabet user inter-face allows you to create one or more semantic definitions for all relevant Jira® connections that have been con-figured in the XML object **JIRAIntegrationConfig** in order to export Alfabet data to Jira. A Jira connection defi-nition should be created for each data connection configured in the XML object **JIRAIntegrationConfig**. You can configure multiple semantic data connections for each data connection configured in the XML element **Data-Connection** in the XML object **JIRAIntegrationConfig**.

A semantic Jira connection allows you to map the Alfabet object classes to the Jira project structure and to specify the stereotypes and possible a configured report that shall return the set of Alfabet objects that may be integrated with Jira. For each Jira connection you create, you must specify the integration pattern to map objects in the Alfabet object hierarchy to the Jira project, project version, or project component. You can choose an architecture-based integration pattern or a project-based integration pattern for the mapping.

Whether your enterprise chooses the architecture-based or project-based integration will depend on your methodology of using Jira. If your company associates demands with products in Alfabet, the ICT object integration pattern may be more relevant, but if demands are associated with projects, a project hierarchy integration pattern may be more suitable.

The following are examples of potential use cases for the integration with Jira. The first example is based on a project-based integration pattern and the second is based on an architecture-based integration pattern:

- The IT department of a banking enterprise documents bugs and change requests for their trading software in Jira and captures their operational plans in Alfabet. In this case, for example, the Jira issues could be imported to Alfabet as features that are planned for applications that are owned and fiscally managed via ICT objects. The following mapping schema provides an example of how Jira projects could be mapped to a project structure in Alfabet:

Alfabet Class	Jira Class
ICT Object	`Project`
Application	`ProjectVersion`
Local Component	`ProjectComponent`
Feature	`Issue`

- A banking enterprise captures demands in order to plan the modification of the bank's trading capabilities. The demands are assigned to projects in Alfabet. The Project stereotype "Program" in Alfabet could be mapped to `Projects` in the Jira instance. The demands will be exported as issues to Jira in order to implement the projects and operationalize the needed changes. The subordinate project stereotype Project could be mapped to the Jira `ProjectVersion` and the project stereotype Project Step would be mapped to the Jira `ProjectComponent`. The following mapping schema provides an example of how Jira projects could be mapped to an ICT object hierarchy in Alfabet:

Alfabet Class	Jira Class
Project Stereotype 1	`Project`

Project Stereotype 2	`ProjectVersion`
Project Stereotype 3	`ProjectComponent`
Demand	`Issue`

The following integration patterns may be specified for a project-based approach:

Integration Type	Jira Project	Jira Project Version	Jira Project Component
Jira Projects, Project Versions, and Project Components	**Project** Stereotype 1	**Project** Stereotype 2	**Project** Stereotype 3
Jira Projects and Project Versions	**Project** Stereotype 1	**Project** Stereotype 2	
Jira Project and Project Components	**Project** Stereotype 1		**Project** Stereotype 3
Jira Projects	**Project** Stereotype 1		
Jira Project Versions		**Project** Stereotype 2	
Jira Project Versions and Project Components		**Project** Stereotype 2	**Project** Stereotype 3
Jira Project Components			**Project** Stereotype 3

The following integration patterns may be specified for an architecture-based approach:

Integration Type	Jira Project	Jira Project Version	Jira Project Component
Jira Projects, Project Versions, and Project Components	ICT Object (or stereotype)	Application (or stereotype)	Local Component (or stereotype)

Integration Type	Jira Project	Jira Project Version	Jira Project Component
	ICT Object (or stereotype)	Component (or stereotype)	Local Component (or stereotype)
	ICT Object (or stereotype)	Standard Platform (or stereotype)	Platform Element (or stereotype)
Jira Projects and Project Versions	ICT Object (or stereotype)	Application (or stereotype)	
	ICT Object (or stereotype)	Component (or stereotype)	
	ICT Object (or stereotype)	Standard Platform (or stereotype)	
Jira Projects	ICT Object (or stereotype)		
Project Versions		Application (or stereotype)	
		Component (or stereotype)	
		Standard Platform (or stereotype)	
Project Versions and Project Components	ICT Object (or stereotype)	Application (or stereotype)	Local Component (or stereotype)
	ICT Object (or stereotype)	Component (or stereotype)	Local Component (or stereotype)
	ICT Object (or stereotype)	Standard Platform (or stereotype)	Platform Element (or stereotype)
Jira Projects, Project Versions, and Project Components		Application (or stereotype)	Local Component (or stereotype)

Integration Type	Jira Project	Jira Project Version	Jira Project Compo- nent
		Component (or stereo- type)	Local Component (or stereotype)
		Standard Platform (or stereotype)	Platform Element (or stereotype)
Project Components		Component (or stereo- type)	Local Component (or stereotype)
		Standard Platform (or stereotype)	Platform Element (or stereotype)

You may specify any level in the Jira project structure to begin the integration with. For example, you could choose to map Alfabet applications to Jira project versions and the application's local components to Jira project components. Depending on the integration pattern specified for the relevant data connection, users can navigate to the object profile of the relevant ICT object, project, application, component, standard platform, etc. in the Alfabet user interface and link the object to the relevant Jira project/project version/project component via the **Link to Jira Project** option in the **Action** menu. The **Link to Jira Project** option will only be available for those objects that are permissible for Jira integration via the stereotype definition of configured report.

Once the Alfabet object has been linked with the Jira project in the user interface, users can regularly synchro- nize the object with the Jira project structure via the **Synchronize with Jira Project Structure** option. The project version and project component will only be synchronized if Alfabet objects exist for the relevant object classes/object class stereotypes. Furthermore, the Alfabet objects will be synchronized with the project's existing project version and component only if the properties specified for the mapping have the same value. For exam- ple, if the integration pattern specifies that applications are to be mapped to Jira project versions based on the Alfabet property `Name`, then all applications with the same name that are owned by the selected ICT object will be mapped to the same Jira project version.

