software AG

Alfabet Interface for RESTful Web Services

Alfabet Reference Manual

Documentation Version Alfabet 10.11.0

CONVENTIONS USED IN THE DOCUMENTATION

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

TABLE OF CONTENTS

Chapter 1:   Introduction

A RESTful API is available for the Alfabet application that provides easy access to the content in the Alfabet database. The API is designed as a web service architecture based on the Representational State Transfer (REST) software architecture type. Session cookies are not used.

The API can be used for the following:

- to get information about the structure of the object class model,

- to get information about objects stored in the Alfabet database,

- to create, update and delete objects and relationships in the Alfabet database,

- to archive objects in the Alfabet database,

- to trigger execution of ADIF jobs based on an ADIF scheme stored in the Alfabet database,

- to trigger start of a workflow via the Alfabet Web Application based on a workflow template stored in the Alfabet database,

- to regenerate user passwords,

- to anonymize user data for selected users,

- to download documents from the **Internal Document Selector**, upload documents to the **Internal Document Selector** and generate a list of documents in the **Internal Document Selector**,

- to update the meta-model configuration of the Alfabet database with the configuration stored in an AMM file,

- to check the availability of the Alfabet components.

This document describes the available data endpoints including the required calls and return values. On basis of the given information, customers can build interfaces that access the Alfabet database via HTTP request to the Alfabet RESTful API of a running Alfabet Web Application. The responses are sent in JSON format.
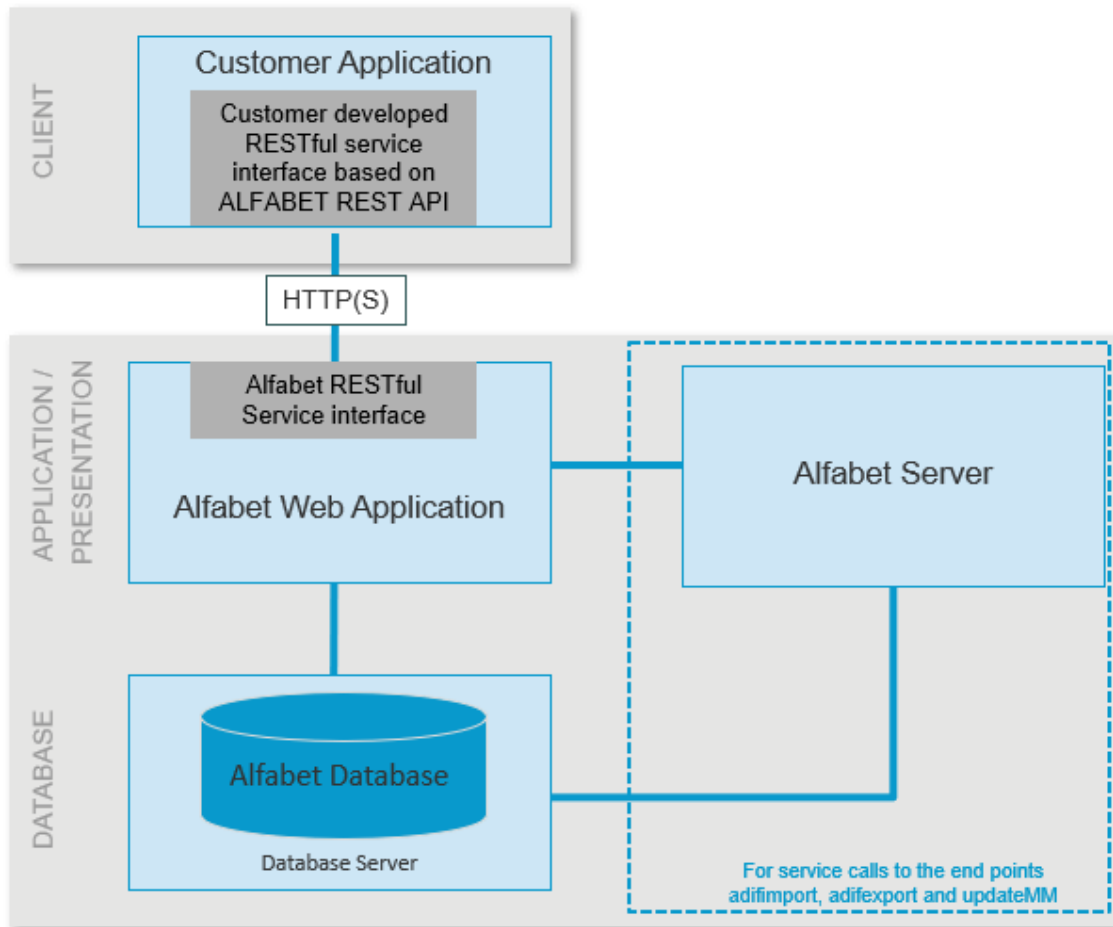
Chapter 2:   Required Licenses

A license for the Alfabet Data Integration Framework (ADIF) must be active to use the Alfabet RESTful API.

> ⓘ In Alfabet 9.10, the Alfabet API for RESTful Web services has been extensively revised to enhance security. In Alfabet 10.11, both the previous version 1 of the Alfabet RESTful API as well as the new version 2 are supported, but this reference manual addresses only the newer version 2 of the Alfabet RESTful API. Please note that only the newer version 2 will be supported as of October 2017. Alfabet releases prior to 9.10 will not provide the functionality described in this reference manual.
>
> The following GET request can be sent to the RESTful API of the Alfabet Web Application in order to check whether the Alfabet Web Application that is being connected to via RESTful services provides an interface based on version 1 or 2:

```
http://<URL of the Alfabet Web application>/api/Versions
```

> The return value lists the available versions of the API on the server-side. This call enables the solution developer using the REST API for Alfabet to determine the functionalities that a specific Alfabet Server is to support. Please note that a version of the API must be enabled in the server alias of the Alfabet Web Application to be listed in the return value.

Chapter 3: Activating the Alfabet RESTful API on Server Side

Service calls to the Alfabet RESTful API of the Alfabet Web Application are only processed by the Alfabet Web Application if an active license for the Alfabet Data Integration Framework (ADIF) is available and the following configurations are done on server side:

- Enable the Alfabet RESTful API in the web.config Files of the Alfabet Web Application

- Configure the Web Server Hosting the Alfabet Web Application to Enable the Alfabet RESTful API

  - Disable the WebDAV module of the Internet Information Services

  - Setting the Required Authorization for the api folder

- Configuring the Server Alias of the Alfabet Web Application to Enable REST API calls

Enable the Alfabet RESTful API in the web.config Files of the Alfabet Web Application

The `web.config` configuration file is located in the Alfabet Web Application directory of the installation directory for the Alfabet components. The `web.config` file can be modified using a standard text editor.

The required settings for the Alfabet RESTful API must be added to the `handlers` element of the `web.config` file of the Alfabet Web Application. The handlers element must have the following child elements apart from child elements that are already included for other processes:

```
<remove name="ExtensionlessUrlHandler-Integrated-4.0" />

<add name="AlfaRest1" type="AlfabetWeb5.api.v1.AlfaRestService, AlfabetWeb5"
verb="*" path="api/v1" />

<add name="ExtensionlessUrlHandler-Integrated-4.0" path="*."
verb="GET,HEAD,POST,DEBUG,PUT,DELETE"
type="System.Web.Handlers.TransferRequestHandler" resourceType="Unspecified"
requireAccess="Script" preCondition="integratedMode,runtimeVersionv4.0"
responseBufferLimit="0" />
```

In addition, the following code must be available in the web.config file to include the correct version of the required third party component Newtonsoft.Json:

```
<configuration>

   ...

   <runtime>

      <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">

         <dependentAssembly>

            <assemblyIdentity name="Newtonsoft.Json"
            publicKeyToken="30ad4fe6b2a6aeed" culture="neutral" />

            <bindingRedirect oldVersion="0.0.0.0-12.0.0.0"
            newVersion="12.0.0.0" />

         </dependentAssembly>

         <dependentAssembly>
```

```
            <assemblyIdentity name="System.Net.Http.Formatting"
            publicKeyToken="31bf3856ad364e35" culture="neutral" />

            <bindingRedirect oldVersion="0.0.0.0-5.2.7.0"
            newVersion="5.2.7.0" />

         </dependentAssembly>

      </assemblyBinding>

   ...

   </runtime>

</configuration>
```

ⓘ This section might be subject to changes during upgrades from one Alfabet version to the other, because of changes to the version of the embedded component. It is recommended to use the example web.config file delivered with the release and adapt it to the current environment to make sure that the above mentioned code and all other required settings are correctly set in the web.config file as required for the specific release.

⚠ If SAML or other single sign on authentication mechanisms are used for Alfabet, this settings might interfere with the settings required for the authentication method. Please contact Software AG Support if problems occur with the REST API settings in the web.config file.

Optionally, security of data transmission can be enhanced by specification of the JSON Web Token (JWT) for sending JSON objects via the Alfabet RESTful service API. By default, the JWT is hard-coded and therefore the same for all Alfabet installations. To change the JWT for an installation, an individual JWT with a minimum length of 128 bits must be base64 encrypted and the encrypted version must be entered into the in the `alfabet.config` file of the Alfabet Web Application:

1) Open the Alfabet Administrator.

2) Click the **Alfabet Aliases** node in the explorer. A workspace with a toolbar opens.

3) In the toolbar, click **Tools** > **Configure alfabet.config**. An editor opens.

4) Click the **Browse** ⬚ button on the right of the **Web Folder** field and select the main directory of the Alfabet Web Application from the directory browser. The `alfabet.config` file in the subdirectory *config* of the selected directory opens in the editor.

5) Add the following code as child element of the `alfaSection` XML element:

```
<add key="ApiJwtBase64Key" value="{Base 64 Encrypted key}"/>
```

6) Click **Save**. The change is saved and the editor is closed.

If your RESTful client will send a high number of service calls per second to the Alfabet RESTful service API, it might be required to increase the maximum allowed number of requests per second. By default processing of incoming RESTful service calls is limited to 300 per second. The limit can be changed in the alfabet.config file of the Alfabet Web Application:

1) Open the Alfabet Administrator.

2) Click the **Alfabet Aliases** node in the explorer. A workspace with a toolbar opens.

3) In the toolbar, click **Tools** > **Configure alfabet.config**. An editor opens.

4) Click the **Browse** ⬚ button on the right of the **Web Folder** field and select the main directory of the Alfabet Web Application from the directory browser. The `alfabet.config` file in the subdirectory *config* of the selected directory opens in the editor.

5) Find the `add` XML element with the `key` XML attribute set to `max_api_requests_per_second` and change the `value` XML attribute to the required number of requests per second. The default is 300:

```
<add key="max_api_requests_per_second" value="300"/>
```

6) Click **Save**. The change is saved and the editor is closed.

> ⓘ Please note that request limits configured for the web server outside of the Alfabet Web Application for the maximum size of a request, the maximum URL length and the maximum length of a query string will apply to all calls to the Alfabet RESTful services.

Configure the Web Server Hosting the Alfabet Web Application to Enable the Alfabet RESTful API

The following settings are required for the Web Server hosting the Alfabet Web Application to enable the Alfabet RESTful API:

*   The WebDAV module of the Internet Information Services® hosting the Alfabet Web Application is not installed.

*   The Alfabet RESTful API of the Alfabet Web Application does not support Windows sign On. Windows sign on must be disabled and Anonymous Access must be enabled for the `api` sub-folder of the Alfabet Web Application on the Web server.

> ⚠ If the Alfabet Web Application is configured to authenticate users via portal authentication, the RESTful API cannot be implemented at all for that Alfabet Web Application.
>
> To use the Alfabet RESTful API in combination with portal authentication, a second Alfabet Web Application accessing the same Alfabet database must be implemented. This additional Alfabet Web Application must run with a different server alias configured to use standard authentication via user name and password for user access to the Alfabet database. The Alfabet RESTful API is provided by this additional Alfabet Web Application. All configuration steps and URL specification described in this manual refer to that additional Alfabet Web Application.

Disable the WebDAV module of the Internet Information Services

The WebDAV module can be deactivated in the Server Roles:

1) On the Web Server host, click the **Start** 🪟 icon that appears when you move the mouse to the lower left corner and click on the **Server Manager** 🖥 icon to open the Server Manager.

2) In the menu on the upper right of the Server Manager, select **Manage** > **Add Roles and Features**.

3) In the **Add Roles and Features Wizard**, select the installation type and the Web server host in the first three pages of the wizard and proceed to the **Server Roles** page, using the **Next** button of the wizard.

4) In the **Server Roles** page, expand **Web Server (IIS)**.

5) Make sure that the **WebDAV Publishing** option is not checked:
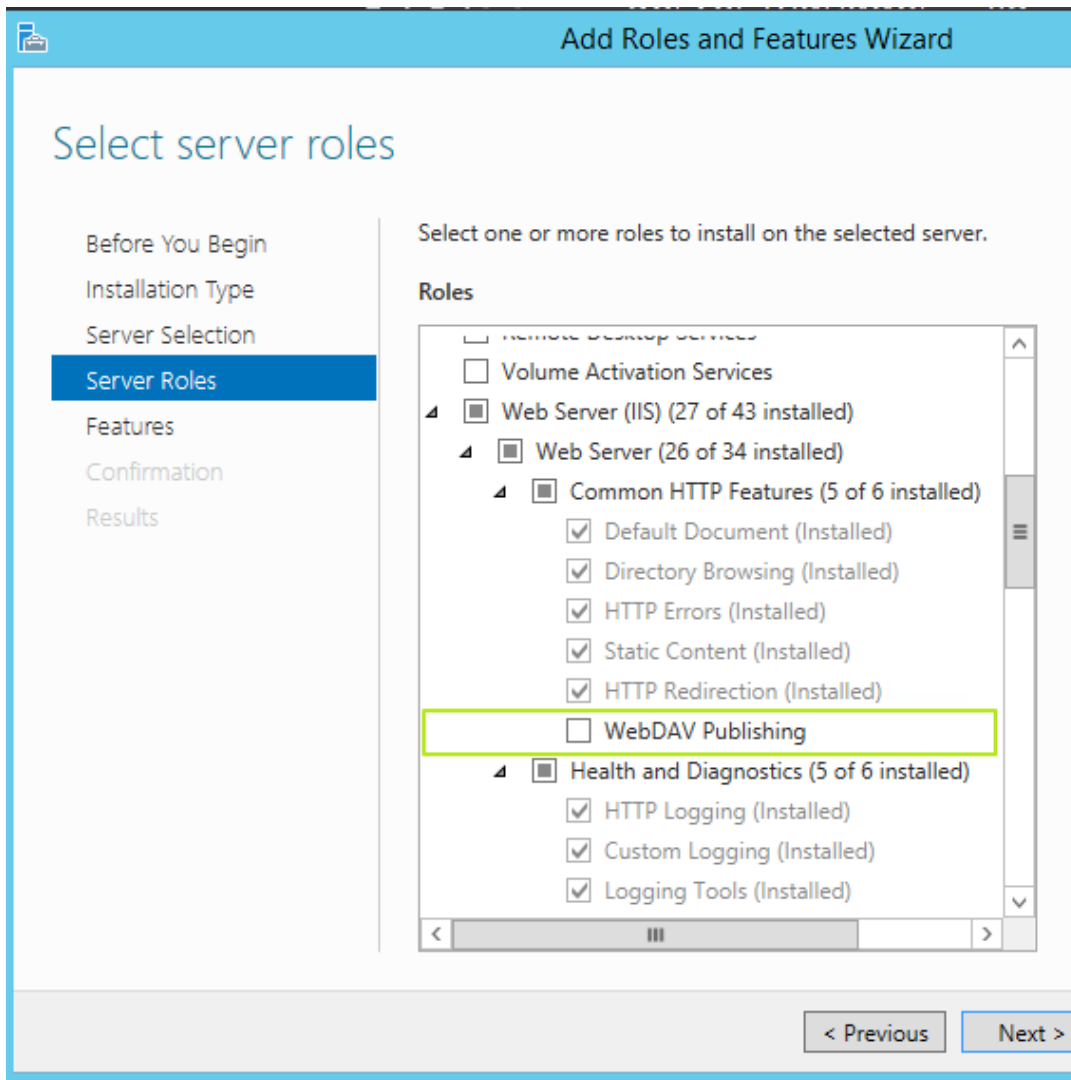
Setting the Required Authorization for the api folder

Independent from the authentication selected for the Alfabet Web Application, the authentication of the `api` sub-folder of the Alfabet Web Application must be set to **Anonymous Authentication**:

1) On the Web Server host, click the **Start** icon that appears when you move the mouse to the lower left corner and click on the **Server Manager** icon to open the Server Manager.

2) In the menu on the upper right of the Server Manager, select **Tools** > **Internet Information Services (IIS) Manager**.

3) In the explorer, expand the node of the Alfabet Web Application and click on the `api` folder node.

4) In the section **IIS** in the middle pane, double-click **Authentication**. The status of the available authentication modes is displayed.
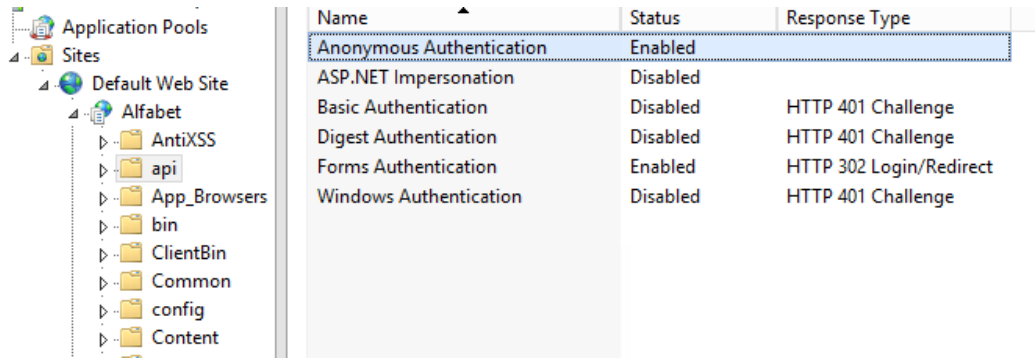
5) Set the **Anonymous Authentication** option to **Enabled** and the **Windows Authentication** option to **Disabled**. Changing of the settings is performed by selecting the option in the list and clicking **Disable** or **Enable** respectively in the **Action** pane on the right.

Configuring the Server Alias of the Alfabet Web Application to Enable REST API calls

The Alfabet API for RESTful Web services must be enabled in the server alias of the Alfabet Web Application. Configuration is performed in the configuration tool Alfabet Administrator:

> For information about how to access and use the Alfabet Administrator, see the section *Working with the Alfabet Administrator* in the reference manual *System Administration*.

1) In the explorer of the Alfabet Administrator, click the **Alfabet Aliases** node.

2) In the table, click the server alias that you want to configure.

3) In the toolbar, click the **Edit** button. An editor opens.

4) Go to the **Server Settings** > **REST API** tab.

5) Set the checkmark for the **Enable REST API v2** option.

> There is a second option **Enable REST API v1**. This option enables the use of the deprecated version of the ARIS - Alfabet Interoperability Interface and is only included for backward compatibility reasons.

6) Select the checkmark of any RESTful service endpoint that shall be activated on the Alfabet Web Application in the **API Access Options** field:

- **Has Meta-Model Access**: Enables access to the `metamodel`, `classes` and `enums` endpoints to read information about the structure of the Alfabet class model including enumerations and culture settings.