The following information is available:

- [Creating a Jira Connection for Project-Based Integration](#)

- [Creating a Jira Connection for Architecture-Based Integration](#)

Creating a Jira Connection for Project-Based Integration

The **Jira Connection** view in the **Integration Solutions Configuration** functionality allows you to define Jira connections that map project stereotypes in Alfabet that are relevant for Jira integration. You can specify an inte- gration mapping of projects in the project hierarchy configured for your Alfabet solution to the Jira project, pro- ject version, and project component. For example, the project stereotype Agile Release Train could be mapped to Jira projects and the project stereotype Program Increment could be mapped to the Jira project version.

You can configure multiple semantic Jira connections for each data connection configured in the XML element **DataConnection** in the XML object **JiraIntegrationConfig**. A semantic Jira connection should be configured for each level in the Jira project structure that the integration can begin with.

The following integration patterns may be specified for a project-based approach:

Integration Type	Jira Project	Jira Project Version	Jira Project Component
Jira Projects, Project Versions, and Project Components	**Project** Stereotype 1	**Project** Stereotype 2	**Project** Stereotype 3
Jira Projects and Project Versions	**Project** Stereotype 1	**Project** Stereotype 2	
Jira Project and Project Components	**Project** Stereotype 1		**Project** Stereotype 3
Jira Projects	**Project** Stereotype 1		
Jira Project Versions		**Project** Stereotype 2	
Jira Project Versions and Project Components		**Project** Stereotype 2	**Project** Stereotype 3
Jira Project Components			**Project** Stereotype 3

To configure a Jira connection for a project-based integration:

1) Go to the **Integration Solutions Configuration** functionality and click the **Jira Connection** node in the **Integration Solutions Configuration** explorer.

2) In the view, click **New** > **Create Jira for Project-Based Integration**.

3) In the **Jira Connection for Project-Based Integrations** editor, define the following fields as needed.

   Basic Data tab:

   - **ID**: Alfabet assigns a unique identification number to each Jira data connection. This number cannot be edited.

   - **Name**: Enter a unique name for the Jira connection. The name should help the user synchronizing the Jira instance that will be targeted by the connection.

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

Authorized Access tab:

- **Authorized User**: Click the **Search** icon to assign an authorized user to the selected Jira 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:

> Please note the following about the fields in the **Connection** tab:
>
> - The integration pattern that you select in the **Integration Type** field will determine which fields must be defined in the **Connection** tab. Please note that if you specify the integration pattern that starts with either a project version or project component, the **Property to Save Jira Project ID**, **Property to Save Jira Project Name**, and **Property to Save Jira Project Key** must be specified for the ascendant Jira project. The **Property to Save Jira Project ID**, **Property to Save Jira Project Name**, and **Property to Save Jira Project Key** fields are filled with the custom properties as well as the standard properties of the type `String` specified in the XML element `AlfabetClassMappingStandardPropertySettings` in the XML object **JiraIntegrationConfig**.
>
> - The **Alfabet Class Aligned with Jira Project** field will always display the object class `Project`. You should define the **Stereotype Filter** field in order to determine which project stereotype may be integrated with Jira. All available project stereotypes availed in the **Stereotype Filter** field will be selected if no project stereotype is selected. Please note that the project stereotypes available in the **Stereotype Filter** fields will be determined by the configuration of project stereotype hierarchies in the XML object **ProjectManager**. For more information, see the section *Creating Project Stereotypes for the Project Hierarchy* in the reference manual *Configuring Alfabet with Alfabet Expand*.
>
> - The definition of the **Stereotype Filter** field and the **Object Filter Report** field determine the Alfabet project's that are allowed to integrate with Jira projects. The **Link to Jira Project** editor option will be available in the **Action** menu in the object profiles of the projects found via the definition of the **Stereotype Filter** field or the **Object Filter Report** field. For more information about configuring the reports that may be selected in the **Object Filter Report** field, see the section Configuring Object Filter Reports.

- **Data Connection**: Select the relevant data connection configured in the XML element **DataConnection** in the XML object **JiraIntegrationConfig**. This data connection will be established when the user selects the integration pattern defined below via the **Action** > **Link to Jira Project** option in the object profile for the relevant project stereotype.

- **Project Filter**: Select one of the Jira project filters defined in the XML object **JiraIntegrationConfig**. The filters limit the number of Jira projects that users can select for the integration in the editor that opens when the **Link to Jira Project** option is executed for a project in Alfabet. For example, a filter could specify that only Jira projects with a project type equal to "Software" or "Business" are returned.

- **Project Filter Description**: Displays information about the filter selected in the **Select Project Filter** field.

- **Integration Type**: Select the level in the Jira project structure to begin the integration with. The project stereotypes displayed in the **Integration Pattern** field will be determined based on the **Integration Type** field. You can select the specific Alfabet project stereotype to map to the relevant level in the Jira project structure in the respective **Stereotype Filter** field.

- **Integration Pattern**: Select the architecture pattern to map Alfabet projects to the Jira project structure selected in the **Integration Type** field.

- **Alfabet Class Aligned with Jira Project**: Displays the object class `Project`.

- **Stereotype Filter**: Select an Alfabet project stereotype to map to Jira projects. All available project stereotypes will be selected if no project stereotype is selected.

- **Object Filter Report**: Select the Alfabet report that returns the relevant objects that may be integrated with Jira objects. This is an optional report that allows you to limit the projects available to be selected for Jira integration.

- **Property to Save Jira Project ID**: Select the Alfabet class property to be used to capture the `ID` property for Jira projects.

- **Property to Save Jira Project Name**: Select the Alfabet class property to be used to capture the `Name` property for Jira projects.

- **Property to Save Jira Project Key**: Select the Alfabet class property to be used to capture the `Key` property for Jira projects.