- **Has GetObjectsByRefs Access**: Enables access to the `objects` endpoint to read information about data stored for objects in the Alfabet database that are found via specification of the object's `REFSTR` in the REST API request. The **Has GetObjectsByRefs Access** option is also required for access to the `archiveobject` endpoint to export information about the object in an archive ZIP file containing relevant views as HTML files.

- **Has GetObjectsByReport Access**: Enables access to the `objects` endpoint to read information about data stored for objects in the Alfabet database that are found via a configured report.

- **Has GetObjectsByFilter Access**: enables access to the `objects` end point to read information about data stored for objects in the Alfabet database that are found via specification of the object class and filter conditions in the call.

- **Has CreateObjects Access**: Enables access to the `update` end point to create new object in the Alfabet database.

- **Has UpdateObjects Access**: Enables access to the `update` endpoint to update data for existing objects in the Alfabet database.

In addition, this access option is required for access to the `regeneratepassword` endpoint. Please note that the **Has PasswordAPI Access** access option must also be selected to enable the `regeneratepassword` endpoint.

- **Has DeleteObjects Access**: Enables access to the `delete` endpoint to delete existing objects in the Alfabet database. The **Has GetObjectsByRefs Access** permission is also required for access to the `archiveobject` endpoint to export information about the object in an archive ZIP file containing relevant views as HTML files if the option for deleting the object after export of the archive is selected.

- **Has AnonymizeUser Access**: Enables access to the `anonymizeuser` endpoint to anonymize users that are found via specification of the user's `REFSTR` in the REST API request in the Alfabet database.

- **Has PasswordAPI Access**: Enables access to the `regeneratepassword` endpoint. Please note that the **Has UpdateObjects Access** option must also be selected to use the `regeneratepassword` endpoint.

- **Has ADIFAPIInvocationAccess**: Enables access to the `adifimport` and `adifexport` endpoints to trigger execution of ADIF jobs based on an ADIF scheme in the Alfabet database. Please note that this access option is also required for asynchronous execution of the ADIF scheme via the **ADIF Jobs Administration** functionality and for execution of ADIF schemes via the **Job Schedule** functionality.

- **Has WorkflowAPIInvocation Access**: Enables access to the `workflow` endpoint to trigger start of a workflow based on a workflow template in the Alfabet database.

- **Has MonitoringAPI Access**: Enables access to the `monitor` endpoint to check whether the Alfabet components are available.

- **Has IDoc Upload Access**: Enables access to the `idocupload` endpoint for upload of documents into the internal document selector in the Alfabet database.

- **Has IDoc Download Access**: Enables access to the `idocdownload` endpoint for download of documents from the internal document selector in the Alfabet database.

- **Has IDoc File List Access**: Enables access to the `idocfilelist` endpoint for export of information about the content of the internal document selector in the Alfabet database.

- **Has Batch Utilities API Access**: Enables access the RESTful service functionality required to execute batch processes via the **Job Schedule** functionality and to asynchronously export or import data capture templates.

- **Has Update Meta-Model Access**: Enables access to the `updateMM` endpoint to update the meta-model of the target database with the meta-model configuration stored in an AMM file.

- **Has View Snapshot Access**: Enables access to the endpoint that is used by the Microsoft Teams® integration feature to include snapshots of Alfabet views into Microsoft Teams channel. This option should be enabled if the Microsoft Teams integration is implemented.

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

Chapter 4:   Authorization

Authorization for the Alfabet Data Integration Framework is done per user, per class and per object:

- Access to the Alfabet RESTful API must be explicitly granted to a named Alfabet user in the user configuration in the Alfabet database. The permissions can be restricted to a subset of the available endpoints.

Prior to calls for data request, an authorization call must be performed. In the authorization call the user name and token is submitted to the Alfabet RESTful API of the Alfabet Web Application, and a unique authorization key for data request calls is sent back. This unique authorization key can be used for a configurable time period for data request calls following the authorization call. After the key expires, a new authorization call must be performed.

- Configuration objects like ADIF schemes, workflow templates, and configured reports are only executable via REST API calls if the attribute settings of the configuration object includes permission for execution via REST API.

- Data about an object class can only be read, created or changed via the Alfabet RESTful API if the class settings for the object class are configured to grant the permissions. An object class can have multiple class settings. Class settings are assigned to view schemes. A view scheme is then assigned to a user profile. One or multiple user profiles can be assigned to a named user.

In parallel to login to the Alfabet user interface, REST API calls are also performed for a specific user with a specific user profile. The user profile is either defined in the REST API request or a default is selected from the user profiles assigned to the user. Whether the user is allowed to read, create or edit data for an object class, depends on the class settings relevant for the user profile.

- If the mandate capability is implemented in your company, the user can only read data about an object if the mandate settings for the user and the object permit access to the object. If the user has multiple mandates assigned, the mandate valid for the REST API call can be specified in the call. If the mandate specified in the call is not assigned to the user, an error is thrown.

  > For an overview of the mandate capability and the configuration of mandates, see *Implementing the Mandate Capability for a Federated Architecture* in the reference manual *Configuring Alfabet with Alfabet Expand*.

- Whether a user can edit data about an object depends on the implemented access permission concepts in your Alfabet solution, like for example right rules, the membership to authorized user groups, or the assignment of tasks about an object in the contexts of workflows or assignments.

  > For a complete overview of the object permission concepts in Alfabet, see the chapter *Configuring Access Permissions for Alfabet* in the reference manual *Configuring Alfabet with Alfabet Expand*.

Required Configuration on Server Side

Prior to sending service calls to the Alfabet RESTful API, the Alfabet solution must be configured to grant all required access permissions:

- Generating a REST API Password for a User

- [Enabling Reports, ADIF Schemes, and Workflow Templates to be Executed via RESTful Service Calls](#)

- [Enabling Access to Folders in the Internal Document Selector](#)

- [Configuring Per Object Class Permissions for Reading From or Writing Into the Alfabet database](#)

- [Controlling Access Via Mandates](#)

- [Controlling Access Per Object via Access Permission Concepts](#)

Generating a REST API Password for a User

Authorization is done per user and requires that access to the Alfabet RESTful API is explicitly granted to a named Alfabet user in the user configuration in the Alfabet database. The editor fields required for configuring the authorization are only available in the user editor if a valid license for ADIF is available.

User configuration is performed in the **User Administration** functionality in the user interface that is accessible via the `Admin` user profile or the **User Management** functionality that is available via the connected server alias node in the Alfabet Administrator.

> ⓘ For more information about how to create a named Alfabet user, see the chapter *Defining and Managing Users* in the reference manual *User and Solution Administration* or *Managing New and Existing Users* in the reference manual *System Administration*.

For security reasons and to avoid technical conflicts, take the following into account when granting REST API access permissions to a named user:

- The user should be exclusively used for calls to the Alfabet RESTful API and should not be used in parallel to access the Alfabet application via the Alfabet user interface or to access the Alfabet database via other interfaces provided by Alfabet, like ADIF or the ARIS - Alfabet Interoperability Interface.

- Multiple users with access to a subset to the endpoints of the Alfabet RESTful API that can either have read only or read/write access permissions can be defined. If multiple client applications are configured to send requests to the Alfabet RESTful API, individual users with access permissions exactly matching the requirements should be used for each client application.

- One of the users that are configured to have access to the Alfabet restful services should be defined for the execution of self-reflective events. Many Alfabet processes are based on the execution of RESTful service calls in the background. These processes are authorized using the user specified to be used for self-reflective events.

After having created a named user, the following settings must be performed to grant access to the Alfabet RESTful API:

1) In the table of the **User Administration** functionality, select the user that shall be used to send request via the Alfabet RESTful API.

2) In the toolbar, click the **Edit** ✎ button. The **User** editor opens.

3) In the editor, go to the **Permissions** tab.

4) Select the **Has API V2 Access** checkbox.

5) Specify details about the access permissions with the following attributes:

- **API Token Duration (minutes)**: The RESTful service interface on client side must be implemented to send a request for authorization code prior to sending a data request. The authorization code received in the response of the authorization request can be used in data request calls to the Alfabet RESTful API posted within a limited amount of time after receiving the authorization code. Enter the number of minutes the authorization code should be valid. By default the authorization code can be used for 20 minutes.

- **API Access Options**: By default, most of the options are checked in this field to give the user access to all central functionality provided for the Alfabet RESTful services. Deselect all options that the user should not have permissions to perform:

    - **Has Meta Model Access**: If selected, the user has access to the `metamodel`, `classes` and `enums` endpoints to read information about the structure of the Alfabet class model including enumerations and culture settings.

    - **Has GetObjectsByRefs Access**: If selected, the user has access to the `objects` endpoint to read information about data stored for objects in the Alfabet database that are found via specification of the object's `REFSTR` in the REST API request. The **Has GetObjectsByRefs Access** permission is also required for access to the `archiveobject` endpoint to export information about the object in an archive ZIP file containing relevant views as HTML files.

    - **Has GetObjectsByReport Access**: If selected, the user has access to the `objects` endpoint to read information about data stored for objects in the Alfabet database that are found via a configured report.

    - **Has GetObjectsByFilter Access**: If selected, the user has access to the `objects` endpoint to read information about data stored for objects in the Alfabet database that are found via specification of the object class and filter conditions in the call.

    - **Has CreateObjects Access**: If selected, the user has access to the `update` endpoint to create new object in the Alfabet database.

    - **Has UpdateObjects Access**: If selected, the user has access to the `update` endpoint to update data for existing objects in the Alfabet database.

In addition, this permission is required for access to the `regeneratepassword` endpoint. Please note that the **Has PasswordAPI Access** permission must also be selected to use the `regeneratepassword` endpoint.

    - **Has DeleteObjects Access**: If selected, the user has access to the `delete` endpoint to delete existing objects in the Alfabet database. The **Has DeleteObjects Access** permission is also required for access to the `archiveobject` endpoint to export information about the object in an archive ZIP file containing relevant views as HTML files if the option for deleting the object after export of the archive is selected.

    - **Has AnonymizeUser Access**: If selected, the user has access to the `anonymizeuser` endpoint to anonymize users that are found via specification of the user's `REFSTR` in the REST API request in the Alfabet database.

    - **Has PasswordAPI Access**: If selected, the user has access to the `regeneratepassword` endpoint. Please note that the **Has UpdateObjects Access** permission must also be selected to use the `regeneratepassword` endpoint.

- **Has ADIFAPIInvocationAccess**: If selected, the user has access to the `adifimport` and `adifexport` endpoints to trigger execution of ADIF jobs based on an ADIF scheme in the Alfabet database.

- **Has WorkflowAPIInvocation Access**: If selected, the user has access to the REST endpoint `workflow` to trigger start of a workflow based on a workflow template in the Alfabet database.

- **Has MonitoringAPI Access**: If selected, the user has access to the `monitor` end point to check whether the Alfabet components are available.

- **Has IDoc Upload Access**: If selected, the user has access to the `idocupload` endpoint for upload of documents into the **Internal Document Selector** in the Alfabet database.

- **Has IDoc Download Access**: If selected, the user has access to the `idocdownload` endpoint for download of documents from the **Internal Document Selector** in the Alfabet database.

- **Has IDoc File List Access**: If selected, the user has access to the `idocfilelist` endpoint for export of information about the content of the internal document selector in the Alfabet database.

- **Has Batch Utilities API Access**: If selected, the user has access to the RESTful service functionality required to execute batch processes via the **Job Schedule** functionality and to asynchronously export or import data capture templates.

- **Has Update Meta-Model Access**: If selected, the user has access to the `updateMM` endpoint to update the meta-model of the target database with the meta-model configuration stored in an AMM file.

- **Has View Snapshot Access**: Enables access to the endpoint that is used by the Microsoft Teams® integration feature to include snapshots of Alfabet views into Microsoft Teams Teams channel. This option should be enabled for the user configured to execute self-reflective events if the Microsoft Teams integration is implemented.

> ⓘ Access to object data, ADIF schemes, workflows, reports and the Internal Document Selector require additional permission settings described in the following sections. An overview of all settings required for individual endpoints is given in the section <u>Overview of Access Permissions Required for Each Endpoint</u>.

6) Click the **Generate API Password** button. A code is generated and stored in the Alfabet database.

7) Copy the code and store the information about user name and code for use on client side.

8) If the user shall be used for execution of events of the type `SelfReflective`, select the user in the table and select **Action** > **Set as Executes Self-Reflective Events User** in the toolbar.

> ⓘ Only one user can be selected for execution of events of the type `SelfReflective`. If you assign this functionality to a user while another user has already been selected to execute self-reflective events, the setting is removed from that user when it is set for the user you are currently assigning it to.

The generated code must be used for authorization of the client in requests sent to the Alfabet RESTful API of the Alfabet Web Application in combination with the user name.

The generation of the password can be performed for a user even if the **Has API Access** check-box is not checked, but the generated code is only valid for access to the Alfabet RESTful API when the **Has API V2 Access** checkbox is set for the user.

Enabling Reports, ADIF Schemes, and Workflow Templates to be Executed via RESTful Service Calls

Some endpoints include execution of user configuration objects created by a solution administrator in Alfabet Expand:

- Configured reports can be executed via a service call to the endpoint `objects` to export predefined object data.

- ADIF Schemes can be executed via a service call to the endpoints `adifimport` or `adifexport`.

- Workflows can be started for a workflow template via a service call to the endpoint `workflow`.

The access permissions required for execution via RESTful service calls must be defined in the attributes of the configured report, ADIF scheme or workflow template in Alfabet Expand:

1) Click the configured report, workflow template or ADIF scheme that you want to execute via RESTful API in the respective explorer of Alfabet Expand.

2) In the attribute window, set the **Applicable for REST API** attribute to `True`.

3) Check whether the following additional configuration requirements are met:

- **For workflow templates**: Only workflow templates with the **Automatic Start** attribute set to `True` can be executed via RESTful service calls.

- **For ADIF schemes**: None.

- **For configured reports**: The configured report must return a tabular dataset.

4) In the toolbar, click the **Save** button to save your changes.

Enabling Access to Folders in the Internal Document Selector

Folder access permissions specified in the **Internal Document Selector** are evaluated for REST API calls to the `idocupload`, `idocdownload` and `idocfilelist`, and to the end points `adifimport` and `adifexport`, if import or export involve files located in the **Internal Document Selector**.

Folder permissions are set in the **Internal Documents** functionality on the Alfabet user interface:

1) In the explorer of the **Internal Documents** functionality, click the parent folder of the folder that you want to set access permissions for.

2) In the table, click the folder that you want to set access permissions for.

3) In the toolbar, click the **Edit** button.

4) In the editor, select the checkmarks of the following access permissions in the **Default Access Permissions** field:

- For the `idocupload` and `adifexport` endpoints: **Manage Items**

- For the `idocdownload` and `adifimport` endpoints: **Open Items**

- For the `idocfilelist` endpoint: **List Items**

5) Click **OK** to close the editor.

> ⚠️ For security reasons, a blacklist and whitelist concept is available in Alfabet to restrict the uploading and downloading of files to the **Internal Document Selector** for specific file extensions. File extensions can be specified in a blacklist as not permissible for uploading and downloading. Alternatively, a whitelist can be configured that explicitly and exclusively allows specified file extensions for upload and download. These settings are also valid for upload and download of files via the Alfabet RESTful services.
>
> For details and information about the definition of the blacklist and the whitelist, see *Configuring the Permissibility of Files and Web Links in Alfabet* in the reference manual *Configuring Alfabet with Alfabet Expand*.

Configuring Per Object Class Permissions for Reading From or Writing Into the Alfabet database

If you would like object data to be read or written via the Alfabet RESTful API, you must define a user profile that grants the required permissions via the class settings assigned to the user profile's view scheme.

In Alfabet, the user profile a user is logged in with and the view scheme and class settings assigned to this user profile determine the scope of functionality available to the user.

> ℹ️ For detailed information about the configuration of class settings, view schemes and user profiles see the chapter *Configuring User Profiles for the User Community* in the reference manual *Configuring Alfabet with Alfabet Expand*.
>
> Please note that the predefined user profile `Admin` and any user profile that is marked as administrative user profile provides access to all objects via the Alfabet user interface independent of access permission concepts and should not be used in the context of the Alfabet RESTful services.

Most of the configuration for a user profile and the underlying view scheme and class settings is irrelevant for the use of the Alfabet RESTful API. The setting required to grant permission to create, edit or delete objects for an object class is exclusively specified via two attributes in the class setting of the object class. It is recommended to define class settings for exclusive use via the Alfabet RESTful API and assign them only to a view scheme and a user profile explicitly defined for use via the Alfabet RESTful API. The user profile can then be assigned to users authorized to read or alter data via the Alfabet RESTful API only.

Configuration of class settings, view schemes and user profiles is done in Alfabet Expand. The following configuration is required:

1) Go to the **Presentation** tab and expand the **Class Settings** node.

2) In the **Class Settings** folder, navigate to the object class folder that you want to define class settings for, right-click the public or private class setting template that you want to copy and

select the **New Class Settings as Copy** option to create a new class setting for the selected object class.

> The **New Class Settings for Stereotype as Copy** option must not be used in this context. The REST API permissions set for object class stereotypes are ignored. Permissions can only be defined on the level of the object class.

3) A copy of the class settings template   is added to the class setting folder. All attributes specified for the class setting that you copied will also be copied to the new class setting.

4) Click the new class setting template to open the attribute window and enter a technical name for the class setting in the **Name** attribute.

> It is recommended that the name of the class setting indicates the view scheme that it is being defined for and the name of the view scheme indicates the user profile it is assigned to. In other words, the name of the class setting should thus indicate the user profile that it is associated with.
>
> A validation mechanism checks for correct syntax when defining a technical name. For a list of the special characters that are not allowed, see the section *Defining Attributes for Configuration Objects* in the chapter *Getting Started with Alfabet Expand* in the reference manual *Configuring Alfabet with Alfabet Expand*.

5) Set one or both of the following attributes to `True`, depending on the rights that should be granted for the class:

- **Allow Read via Rest API**: Set to `True` to allow reading data of objects of this object class via the Alfabet RESTful API endpoint `objects`.

- **Allow Write via Rest API**: Set to `True` to allow update data of objects of this object class, creation of objects and deletion of objects via the `delete` and `update` endpoints.

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

7) Repeat steps 2) - 6) for all object classes that shall be editable via the Alfabet RESTful API.

8) Go to the **Presentation** tab and expand the **View Schemes** node.

9) To create the view scheme, do one of the following:

- Right-click the **View Schemes** folder and select **New View Scheme** to create a new view scheme from scratch, or

- Click a view scheme you want to copy and select **New View Scheme as Copy** to create a new view scheme based on an existing view scheme.

10) You will see the new view scheme   added to the **View Schemes** folder. In the attribute window, enter a name in the **Name** attribute.

> A validation mechanism checks for correct syntax when defining a technical name. The technical name of a configuration object may not begin with an empty space nor include any of the following special characters: **\ / * ? " > < |':**
>
> Furthermore, please note that names may not begin with an empty space nor include special characters. For a list of the special characters that are not allowed, see the

section *Defining Attributes for Configuration Objects* in the chapter *Getting Started with Alfabet Expand*.