- **Alfabet Class Aligned with Jira Project Version**: Displays the object class `Project`.

- **Stereotype Filter**: Select an Alfabet project stereotype to map to Jira project versions. All available project stereotypes will be selected if no project stereotype is selected.

- **Object Filter Report**: Select the Alfabet report that returns the relevant objects that may be integrated with Jira project versions. This is an optional report that allows you to limit the project versions available to be selected for Jira integration.

- **Property to Save Jira Project ID**: Select the Alfabet class property to be used to capture the `ID` property for Jira project versions.

- **Property to Save Jira Project Name**: Select the Alfabet class property to be used to capture the `Name` property for Jira project versions.

- **Property to Create Jira Version on Synchronization**: Select the Alfabet class property to be used to capture the `Name` property when creating new Jira project versions.

- **Alfabet Class Aligned with Jira Project Component**: Displays the object class `Project`.

- **Stereotype Filter**: Select an Alfabet project stereotype to map to Jira project components. All available project stereotypes will be selected if no project stereotype is selected.

- **Object Filter Report**: Select the Alfabet report that returns the relevant objects that may be integrated with Jira project components. This is an optional report that allows you to limit the project components available to be selected for Jira integration.

- **Property to Save Jira Project ID**: Select the Alfabet class property to be used to capture the `ID` property for Jira project components.

- **Property to Save Jira Project Name**: Select the Alfabet class property to be used to capture the `Name` property for Jira project components.

- **Event Templates for the Synchronization**: Select the Alfabet event templates that shall call the relevant ADIF import/export schemes that are to be triggered when synchronizing an Alfabet object with the linked Jira object's data. For details about the configuration of event templates necessary to export Alfabet data to Jira, see the section Configuring the Event Templates to Trigger the ADIF Export Schemes for Synchronization.

- **Event Template Sequence**: Reorder the sequence that the selected Alfabet event templates are triggered. To do so, click an event template in the window and click the

4) Repeat this for all necessary configured **DataConnection** elements in the XML object **JiraIntegrationConfig**.

Creating a Jira Connection for Architecture-Based Integration

The **Jira Connection** view in the **Integration Solutions Configuration** functionality allows you to define Jira connections that specify the mapping of the object classes or their object class stereotypes in Alfabet that are relevant for Jira integration. You can specify an integration mapping based on predefined architecture object hierarchies in Alfabet to map to the Jira project, project version, and project component. A semantic definition should be configured for each level in the Jira project structure that the integration can begin with.

The following integration patterns may be specified for an architecture-based approach:

Integration Type	Jira Project	Jira Project Version	Jira Project Component
Jira Projects, Project Versions, and Project Components	ICT Object (or stereotype)	Application (or stereotype)	Local Component (or stereotype)
	ICT Object (or stereotype)	Component (or stereotype)	Local Component (or stereotype)
	ICT Object (or stereotype)	Standard Platform (or stereotype)	Platform Element (or stereotype)
Jira Projects and Project Versions	ICT Object (or stereotype)	Application (or stereotype)	
	ICT Object (or stereotype)	Component (or stereotype)	
	ICT Object (or stereotype)	Standard Platform (or stereotype)	
Jira Projects	ICT Object (or stereotype)		

Integration Type	Jira Project	Jira Project Version	Jira Project Component
Project Versions		Application (or stereotype)	
		Component (or stereotype)	
		Standard Platform (or stereotype)	
Project Versions and Project Components	ICT Object (or stereotype)	Application (or stereotype)	Local Component (or stereotype)
	ICT Object (or stereotype)	Component (or stereotype)	Local Component (or stereotype)
	ICT Object (or stereotype)	Standard Platform (or stereotype)	Platform Element (or stereotype)
Jira Projects, Project Versions, and Project Components		Application (or stereotype)	Local Component (or stereotype)
		Component (or stereotype)	Local Component (or stereotype)
		Standard Platform (or stereotype)	Platform Element (or stereotype)
Project Components		Component (or stereotype)	Local Component (or stereotype)
		Standard Platform (or stereotype)	Platform Element (or stereotype)

You can configure multiple semantic data connections for each data connection configured in the XML element **DataConnection** in the XML object **JiraIntegrationConfig**.

To configure a Jira database connection for an architecture-based integration:

1) Go to the **Integration Solutions Configuration** functionality and click the **Jira Connection** node in the **Integration Solutions Configuration** explorer.

2) In the view, click **New** > **Create Jira for Architecture-Based Integration**.

3) In the **Jira Connection for Architecture-Based Integrations** editor, define the following fields as needed.

Basic Data tab:

- **ID**: Alfabet assigns a unique identification number to each Jira data connection. This number cannot be edited.

- **Name**: Enter a unique name for the Jira connection. The name should help the user synchronizing the Jira instance that will be targeted by the connection.

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

Authorized Access tab:

- **Authorized User**: Click the **Search** icon to assign an authorized user to the selected Jira 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:

> Please note the following about the fields in the **Connection** tab:
>
> - The integration pattern that you select in the **Integration Type** field will determine which fields must be defined in the **Connection** tab. Please note that if you specify the integration pattern that starts with either a project version or project component, the **Property to Save Jira Project ID**, **Property to Save Jira Project Name**, and **Property to Save Jira Project Key** must be specified for the ascendant Jira project. The **Property to Save Jira Project ID**, **Property to Save Jira Project Name**, and **Property to Save Jira Project Key** fields are filled with the custom properties as well as the standard properties of the type `String` specified in the XML element `AlfabetClassMappingStandardPropertySettings` in the XML object **JiraIntegrationConfig**.
>
> - You may define the **Stereotype Filter** field in order to determine which object class stereotype may be integrated with Jira. All available object class stereotypes available in the **Stereotype Filter** field will be selected if no object class stereotype is selected.
>
> - The definition of the **Stereotype Filter** field and the **Object Filter Report** field determine the Alfabet object's that are allowed to integrate with Jira projects. The **Link to Jira Project** editor option will be available in the **Action** menu in the object profiles of the objects found via the definition of the **Stereotype Filter** field or the **Object Filter Report** field. For more information about configuring the reports that may be selected in the **Object Filter Report** field, see the section Configuring Object Filter Reports.
>
> - **Alfabet Class Aligned with Jira Project**: Specify the Alfabet class in the specified integration pattern that is matched to Jira projects. Whether this field can be defined or not will depend the selection in the **Integration Type** field.