Furthermore, if the technical name of a configuration object is changed, the name will be correctly updated in other configuration objects referencing the changed object during design time. Please note that the name of a changed configuration object will not be updated in guide pages nor if the guide pages reference a configuration object listed in the **Show Usage** functionality. If you plan to change the name of a configuration object, the reference in the guide page should be changed prior to changing the configuration object name.

11) You should now systematically determine whether each object class/object class stereotype requires a custom class setting or whether the standard class setting suffices for your needs. To open the **View Scheme** editor for the selected view scheme, double-click the view scheme you are configuring. The **View Scheme** editor opens in the center pane.

12) The table labelled **Object Class Configuration** displays object classes/object class stereotypes on the first level. Click the + to expand the table below the object class or object class stereotype that you want to configure. When you expand the object class node, you will see all existing standard class settings and custom class settings configured by your enterprise for the selected object class. To specify which class setting should be implemented in the selected view scheme, click in the **Use in View Scheme** column, set an X for the class setting to be implemented for the selected view scheme. Carry out this procedure for all object classes with a class setting configuration for the Alfabet RESTful API.

13) In the toolbar, click the **Save** button to save your changes.

14) Go to the **Admin** tab, right-click the **User Profiles** node and select **New User Profile**. A new user profile is displayed in the explorer.

15) In the attribute window, define the following attributes:

- **Name**: Enter a caption for the user profile. This is the name of the user profile that users will see when logging in to Alfabet.

- **Type**: If the user profile should have editing permissions, select `Read/Write`. If the user profile should only have viewing permissions, select `ReadOnly`.

- **View Scheme**: Click the **Drop-Down** button to select the relevant view scheme that applies to the user profile when accessed by an external application or via a hyperlink in an e-mail notification.

16) In the toolbar, click the **Save** button to save your changes.

After having defined the user profile, you must assign the user profile to the user that is already defined to have access to the Alfabet RESTful API as described in the section Generating a REST API Password for a User.

User configuration is performed in the **User Administration** functionality in the user interface that is accessible via the `Admin` user profile.

1) In the table of the **User Administration** functionality, select the user that shall be used to send requests to alter data via the Alfabet RESTful API.

2) In the toolbar, click the **Navigate** button. The object profile for the user opens.

3) Click the **Assigned User Profiles** link in the **User's Solution Configuration** section. The **Assigned User Profiles** page view opens.

4) In the toolbar, click **New** > **Assign User Profiles**. The **User Profile** selector opens.

5) Select the user profile that you have configured for changing data via the Alfabet RESTful API.

6) Click **OK** to save your selection.

Controlling Access Via Mandates

If your company is using the mandates concept, the mandate settings for users and objects in the Alfabet database are also taken into account for responses from the Alfabet RESTful API requesting object data or creating, deleting or updating objects.

> (i) For more information about the mandate concept and the required configuration steps see *Implementing the Mandate Capability for a Federated Architecture* in the reference manual *Configuring Alfabet with Alfabet Expand*.

In the payload of the service call, a mandate can be defined for the call with the field `CurrentMandate` as described in the description of service calls in the section Service Calls and Return Values. This setting is optional, but it is recommended to define the mandate in the call. The system checks whether the mandate defined in the call is also specified for the user used to send the service call. The user must either have the mandate assigned or be specified as mandate master. Otherwise the service call will be rejected.

When the mandate concept is implemented, the following rules apply to return of data via calls of the Alfabet RESTful API:

	No CurrentMandate is defined in the service call	A CurrentMandate is defined in the service call
The user is a mandate master and additionally has one or multiple mandates assigned.	The user will see data about all objects regardless of the mandate assignment of the objects in the return calls.	The user will see data about all objects assigned to the mandate defined with `CurrentMandate` and about all objects with no mandate assigned.
The user has a single mandate assigned and is not the mandate master.	The user will see data about all objects assigned to the mandate assigned to him and about all objects with no mandate assigned.	If the mandate specified with `CurrentMandate` is assigned to the user, the user will see data about all objects assigned to that mandate and about all objects with no mandate assigned. If the mandate specified with `CurrentMandate` is not assigned to the user, the service call is rejected with a message that informs about the fact that the `CurrentMandate` is not assigned to the user

	No CurrentMandate is defined in the service call	A CurrentMandate is defined in the service call
The user has multiple mandates assigned and is not the mandate master.	The user will see data about all objects assigned to the default mandate assigned to him and about all objects with no mandate assigned. The first mandate in alphabetical order of the mandates assigned to the user is the default mandate used.	If the mandate specified with `CurrentMandate` is assigned to the user, the user will see data about all objects assigned to that mandate and about all objects with no mandate assigned. If the mandate specified with `CurrentMandate` is not assigned to the user, the service call is rejected with a message that informs about the fact that the `CurrentMandate` is not assigned to the user
The user has no mandate assigned and is not a mandate master.	The user will see data about all objects that do not have a mandate assigned.	The service call is rejected with a message that informs about the fact that no mandates are assigned to the user.

If data about an object is requested with a service call to the `objects` endpoint and the mandate settings of the object do not match the mandate settings of the user or the mandate specified in the service call, no data is returned. Instead, two messages about the mandate mismatch are added to the return value:

```
"RejectedObjects": [

    {

    "Id": "",

    "RefStr": "76-3473-0",

    "Message": "Mandate mismatch"

    }

],

"AccessDenied": {

"76-3473-0": "Mandate mismatch"

}
```

If data of an object shall be updated with a service call to the `update` endpoint or an object shall be deleted with a service call to the `delete` endpoint, the return value also includes the `AccessDenied` field to inform about the mandate mismatch for individual objects.

Controlling Access Per Object via Access Permission Concepts

When access permissions per object class have been granted to the user executing the service call to the Alfabet RESTful API as described in the section Configuring Per Object Class Permissions for Reading From or Writing Into the Alfabet database and the users mandate concepts also permit access as described in the section Controlling Access Via Mandates, write access to objects might nevertheless be rejected because

of per object access permissions. The access permission concepts for a user that are implemented in your Alfabet solution are also evaluated for access to objects via the REST API service calls.

> ⓘ For an overview of available access permission concepts in Alfabet and the required configurations to implement them, see the section *Configuring Access Permissions for Alfabet* in the reference manual *Configuring Alfabet with Alfabet Expand*.

The following must be taken into account:

- User profiles can be configured to grant either ReadWrite or ReadOnly access to Alfabet objects. The configuration of the user profile used for the Alfabet RESTful API must be configured to grant the required access permissions.

- The user must have write access permissions to an object based on for example the authorized user concept or right rules to update or delete object data via a REST API service call.

Usually, a REST API specific user is used to for service calls to the Alfabet RESTful API. This user is not used for creating and updating objects on the Alfabet user interface. Therefore write access permission based for example on the assignment as authorized user or availability of open assignments for the object are not relevant in the context of the Alfabet RESTful API. To grant access to objects for the REST API specific user, you can for example assign the user to relevant authorized user groups or define a right rule to allow write access to objects to the specific user logged in with the specific user profile.

Required Implementation on Client Side

The RESTful service interface on client side must be implemented to send a request for authorization code prior to sending a data request. By default, the authorization code received in the response of the authorization request can be used in data request calls to the Alfabet RESTful API posted within 20 minutes after receiving the authorization code. The time period can be changed for the user in the **User** editor.

> ⓘ For more information about changing the validity period for the authorization code, see Required Configuration on Server Side.

The authorization call must be send to the following URL:

URLOfTheAlfabetWebApplication/api/token

> ⚠ The specification of the URL is case sensitive.

The method for the call is ***POST***.

The `Content-Type` field of the HTTP header must be defined as:

```
Content-Type: application/x-www-form-urlencoded
```

The payload of the request must be defined as:

```
grant_type=password&username=Alfabet&password=AFPUVOIM462LG3EOMY3A6G4T4WNATJ
I3
```

The answer is a JSON response with the following structure:

```
    {
    "token_type": "bearer"
    access_token":"adf93nfpornpor"
    "expires_in": 1200
    }
```

The `expires_in` field returns the time the token is valid in seconds.

The calls to the Alfabet RESTful API must be send to the following URL amended with the specification of the service call as described in the following chapters:

> *URLOfTheAlfabetWebApplication*/api/v2

⚠️ The specification of the URL is case sensitive.

The data in the response of the authorization call must be added as authorization key to the header of all requests sent to the Alfabet RESTful API in the following format:

> Authorization:*TypeValue*

with

Type = the value in the `token_type` field

Value = the value in the `access_token` field

👁️ For the answer above, the Authorization would be:

> Authorization: bearer adf93nfporesopor

The authorization key expires after approximately 20 minutes if not otherwise configured in the **User** editor for the user used for key generation. The client interface should be configured to request a new authorization key in regular intervals or prior to each request.

Overview of Access Permissions Required for Each Endpoint

The configuration of access permissions for the Alfabet RESTful services is a complex process to enable establishing of a maximum of security for each use case. To ease configuration for a specific RESTful service targeting the Alfabet RESTful API, the following table lists all available kinds of service calls and the access permissions that must be configured.

Functionality	Endpoint	API Access Options for the named user having the option Has API V2 Access enabled	Attribute to be activated in the relevant class settings of the object class	Additional permission concepts that apply
Exporting Information about the Complete Alfabet Class Model Including Enumerations and Culture Settings	`metamodel`	Has Meta Model Access		
Exporting Information about All or Multiple Classes of the Alfabet Class Model	`classes`	Has Meta Model Access		
Exporting Information about All or Multiple Enumerations in the Alfabet Class Model	`enums`	Has Meta Model Access		
Exporting Data About Objects with Defined REFSTR Values	`objects`	Has GetObjectsByRefs Access	Allow Read via Rest API	Mandate settings for user and object
Exporting Information About Objects Found By A Configured Report	`objects`	Has GetObjectsByReport Access		Mandate settings for user and object Access permission of user and execution permission for REST API for configured report
Exporting Data About Objects Of a Defined Object Class Matching A Filter Definition	`objects`	Has GetObjectsByFilter Access	Allow Read via Rest API	Mandate settings for user and object

Functionality	Endpoint	API Access Options for the named user having the option Has API V2 Access enabled	Attribute to be activated in the relevant class settings of the object class	Additional permission concepts that apply
[Deleting Objects from the Alfabet database](#)	`delete`	Has DeleteObjects Access	Allow Write via Rest API	Access permissions of user for object
[Archiving Objects from the Alfabet database](#)	archive	Has GetObjectsByRefs Access (for all calls) Has DeleteObjects Access (only if objects shall be deleted after archiving)	Allow Read via Rest API (for all calls) Allow Write via Rest API (only if objects shall be deleted after archiving)	Access permissions of user for object
[Creating a new Object in the Alfabet database](#)	`update`	Has CreateObjects Access	Allow Write via Rest API	Access permissions of user for object
[Changing the Properties of an existing Object in the Alfabet database](#)	`update`	Has UpdateObjects Access	Allow Write via Rest API	Access permissions of user for object
[Starting an ADIF Import via RESTful Service Call](#)	`adifimport`	Has ADIFAPIInvocation Access		Execution permission for REST API for ADIF import scheme For ADIF import from a file located in the **Internal Document Selector**: **Open Items** access permissions on the IDOC folder and file extension permission via the blacklist and whitelist

Functionality	Endpoint	API Access Options for the named user having the option Has API V2 Access enabled	Attribute to be activated in the relevant class settings of the object class	Additional permission concepts that apply
Starting an ADIF Export via REST-ful Service Call	`adifexport`	Has ADIFAPIInvocation Access		Execution permission for REST API for ADIF export scheme For ADIF export to a file located in the **Internal Document Selector**: **Manage Items** access permissions on the IDOC folder and file extension permission via the blacklist and white-list
Starting a Work-flow via RESTful Service Call	`workflow`	Has WorkflowAPIInvoca-tion Access		Execution permission for REST API for work-flow template
Regenerating the Password of an Alfabet User	`regenerate-password`	Has PasswordAPI Access Has UpdateObjects Access	Allow Read via Rest API for the object class `Person` Allow Write via Rest API for the object class `Person`	
Anonymizing User Data For Selected Users	`anony-mizeuser`	Has AnonymizeUser Access		Anonymization must be activated for the object class person.
Exporting Infor-mation about the Content of the Internal Docu-ment Selector	`idocfilelist`	Has IDOC File List Access		**List Items** access per-missions on the IDOC folder and file exten-sion permission via the blacklist and whitelist
Uploading Docu-ments to the	`idocupload`	Has IDOC Upload Access		**Manage Items** access permissions on the IDOC folder and file

Functionality	Endpoint	API Access Options for the named user having the option Has API V2 Access enabled	Attribute to be activated in the relevant class settings of the object class	Additional permission concepts that apply
Internal Document Selector				extension permission via the blacklist and whitelist
Downloading Documents from the Internal Document Selector	idocdownload	Has IDOC Download Access		**Open Items** access permissions on the IDOC folder and file extension permission via the blacklist and whitelist
Checking Whether the Alfabet components are Running	monitor	Has MonitoringAPI Access		
Updating the Meta-Model	updateMM	Has Update Meta-Model Access		

Chapter 5:   Configuring Handling of Date, Time and Number Formats For API Calls

By default, date and time formats have to be defined and are returned in the standard formats defined by your language settings.

The Alfabet Web Application can be configured both to return date and time information in other formats and even to accept other date and time formats in requests. Different sets of date and time format can be configured for concurrent use by different applications that all send requests to the Alfabet RESTful API and use a different date/time format.

> ⚠ Please note however, that for requests to the endpoint `objects` that request results from a configured report with parameters, the parameter settings in the **ReportArgs** field must be defined in the standard formats specified above independent from the date/time format specified for the request.

To define a date and time format for the Alfabet RESTful API, an **API Culture** must be defined for the Alfabet solution with the tool Alfabet Expand. The **API Culture** must then be referenced in the service call to use the defined date, date time and time formats of that API Culture:

* [Configuring a New API Culture](#)

* [Using an API Culture in a Service Call](#)

Configuring a New API Culture

Date and Time definition for the Alfabet RESTful API service calls are defined in the Meta-Model explorer of the tool Alfabet Expand.

> ℹ For information about how to access and work with Alfabet Expand, see the reference manual *Configuring Alfabet with Alfabet Expand*.

> ℹ Please consider the following syntax for the specification of date and time formats.
>
> * To specify the 24-hour clock system, use "H" or "HH" for hours.
>
> * To specify the 12-hour clock system, use "h" or "hh". This setting requires additional specification of AM or PM when a time is written.
>
> * Use "M" for month and "m" for minute.

1) In Alfabet Expand, go to the Meta-Model tab.

2) Right click the **API Cultures** node in the explorer and select **Add New API Culture**. A new API Culture element is added to the explorer and the attributes of the new elements are displayed in the attribute window.

3) In the attribute **API Culture Name**, change the default name of the API Culture element to a unique and meaningful name. The name of the API Culture element is used to reference the API Culture in service calls to the Alfabet RESTful API.

4) Select one of the following from the dropdown list in the **Date Format** field to define date and date time formats:

- `Default`: The default date pattern yyyy-MM-dd and date time pattern yyyy-MM-dd HH:mm:ss.fff is used.

- `Pattern`: A customer defined pattern is used. If you select this option, two new attributes **Date Pattern** and **Date/Time Pattern** are displayed. Write the required date pattern for date only values into the **Date Pattern** attribute and the required date time pattern into the **Date/Time Pattern** attribute.

- `Posix`: Dates and date time values are accepted and returned in posix format.

- `Windows`: The Windows date pattern MM/dd/yyyy and date time pattern MM/dd/yyyy HH:mm is used.

5) Select one of the following from the dropdown list in the **Number Format** field to define number formats:

- `Default`: The default number format is used. The decimal symbol is a dot, the number of decimals is not limited and the group symbol is a comma (e.g. 3,125.987).

- `Custom`: A customer defined pattern is used. If you select this option, three new attributes **Number Grouping Symbol**, **Number of Decimal Digits** and **Number Decimal Symbol** are displayed. Define the required number format in the new fields. If a field is undefined, the default is used for this setting.

6) Select one of the following from the dropdown list in the **Time Format** field to define time formats:

- `Default`: The default time pattern HH:mm:ss is used.

- `Pattern`: A customer defined pattern is used. If you select this option, a new attribute **Time Pattern** is displayed. Write the required time pattern into the field.

7) In the toolbar, click the **Save** button to save your changes.

Using an API Culture in a Service Call

API Cultures defined for the Alfabet solution can be used in service calls to the endpoints `objects`, `delete` and `update`. To use an API Culture in a service call, a field `ApiCulture` must be added to the JSON object in the payload of the call. The field value must specify the name of a defined API Culture.

For example the following request for data about objects of the defined REFSTR values returns results with dates defined in Posix format:

```
{
    "ApiCulture": "ApiCulture_Posix",
    "EmptyValues": true,
    "Refs": ["76-3472-0","76-3473-0","76-3474-0"] }
}
```

This will result in a return value with Posix date formats:

```
{
    "Objects": [
        {
        "ClassName": "Application",
        "RefStr": "76-3473-0",
        "Values": {
            "applicationgroups": null,
            "baseapplicationid": null,
            "color": null,
            "creation_date": " 1504696528 ",
            "creation_user": "ALFABET", "description": null,
            ...
            }
        ...
        }
    ]
}
```

Chapter 6: Service Calls and Return Values

Service calls are sending the information about what action to perform in the Alfabet database or what information to send back to the URL of the Alfabet RESTful API of an Alfabet Web Application. The endpoints that are available to access and to change data are predefined and require a specific set of parameters to be sent to the Alfabet RESTful API.

To make a service call, a valid URL consisting of the address of the Alfabet RESTful API and the specification of the request endpoint must be sent to the web server via an HTTP client. The header of the request must contain a valid authorization as described in the chapter above. For some endpoints, the parameters required for execution of the request must be submitted in JSON format in the payload of the request.

A valid URL has the following structure:

> *ServerAdress*/api/v2/*EndpointName*/?*Parameters*

with:

Variable in URL	Required Value
Server-Adress	The web address of the Alfabet Web Application. The specification of the web address is case sensitive.
api/v2	The specification of the Alfabet RESTful API. The current version is api/v2.
End-point-Name	The endpoint to be called. Allowed values are: • metamodel for calls returning information about all classes and cultures in the Alfabet meta-model. For more information see Exporting Information about the Complete Alfabet Class Model Including Enumerations and Culture Settings. • classes for calls returning information about all or a subset of the classes in the Alfabet meta-model. • enums for calls returning information about all or a subset of enumerations defined in the Alfabet meta-model • objects for calls returning property values of one or multiple defined object in the Alfabet database or the result of a query executed via an Alfabet configured report. • delete for deleting objects and relations from the Alfabet database. • archiveobject for exporting information about selected objects in the Alfabet database in a ZIP file containing the most relevant page views for the objects in HTML format. The endpoint can optionally delete the objects from the Alfabet database after archiving. • update for updating data in the Alfabet database and to create new objects and relations.

Varia-ble in URL	Required Value
	• `regeneratepassword` to regenerate the password of selected users in the Alfabet database.
	• `anonymizeuser` for anonymizing the data of one or multiple selected users in the Alfabet database.
	• `workflow` to start a workflow based on an existing workflow template in the Alfabet database.
	• `adifimport` to start an ADIF job based on an ADIF import scheme in the Alfabet database.
	• `adifexport` to start an ADIF job based on an ADIF export scheme in the Alfabet database.
	• `monitor` to check whether the Alfabet components that might be involved in the execution of a RESTful service request are running and can be accessed.
Param-eters	The specification of parameters only applies to the methods `metamodel`, `classes` and `enums`. A parameter is defined as: *ParameterName=ParameterValue* If multiple parameters are defined, they are concatenated with an & character: *ParameterName1=ParameterValue1&ParameterName2=ParameterValue2*

For the requests of all endpoints except `metamodel`, `classes` and `enums`, the parameters for the request are defined in the payload of the request in JSON format. Details about the calls are given for each request in the following chapters.

Each endpoint provides data in JSON format. If the response includes data of the type date, the dates will by default be strings of the format yyyy-mm-dd. The date format can be changed with a parameter setting in the payload of the request. If the response includes object class property names, these are returned in lower case letters.

The endpoints are described in detail in the following sections, including information about the HTTP method used and the structure of the service call and the return value:

- [Exporting Information about the Complete Alfabet Class Model Including Enumerations and Culture Settings](#)

- [Exporting Information about All or Multiple Classes of the Alfabet Class Model](#)

- [Exporting Information about All or Multiple Enumerations in the Alfabet Class Model](#)

- [Exporting Information About Object Data Stored in the Alfabet database](#)

  - [Exporting Data About Objects with Defined REFSTR Values](#)

  - [Exporting Data About Objects Of a Defined Object Class Matching A Filter Definition](#)

Exporting Information about the Complete Alfabet Class Model Including Enumerations and Culture Settings

The endpoint `metamodel` returns information about the definition of all Alfabet object classes, the enumeration and the culture settings in the meta-model.

> ⓘ For detailed information about the structure of the Alfabet meta-model and how data is stored in the Alfabet database tables, see the chapter *The Alfabet Meta-Model in the Alfabet Database* in the reference manual *Alfabet Data Integration Framework* or the reference manual *Alfabet Meta-Model*.

Endpoint name: `metamodel`

HTTP method: `GET`

Service call:

> *ServerAdress*/api/v2/metamodel/?emptyValues=true

with the following parameters:

Parameter	Mandatory/Optional	Required Value
`emptyValues=true/false`	Optional	If `emptyValues=true` is added to the service call, all relevant attributes of the object classes, enumerations and culture settings are returned even if they are not set. If `emptyValues=false` is added, only attributes that are set are returned. By default the attribute is set to `false`.

Header Fields:

> `Authorization:`*TypeValue*

> ⓘ For information about the required Authorization header content, see Authorization.

Return value:

The return value is a JSON object with four fields:

- `alias`: The name of the server alias of the Alfabet Web Application answering the request.

- `cultures`: A JSON list of cultures defined in the Alfabet meta-model.

- `classes`: A JSON list of classes defined in the Alfabet meta-model.

- `enums`: A JSON list of enumerations defined in the Alfabet meta-model.

> 👁 ```
>   {
>   "alias": "Alfabet"
> ```

```
-"cultures": [...]

-"classes": [...]

-"enums": [...]

}
```

The fields `cultures`, `classes` and `enums` all consist of a JSON list of objects. Each object class, enumeration and culture setting in the Alfabet meta-model is an object in the respective JSON list. The attributes of the meta-model object are fields of the JSON object with a name corresponding to the name of the attribute. Attributes specify the technical data about the meta-model object that is used to build the database tables and to process the object data within the Alfabet solution. The list of attributes is limited to the attributes that may be relevant for external applications. For example the date of the last update or tags defined for the object are not exported.

Subordinate objects, like for example enumeration items for enumerations are listed as a field that contains the subordinate objects as a JSON list of objects.

The following information is provided:

Object Type in the Alfabet meta-model	Information provided in the response	Example

Culture Settings	Each culture setting in the Alfabet meta-model is a JSON object in the JSON list of the field `cultures`.
	
	 Each relevant attribute defined for the culture setting in the Alfabet meta-model is defined as a field in the JSON object. The field names are similar to the names of the attributes.
		```"cultures": [7]```
		 ``` 0: {```
		 ``` "BaseCulture": 1031```
		 ``` "DateFormatCulture": 1031```
		 ``` "NumberFormatCulture": 1031```
		 ``` "HelpCulture": 1031```
		 ``` "IsDefault": false```
		 ``` "IsInstanceTranslation": true```
		 ``` "Picture": "Flag_German"```
		 ``` "TimePattern": "HH:mm:ss"```
		 ``` "MeasurementUnit": "Metric"```
		 ``` }```
		 ``` ...```

Object Classes	Each object class in the Alfabet meta-model that is visible in the **Classes** explorer of the **Meta-model** tab of the configuration tool Alfabet Expand is a JSON object in the	```"classes": [266]```
		 ``` 0: {```
		 ``` "Name": "Domain"```
		 ``` "Id": 359```
		 ``` "Comment": ""```

Object Type in the Alfabet meta-model	Information provided in the response	Example

	JSON list of the field `classes`.
	
	Information about object classes includes the following:
	
	• **Attributes of the object class**
	
	The relevant attributes defined for the object class in the Alfabet meta-model are each defined as a field in the JSON object. The field names are similar to the names of the attributes.
	
	Please note that object classes having the attribute **Automatically Managed** set to `true` are object classes that shall only be changed by mechanisms triggered by the Alfabet software components and must not be changed by any third party component.
	
	• **Object class properties**
	
	For each Alfabet object class a database table exists in the Alfabet database. Each object class property corresponds to a column in the database table, that means that the object class properties store the information about individual objects of the class. All object class properties that are visible in the **Classes** explorer of the **Metamodel** tab of the configuration tool Alfabet Expand are included into the field `Properties` as a JSON
		`"Caption": "Domain"`
		`"AutomaticallyManaged": false`
		`"TechName": "DOMAIN"`
		`"Audit": true`
		`"HasMandates": true`
		`"Hint": ""`
		`"IDPrefix": "DOM"`
		`"Stereotypes":[5]`
		    `0: {`
		    `"Name": "Area"`
		    `"Caption": "Area"`
		    `"CaptionPlural":"Areas"`
		    `"comment": ""`
		    `"HasMandates": false`
		    `}`
		    `...`
		`"Properties": [31]`
		    `0: {`
		    `"Name": "ID"`
		    `"Guid":`
		    `"6736AEFBF52C416C94D161FAABAF3D7F"`
		    `"TechName": "ID"`
		    `"Caption": "ID"`
		    `"Comment": ""`
		    `"Hint": ""`
		    `"Alias": ""`
		    `"Type": "String"`
		    `"AutomaticallyManaged": false`
		    `"DefaultValue": null`
		    `"Validator": ""`
		    `"EnumInfo": ""`
		    `}`
		    `...`

Object Type in the Alfabet meta-model	Information provided in the response	Example

	list of object, each JSON object representing a property and the fields of the JSON object representing a subset of the attributes of the property that might be relevant for external processes.
	
	Please note that object class properties having the attribute **Automatically Managed** set to `true` are properties that shall only be changed by the Alfabet software components and must not be changed by any third party component.
	
	• **Object class stereotypes**
	
	If stereotypes are configured for the object class, the stereotypes are included as a JSON list of objects in the field `Stereotypes` of the class object. Each attribute of the stereotype is exported as a field in the JSON object.
	
	• **Class Keys**
	
	A class key with an attribute **Unique** set to `True` specifies one or a combination of object class properties that must be unique for that class. If the data does not fulfill the requirements specified in the class key definition, the object cannot be created in the database. The Alfabet meta-model also allows class keys to be specified that do not require uniqueness. In this case, the
		``
		"Keys": [3]
		    0: {
		    "Name": "Domain_Key1"
		    "TechName": "C359_K1"
		    "Content": "BelongsTo"
		    "Unique": false
		    "IsCaseIns": false
		    "Descending": false
		    "IsActivated": true }
		    ...
		}
		``

Object Type in the Alfabet meta-model	Information provided in the response	Example
	class key attribute **Unique** will be set to `False`. The purpose of such a class key is to speed up the search functionality by creating an index for each class key. In the return value, each class key is an object and listed in an array of objects in the field `"unique keys"` in the object class object.	
| Enu-mera-tions | Each enumeration in the Alfabet meta-model is a JSON object in the JSON list of the field enums.<br><br>Information about enu-meration includes the fol-lowing:<br><br>• **Attributes of the enumeration**<br><br>Each relevant attribute defined for the enumera-tion in the Alfabet meta-model is defined as a field in the JSON object. The field names are similar to the names of the attrib-utes.<br><br>• **Enumeration items and their attributes**<br><br>The enumeration items defined for the enumera-tion are listed in the field Items. The field Items provides a JSON list of objects. Each enumera-tion item is an object in that list and the fields of the object return the rele-vant attributes of the enumeration item. The | ```
"enums": [90]
    0: {
        "Name": "AlfaDocCategory"
        "Guid":
        "6E7E549625034B788C5725F2541BD412"
        "Comment": ""
        "Hint": ""
        "HelpFile":
        "Enum_AlfaDocCategory.html"
        "Items": [7]
            0: {
            "Value": ""
            "Comment": ""
            "Hint": ""
            }
            1: {
            "Value": "Manual"
            "Comment": ""
            "Hint": ""
            }
            ...
    }
``` |

Object Type in the Alfabet meta-model	Information provided in the response	Example
	field names are similar to the names of the attributes.	

Exporting Information about All or Multiple Classes of the Alfabet Class Model

The endpoint `classes` returns information about the definition of all or a subset of the Alfabet object classes in the Alfabet meta-model.

> ℹ️ For detailed information about the structure of the Alfabet meta-model and how data is stored in the Alfabet database tables, see the chapter *The Alfabet Meta-Model in the Alfabet Database* in the reference manual *Alfabet Data Integration Framework* or the reference manual *Alfabet Meta-Model*.

Endpoint name: `classes`

HTTP method: `GET`

Service call:

> *ServerAdress*/api/v2/classes/?names=Domain,Application&emptyValues=true

with the following parameters:

Parameter	Mandatory/Optional	Required Value
`names= ClassName, ClassName`	Optional	A comma separated list of object classes for that information shall be returned. The object class must be defined by the value of the **Name** attribute of the object class. If the parameter is not added to the call, information about all object classes in the Alfabet meta-model is returned.
`emptyValues=true/false`	Optional	If `emptyValues=true` is added to the service call, all relevant attributes of the object classes are returned even if they are not set. If `emptyValues=false` is added, only attributes that are set are returned. By default the attribute is set to `false`.

Header Fields:

```
Authorization:TypeValue
```

For information about the required Authorization header content, see Authorization.

Return value:

The return value is a JSON object with two fields:

- `alias`: The name of the server alias of the Alfabet Web Application answering the request.

- `classes`: A JSON list of classes defined in the Alfabet meta-model.

```
{
    "alias": "Alfabet"
    -"classes": [...]
}
```

The field classes consists of a JSON list of objects. Each object class in the Alfabet meta-model is an object in the JSON list. The attributes of the meta-model object class are fields of the respective JSON object. The name of the field corresponds to the name of the attribute. Attributes specify the technical data about the meta-model object that is used to build the database tables and to process the object data within the Alfabet solution. The list of attributes is limited to the attributes that may be relevant for external applications. For example the date of the last update or tags defined for the object are not exported.

Subordinate objects, like for example stereotypes defined for the class are listed as a field that contains the subordinate objects as a JSON list of objects.

The following information is provided:

If the parameter `names` is not specified in the request, each object class in the Alfabet meta-model that is visible in the **Classes** explorer of the **Metamodel** tab of the configuration tool Alfabet Expand is a JSON object in the JSON list of the field `classes`. The classes are listed in alphabetical order.

If the parameter `names` is specified in the request, the list of classes contains only the classes defined in the `names` parameter in the order specified in the `names` parameter.

Information about object classes includes the following:

- **Attributes of the object class**

```
"classes": [266]
    0: {
        "Name": "Domain"
        "Id": 359
        "Comment": ""
        "Caption": "Domain"
        "AutomaticallyManaged": false
        "TechName": "DOMAIN"
        "Audit": true
        "HasMandates": true
        "Hint": ""
        "IDPrefix": "DOM"
        "Stereotypes":[5]
            0: {
            "Name": "Area"
            "Caption": "Area"
```

The relevant attributes defined for the object class in the Alfabet meta-model are each defined as a field in the JSON object. The field names are similar to the names of the attributes.

Please note that object classes having the attribute **Automatically Managed** set to `true` are object classes that shall only be changed by mechanisms triggered by the Alfabet software components and must not be changed by any third party component.

- **Object class properties**

For each Alfabet object class a database table exists in the Alfabet database. Each object class property corresponds to a column in the database table, that means that the object class properties store the information about individual objects of the class. All object class properties that are visible in the **Classes** explorer of the **Metamodel** tab of the configuration tool Alfabet Expand are included into the field `Properties` as a JSON list of object, each JSON object representing a property and the fields of the JSON object representing a subset of the attributes of the property that might be relevant for external processes.

Please note that object class properties having the attribute **Automatically Managed** set to `true` are properties that shall only be changed by the Alfabet software components and must not be changed by any third party component.

- **Object class stereotypes**

If stereotypes are configured for the object class, the stereotypes are included as a JSON list of objects in the field `Stereotypes` of the class object. Each attribute of the

```
    "CaptionPlural":"Areas"
    "comment": ""
    "HasMandates": false
    }
    ...
"Properties": [31]
    0: {
    "Name": "ID"
    "Guid":
    "6736AEFBF52C416C94D161FAABAF3D7F"
    "TechName": "ID"
    "Caption": "ID"
    "Comment": ""
    "Hint": ""
    "Alias": ""
    "Type": "String"
    "AutomaticallyManaged": false
    "DefaultValue": null
    "Validator": ""
    "EnumInfo": ""
    }
    ...
"Keys": [3]
    0: {
    "Name": "Domain_Key1"
    "TechName": "C359_K1"
    "Content": "BelongsTo"
    "Unique": false
    "IsCaseIns": false
    "Descending": false
    "IsActivated": true }
    ...
}
```

stereotype is exported as a field in the JSON object.

- **Class Keys**

A class key with an attribute **Unique** set to `True` specifies one or a combination of object class properties that must be unique for that class. If the data does not fulfill the requirements specified in the class key definition, the object cannot be created in the database. The Alfabet meta-model also allows class keys to be specified that do not require uniqueness. In this case, the class key attribute **Unique** will be set to `False`. The purpose of such a class key is to speed up the search functionality by creating an index for each class key. In the return value, each class key is an object and listed in an array of objects in the field `"unique keys"` in the object class object.

Exporting Information about All or Multiple Enumerations in the Alfabet Class Model

The endpoint `enums` returns information about the definition of all or a subset of the enumerations in the Alfabet meta-model. Enumerations define a predefined set of values that are allowed to be set for an object class property.

> ⓘ For detailed information about the structure of the Alfabet meta-model and how data is stored in the Alfabet database tables, see the chapter *The Alfabet Meta-Model in the Alfabet Database* in the reference manual *Alfabet Data Integration Framework* or the reference manual *Alfabet Meta-Model*.

Endpoint name: `enums`

HTTP method: `GET`

Service call:

> *ServerAdress*`/api/v2/enums?names=Enum1,Enum2&emptyValues=true`

with the following parameters:

Parameter	Manda-tory/Op-tional	Required Value
names= EnumName, EnumName	Optional	A comma separated list of enumeration for that infor-mation shall be returned. The enumeration must be de-fined by the value of the **Name** attribute of the enu-meration. If the parameter is not added to the call, in-formation about all enumerations in the Alfabet meta-model is returned.
emptyValues=true/false	Optional	If emptyValues=true is added to the service call, all relevant attributes of the enumeration and enumera-tion items are returned even if they are not set. If emp-tyValues=false is added, only attributes that are set are returned. By default the attribute is set to false.

Header Fields:

```
Authorization:TypeValue
```

ⓘ   For information about the required Authorization header content, see Authorization.

Return value:

The return value is a JSON object with two fields:

- alias: The name of the server alias of the Alfabet Web Application answering the request.

- classes: A JSON list of classes defined in the Alfabet meta-model.

```
{
    "alias": "Alfabet"
    -"enum": [...]
}
```

The field classes consists of a JSON list of objects. Each enumeration in the Alfabet meta-model is an ob-ject in the JSON list. The attributes of the enumeration are fields of the respective JSON object. The name of the field corresponds to the name of the attribute. Attributes specify the technical data about the meta-model object that is used to build the database tables and to process the object data within the Alfabet so-lution. The list of attributes is limited to the attributes that may be relevant for external applications. For example the date of the last update or tags defined for the object are not exported.

Subordinate objects, like for example enumeration items defined for the enumeration are listed as a field that contains the subordinate objects as a JSON list of objects.

The following information is provided:

If the parameter names is not specified in the request, each enumeration in the

```
"enums": [90]
```

Alfabet meta-model that is visible in the **Classes** explorer of the **Metamodel** tab of the configuration tool Alfabet Expand is a JSON object in the JSON list of the field `enums`. The enumerations are listed in alphabetical order.

If the parameter `names` is specified in the request, the list of enumerations contains only the enumerations defined in the `names` parameter in the order specified in the `names` parameter.

Information about object classes includes the following:

- **Attributes of the enumeration**

Each relevant attribute defined for the enumeration in the Alfabet meta-model is defined as a field in the JSON object. The field names are similar to the names of the attributes.

- **Enumeration items and their attributes**

The enumeration items defined for the enumeration are listed in the field Items. The field Items provides a JSON list of objects. Each enumeration item is an object in that list and the fields of the object return the relevant attributes of the enumeration item. The field names are similar to the names of the attributes.

```
0: {
    "Name": "AlfaDocCategory"
    "Guid":
    "6E7E549625034B788C5725F2541BD412"
    "Comment": ""
    "Hint": ""
    "HelpFile":
    "Enum_AlfaDocCategory.html"
    "Items": [7]
        0: {
        "Value": ""
        "Comment": ""
        "Hint": ""
        }
        1: {
        "Value": "Manual"
        "Comment": ""
        "Hint": ""
        }
        ...
}
```

Exporting Information About Object Data Stored in the Alfabet database

The endpoint `objects` returns the information stored about one or multiple objects in a database table of the Alfabet database. There are three methods to select the objects for that information is returned by this endpoint:

- **Objects By References**

This method requires that the value of the `REFSTR` property of the object is known. The `REFSTR` is a unique identifier for objects in the Alfabet database. It has to be defined in the payload of the request to export all property values set for the object.

- **Objects By Filter**

The method returns all property values set for all objects of a specified object class matching search conditions that are defined in the service call. The search filter in the service call can only search in object class properties returning a text, that means object class properties of the type `Text` or `String`.

- **Objects By Report**

The payload of the request can point to a configured report of the type `NativeSQL` or `Query` in the Alfabet database. The return value includes all information included in the dataset of the report or a subset thereof.