- **Data Connection**: Select the relevant data connection configured in the XML element **DataConnection** in the XML object **JiraIntegrationConfig**. This data connection will be

established when the user selects the integration pattern defined below via the **Action** > **Link to Jira Project** option in the object profile for the relevant object.

- **Project Filter**: Select one of the Jira project filters defined in the XML object **JiraIntegrationConfig**. The filters limit the number of Jira projects that users can select for the integration in the editor that opens when the **Link to Jira Project** option is executed for an object. For example, a filter could specify that only Jira projects with a project type equal to "Software" or "Business" are returned.

- **Project Filter Description**: Displays information about the filter selected in the **Select Project Filter** field.

- **Integration Type**: Select the level in the Jira project structure to begin the integration with. The object classes displayed in the **Integration Pattern** field will be determined based on the **Integration Type** field. You can select the specific object stereotype to map to the relevant level in the Jira project structure in the respective **Stereotype Filter** field.

- **Integration Pattern**: Select the architecture pattern to map Alfabet objects to the Jira project structure selected in the **Integration Type** field.

- **Alfabet Class Aligned with Jira Project**: Displays the Alfabet class in the specified integration pattern that is mapped to Jira projects. Whether this field can be defined or not will depend on the selection in the **Integration Type** field.

- **Stereotype Filter**: Select an object stereotype to map to Jira projects. All available object class stereotypes will be selected if no object class stereotype is selected.

- **Object Filter Report**: Select the Alfabet report that returns the relevant objects that may be integrated with Jira projects. This is an optional report that allows you to limit the objects available to be selected for Jira integration.

- **Property to Save Jira Project ID**: Select the Alfabet class property to be used to capture the `ID` property for Jira projects.

- **Property to Save Jira Project Name**: Select the Alfabet class property to be used to capture the `Name` property for Jira projects.

- **Property to Save Jira Project Key**: Select the Alfabet class property to be used to capture the `Key` property for Jira projects.

- **Alfabet Class Aligned with Jira Project Version**: Displays the Alfabet class in the specified integration pattern that is mapped to Jira project versions. Whether this field can be defined or not will depend on the selection in the **Integration Type** field.

- **Stereotype Filter**: Select an object class stereotype to map to Jira project versions. All available object class stereotypes will be selected if no object class stereotype is selected.

- **Object Filter Report**: Select the Alfabet report that returns the relevant objects that may be integrated with Jira project versions. This is an optional report that allows you to limit the objects available to be selected for Jira integration.

- **Property to Save Jira Project ID**: Select the Alfabet class property to be used to capture the `ID` property for Jira project versions.

- **Property to Save Jira Project Name**: Select the Alfabet class property to be used to capture the `Name` property for Jira project versions.

- **Property to Create Jira Version on Synchronization**: Select the Alfabet class property to be used to capture the `Name` property when creating new Jira project versions.

- **Alfabet Class Aligned with Jira Project Component**: Displays the Alfabet class in the specified integration pattern that is mapped to Jira project components. Whether this field can be defined or not will depend on the selection in the **Integration Type** field.

- **Stereotype Filter**: Select an object class stereotype to map to Jira project components. All available object class stereotypes will be selected if no object class stereotype is selected.

- **Object Filter Report**: Select the Alfabet report that returns the relevant objects that may be integrated with Jira project components. This is an optional report that allows you to limit the objects available to be selected for Jira integration.

- **Property to Save Jira Project ID**: Select the Alfabet class property to be used to capture the `ID` property for Jira project components.

- **Property to Save Jira Project Name**: Select the Alfabet class property to be used to capture the `Name` property for Jira project components.

- **Event Templates for the Synchronization**: Select the Alfabet event templates that shall call the relevant ADIF import/export schemes that are to be triggered when synchronizing an Alfabet object with the linked Jira object's data. For details about the configuration of event templates necessary to export Alfabet data to Jira, see the section Configuring the Event Templates to Trigger the ADIF Export Schemes for Synchronization.

- **Event Template Sequence**: Reorder the sequence that the selected Alfabet event templates are triggered. To do so, click an event template in the window and click the

4) Repeat this for all necessary configured **Connection** elements in the XML object **JiraIntegrationConfig**.

Linking to and Synchronizing the Jira Project

Once an integration pattern has been configured in the **Integration Solutions Configuration** functionality, you can link the relevant Alfabet object to a Jira project. Depending on the configuration of the integration pattern for the semantic Jira connection, the object that may be linked may be a project stereotype, ICT object, application, component, local component, standard platform, or standard platform element. The **Link to Jira Project** option will be available in the object profile for all objects that are specified via the stereotype definition or configured report definition of a semantic Jira connection.

⚠️ The Alfabet Server must be running and able to connect to the Internet.

To link to a Jira project:

1) In the Alfabet user interface, navigate to the object profile of the relevant object that has been specified via a semantic Jira connection.

2) In the toolbar, click **Action** > **Link to Jira Project**. Please note that if you are linking an Alfabet object that shall be integrated with a Jira project version, you will see **Action** > **Link to Jira Project Version** and if you are linking an Alfabet object that shall be integrated with a Jira project component, you will see **Action** > **Link to Jira Project Component**. The **Link to Jira Project** editor opens.