The following information is available:

- [Exporting Data About Objects with Defined REFSTR Values](#)

- [Exporting Data About Objects Of a Defined Object Class Matching A Filter Definition](#)

- [Exporting Information About Objects Found By A Configured Report](#)

    - [Configuring a Report that Can be Called in a RESTful Service Request](#)

    - [Service Call for Direct Execution of a Configured Report](#)

    - [Service Calls for Execution of Offline Executed Reports](#)

        - [Service Call for Triggering Offline Execution](#)

        - [Checking the Status of the Asynchronous Report Execution](#)

        - [Exporting Results from Asynchronous Report Execution](#)

    - [Return Value for the ReportResultDataSet Asynchronous Report Execution Returning the Data Defined In the Report](#)

    - [Return Value for the ReportResultObjects Returning All Properties of the Objects Found in the Report](#)

    - [Return Value for the ReportResult Returning the REFSTR of the Objects Found in the Report](#)

Exporting Data About Objects with Defined REFSTR Values

Endpoint name: `objects`

HTTP method: `POST`

Service call:

> *ServerAdress*`/api/v2/objects`

Header Fields:

> `Authorization:`*TypeValue*

> For information about the required Authorization header content, see [Authorization](#).

> `Content-Type: application/json; charset=utf-8`

Payload

The payload is a JSON object with the following structure:

```
{
"CurrentProfile":"UserProfile1",
"CurrentMandate":"Mandate1",
"ApiCulture": "ApiCulture_Posix",
"Language": 1033,
"DataCulture": "de-DE",
"EmptyValues": true,
"Refs": ["76-2518-0","76-3246-0","405-3-0"]
}
```

It may have the following fields:

Field	Manda-tory/Op-tional	Required Value
`"Refs":[" RefstrOfOb-ject "," RefstrOfOb-ject "]`	Mandatory	An array containing one or multiple values of the `REFSTR` property of objects in the Alfabet database. The response will include information about the objects with the objects specified by their `REFSTR`.
`"Emp-tyValues":true/false`	Optional	If `EmptyValues` is set to true, all object class properties of the object are returned even if they are not set. If `Emp-tyValues` is set to false, only object class properties of the object that are set are returned. By default the attrib-ute is set to false.
`"CurrentPro-file":" UserProfile-Name "`	Mandatory	The name of a user profile assigned to the Alfabet user sending the call that shall be used to access Alfabet. The user profile is required to evaluate whether the user that is sending the service call is allowed to read data about an object. For details about the access permissions that de-pend on the user profile, see the section Authorization.
`"CurrentMandate":" Man-dateName "`	Optional	This field is only relevant if access to objects is controlled by the mandate concept for federated architectures im-plemented in Alfabet. The field must specify the name of a mandate assigned to the Alfabet user sending the call. For details about the consideration of mandate settings in REST API service calls, see the section Controlling Access Via Mandates.
`"Language": "Lan-guageLCID "`	Optional	This field is only relevant if instance translation is used in your Alfabet application. The field has to specify the lan-guage code (LCID) decimal of the culture for that transla-tions are available for the name and description of the

Field	Manda- tory/Op- tional	Required Value
		object in the Alfabet database. By default the standard language of the database is returned.
		Please note that no values are returned when an instance translation is not available for an object class property, even if a value is provided in the standard language.
		To return all values in the original values and add transla- tions in a second language, the parameter `DataCulture` shall be used instead of the parameter `Language`. `Language` supersedes `DataCulture`. If both fields are availa- ble, only `Language` is used.
`"DataCulture":" Lan- guageCode "`	Optional	This field is only relevant if instance translation is used in your Alfabet application. The field has to specify the lan- guage code of the culture for that translations are availa- ble for the name and description of the object in the Alfabet database (for example `de-DE` for German), or `ALL` to return translations for all languages that provide trans- lation. Translations are added to the return value in addi- tion to the values in the original language in a separate field `Translations`.
		The field `Language` supersedes `DataCulture`. If both fields are available, only `Language` is used and `DataC- ulture` is ignored.
`"ApiCulture":"APICul- tureName"`	Optional	This field specifies an API Culture that was defined in Alfabet Expand as part of the configuration of the Alfabet meta-model. The API Culture allows to define data, date/time and time patterns as well as number formats that deviate from the Alfabet standard. If this field is added to the request, dates, times and numbers are writ- ten into the return value in the formats specified in the API Culture definition. For more information, see the section Configuring Handling of Date, Time and Number Formats For API Calls.
`"SortOrder": " Ob- jectClassPropertyName "`		Define the name of an object class property to sort results by this property alpha-numerical in ascending order in the return value.
		By default objects are sorted by ascending `REFSTR`.
		If objects do not have a value for the selected object class property, these objects are sorted by ascending `REFSTR` and listed first.
		If either `Language` or `DataCulture` is set, and data trans- lation is enabled for the object class property, the values are sorted by translated value. If no translation is provided

Field	Manda-tory/Op-tional	Required Value
		for some of the object class property values, these objects are sorted by ascending REFSTR and listed first.

Return value:

The return value is a JSON object with two fields:

- Count: The number of objects found in the database for that data is returned. The count may differ from the number of defined REFSTR values in the request: if an object is not found because the REFSTR defined in the request does not exist in the Alfabet database, the object is not included in the count and in the returned object data.

- Objects: A JSON list of objects, each object representing a database object for that data is returned.

```
{
    "Objects": [...]
    "Count": 3
}
```

The field Objects contains one JSON object per database object found for the call. The object contains the following fields:

- ClassName: The name of the Alfabet object class the object belongs to.

- RefStr: The value of the REFSTR of the object.

- Values: A JSON object that includes all information defined about the object in the database via object class properties. Each property is a field with a name corresponding to the property name. The value of the field informs about the value of the property. The return value for a property depends on the value type of the property:

Property Type	Return value
String, Text	The string, text is returned as string (in inverted commas): `"description":"This application manages customer relations."`
StringAr-ray	All selected options of the string array are returned in one string: `"subcategories": "APP_SecurityAssessment APP_CloseSubworkflow"`

Property Type	Return value
Boolean	Boolean values are 1 for true, 0 for false or `null` if the property is not set: `"variant": 1`
Real, Integer	Real and integer values are returned as number: `"xpos": 485.4`
Date, DateTime	Date and date time information is both returned as date strings of the format yyyy-mm-dd: `"last_update": "2012-02-21"`
URL	URLs are returned as string (in inverted commas): `"url": "http://company.com/ReportServer"`
Reference	The `REFSTR` of the object the reference is targeting is returned: `"ictobject": "26-608-0"`

ReferenceArray	For reference arrays, the `REFSTR`s of all objects that the current object references with this property are listed in an array:
	
	`"applicationgroups": [3]`
	`    0: "95-38-0"`
	`    1: "95-8-0"`
	`    2: "95-9-0"`
	
	The return value is independent from the setting of the attribute **RefSupport** of the property. If the attribute **RefSupport** of the property is set to `false`, the `REFSTR` value of the reference targets are directly stored in the database table of the object class in a column for the property. If the attribute **RefSupport** is set to `true`, the references are stored in the `RELATIONS` table. This difference is not visible in the return value for the `objects` endpoint.
	
	If information about the objects referenced by the current object shall be included into the return value, you can use the method to find objects via a configured report that is also available for the endpoint `objects`.

- `Translations`: This field is only available if the field `DataCulture` is defined in the payload of the service call. The field contains a JSON list of object with one object per data culture. Each object has a field `DataCulture` that returns the language for that translations are returned and one field `Values` that is a JSON object with a field for each translated property. The value of the field informs about the translated value of the property.

- `GenericAttributes`: If attributes are defined for the object via the object class `GenericAttribute`, all generic attributes with the object class property `Owner` set to the current object are listed in this field as a JSON list of objects. Each object in the list returns one object of the class `GenericAttribute`, with the fields `Refstr`, `Values` and `Translations` that return data about the generic attribute in the same format as returned for the main object.

Exporting Data About Objects Of a Defined Object Class Matching A Filter Definition

Endpoint name: `objects`

HTTP method: `POST`

Service call:

> *ServerAdress*`/api/v2/objects`

Header Fields:

> `Authorization:`*TypeValue*

> (i) For information about the required Authorization header content, see Authorization.

> `Content-Type: application/json; charset=utf-8`

Payload

The payload is a JSON object with the following structure:

```
{
"CurrentProfile":"UserProfile1",
"CurrentMandate":"Mandate1",
"ApiCulture": "ApiCulture_Posix",
"Language": 1033,
"DataCulture": "de-DE",
"EmptyValues": true,
"Class": "Application",
"Limit":"80"
"FilterTextProperties": [
    {"Name":"Content*","ShortName":"CMS*"},
    {"Name":"Document*"}
    ]
}
```

It may have the following fields:

Field	Mandatory/Optional	Required Value
`"Class":"ObjectClassName"`	Mandatory	The value of the Name attribute of the Alfabet object class for that object data shall be returned. Only one object class can be defined for a service call.
`"FilterTextProperties":[ {"PropertyName":"FilterValue","PropertyName":"FilterValue"}, {{"PropertyName":"FilterValue","PropertyName":"FilterValue"}]`	Mandatory	An array containing the filter definition. The array must contain at least one JSON object with one field. Each field defines a filter condition with the syntax: `"ObjectClassPropertyName":"SearchCondition"` Please note the following about the filter definitions: • The search filter in the service call may only include object class properties returning a text, that means object class properties of the type `Text` or `String`. • Asterisks can be used as wildcard in the search strings. • If the search string is empty, the search returns only objects for that this object class property is not set. • Each JSON object in the filter array can have multiple fields. The fields are evaluated with an AND condition. For example the filter `{"Name":"Content*","ShortName":"CMS*"}` finds objects with a name starting with `Content` AND a short name starting with `CMS`. • The filter array can include multiple JSON objects. The filter definitions in different JSON objects are evaluated with an OR condition. For example the filter `{"Name":"Content*"},{"ShortName":"CMS*"}` finds objects with a name starting with `Content` OR a short name starting with `CMS`.
`"Limit":Number`	Optional	This field defines the maximum number of objects for that the response of the call shall return data. This value may be set to limit data return for filter definitions returning a high number of objects. In combination with the field `Offset` this field can be used to fetch the result of a big search in multiple steps. If `Limit` is not set, a maximum number of 1000 result data sets of the configured report are returned.
`"Offset":Number`	Optional	This field defines the start position for returning values in the result data set of the configured report. For example if the `Limit` field is set to 20 and the `Offset` is set to 10, the results listed in row 11 to 30 in the tabular output of the configured

Field	Mandatory/Optional	Required Value
		report are returned in the service call. By default, the start position is the first row in the data set with the number 0.
`"Emp-tyValues":true/false`	Optional	If `EmptyValues` is set to true, all object class properties of the object are returned even if they are not set. If `EmptyValues` is set to false, only object class properties of the object that are set are returned. By default the attribute is set to false.
`"CurrentPro-file":" UserProfile-Name "`	Mandatory	The name of a user profile assigned to the Alfabet user sending the call that shall be used to access Alfabet. The user profile is required to evaluate whether the user that is sending the service call is allowed to read data about an object. For details about the access permissions that depend on the user profile, see the section Authorization.
`"CurrentMan-date":" Man-dateName "`	Optional	This field is only relevant if access to objects is controlled by the mandate concept for federated architectures implemented in Alfabet. The field must specify the name of a mandate assigned to the Alfabet user sending the call. For details about the consideration of mandate settings in REST API service calls, see the section Controlling Access Via Mandates.
`"Language": "Lan-guageLCID "`	Optional	This field is only relevant if instance translation is used in your Alfabet application. The field has to specify the language code (LCID) decimal of the culture for that translations are available for the name and description of the object in the Alfabet database. By default the standard language of the database is returned. Please note that no values are returned when an instance translation is not available for an object class property, even if a value is provided in the standard language. To return all values in the original values and add translations in a second language, the parameter `DataCulture` shall be used instead of the parameter `Language`. `Language` supersedes `DataCulture`. If both fields are available, only `Language` is used.
`"DataCulture":" Lan-guageCode "`	Optional	This field is only relevant if instance translation is used in your Alfabet application. The field has to specify the language code of the culture for that translations are available for the name and description of the object in the Alfabet database (for example `de-DE` for German), or `ALL` to return translations for all languages that provide translation. Translations are added to the return value in addition to the values in the original language in a separate field `Translations`.

Field	Manda-tory/Op-tional	Required Value
		The field `Language` supersedes `DataCulture`. If both fields are available, only `Language` is used and `DataCulture` is ignored.
`"ApiCulture":"API-CultureName"`	Optional	This field specifies an API Culture that was defined in Alfabet Expand as part of the configuration of the Alfabet meta-model. The API Culture allows to define data, date/time and time patterns as well as number formats that deviate from the Alfabet standard. If this field is added to the request, dates, times and numbers are written into the return value in the formats specified in the API Culture definition. For more information, see the section Configuring Handling of Date, Time and Number Formats For API Calls.
`"SortOrder": " ObjectClassPropertyName "`		Define the name of an object class property to sort results by this property alpha-numerical in ascending order in the return value. By default objects are sorted by ascending `REFSTR`. If objects do not have a value for the selected object class property, these objects are sorted by ascending `REFSTR` and listed first. If either `Language` or `DataCulture` is set, and data translation is enabled for the object class property, the values are sorted by translated value. If no translation is provided for some of the object class property values, these objects are sorted by ascending `REFSTR` and listed first.

Return value:

The return value is a JSON object with three fields:

- `Count`: The number of objects found in the database for that data is returned.

- `RejectedObjects`: The objects that were found but for that no data was returned because of missing access permissions. The field contains a JSON list of objects with one JSON object for each rejected Alfabet object. The object has a field `RefStr` that returns the `REFSTR` of the Alfabet object and a field `Message` that gives information about the rejection.

- `Objects`: A JSON list of objects, each object representing a database object for that data is returned.

```
{
    "Objects": [...],
    "Count": 3,
    "RejectedObjects":[Array[0]]
}
```

The field `Objects` contains one JSON object per database object found for the call. The object contains the following fields:

- `ClassName`: The name of the Alfabet object class the object belongs to.

- `RefStr`:The value of the `REFSTR` of the object.

- `Values`: A JSON object that includes all information defined about the object in the database via object class properties. Each property is a field with a name corresponding to the property name. The value of the field informs about the value of the property. The return value for a property depends on the value type of the property:

Property Type	Return value
String, Text	The string, text is returned as string (in inverted commas): `"description":"This application manages customer relations."`
StringArray	All selected options of the string array are returned in one string: `"subcategories": "APP_SecurityAssessment APP_CloseSubworkflow"`
Boolean	Boolean values are 1 for true, 0 for false or `null` if the property is not set: `"variant": 1`
Real, Integer	Real and integer values are returned as number: `"xpos": 485.4`
Date, DateTime	Date and date time information is both returned as date strings of the format yyyy-mm-dd: `"last_update": "2012-02-21"`
URL	URLs are returned as string (in inverted commas): `"url": "http://company.com/ReportServer"`
Reference	The `REFSTR` of the object the reference is targeting is returned: `"ictobject": "26-608-0"`
ReferenceArray	For reference arrays, the `REFSTR`s of all objects that the current object references with this property are listed in an array: `"applicationgroups": [3]`

Property Type	Return value

```
0: "95-38-0"

1: "95-8-0"

2: "95-9-0"
```

The return value is independent from the setting of the attribute **RefSupport** of the property. If the attribute **RefSupport** of the property is set to `false`, the REF-STR value of the reference targets are directly stored in the database table of the object class in a column for the property. If the attribute **RefSupport** is set to `true`, the references are stored in the RELATIONS table. This difference is not visible in the return value for the `objects` endpoint.

If information about the objects referenced by the current object shall be included into the return value, you can use the method to find objects via a configured report that is also available for the endpoint `objects`.

- `Translations`: This field is only available if the field `DataCulture` is defined in the payload of the service call. The field contains a JSON list of object with one object per data culture. Each object has a field `DataCulture` that returns the language for that translations are returned and one field `Values` that is a JSON object with a field for each translated property. The value of the field informs about the translated value of the property.

- `GenericAttributes`: If attributes are defined for the object via the object class `GenericAttribute`, all generic attributes with the object class property `Owner` set to the current object are listed in this field as a JSON list of objects. Each object in the list returns one object of the class `GenericAttribute`, with the fields `Refstr`, `Values` and `Translations` that return data about the generic attribute in the same format as returned for the main object.

Exporting Information About Objects Found By A Configured Report

A configured report returning a tabular dataset that is configured via the configuration tool Alfabet Expand and stored in the Alfabet database can be used to export all or a subset of the data available about objects in the Alfabet database. This method returns data about all objects found as base objects of the report.

> For Alfabet queries, the base objects in the reports are the objects of the object class defined in the FIND clause. In native SQL queries, the first column of the result set is not displayed in the query results. It must specify the REFSTR of the object class selected as base class.
>
> Please note that the base class can be changed to any other class for that data is added to the result data set via the instruction SetRowReference.

The report can be used by the objects endpoint of the Alfabet RESTful API to deliver the following data about the objects found by the query of the report:

- The response returns the data that is included in the report. The advantages of this method are the following:

- The subset of objects returned can depend on a specific parameter, like for example all applications in an application group or all components that are having a defined indicator set to a specific value.

- The information returned about the objects is configurable. That means that data can be provided in the way and with the field names defined via the report. For example a reference to another object is stored in the database table of the object in a column with the technical name of the property and the value set to the `REFSTR` of the referenced object. The dataset in the query can return this information in a column of a customer defined name with the name of the referenced object instead of the `REFSTR`.

- The response returns information about all property values for all base objects found in the report independent from the data defined in the dataset of the report. This method delivers the same result per object as the method finding objects by `REFSTR`. The advantages of this method are the following:

  - The `REFSTR` of the objects must not be known to the client application when sending the request.

  - The subset of objects returned can depend on a specific parameter, like for example all applications in an application group or all components that are having a defined indicator set to a specific value.

- The response returns all `REFSTR` values of all base objects found in the report. This kind of response is useful if the `REFSTR` values of a defined group of objects are required as input for example for external applications that shall provide links that open the Alfabet user interface. For more information see Accessing the Alfabet User Interface From the External Application.

For the execution of this endpoint methods, a configured report must be available before executing the RESTful service call. The required configuration and the call are described in the following:

- Configuring a Report that Can be Called in a RESTful Service Request

- Service Call for Direct Execution of a Configured Report

- Service Calls for Execution of Offline Executed Reports

  - Service Call for Triggering Offline Execution

  - Checking the Status of the Asynchronous Report Execution

  - Exporting Results from Asynchronous Report Execution

- Return Value for the ReportResultDataSet Asynchronous Report Execution Returning the Data Defined In the Report

- Return Value for the ReportResultObjects Returning All Properties of the Objects Found in the Report

- Return Value for the ReportResult Returning the REFSTR of the Objects Found in the Report

Configuring a Report that Can be Called in a RESTful Service Request

The configuration of configured report and the underlying queries in Alfabet Expand is described in detail in the reference manual *Configuring Alfabet with Alfabet Expand* in the chapters *Configuring Reports* and

Defining Queries. This information is not repeated here. The following description is limited to additional settings required for the report.

The following special requirements apply to the settings of the attributes of the report:

- The attribute **Applicable for REST API** of the configured report must be set to `True`.

- The report must be of the **Type** `Query` or `NativeSQL`.

- Long running configured reports can be configured to be executed offline. These configured reports are executed asynchronously via the Alfabet RESTful services and require two service calls to export data, one for triggering the execution and one for downloading the results. The execution of this type of report is described separately in the following. Please note that there are a number of restrictions for data export from configured reports that are executed asynchronously. Results can only be downloaded in the original language. Translations are not provided. The options to use the configured report as an object list and download only the REFSTR values or the complete set of object class property values of the objects found by the configured report are not available.

The following special requirements apply to the query definition:

- SQL queries must return a `REFSTR` value as first argument. It is possible to create a report based on native SQL returning data without coupling the data to a base object, for example by setting the first argument in the `SELECT` statement to `NULL`. The return value of the Alfabet RESTful service call is structuring the returned data by base object found by in the report. Therefore this kind of report would result in no objects found. As a result, no data is displayed unless the instruction `SetRowReference` is used to define the base object using another row in the dataset to identify the base objects.

- Alfabet query language parameters can be used in Alfabet queries and native SQL queries. If the configured report contains parameter definitions, it is not required to define a **Report View** with filter definitions. The parameters are directly set in the service call.

- Alfabet query language instructions can be used in Alfabet queries and native SQL queries. The dataset after execution of the instructions is used for the generation of the return values. For example if the `JoinColumns` instruction is used in the configured report, the data joined in one column will also be displayed in one field of the result JSON object. The instruction `SetRowReference` alters the base object of the report data set and also for the return value of the service call.

- The configured report must return a simple table. Grouped datasets are not processed correctly. Data for subordinate levels is ignored.

Please note that the user that is used in the authorization of the RESTful service request must have a user profile assigned that has access permissions to the configured report. If none of the user profiles assigned to the user allows access to the configured report, no data is returned by the service request.

> (i) For information about configuring access permissions for configured reports, see the chapter *Defining and Managing User Access to Configured Reports* in the reference manual *User and Solution Administration*. While the access permissions for a configured report on the Alfabet user interface depend on multiple different factors, the access permissions for access to the configured report via the RESTful service API only depend on the user profile related access permissions defined for the configured report.

Service Call for Direct Execution of a Configured Report

Endpoint name: `objects`

HTTP method: `POST`

Service call:

`ServerAdress/api/v2/objects`

Header Fields:

`Authorization:TypeValue`

> (i) For information about the required Authorization header content, see [Authorization](#).

`Content-Type: application/json; charset=utf-8`

Payload

The payload is a JSON object with the following structure:

```
{
"CurrentProfile":"UserProfile1",
"CurrentMandate":"Headquarters",
"Language": 1033,
"DataCulture": "fr-FR",
"ApiCulture": "APICultureReporting",
"EmptyValues": true,
"Report": "Report_1",
"ReportResult": "DataSet",
"Limit": 500,
"Offset": 250,
"ReportArgs":
    {
    "arg1": "val1",
    "arg2": "val2",
    }
}
```

It may have following fields:

Field	Manda-tory/Op-tional	Required Value
`"Report":" Report-Name "`	Manda-tory	The name of the configured report that the call is reading data from. The name of the configured report is the value of the property **Name** of the configured report.
`ReportResult:"Da-taSet/References/Ob-jects"`	Manda-tory	`ReportResult` must be one of the following: • ***DataSet*** to return all information included in the report. • ***References*** to return a list of `REFSTR` of the objects found in the report • ***Objects*** to return all property values set for all objects found in the report
`"Limit":Number`	Optional	This field defines the maximum number of objects for that the response of the call shall return data. This value may be set to limit data return for reports including a high number of objects. In combination with the field `Offset` this field can be used to fetch the result of a big report in multiple steps. If `Limit` is not set, a maximum number of 1000 result data sets of the configured report are returned.
`"Offset":Number`	Optional	This field defines the start position for returning values in the result data set of the configured report. For example if the `Limit` field is set to 20 and the `Offset` is set to 10, the results listed in row 11 to 30 in the tabular output of the con-figured report are returned in the service call. By default, the start position is the first row in the data set with the number 0.

`"ReportArgs":`
 	
`    {`
`    "arg1": "val1",`
`    "arg2": "val2",`
`    }`
	Optional	If the configured report contains filter definitions, the values to be set for the filters when executing the report via the service call must be set with this field. The field value is a JSON object with one field for each filter parameter of the configured report. The field name must be identical to the parameter name without the prefix @ or: and the field value must be the value that shall substitute the parameter in the query.
		
		Please note the following:
		
		• Values can be of the type integer, boolean or string.
		
		• The wildcards % and * can be used in string values.

Field	Manda-tory/Op-tional	Required Value
		• All filter definitions of the configured report must be provided via the `ReportArgs` in the service call. If you do not want to set one of the filter values, you can define a wildcard only as value to return all results.
`"Emp-tyValues":true/false`	Optional	If `EmptyValues` is set to `true`, all object class properties of the object are returned even if they are not set. If `Emp-tyValues` is set to `false`, only object class properties of the object that are set are returned. By default the attribute is set to `false`.
`"CurrentPro-file":" UserProfile-Name "`	Manda-tory	The name of a user profile assigned to the Alfabet user sending the call that shall be used to access Alfabet. The user profile is required to evaluate whether the user that is sending the service call is allowed to read data about an object. For details about the access permissions that depend on the user profile, see the section <u>Authorization</u>.
`"CurrentMan-date":" MandateName "`	Optional	This field is only relevant if access to objects is controlled by the mandate concept for federated architectures implemented in Alfabet. The field must specify the name of a mandate assigned to the Alfabet user sending the call. For details about the consideration of mandate settings in REST API service calls, see the section <u>Controlling Access Via Mandates</u>.
`"Language":" Lan-guageLCID "`	Optional	This field is only relevant if instance translation is used in your Alfabet application and if the service call shall return all properties of the objects found in the configured report, that means `ReportResult` is set to `Objects`. The field must specify the language code (LCID) decimal of the culture for that translations are available in the Alfabet database. By default the standard language of the database is returned. Please note the following for the different settings of the ReportResult field: • `DataSet`: If the configured report is based on a native SQL query, instance translations are only returned if the translations are included into the native SQL query via the Alfabet specific */CULTURE_CODE/* statement. For more information about configuring native SQL queries to return data translation values, see *Displaying Translated Object Data In the Current Language of the User Interface* in the reference manual *Configuring Alfabet with Alfabet Expand*.

Field	Manda-tory/Op-tional	Required Value
		• `Objects`: For object class properties defined as translatable, values are returned in the selected language only. If a translation is not available for a translatable object class property, no value will be returned, even if a value is provided in the standard language. To return all values in the original values and add transla-tions in a second language, the parameter `DataCulture` shall be used instead of the parameter `Language`. `Language` supersedes `DataCulture`. If both fields are avail-able, only `Language` is used.
`"DataCulture":" Lan-guageCode "`	Optional	This field is only relevant if instance translation is used in your Alfabet application and translated values shall be re-turned. The way data is returned depends on the setting of the `ReportResult` field: • `Objects`: The field has to specify the language code of the culture for that translations are available in the Alfabet database (for example `de-DE` for German), or `ALL` to return translations for all languages that provide translation. Translations are added to the return value in addition to the values in the original language in a separate field `Translations`. • `DataSet`: The field has to specify the language code of the culture for that translations are available in the Alfabet database (for example `de-DE` for German). The translated value will be returned instead of the original language value in the return data set if the object class property is translatable and translation is provided. If no translation is available, the original language will be used to return the value. If the configured report is based on a native SQL query, in-stance translations are only returned if the translations are included into the native SQL query via the Alfabet specific */CULTURE_CODE/* statement. For more information about configuring native SQL queries to return data translation val-ues, see *Displaying Translated Object Data In the Current Language of the User Interface* in the reference manual *Configuring Alfabet with Alfabet Expand*. The field `Language` supersedes `DataCulture`. If both fields are available, only `Language` is used and `DataCulture` is ig-nored.

Field	Manda-tory/Op-tional	Required Value
`"ApiCulture":" APICul-tureName "`	Optional	This field specifies an API Culture defined in Alfabet Expand as part of the configuration of the Alfabet meta-model. The API Culture allows to define date, date/time and time patterns as well as number formats that deviate from the Alfabet standard. If this field is added to the request, dates, times and numbers are written into the return value in the formats specified in the API Culture definition. For more information, see the section [Configuring Handling of Date, Time and Number Formats For API Calls](#). Please note however, that values defined in the `ReportArgs` field must be defined in Alfabet default formats independent from the API Culture.

Service Calls for Execution of Offline Executed Reports

Long running configured reports can be configured to be executed offline. These configured reports are executed asynchronously and require a number of consecutive calls to the `offlineExecution` endpoint in the given order. Details about the endpoints are given in the following sections:

1) Schedule asynchronous execution of the configured report via a call to the `offlineExecution` endpoint with a JSON payload defining the execution details. A session token is returned. When the returned estimated execution time has elapsed, the next service call can be scheduled.

2) Check the status for the report execution with the returned session token via a call to the `offlineExecution` endpoint with the session token as parameter. When the returned status is `Succeeded`, the next service call for download can be scheduled.

3) Download the results via a call to the endpoint `offlineExecution` with the session token and additional download parameters. For configured reports returning a high number of records the results can be exported step-wise via multiple service calls.

The following sections inform about the required calls:

* [Service Call for Triggering Offline Execution](#)

* [Checking the Status of the Asynchronous Report Execution](#)

* [Exporting Results from Asynchronous Report Execution](#)

Service Call for Triggering Offline Execution

Endpoint name: `offlineExecution`

HTTP method: `POST`

Service call:

```
ServerAdress/api/v2/offlineExecution
```

Header Fields:

```
Authorization:TypeValue
```

For information about the required Authorization header content, see Authorization.

```
Content-Type: application/json; charset=utf-8
```

Payload

The payload is a JSON object with the following structure:

```
{
"CurrentProfile":"UserProfile1",
"CurrentMandate":"Headquarters",
"Report": "Report_1",
"ReportArgs":
    {
    "arg1": "val1",
    "arg2": "val2",
    }
}
```

It may have the following fields:

Field	Manda-tory/Op-tional	Required Value
`"Report":" Report-Name "`	Mandatory	The name of the configured report that the call is reading data from. The name of the configured report is the value of the property **Name** of the configured report.

`"ReportArgs":`
 	
`    {`
`    "arg1":`
`    "val1",`
`    "arg2":`
`    "val2",`
`    }`
	Optional	If the configured report contains filter definitions, the values to be set for the filters when executing the report via the service call must be set with this field. The field value is a JSON object with one field for each filter parameter of the configured report. The field name must be identical to the parameter name without the prefix @ or: and the field value must be the value that shall substitute the parameter in the query.
		
		Please note the following:
		
		• Values can be of the type integer, boolean or string. Dates must be provided as string in the format.
		
		• The wildcards % and * can be used in string values.

Field	Manda-tory/Op-tional	Required Value
		• All filter definitions of the configured report must be provided via the `ReportArgs` in the service call. If you do not want to set one of the filter values, you can define a wildcard only as value to return all results.
`"CurrentPro-file":" UserPro-fileName "`	Mandatory	The name of a user profile assigned to the Alfabet user sending the call that shall be used to access Alfabet. The user profile is required to evaluate whether the user that is sending the service call is allowed to read data about an object. For details about the access permissions that depend on the user profile, see the section Authorization.
`"CurrentMan-date":" Man-dateName "`	Optional	This field is only relevant if access to objects is controlled by the mandate concept for federated architectures implemented in Alfabet. The field must specify the name of a mandate assigned to the Alfabet user sending the call. For details about the consideration of mandate settings in REST API service calls, see the section Controlling Access Via Mandates.

Return value:

The return value is a JSON object in the following format:

```
{
    "Count": 0,
    "Name": "RESTReportLongRunning",
    "Description": "This configured report is executed offline",
    "OfflineExecutionResultStatus": "Queued",
    "OfflineExecutionToken": "ux4zv0r13ocrv1t6dca3zrzi5",
    "OfflineExecutionErrorMessage": ""
    "EstimatedOfflineExecutionTime": "00:05:00"
}
```

The following information is provided:

- `OfflineExecutionToken`: A token that has to be added to the service calls to check the report execution status and to export the report results.

- `EstimatedOfflineExecutionTime`: The estimated execution time in minutes that to be used as default setting in the **Estimated Execution Time** attribute of the configured report. The next call for checking the execution result should be scheduled after the estimated offline execution time.

- `Name`: The name of the configured report scheduled for execution.

- `Description`: The text provided with the **Description** attribute of the configured report.

- `OfflineExecutionResultStatus`: The status of the offline execution is `Queued` after successful scheduling of report execution.

- `OfflineExecutionErrorMessage`: If an error occures, this field will provide details about the error. After successful execution this field will be empty.

- `Count`: This field is not relevant.

Checking the Status of the Asynchronous Report Execution

Endpoint name: `offlineExecution`

HTTP method: `GET`

Service call:

> *ServerAdress*/api/v2/offlineExecution?offlineExecutionId=*OfflineExecutionToken*

with the following parameters:

Parameter	Manda- tory/Optional	Required Value
`offlineExecutionId= Of- flineExecutionToken`	Mandatory	The `OfflineExecutionToken` from the return value of the service call scheduling report execution.

Header Fields:

> `Authorization:`*TypeValue*

> ⓘ   For information about the required Authorization header content, see Authorization.

> `Content-Type: application/json; charset=utf-8`

Return value:

The return value is a JSON object in the following format:

```
{
"Count": 170,
"OfflineExecutionResultStatus": "Succeeded",
"OfflineResultExpirationDateTime": "2020-07-22T10:51:39.29"
}
```

The following information is provided:

- `Count`: The number of records returned.

- `OfflineExecutionResultStatus`: Returns `Succeeded` if results are available. During execution, the status is `NotReady`.

- `OfflineResultExpirationDateTime`: Informs about the date and time at which the report results will be deleted. Offline execution results are available for a time period defined in the configured report with the **Offline Result Retention Time** attribute.

- `Error Message`: If the call has failed, this field gives details about the error that occurred.

- `Error Code`: If the call has failed, this field returns the error code of the web services.

Exporting Results from Asynchronous Report Execution

Endpoint name: `offlineExecution`

HTTP method: `GET`

Service call:

```
ServerAdress/api/v2/offlineExecution?offlineExecutionId=OfflineExecutionToke
n&offset=StartRowNumber&limit=NumberOfResults
```

with the following parameters:

Parameter	Manda-tory/Op-tional	Required Value
`offlineExecu-tionId= Of-flineExecu-tionToken`	Mandatory	The `OfflineExecutionToken` from the return value of the service call scheduling report execution.
`limit= Number-OfResults`	Mandatory	The number of records from the result data set of the configured report that shall be returned by the call.
`offset= Star-tRowNumber`	Mandatory	The start position for returning values in the result data set of the configured report. For example if the `limit` parameter is set to 20 and the `offset` parameter is set to 10, the results listed in row 11 to 30 in the tabular output of the configured report are returned in the service call. By default, the start position is the first row in the data set with the number 0.

Header Fields:

```
Authorization:TypeValue
```

ⓘ   For information about the required Authorization header content, see Authorization.

```
Content-Type: application/json; charset=utf-8
```

Return value:

For information about the return value, see XXX.

Return Value for the ReportResultDataSet Asynchronous Report Execution Returning the Data Defined In the Report

The return value is a JSON object with four fields:

- `Count`: The number of rows in the data set of the configured report the service call is based on.

- `Name`: The name of the configured report the data is derived from as defined in the attribute **Name** of the configured report.

- `Description`: The description for the configured report the data is derived from as defined in the attribute **Description** of the configured report.

- `Objects`: A JSON list of objects, each object representing a row in the data set of the configured report. Information about the base object of the row and the values in the row of the report are given.

```
{
    "Objects": [...]
    "Count": 3,
    "Name": "ReportName",
    "Description": "Report Description"
}
```

The field `Objects` contains one JSON object per row in the data set of the configured report. The object contains the following fields:

- `ClassName`: The name of the Alfabet object class the base object of the current row belongs to.

- `RefStr`:The value of the `REFSTR` of the base object of the current row.

- `Values`: A JSON object that includes all information defined in the cells of the current row of the configured report. Each column in the dataset is a field with a name corresponding to the column name. The value of the field informs about the value in the current row. All values are returned as strings.

Return Value for the ReportResultObjects Returning All Properties of the Objects Found in the Report

The return value lists all object class properties set for all base objects of the configured report.

The return value is a JSON object with four fields:

- `Count`: The number of rows in the data set of the configured report the service call is based on. The count may differ from the number of objects for that data is returned. The report might display multiple rows for the same base object. In that case each base object is only considered once.

- `Name`: The name of the configured report the data is derived from as defined in the attribute **Name** of the configured report.

- `Description`: The description for the configured report the data is derived from as defined in the attribute **Description** of the configured report.

- `Objects`: A JSON list of objects, each object representing a database object for that data is returned.

```
{
    "Objects": [...]
    "Count": 3
}
```

The field `Objects` contains one JSON object per database object found for the call. The object contains the following fields:

- `ClassName`: The name of the Alfabet object class the object belongs to.

- `RefStr`:The value of the `REFSTR` of the object.

- `Values`: A JSON object that includes all information defined about the object in the database via object class properties. Each property is a field with a name corresponding to the property name. The value of the field informs about the value of the property. The return value for a property depends on the value type of the property:

Property Type	Return value
String, Text	The string, text is returned as string (in inverted commas): `"description":"This application manages customer relations."`
StringArray	All selected options of the string array are returned in one string: `"subcategories": "APP_SecurityAssessment APP_CloseSubworkflow"`
Boolean	Boolean values are 1 for true, 0 for false or `null` if the property is not set: `"variant": 1`
Real, Integer	Real and integer values are returned as number: `"xpos": 485.4`
Date, DateTime	Date and date time information is both returned as date strings of the format yyyy-mm-dd:

Property Type	Return value
	`"last_update": "2012-02-21"`
URL	URLs are returned as string (in inverted commas): `"url": "http://company.com/ReportServer"`
Reference	The `REFSTR` of the object the reference is targeting is returned: `"ictobject": "26-608-0"`

Refer-enceArray	For reference arrays, the `REFSTR`s of all objects that the current object references with this property are listed in an array:
	
	`"applicationgroups": [3]`
	    `0: "95-38-0"`
	    `1: "95-8-0"`
	    `2: "95-9-0"`
	
	The return value is independent from the setting of the attribute **RefSupport** of the property. If the attribute **RefSupport** of the property is set to `false`, the `REFSTR` value of the reference targets are directly stored in the database table of the object class in a column for the property. If the attribute **RefSupport** is set to `true`, the references are stored in the `RELATIONS` table. This difference is not visible in the return value for the `objects` endpoint.
	
	If information about the objects referenced by the current object shall be included into the return value, you can use the method to find objects via a configured report that is also available for the endpoint `objects`.

- `GenericAttributes`: If attributes are defined for the object via the object class `GenericAttribute`, all generic attributes with the object class property `Owner` set to the current object are listed in this field as a JSON list of objects. Each object in the list returns one object of the class `GenericAttribute`, with the fields `Refstr`, `Values` and `Translations` that return data about the generic attribute in the same format as returned for the main object.

Return Value for the ReportResult Returning the REFSTR of the Objects Found in the Report

The return value includes all `REFSTR` values for the base object of each row in the configured report.

The return value is a JSON object with four fields:

- `Count`: The number of rows in the data set of the configured report the service call is based on.

- `Name`: The name of the configured report the data is derived from as defined in the attribute **Name** of the configured report.

- `Description`: The description for the configured report the data is derived from as defined in the attribute **Description** of the configured report.