3) In the **Integration Connection** field, select the relevant connection to use to connect to Jira. All Jira projects meeting the conditions specified for the project filters specified for the selected integration connection will be displayed. The Jira projects can be further limited by specifying criteria in the **Search Pattern** field.

4) In the dataset, select the relevant Jira project/Jira project version/Jira project component that you want to link the selected object to and click **OK**. The **Link to Jira Project** action creates a new record in the database table for the `IntegrationConnectionUsage` class. The table maps the relationship between the selected integration connection, the ICT object/project in Alfabet, and the identifier of the Jira project.

5) To synchronize the Jira project structure (the relevant Jira project version and Jira project component), click **Action** > **Synchronize with Jira Project Structure**. The synchronization creates the relevant project versions and project components based on the integration pattern specified for the data connection you are using. For example, if the Alfabet object to start the integration with is an application and the application has been linked to a Jira project version, the application's local components will be synchronized with the Jira project components of the Jira project version.

The project version and project component will only be synchronized if Alfabet objects exist for the relevant object classes/object class stereotypes. For example: the integration pattern specifies the mapping of the object class Application to Jira project versions and the object class stereotype Technical Component of the class Local Component to the Jira project component. If the application GL Trade Products is linked to a Jira project version, then upon synchronization all local components assigned to the application would be created as Jira project components for the Jira project version. However, if no technical components have been defined for any of the applications, then no Jira project components will be created for the Alfabet local component.

Furthermore, the Alfabet objects will be synchronized with a project's existing project version and component only if the properties specified for the mapping have the same value. For example, if the integration pattern specifies that ICT objects are to be mapped to Jira projects based on the Alfabet property `Name`, then all applications with the same name that are owned by the selected Alfabet ICT object will be mapped to the same Jira project version.

6) To trigger the event templates specified for synchronization for the relevant Jira integration connection, click **Action** > **Synchronize with Jira Project Data**. The event templates must be configured to trigger the ADIF export schemes associated with the relevant configured reports. The configured report associated with the ADIF export scheme should return a set of objects that can be mapped to Jira Issues.

Chapter 13: Configuring Interoperability with Microsoft Project

Interoperability with Microsoft® Project allows your enterprise to synchronize strategic project portfolio planning with operational project management and evaluate all of the projects in the enterprise in the context of corporate strategy and strategic investment. The capability allows the enterprise to export projects planned in Alfabet to Microsoft Project as well as to import projects managed in Microsoft Project to Alfabet.

> The configuration of the project management capability including project stereotypes, release statuses, wizards, etc. should be completed before configuring integration of Alfabet projects with Microsoft projects. For more information, see the section *Configuring the Project Management Capability* in the reference manual *Configuring Alfabet with Alfabet Expand*.

The following is required to import MS projects to Alfabet, synchronize imported Alfabet projects with their updated projects in MS Project, or export Alfabet projects to MS Project:

- Specify data connections and import patterns in the XML object **MicrosoftProjectPlanConfig**.

- Ensure that a wizard has been created and is assigned to the class setting for the class Project. This is mandatory in order to import projects via MS Project. An additional step will be prepended to the first wizard step that allows the import to be triggered in order to create the new project in Alfabet.

- Ensure that release status definitions have been created for the class `MicrosoftProjectPlan_Mappings` in the XML object **ReleaseStatusDefs**.

- Ensure that a view is available to the relevant user profiles that provides the functionality to import MS projects, synchronize Alfabet projects with updated MS projects, and export Alfabet projects to Microsoft Server. For more information about configuring user profiles, see the section *Configuring User Profiles for the User Community* in the reference manual *Configuring Alfabet with Alfabet Expand*. These views include the following:

  - **Capture Projects** functionalities (`PRJ_CaptureProjects` and `PRJ_CaptureProjects_Ex`)

  - **Projects** page views for project groups and buckets (`PRJG_Projects` and `BKT_Projects`)

  - **Project** object profile (`PRJ_ObjectView`) or a project stereotype object profile

- Specify the MS project methodology to use for the import/export/synchronization of MS projects via the **MS Project Methodology** node in the **Integration Solutions Configuration** functionality in the user interface. For more information, see the section *Specifying MS Project Methodologies for Interoperability with Microsoft Project* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.

> ⚠ Ensure that the file extensions.mpp and.mpt are not listed in the blacklist defined in the XML object **FileExtensionLists**. If a whitelist is specified the XML object **FileExtensionLists**, then the file extensions.mpp and.mpt must be included as permissible file extensions. For more information, see the section *Specifying the Permissible File Extensions for Uploading/Downloading Files* in the reference manual *Configuring Alfabet with Alfabet Expand*.

The following information is available:

- [Configuring the Connections for Interoperability with Microsoft Project](#)

- [Specifying a Wizard for Interoperability with Microsoft Project](#)

- [Specifying Release Status for Interoperability with Microsoft Project](#)

Configuring the Connections for Interoperability with Microsoft Project

Specify data connections and import patterns in the XML object **MicrosoftProjectPlanConfig**. These are required to define the semantic connections for integration with Microsoft Project.

The XML object **MicrosoftProjectPlanConfig** allows a solution designer to specify data connections and the data mapping required for the import and export of Alfabet projects to/from Microsoft Project. Furthermore, the options for import matching rules require for the import of Microsoft projects/project tasks as skill requests to Alfabet can be defined. Please note the following:

- You can specify that a connection is created to one or more Microsoft Project Server instances in order to import data from the server instance. Alternatively, data can be imported to Alfabet via one or more MPP files. Please note that the connection to the Microsoft Project Server only supports the import of Microsoft projects and project tasks to Alfabet. Export from Alfabet to the Microsoft Project Server is not supported.

- You must specify the mapping between Alfabet property types and Microsoft project data types to import/export data from either a Microsoft Project Server instance or an MPP file.