- `Refs`: A JSON array containing the `REFSTR` values of the base objects in the report. The array can contain a `REFSTR` value multiple times if the base object of multiple rows is identical.

```
{
    "Refs": ["95-36-0","95-43-0","95-41-0"]
    "Count": 3
}
```

Deleting Objects from the Alfabet database

The endpoint `delete` provide a means to delete objects and relations from the Alfabet database. A relation is a reference from one object to another object that is stored in a property of the type Reference Array. Most reference arrays are stored in the relations table of the Alfabet database.

> For detailed information about the storage of relations in the Alfabet database see the chapter *The Alfabet Meta-Model in the Alfabet Database* in the reference manual *Alfabet Data Integration Framework*.

The request must include the definition of the database manipulation in the body of the request in JSON format.

Endpoint name: `delete`

HTTP method: `POST`

Service call:

> *ServerAdress*/api/v2/delete

Header Fields:

> Authorization:*TypeValue*

> For information about the required Authorization header content, see Authorization.

> Content-Type: application/json; charset=utf-8

Payload

The payload is a JSON object with the following structure:

```
{
"CurrentProfile": "UserProfile1",
"CurrentMandate": "Mandate1",
"Refs": ["76-2518-0","76-3246-0","405-3-0"],
"Relations": [{
```

```
        "FromRef": "76-2518-0",

        "Property": "BelongsTo",

        "ToRef": "76-3246-0"

        },

        {...}]

    }
```

It may have following fields:

Field	Manda-tory/Op-tional	Required Value
`"Refs":[" RefstrOfOb-ject "," RefstrOfObject "]`	Optional	An array containing one or multiple values of the `REFSTR` property of objects in the Alfabet database that shall be deleted.

`"Relations": [{`
 	
`    "FromRef":`
`    "RefstrOfObject",`
 	
`    "Property":`
`    "PropertyName",`
 	
`    "ToRef":`
`    "RefstrOfObject"`
`    },`
`    {...}]`
	Optional	A JSON list of objects, each objects representing a relation to be deleted. For each relation that shall be deleted, the following fields must be defined:
		
		• `FromRef`: The `REFSTR` of the object for that the relation is defined via one of its object class properties.
		
		• `Property`: The value of the **Name** attribute of the object class property for that the relation is stored.
		
		• `ToRef`: The `REFSTR` of the object to that the relation is built.

`"CurrentProfile":" UserPro-fileName "`	Manda-tory	The name of a user profile assigned to the Alfabet user sending the call that shall be used to access Alfabet. The user profile is required to evaluate whether the user that is sending the service call has write permissions to an object. For details about the access permissions that depend on the user profile, see the section Authorization.
`"CurrentMandate":"Man-dateName"`	Optional	This field is only relevant if access to objects is controlled by the mandate concept for federated architectures implemented in Alfabet. The field must specify the name of a mandate assigned to the Alfabet user sending the call. For details about the consideration of mandate settings in REST API service calls, see the section Controlling Access Via Mandates.

Return value:

The return value is a JSON object with one field `"Count"` that informs about the number of objects and relations that were deleted.

Creating and Updating Object Data in the Alfabet database

The endpoint `update` provide a means to alter the content of the Alfabet database. New objects can be created for Alfabet object classes, properties of existing objects can be changed, and relations between objects can be created. The request must include the definition of the database manipulation in the body of the request in JSON format.

As a result of the service call, the content of the Alfabet database is changed and a return value is sent that confirms the change.

Endpoint name: `update`

HTTP method: `PUT`

Service call:

> *ServerAdress*`/api/v2/update`

Header Fields:

> `Authorization:`*TypeValue*

ⓘ For information about the required Authorization header content, see Authorization.

> `Content-Type: application/json;charset=utf-8`

Payload

The payload is a JSON object with the following structure:

```
{
"CurrentProfile":"UserProfile1",
"CurrentMandate": "Mandate1",
"ApiCulture": "ApiCultureName",
"Objects": [...],
"Relations": [...],
}
```

It may have following fields:

Field	Mandatory/Optional	Required Value
`"Objects"`	Optional	An array containing the specification of the objects and the data that shall be changed for the objects. Details are described below in the sections

Field	Manda-tory/Op-tional	Required Value
		Creating a new Object in the Alfabet database and Changing the Properties of an existing Object in the Alfabet database.
"Rela-tions"	Optional	An array containing the specification of the references that shall be changed. Details are described below in the section Creating or Updating a Relation Between Objects in the Alfabet database.
"Cur-rentPro-file"	Mandatory	The name of a user profile assigned to the Alfabet user sending the call that shall be used to access Alfabet. The user profile is required to evaluate whether the user that is sending the service call has write permissions to an object. For details about the access permissions that depend on the user profile, see the section Authorization.
"Current-Mandate"	Optional	This field is only relevant if access to objects is controlled by the mandate concept for federated architectures implemented in Alfabet. The field must specify the name of a mandate assigned to the Alfabet user sending the call. For details about the consideration of mandate settings in REST API service calls, see the section Controlling Access Via Mandates.
"ApiCul-ture"	Optional	This field specifies an API Culture that was defined in Alfabet Expand as part of the configuration of the Alfabet meta-model. The API Culture allows to define data, date/time and time patterns as well as number formats that deviate from the Alfabet standard. If this field is added to the request, dates, times and numbers can be defined in the object data definitions for update of object data in the formats specified in the API Culture definition. For more information, see the section Configuring Handling of Date, Time and Number Formats For API Calls.

The following sections describe the different kind of operations and the required JSON request:

- Creating a new Object in the Alfabet database

- Changing the Properties of an existing Object in the Alfabet database

- Creating or Updating a Relation Between Objects in the Alfabet database

A single JSON request can include multiple different operations, that means you can for example create new objects and update data of existing objects in the same request.

> ⓘ The update of the data in the Alfabet database requires knowledge about the object class config-uration of the Alfabet meta-model and/or existing object data.
>
> - Information about the current definition of object classes and object class properties can be retrieved via the endpoint `metamodel`.

- Information about the current object data can be retrieved via the endpoint `select` or `object`.

- For basic information about the structure of the Alfabet class model that is required to perform data manipulation directly on the database level, see *The Alfabet Meta-Model in the Alfabet Database* in the reference manual *Alfabet Data Integration Framework*.

Return value:

The return value is a JSON object informing about changes performed and errors that occurred, for example when creating a two new objects, the return value is:

```
{
    "NewObjects": {
        1: "95-61-0"
        2: "95-62-0" }
    "Count": 2
}
```

The return value can have the following fields:

Field	Value
"NewObjects"	This field contains a JSON object with one field for each new object. The field name is the Id defined in the request and the field value is the value of the REFSTR property of the new object in the Alfabet database. If the request only updates objects, this field is empty.
"Access Denied"	This field contains a JSON object with a field for each object for that access is denied for example because of mandate settings. The field name is the REFSTR value of the object and the field value is the reason for denial of access.
"RejectedObjects"	This field informs about object that could not be created or changed. The field contains a JSON list of objects with one JSON object for each rejected change to an Alfabet object. The object has a field RefStr that returns the REFSTR of the Alfabet object and a field Message that gives information about the rejection.
"NotFound"	If a relation shall be created and one or both of the involved objects does not exist in the Alfabet database, the call displays a NotFound field containing an array with the REFSTR values that could not be found.
"Count"	This fields informs about the number of objects and relations that were updated or created. Please note that the count for update and deletion of relations is not always identical to the number of relations that were requested to be updated or deleted via the call. This is due to internal mechanisms like for example back relations that lead to two relations to

Field	Value
	be changed if one relation is requested to be changed. The returned count can therefore not be used to check the completeness of request execution.

Creating a new Object in the Alfabet database

The JSON object of the request to create a one or multiple new objects contains the following fields:

- `CurrentProfile`: The name of the user profile that shall be used for the user when performing the call. The permission to create an object of an object class in the Alfabet database is evaluated via the user profile. Detailed information is given in the section. The field is optional. If it is not included, the last user profile the user was logged in with is used for the call. If the information about the user profile of the last log in is not found, the user profile defined as default profile for the user in the user settings of the user is used as default for the field.

- `Objects`: A definition of object data that shall be created. Details are given below.

The value of the field `Objects` is a JSON list of objects, each object defining one Alfabet object to be created with the following fields:

Field name	Required Value	Remark
Class-Name	The value of the **Name** attribute of the Alfabet object class for that the object shall be created.	
Id	An integer that is a unique identifier inside the JSON request.	Each object created in a single JSON request must have a different id. The id is only used within the single JSON request. It is not having any impact on the Alfabet database. As soon as the objects are created, the ids are no longer coupled to the objects and can be used in other service calls.
Values	A JSON object with one field for each property that shall be defined. The field name is identical to the **Name** of the object class property written in lower case letters. The field value is identical to the value of the object class property of the created object.	At least all properties defined as mandatory must be set for an object to create the object. The property REFSTR cannot be defined in the JSON request. It is set automatically during creation of the object and returned in the return value of the service call. Technical properties like creation date and creation user are also automatically set during creation of the object.

Field name	Required Value	Remark
		Properties with the attribute **Automatically Managed** set to `true` must not be set via this mechanism. Properties of the type Reference and ReferenceArray cannot be updated with this mechanism. For information about the update of properties of the type Reference and ReferenceArray see [Creating or Updating a Relation Between Objects in the Alfabet database](#). The value must match the format and restrictions that apply to the object class property. For example a string length restriction might be implemented for an object class property of the type string. Dates must be defined in the format yyyy-mm-dd.
Translations	A list of JSON objects, each object representing a translation and having two fields: the field `DataCulture` that defines the language code of the data culture (for example `fr-FR`) and the field `Values`, that contains a field for each translation to be added with the field name being identical to the property name and the field value being identical to the translation that shall be added.	This field is optional and shall be added if translations of object class names and descriptions shall be provided for one or multiple data cultures specified in Alfabet Expand that are defined to allow data translation.
GenericAttributes	A JSON list of objects, with one JSON object for each generic attribute to create or update. Each JSON object has three fields: • `RefStr`: The field must be defined with an empty value to create a new generic attribute. • `Values`: A JSON object with one field for each property that shall be updated. Required fields are `Name`, `Type` and `Value`. The field `Group` is optional. The property `Owner` of the object class `GenericAttribute` must not be specified. It is set by the import mechanism to the `REFSTR` to the object the generic attribute is defined for. • `Translations`: If the name of the generic attribute shall be translated, a field translation can be specified as described above for the main object.	The object class `GenericAttribute` stores properties for different object classes. If generic attributes shall be defined for objects created via an endpoint `update`, generic reference data must be created directly within the request for the object it belongs to.

The following example shows the required JSON request for creating two application groups with a translation into two languages and a generic attribute:

```json
{
    "CurrentProfile": "RESTAccessProfile";
    "Objects": [{
        "ClassName": "ApplicationGroup",
        "Id": "1",
        "Values": {
            "name": "Test Group 1",
            "shortname": "TestG1"
            }
        "Translations":[
            {"DataCulture": "de-DE",
            "Values":{"name": "Testgruppe 1"}
            },
            {"DataCulture": "fr-FR",
            "Values":{"name": "Groupe Controle 1"}
            }
            ]
        ]
        },
        {
        "ClassName": "ApplicationGroup",
        "Id": "2",
        "Values": {
            "name": "Test Group 2",
            "shortname": "TestG2"
            },
        "Translations":[
            {"DataCulture": "de-DE",
            "Values":{"name": "Testgruppe 2"}
            },
            {"DataCulture": "fr-FR",
            "Values":{"name": "Groupe Controle 2"}
            }
            ],
        "GenericAttributes": [{
```

```
                    "RefStr": "",

                    "Values": {"Name": "Integer Attribute 1",

                    "Group": "",

                    "Type": "Integer",

                    "Value": "99999"},

                    "Translations": [{

                        "DataCulture": "de-DE",

                        "Values": {"Name": "Integer-Attribut"}

                        }]

                    }]

                }]

            }
```

Changing the Properties of an existing Object in the Alfabet database

To update property values for an existing object in the Alfabet database, the JSON object of the request must contain a field `Objects`. The value of this field is a JSON list of objects, each object defining one Alfabet object to be updated with the following fields:

Field name	Required Value	Remark
REF-STR	The value of the `REFSTR` property of the object that shall be updated.	
Val-ues	A JSON object with one field for each property that shall be updated. The field name is identical to the **Name** of the object class property written in lower case letters. The field value is identical to the new value that shall be set for the object class property.	• Properties with the attribute **Automatically Managed** set to `true` must not be set via this mechanism. • Properties of the type Reference and ReferenceArray cannot be updated with this mechanism. For information about the update of properties of the type Reference and ReferenceArray see Creating or Updating a Relation Between Objects in the Alfabet database. • The value must match the format and restrictions that apply to the object class property. For example a string length restriction might be implemented for an object class property of the type string.

Field name	Required Value	Remark
	• Dates must be defined in the format yyyy-mm-dd. • Object class properties of the data type Url or Email consist of a name and the actual web/email address separated with `\r\n`. For example: `Software Ag\r\nwww.softwareag.com` If no `\r\n` is available in the definition, the provided string is saved as web/email address.	
Trans-lations	A list of JSON objects, each object representing a translation and having two fields: the field `DataCulture` that defines the language code of the data culture (for example `fr-FR`) and the field `Values`, that contains a field for each translation to be added with the field name being identical to the property name and the field value being identical to the translation that shall be added.	This field is optional and shall be added if translations of object class names and descriptions shall be provided for one or multiple data cultures specified in Alfabet Expand that are defined to allow data translation.
Ge-neri-cAt-trib-utes	A JSON list of objects, with one JSON object for each generic attribute to create or update. Each JSON object has three fields: • `RefStr`: For updates, the value of the field must be the `REFSTR` of the existing generic attribute. For creating a new generic attribute, the field must be defined with an empty value. • `Values`: A JSON object with one field for each property that shall be updated. Required fields are `Name`, `Type` and `Value`. The field `Group` is optional. The property Owner of the object class Generic Attribute must not be specified. It is set by the import mechanism to the `REFSTR` to	The object class `GenericAttribute` stores properties for different object classes. If generic attributes shall be defined via an endpoint `update`, generic reference data must be updated or created directly within the update request for the object it belongs to.

Field name	Required Value	Remark
	the object the generic attribute is defined for. • `Translations`: If the name of the generic attribute shall be translated, a field translation can be specified as described above for the main object.	

The following example shows the required JSON request for updating two application groups including translation of the names for the two objects:

```
{
    "Objects": [{
        "RefStr": "95-61-0",
        "Values": {
            "shortname": "TG1",
            "status": "Planned"
            }
        "Translations":[
            {"DataCulture": "de-DE",
            "Values":{"name": "Testgruppe 1"}
            },
            {"DataCulture": "fr-FR",
            "Values":{"name": "Groupe Controle 1"}
            }
            ]
        "GenericAttributes": [{
            "RefStr": "",
            "Values": {"Name": "Integer Attribute 1",
            "Group": "",
            "Type": "Integer",
            "Value": "99999"},
            "Translations": [{
                "DataCulture": "de-DE",
                "Values": {"Name": "Integer-Attribut"}
                }]
```

```
            }]
        },
        {
        "RefStr": "95-62-0",
        "Values": {
            "shortname": "TG2",
            }
        "Translations":[
            {"DataCulture": "de-DE",
            "Values":{"name": "Testgruppe 1"}
            },
            {"DataCulture": "fr-FR",
            "Values":{"name": "Groupe Controle 1"}
            }
            ]
        }]
    }
```

Creating or Updating a Relation Between Objects in the Alfabet database

Relations can be created between objects that already exist in the Alfabet database. If an object does not exist, the relation is not set and the REFSTR of the object or objects that could not be found is returned in an array in a JSON field `NotFound` in the return value.

A relation between two objects is established via a property of one of the objects that is of the type Reference or ReferenceArray. Although different mechanisms exist for storage of relations for properties of the type Reference and ReferenceArray, these differences are of no importance for the definition of the relation in the service call. This is handled by the Alfabet RESTful API on server side.

If a relation is defined that already exists in the Alfabet database, the following happens:

- If the object class property storing the relation is of the **Type** `Reference`, the existing reference is overwritten.

- If the object class property storing the relation is of the **Type** `ReferenceArray`, the new reference is added to the array and existing references persist. If you want a relation of the **Type** `ReferenceArray` to be substituted, you must delete the old relation with a service request of the endpoint `delete` and create a new one with this service request.

For the request send with the service call, the following information must be provided:

- Definition of the "from" object. This is the object for that the property establishing the relation is defined.

- Definition of the "to" object. This is the object to that the relation is established.

- Definition of the property establishing the relation.

To create a new relation, the JSON object of the request must contain a field `Relations`. The value of this field is a JSON list of objects, each object defining one relation between Alfabet objects to be created or updated with the following fields:

Field name	Required Value
FromRef	The value of the REFSTR property of the Alfabet object for that the property establishing the relation is defined.
Property	The value of the **Name** attribute of the object class property establishing the relation.
ToRef	The value of the REFSTR property of the Alfabet object to that the relation is established.

The following example shows a JSON request that updates a property of the type Reference and creates a relation for a property of the type ReferenceArray. The updated object is an application group. The property `ResponsibleUser` is of the type Reference. The existing reference to a responsible user is therefore overwritten with the relation defined in the request. The property `Applications` is of the type ReferenceArray. The relation defined in the request is added to already existing relations:

```
{
    "Relations": [{
        "FromRef": "95-61-0",
        "Property": "ResponsibleUser",
        "ToRef": "421-862-0"
    },
    {
        "FromRef": "95-61-0",
        "Property": "Applications",
        "ToRef": "76-2518-0"
    }]
}
```

Archiving Objects from the Alfabet database

The endpoint `archiveobject` provide a means to create an archive ZIP file for one or multiple selected objects from the Alfabet database. The object can optionally be deleted after creating the archive ZIP file.

When an Alfabet object is archived, a ZIP file is created containing HTML files that display the object profile for the archived object as well as the object profiles of its dependent objects. Each archived object profile

displays a preconfigured set of page views, whereby the visibility of these views will depend on the class setting configured for the object class. If a page view displays dependent objects, a user can click the dependent object in the HTML view in order to open another HTML file showing the archived object profile of the selected dependent object.

Alfabet objects are typically archived by a solution administrator in the **Simple Search** functionality of the **Admin** user profile. If an archive is created via the Alfabet user interface, the archived Alfabet object is deleted from the Alfabet database after generation of the archive ZIP file. In the **Archive Manager** functionality of the **Admin** user profile, the archive ZIP file is then available for download to a local disk. After extracting the ZIP file, the relevant HTML file can then be viewed in a browser window. The archive ZIP file contains one folder for each culture setting supported by your enterprise.

This functionality is especially useful in order to remove outdated objects from the Alfabet database and thus enhance the performance of Alfabet. The archived data can be saved and displayed when needed in a Web browser.

> For information about the archiving of Alfabet objects via the Alfabet user interface, see *Deleting and Archiving Alfabet Objects* in the reference manual *User and Solution Administration*.

Archiving of objects via a RESTful service call provides more flexibility:

- An archive can be created without deleting the object.

- Archives can either be made available in the **Archive Manager** functionality of the Alfabet user interface only, or they can additionally be directly stored in a defined folder on the local file system directly during the execution of the RESTful service call.

The request must include the definition of the archiving and database manipulation in the body of the request in JSON format.