- You must specify import matching rules that describe how the resource data captured in Microsoft Project shall be used to find the skill, organization, and person referenced by a skill request created in the context of importing a Microsoft project task. You must define at least one property or matching pattern for each class. The following is possible to find the relevant skill, organization, and person in the Alfabet database based on the resource data associated with the project task imported to Alfabet:

  - Specify object class properties for the classes `Skill` (mandatory), `Person` (optional), and `OrgaUnit` (optional) that may be used to search for skills, persons, and organizations in the Alfabet database.

     The `Name` property is defined for the class `Skill`. A project task is imported as a skill request to Alfabet. The project task has a resource Java Programming. Upon import, the system will search for a skill with the name JAVA Programming. If an existing skill is found, the new skill request will reference the skill Java Programming.

  - Specify matching patterns that consist of multiple properties and a delimiter for the classes `Skill` (mandatory), `Person` (optional), and `OrgaUnit` (optional) that may be used to search for skills, persons, and organizations in the Alfabet database.

     A matching pattern consisting of the properties `Name` and `ID` defined for the class `Skill` and the underscore (_) delimiter. A project task is imported as a skill request to Alfabet. The project task has a resource Java Programming_123. Upon import, the system will search for a skill with the name Java Programming and the ID 123. If an existing skill is found with a matching name and ID, the new skill request will reference the skill Java Programming. Please note however, if the resource was named Java Programming/123, no skill would be found because the slash delimiter is used in the resource name and not an underscore.

     Please note the following regarding the implementation of import matching rules:

    - One or more properties and matching patterns can be selected by the user defining the methodology in the **MS Project Methodology** editor in the **Integration Solutions Configuration** functionality:

    - Criteria such as `Contains`, `Starts With`, etc. can be defined for the selected properties and matching patterns by the user defining the methodology in the **MS**

Project Methodology editor in the **Integration Solutions Configuration** functionality:

- The first relevant skill, organization, or person found to match the criteria in the Alfabet database will be used.

- Every skill request that is imported to Alfabet requires a reference to a skill. This is mandatory. The reference to a person or organization is optional for a skill request. A default skill can be selected by the user importing the Microsoft project if no skills are found by import matching rules.

⚠ Please note that if changes are made to the XML object **MicrosoftProjectPlanConfig** after the MS project methodologies have been created in the **MS Project Methodology** view in the **Integration Solutions Configuration** functionality, you should review the import and export mapping rules of the MS project methodologies and make adjustments as needed. Changes made to the XML object **MicrosoftProjectPlanConfig** will not be automatically updated to existing MS project methodologies. For more information about configuring MS project methodologies, see the section *Specifying MS Project Methodologies for Interoperability with Microsoft Project* in the reference manual *Configuring Evaluation and Reference Data in Alfabet*.

To define the XML object **MicrosoftProjectPlanConfig**:

1) In Alfabet Expand, go to the **Presentation** tab and expand the explorer nodes **XML Objects** > **IntegrationSolutions**.

2) Right-click **MicrosoftProjectPlanConfig** and select **Edit XML...**. The XML object **MicrosoftProjectPlanConfig** opens.

ⓘ 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 **MicrosoftProjectPlanConfig**. 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.

3) **If projects are to be imported from Microsoft Project Server** (optional): Add an XML element `DataConnections` to the root XML element `MicrosoftProjectPlanConfig`. For each Microsoft® Project Server instance, add a child XML element `DataConnections` and specify the following XML attributes:

- `Name`: The name of the connection to the Microsoft® Project Server. A unique name must be specified for the connection. The name will be displayed in the **MS Project Server Connection** field in the Project wizard that opens when the import via MS Project Server option is triggered.

- `IsActive`: Enter "`true`" to activate interoperability with Microsoft® Project Server. One Microsoft® Project Server may be active at any given time. If multiple connections are set to `IsActive = true`, then the first active connection will be used.

- `ServiceURL`: Specify the URL to the Microsoft® Project Server instance.

- `ServiceUserName`: Specify the user name used to access the Microsoft® Project Server instance.

- `ServiceUserDomain`: 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.

- `ServicePassword`: Specify the password used to access the Microsoft® Project Server instance.

- `DBDataSource`: Specify the database data source to access for the Microsoft® Project Server instance.

- `DBInitialCatalog`: Enter the database catalog name to access for the Microsoft® Project Server instance.

- `DBSchema`: Enter the database schema name to access for the Microsoft® Project Server instance.

- `DBUserName`: Enter the database user name to access for the Microsoft® Project Server instance.

- `DBPassword`: Enter the database user password to access for the Microsoft® Project Server instance.

4) Add an XML element `DataTypeMappings` to the root XML element `MicrosoftProjectPlanConfig`. Add the XML attribute `UnknownMPPTypeAsString` and specify "`true`" if data types not specified in an XML element `DataType` shall be imported to Alfabet from Microsoft® Project with the data type `String`. Enter "`false`" if unknown data types shall not be imported to Alfabet.

5) Create an XML element `DataType` for each Microsoft Project data type to be mapped to an Alfabet data type when importing/exporting data. Specify the following XML attributes for each XML element `DataType`:

- `MPPType`: Enter the Microsoft Project data type to match to the Alfabet data type specified in the XML attribute `ADIFType`

- `ADIFType`: Enter the Alfabet data type to match to the Microsoft roject data type specified in the XML attribute `MPPType`.

> (i) Please note that an ADIF configuration is not required to import or export data from Microsoft Project.

6) Add an XML element `ResourcMatchOptions` to the root XML element `MicrosoftProjectPlanConfig` in order to define the import matching rules that can potentially be used to match the resources from Microsoft® Project with skills, organizations or persons in Alfabet when a skill request is created in Alfabet based on a project task in Microsoft Project. You must define at least one property or matching pattern for each class. The specified properties and matching patterns will be available for selection in the **Import Matching Rules** section in the **MS Project Methodology** editor available in the Alfabet user interface and can be assigned to a methodology as needed. Define the following child XML elements for the XML element `ResourcMatchOptions`.

- `Class`: Create an XML element `Class` for each of the following classes: `Skill`, `Person`, and `OrgaUnit`. Define the following XML attributes for each XML element `Class`:

  - `Name`: Enter the technical name of the Alfabet class (for example: `Skill`, `Person`, and `OrgaUnit`.) The technical name must be correctly spelled. The XML attribute `Name` may not exceed 64 characters.

  - `Properties`: Define a comma-separated list of the properties of the relevant Alfabet class that may potentially be mapped to resources in Microsoft® Project.

- `MatchPatterns`: Create one or more `Pattern` XML elements to specify the patterns to use to match properties not specified in the XML attribute `Properties`. For each XML element `Pattern`,

specify the following: take multiple properties in Alfabet and concatenate with delimiter to match with strings in MS.

- `Name`: Enter the name of the matching rule. The XML attribute `Name` may not exceed 64 characters. Please note that if the names of matching patterns have more than 64 characters, errors may occur when specifying the MS project methodology.

- `Properties`: Specify the properties that shall make up the pattern to be used to find for the skills (or persons or organizations) in the Alfabet database that match the resource in MS project.

- `Delimter`: Specify one or more delimiters used in resources. Only resources with the specified delimiters qualify as resources for which matching skills can be found.

7) In the toolbar of Alfabet Expand, click the **Save** button to save your changes.

Specifying a Wizard for Interoperability with Microsoft Project

Please note that in order to import MPP files to Alfabet, a wizard must be configured and assigned to the class setting for the class Project. This is mandatory in order to import projects via MS Project. An additional step will be prepended to the first wizard step that allows the import to be triggered in order to create the new project in Alfabet. For the class setting, the **Default Editor Type** attribute must be set to `Wizard` and the relevant wizard must be specified for the **Wizard** attribute.

When the user selects the **Import Project from MS Project File** or **Import Project from MS Project Server** options in the relevant views, the relevant wizard will be automatically prepended with an additional wizard step prompting the user to select the MPP file, the desired MS Project methodology, and the default skill to be used for any resources in the MPP file that cannot be matched based on the matching rules present in the MS Project methodology. Once the **Next** button is clicked in the prepended wizard step, the project will be created and the wizard will move to the configured first wizard step so that the new project can be edited. For more information about the configuration of the project management capability as well as the specification of project stereotypes via the XML object **ProjectManager**, see the section *Configuring the Project Management Capability* in the reference manual *Configuring Alfabet with Alfabet Expand*.

Specifying Release Status for Interoperability with Microsoft Project

Ensure that release status definitions have been defined for the class **MS Project Methodology**. In the XML object **ReleaseStatusDefs**, create an XML element **ReleaseStatusDef** for the class `MicrosoftProject-Plan_Mappings`. Ensure that an XML attribute `ApprovedStatus` is defined. Only MS project methodologies for which the approved status has been defined may be selected for the import or export of MS projects.

For more information about specifying the XML object **ReleaseStatusDefs**, see the section *Configuring Release Status Definitions for Object Classes* in the reference manual *Configuring Alfabet with Alfabet Expand*.

Chapter 14:   Configuring Interoperability with a Translation Service

The XML object **AlfaTranslationServicesConfig** allows you to specify interoperability with a translation service for the purposes of providing an automatic 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 about the configuration required, see the sections *Translating Custom Strings via the Automated Translation Capability* and *Configuring the Translation of Object Data* in the chapter *Localization and Multi-Language Support for the Alfabet Interface* of the reference manual *Configuring Alfabet with Alfabet Expand*.

The following prerequisites must be fulfilled in order to fetch translations via the automated translation capability:

- 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.

(i)   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 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

  > ℹ️ 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 <u>Configuring Server Variables for Integration and Interoperability Solutions</u>.

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.

  > ℹ️ 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.

Chapter 15:   Configuring Integration of an External BOT

An external BOT configured either in Microsoft Virtual Agent® or Google OpenDialog® can be integrated in the Alfabet user interface. External BOTs are accessible in the Alfabet user interface in a small window that opens if the user select the bot icon in the slide-in toolbar.
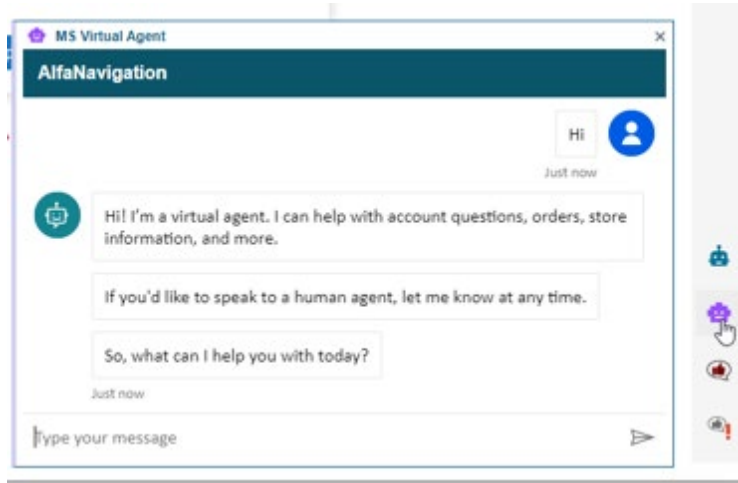
To integrate an external BOT, access to the external BOT must be configured in the XML object **ExternalVirtualAgentConfig** in Alfabet Expand:

1) Go to the **Presentation** tab and expand the **XML Objects** node.

2) Expand the **IntegrationSolutions** node and click the **ExternalVirtualAgentConfig** node. the XML editor opens.

3) Define an XML element **ExternalVirtualAgentConfig** with either a child XML element `MSInfo` for integration of a BOT configured in Microsoft Virtual Agent or a child XML element `GoogleInfo` for integration of a BOT configured in Google OpenDialog. Both XML elements have the same XML attributes:

   - `IsActive`: Set this XML attribute to `true` to enable use of the BOT on the Alfabet user interface. If multiple BOTs are defined in the XML object, only one of the definitions can be defined as active.

   - `Url`: Enter the URL of the external BOT.

     ```
     <ExternalVirtualAgentConfig>

         <MSInfo IsActive="true" Url="http://MS-service.com"/>

         <GoogleInfo IsActive="false" Url="http://google-service.com"/>

     </ExternalVirtualAgentConfig>
     ```

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

Chapter 16:   Configuring Integration with OAS Based RESTful Services

An API is available to interface with any external application offering a RESTful service API with an OpenAPI specification file (OAS). Import of data to and export of data from Alfabet can be configured by the customer based on the OpenAPI specification. The definition of the API requires the following configuration steps in Alfabet Expand

- The new XML object **GenericAPIIntegrationConfig** allows the connection to the RESTful API of the external application including access permission specification and optional specification of a user agent string to be defined. HTTP and OAuth authentification is supported. Server variables can be used in the definition to avoid direct specification of sensitive access data in the XML object. The connection can optionally be routed via a proxy. Connections to different RESTful APIs can be defined in the XML object **GenericAPIIntegrationConfig**.

- A resource bundle must be defined via the context menu of the **Resource Bundles** node in the **Reusable Elements** explorer. The **Bundle Type** must be set to `OpenAPISpecification` and the **Assistant** attribute must be set to `GenericAPIIntegration_OASAssistant`. The assistant can then be opened via the context menu of the resource bundle to define the data integration. The OpenAPI specification file must first be imported to the assistant. The calls to the API endpoints can then be configured for both data import and export based on the Swagger definition in the OpenAPI specification file. Multiple calls to different endpoints can be defined in the assistant.

- A database connection definition of the new database connection type **Generic API Integration Database Connection** must be defined in the **Integration Solutions Configuration** functionality in the Alfabet user interface. The database connection specifies which connection definition from the XML object **GenericAPIIntegrationConfig**, which resource bundle, and which call to an endpoint defined in the resource bundle are relevant for the connection. One database connection must be defined for each call defined in a resource bundle. Authorization details must also be specified.

- To specify Alfabet data to export to the external application and to specify Alfabet data that needs to be provided with calls to the RESTful API of the external application, a configured report must be defined that provides the data in the required format. Many of the resources in an OpenAPI REST API will use parameters to navigate the resource network of the API. For example, an API for Project Tasks would typically use a {Project} parameter in the call structure. These parameters must be replaced at runtime and will be provided via a report. The URL for the REST API call is generated at runtime from the URL with parameter placeholders defined in the XML object **GenericAPIIntegrationConfig** and the parameter values returned via the report. The report must return a simple tabular dataset and must be assigned to a category defined in the XML object **UseCaseCategories** for the new use case `GenericOASIntegration`.

- An ADIF import or export scheme must be generated via the `GenericAPIIntegration_Assistant` that handles both import and export of data. The assistant will automatically generate the information about temporary tables generated during import for ADIF import. In addition, handling of response requests for data export can be configured. Response information can be mapped to standard or custom object class properties of the exported object class.

Appendix 1:   ServiceNow Ressources Accessed by the Integration API

This information is relevant to configure routing of requests to ServiceNow via an API gateway.

The following resources of ServiceNow must be available via the API gateway:

Resource	Method	Operation Type	Parameters	API/Resource Parameter	Parameter Type	Parameters Type	Is Optional
/oauth_token.do	POST	Import and Export	client_id	Resource	Header	String	Yes
		Import and Export	client_secret	Resource	Header	String	Yes
		Import and Export	grant_type	Resource	Header	String	Yes
		Import and Export	password	Resource	Header	String	Yes
		Import and Export	username	Resource	Header	String	Yes
/{tablename}.do	GET	Import	SCHEMA	Resource	Query String	String	Yes

Resource	Method	Operation Type	Parameters	API/Resource Parameter	Parameter Type	Parameters Type	Is Optional
		Import	{tablename}.do	Resource	Path	String	No
		Import and Export	CSV	Resource	Query String	String	Yes
		Import	jvar_report_id	Resource	Query String	String	Yes
		Import	sysparm_record_count	Resource	Query String	String	Yes
		Export	WSDL	Resource	Query String	String	Yes
/api/now/v1/table/{tablename}	GET	Import	{tablename}	Resource	Path	String	No
		Import	sysparm_exclude_reference_link	Resource	Query String	String	Yes
		Import	sysparm_display_value	Resource	Query String	String	Yes
		Import	sysparm_limit	Resource	Query String	Integer	Yes

Resource	Method	Operation Type	Parameters	API/Resource Parameter	Parameter Type	Parameters Type	Is Optional
		Import	sysparm_offset	Resource	Query String	Integer	Yes
		Import	sysparm_fields	Resource	Query String	String	Yes
		Import and Export	sysparm_query	Resource	Query String	String	Yes
		Export	source_table	Resource	Query String	String	Yes
		Export	map	Resource	Query String	String	Yes
		Export	GOTOname	Resource	Query String	String	Yes
		Export	name	Resource	Query String	String	Yes
		Export	sys_id	Resource	Query String	String	Yes
/api/now/v1/stats/{entryid}	GET	Import	{entryid}	Resource	Path	String	No
		Import	sysparm_count	Resource	Query String	String	Yes

Resource	Method	Operation Type	Parameters	API/Resource Parameter	Parameter Type	Parameters Type	Is Optional
		Import	sysparm_query	Resource	Query String	String	Yes

Index