Endpoint name: `archiveobject`

HTTP method: `POST`

Service call:

> *ServerAdress*`/api/v2/archiveobject`

Header Fields:

> `Authorization:`*TypeValue*

> For information about the required Authorization header content, see Authorization.

> `Content-Type: application/json; charset=utf-8`

Payload

The payload is a JSON object with the following structure:

```
{
"CurrentProfile": "UserProfile1",
"CurrentMandate": "Mandate1",
"CurrentCulture": "1033",
```

```
"Refs": ["76-2518-0","76-3246-0","405-3-0"],

"DeleteAfterArchive": "true",

"ZipPath":"C:\\ArchiveObject\\Applications"

}
```

It may have following fields:

Field	Manda-tory/Op-tional	Required Value
`"Refs":[" RefstrO- fObject "," RefstrO- fObject "]`	Mandatory	An array containing one or multiple values of the `REFSTR` property of objects in the Alfabet database that shall be ar-chived or archived and deleted.
`"CurrentPro- file":" UserProfile- Name "`	Mandatory	The name of a user profile assigned to the Alfabet user send-ing the call that shall be used to access Alfabet. The user pro-file is required to evaluate whether the user that is sending the service call has write permissions to an object. For details about the access permissions that depend on the user profile, see the section <u>Authorization</u>.
`"CurrentMan- date":" Man- dateName "`	Optional	This field is only relevant if access to objects is controlled by the mandate concept for federated architectures imple-mented in Alfabet. The field must specify the name of a man-date assigned to the Alfabet user sending the call. For details about the consideration of mandate settings in REST API ser-vice calls, see the section <u>Controlling Access Via Mandates</u>.
`"CurrentCulture": " LCID decimal "`	Optional	The language code (LCID) decimal of the culture that shall be used for messages in the return value. The culture must be a culture for that translations are available via the vocabulary files in the Alfabet database. By default, the message in the return value are English.
`"ZipPath":" Abso- lutePathToFolder "`	Optional	The absolute path to the folder on the local file system in which the archive should be stored in addition to being stored in the Alfabet database. If the folder does not exist, it is cre-ated during execution of the service call. If this field is not set, the archive is only stored in the Alfabet database and available for download via the Archive Manager functionality on the Alfabet user interface. Please note that the back slashes in the path must be written as double back slashes to be accepted as text in the JSON definition.

Field	Manda-tory/Op-tional	Required Value
`"DeleteAfterAr-chive": "true|false"`	Optional	Set the field value to `true` if the object shall be deleted after generation of the archive. By default, the object is not deleted after generation of the archive.

Additional access rights are required for deletion of the object. If you set the field to `true` and the permissions for the user and user profile for processing the call are not sufficient to delete the object, the archive is also not created, even if the permissions for creating archives are given. |

Return value:

The return value is a JSON object with one field `"ResultMessages"` that informs about the success status in a separate field per object in the format "REFSTR of object":"Message".

```
{

"ResultMessages": {

    "95-44-0": "Instance not found",

    "76-3200-0": "Access denied:'Instance cannot be deleted'",

    "95-43-0": "Object successfully archived."

    }

}
```

Regenerating the Password of an Alfabet User

The endpoint `regeneratepassword` provide a means to reset the password of a user or to create an initial password for a new user. A password is automatically assigned to the users and the user receives emails informing the him/her about the login credentials.

ⓘ This functionality includes sending emails to the user via the system. You must ensure that system emails are activated for your Alfabet installation. For more information about activating the sending of emails, see the section *Activating the Dispatch of Email Notifications in Alfabet* in the reference manual *System Administration*. For more information about specifying the message in the emails or configuring custom text templates to use in place of the standard text templates, see the section *Specifying Custom Text Templates for Password Generation* in the reference manual *Configuring Alfabet with Alfabet Expand* and the section *Text Templates for Activation of User Passwords* in the reference manual *Configuring Alfabet with Alfabet Expand - Appendix*.

Password regeneration actions are logged in a log file. This is by default the file `PasswordChangeLog.txt` in the working directory of the Alfabet Web Application. For information about how to change the location of the log file, see *Defining the Location of the Log File for Password Reset and Regeneration Actions* in the reference manual.

ⓘ For general information about the configuration and administration requirements for user login via user name and password, see *Configuring Standard Login* in the reference manual *System Administration*.

The request must include the definition of the archiving and database manipulation in the body of the request in JSON format.

Endpoint name: `regeneratepassword`

HTTP method: `POST`

Service call:

> *ServerAdress*`/api/v2/regeneratepassword`

Header Fields:

> `Authorization:`*TypeValue*

ⓘ For information about the required Authorization header content, see [Authorization](#).

> `Content-Type: application/json; charset=utf-8`

Payload

The payload is a JSON object with the following structure:

```
{
"CurrentProfile": "UserProfile1",
"CurrentCulture": "1033",
"Refs": ["421-2518-0","421-3-0"],
"UserNames:["CUSTOMER","CLIENTE"]
}
```

It may have following fields:

Field	Manda-tory/Op-tional	Required Value
`"Refs":[" RefstrOfOb-ject "," RefstrOfOb-ject "]`	Optional	An array containing one or multiple values of the `REFSTR` property of the users (object class `Person`) in the Alfabet database for which the password should be regenerated.
`"UserNames":["CUS-TOMER","CLIENTE"]`	Optional	An array containing one or multiple user names of users (`USER_NAME` property of the object class `Person`) in the Alfabet database for which the password should be regen-erated.

Field	Manda-tory/Op-tional	Required Value
`"CurrentPro-file":" UserProfile-Name "`	Mandatory	The name of a user profile assigned to the Alfabet user sending the call that shall be used to access Alfabet. The user profile is required to evaluate whether the user that is sending the service call has write permissions to an object. For details about the access permissions that depend on the user profile, see the section Authorization.
`"CurrentCulture": " LCID decimal "`	Optional	The language code (LCID) decimal of the culture that shall be used for messages in the return value. The culture must be a culture for that translations are available via the vo-cabulary files in the Alfabet database. By default, the mes-sage in the return value are English.

Return value:

The return value is a JSON object with one field `"ResultMessages"` that informs about the success status in a separate field per defined user in the format "REFSTR of user":"Message" or "User Name": "Message".

```
{
"ResultMessages": {
    "421-2518-0": "User not found",
    "421-3-0": "Password successfully changed.",
    "CUSTOMER": "Password successfully changed.",
    "CLIENTE": "Password successfully changed."
    }
}
```

Anonymizing User Data For Selected Users

This endpoint can be used to anonymize the data of one or multiple selected Alfabet users. The endpoint will only anonymize the data if the following preconditions are met:

- Anonymization is enabled in the Alfabet database for the object class `Person`. For more information about the anonymization feature and the required configuration to enable it, see *Anonymizing Data* in the reference manual *Configuring Alfabet with Alfabet Expand*.

- The user is not configured to be excluded from anonymization. For more information about excluding users from anonymization, see *Excluding Users from Anonymization* in the reference manual *Configuring Alfabet with Alfabet Expand*.

- The Alfabet user used to execute the call has the required access permission **Has AnonymizeUser Access**. For more information about access permissions for REST API service calls, see Generating a REST API Password for a User.

⚠️ Data anonymization is an irreversible action!

Endpoint name: `anonymizeuser`

HTTP method: `PUT`

Service call:

> *ServerAdress*`/api/v2/anonymizeuser`

Header Fields:

> `Authorization:`*TypeValue*

ℹ️ For information about the required Authorization header content, see Authorization.

> `Content-Type: application/json; charset=utf-8`

Payload

The payload is a JSON object with the following structure:

> `{`
>
> `"Refs": ["421-5-0","421-9-0"]`
>
> `}`

It may have the following fields:

Field	Manda-tory/Op-tional	Required Value
`"Refs":[" RefstrOfOb-ject "," RefstrOfOb-ject "]`	Mandatory	An array containing one or multiple values of the `REFSTR` property of objects of the object class `Person` with the stereotype `User` in the Alfabet database. All users included into the array will be anonymized.

Return value:

The return value is a JSON object with three fields:

- `Count`: The number of users that have been anonymized.

- `RejectedObjects`: A JSON list of objects that were not anonymized, for example because the user is excluded from anonymization. The field contains a JSON list of objects with one JSON object for each rejected Alfabet object. The object has a field `RefStr` that returns the `REFSTR` of the Alfabet object and a field `Message` that gives information about the rejection.

👁️
> `{`
>
> `"Count": 3,`

```
"RejectedObjects":{

    "RefStr": "421-999-0",

    "Message": "Cannot find instance"

    }

}
```

Starting a Workflow via RESTful Service Call

This endpoint can be used to start a workflow that is configured to be automatically started and to allow start via REST API.

Endpoint name: `workflow`

HTTP method: `POST`

Service call:

     ServerAdress`/api/v2/workflow`

Header Fields:

     `Authorization:`*TypeValue*

> For information about the required Authorization header content, see Authorization.

     `Content-Type: application/json; charset=utf-8`

Payload

The payload is a JSON object with the following structure:

```
{

"WorkflowTemplate": "WorkflowTemplateName"

}
```

It may have the following fields:

Field	Mandatory/Op-tional	Required Value
`"WorkflowTemplate":" WorkflowTem-plateName "`	Mandatory	The name of the workflow template that shall be started.

Return value:

The return value is a JSON object that can show the following fields:

-    `ResultMessage`: A message informing about the number of workflows that have been started.

- `ErrorMessage`: If starting of the workflow fails, this field informs about the reason for that.

- `ErrorCode`: If starting of the workflow fails, this field returns the error code for the error. This information is only relevant if you can not fix the problem because of the information in the error message. You can then send the error code and error message to your system administrator for support.

```
{
    "ResultMessage": "3 new workflows have been created."
}
```

Starting an ADIF Import via RESTful Service Call

A set of endpoints can be used to start an ADIF import based on an ADIF import scheme stored in the Alfabet database targeted by the REST API call.

The following attribute settings are required on the ADIF import scheme to perform execution via the RESTful services:

- **Applicable for REST API**: Set to `True`.

- **Estimated Execution Time**: Enter the estimated execution time for the ADIF job in minutes. The estimated execution time will be added to the return value of the RESTful service call triggering the execution to inform the client side about the estimated wait time until a result for the triggered ADIF import can be expected and requested via RESTful service call.

If the ADIF import scheme is configured to import from file, the file can either be streamed with the RESTful service call or uploaded to the **Internal Document Selector** of the Alfabet database prior to starting the ADIF import via the RESTful service call.

If data shall be imported from a file the following preconditions must be met:

- Import can be performed from one or multiple files. All files required for import must be added to a single ZIP archive.

- If the file for import shall be located in the Internal Document Selector, the option **Open Items** must be activated in the **Default Access Permissions** attribute of the folder in the **Internal Document Selector**. For information about setting the required access permissions, see *Uploading Documents and Managing User Permissions to Document Folders in the Internal Document Selector* in the reference manual *User and Solution Administration*.

- If the file for import shall be located in the **Internal Document Selector**, the file can be uploaded to the **Internal Document Selector** via a RESTful service call to the `idocupload` endpoint.

  > Alternatively, documents can be uploaded via the Internal Documents functionality on the Alfabet user interface. For information about uploading documents to the **Internal Document Selector**, see *Uploading Documents and Managing User Permissions to Document Folders in the Internal Document Selector* in the reference manual *User and Solution Administration*.

ADIF import via RESTful service call is either executed asynchronously or synchronously and requires a number of consecutive calls to the following endpoints in the given order. Details about the endpoints are given in the following sections.

⚠️ Please note that the Alfabet Web Application must be configured to connect to a running Alfabet Server to execute ADIF jobs via the RESTful services. For more information, see the reference manual *System Administration*.

1) Schedule asynchronous execution of ADIF import via a call to the `adifimport` endpoint. A session token is returned.

   ADIF import from a file on the local file system and ADIF import from either an external database or from a file in the **Internal Document Selector** of the Alfabet database require different settings for the call to the `adifimport` endpoint and are therefore described separately in the following.

2) Check the execution status for the ADIF import with the returned session token via the `adifsessionresult` endpoint.

3) If the status returned by the call to the `adifsessionresult` endpoint is `FAILED` or `WARNING`, check the log file of the ADIF execution via the endpoint `adiflog`.

The following sections inform about the required calls:

- [Triggering ADIF Import from an External Database or a Document in the Alfabet Database](#)

- [Triggering ADIF Import from a File Stream in the Service Call](#)

- [Checking ADIF Execution Result Status](#)

- [Downloading the Log File for ADIF Execution](#)

Triggering ADIF Import from an External Database or a Document in the Alfabet Database

Endpoint name: `adifimport`

HTTP method: `POST`

Service call:

> *ServerAdress*`/api/v2/adifimport`

Header Fields:

> `Authorization:`*TypeValue*

ℹ️ For information about the required Authorization header content, see [Authorization](#).

> `Content-Type: application/json; charset=utf-8`

Payload

The payload is a JSON object with the following structure:

> `{`

```
"Scheme": "ADIFImportSchemeName",

"UserArgs": {"@AppRef":"76-2760-0"},

"ImportfileName":"IDOC:\\FolderName\\FileName.zip";

"Verbose": false

}
```

It may have the following fields:

Field	Manda-tory/Op-tional	Required Value
`"Scheme":" ADIFImportSchemeName "`	Mandatory	The name of the ADIF import scheme that shall be executed.
`"UserArgs": {" arg1name ":" arg1value "," arg2name ": " arg2value "}`	Optional, only required if the ADIF scheme uses variables.	When the ADIF scheme is configured to use variables, the variables can be specified in the field `UserArgs` as a JSON object with one field for each variable. The field name must be identical to the variable name and the field value defines the variable value for the current execution of the ADIF import scheme.
`"ImportFileName":"IDOC:\\ Folder-Name\\FileName.zip"`	Optional, only required if the data is imported from a file in the **Internal Document Selector** of the Alfabet database.	If data shall be imported from file, the path to the file and the file name in the **Internal Document Selector** of the Alfabet database. The path must start with `IDOC:\` and backslashes must be used between folder names. Please note the following:that • Backslashes must be excaped with another backslash in JSON. If you are including the JSON into code when implementing a RESTful client, further escaping of the backslashes required by JSON may be required for code execution. You might require setting four backslashes between folder names.

Field	Manda-tory/Op-tional	Required Value
`"Verbose": true/false`	Optional, de-fault is `false`	If this field is set to `true`, additional information about the running process is logged in the logging information in the `ALFA_ADIF_SESSION` table. NOTE: Verbose logging is in most cases not required and can lead to a decrease in performance.

Return value:

The return value is a JSON object in the following format:

```
{
"Successful": true,
"SessionToken": "BA7557F7581F46609BCC338E9DBBA96E",
"EstimatedExecutionTime": 0,
"ResultMessage": "ADIF scheme has been sent to the Alfabet Server for asynchronous execution."
}
```

The following information is returned:

- `SessionToken`: Copy this token and use it in the calls to the `adifsessionresult` and `adiflog` endpoints to request information about the success of the ADIF import execution.

- `EstimatedExecutionTime`: Returns the estimated execution time in minutes defined in the ADIF scheme. A call to the `adifsessionresult` or `adiflog` endpoints should be scheduled after the estimated execution time only.

- `Successful`: Returns `true` if the ADIF import has been successfully scheduled for asynchronous execution. For information about the success of the ADIF import execution itself you must send a separate request to the `adifresult` endpoint.

- `ResultMessage`: Returns a description of the action that has been executed successfully.

- `ErrorMessage`: If the call fails, the return value will include information about the reason for the failure in this field.

- `ErrorCode`: If the call fails, the return value will include the web server error code in this field.

Please note that trying to start an ADIF export scheme with the `adifimport` endpoint will fail. The error message does not inform about the wrong type of scheme but states that the ADIF scheme is not found. This is due to the fact that the REST API searches for the ADIF scheme name in the subset of ADIF import schemes only and therefore the ADIF export scheme is not found.

Triggering ADIF Import from a File Stream in the Service Call

Endpoint name: `adifimport`

HTTP method: `POST`

Service call:

> *ServerAdress*`/api/v2/adifimport`

Header Fields:

> `Authorization:`*TypeValue*

> (i) For information about the required Authorization header content, see Authorization.

> `Content-Type: multipart/form-data; charset=utf-8;`

Payload:

The payload consists of two parts:

1) A JSON object providing details about the ADIF import.

2) The content of the import ZIP file.

> The way the payload is added depends on the type of client or client code.
>
> For RESTful service client applications and object oriented programming languages, the content is added as two separate files: a JSON file containing the JSON object and the ZIP file containing the files for import.
>
> For clients implemented in string oriented programming languages, the payload needs to start and end with a delimiter defined in the Content-Type header field (`Content-Type: multi-part/form-data; charset=utf-8; delimiter: DelimiterStringNotPartOfAnyContent`), and the delimiter must also be used to separate the two parts of the payload with the JSON directly defined between the delimiters and the import ZIP file provided as file stream. The two parts of the payload must be the following, defined in the given order:
>
> > `---`*DelimiterStringNotPartOfAnyContent*`--`
> >
> > `{`
> >
> > `"Scheme": "ADIFImportSchemeName",`
> >
> > `"ImportFileName": "FileName.zip",`
> >
> > `"Verbose": true`
> >
> > `}`
> >
> > `---`*DelimiterStringNotPartOfAnyContent*`--`
> >
> > *file content stream data*
> >
> > `---`*DelimiterStringNotPartOfAnyContent*`--`

The JSON may have the following fields:

Field	Manda-tory/Op-tional	Required Value
`"Scheme":" ADIFImportSchemeName "`	Mandatory	The name of the ADIF import scheme that shall be executed.

`"UserArgs":`
`{" arg1name ":" arg1value "," arg2name ":`
`" arg2value "}`
	Optional, only required if the ADIF scheme uses variables.	When the ADIF scheme is configured to use variables, the variables can be specified in the field `UserArgs` as a JSON object with one field for each variable. The field name must be identical to the variable name and the field value defines the variable value for the current execution of the ADIF import scheme.

`"ImportFileName":" FileName.zip"`	Mandatory	The name of the import file for the RESTful service call.
`"Verbose": true/false`	Optional, default is `false`.	If this field is set to `true`, additional information about the running process is logged in the logging information in the `ALFA_ADIF_SESSION` table. NOTE: Verbose logging is in most cases not required and can lead to a decrease in performance.

Return value:

The return value is a JSON object in the following format:

```
{
"Successful": true,
"SessionToken": "BA7557F7581F46609BCC338E9DBBA96E",
"EstimatedExecutionTime": 0,
"ResultMessage": "ADIF scheme has been sent to the Alfabet Server for
asynchronous execution."
}
```

The following information is returned:

- `SessionToken`: Copy this token and use it in the calls to the `adifsessionresult` and `adiflog` endpoints to request information about the success of the ADIF import execution.

- `EstimatedExecutionTime`: Returns the estimated execution time in minutes defined in the ADIF scheme. A call to the `adifsessionresult` or `adiflog` endpoints should be scheduled after the estimated execution time only.

- `Successful`: Returns `true` if the ADIF import has been successfully scheduled for asynchronous execution. For information about the success of the ADIF import execution itself you must send a separate request to the `adifresult` endpoint.

- `ResultMessage`: Returns a description of the action that has been executed successfully.

- `ErrorMessage`: If the call fails, the return value will include information about the reason for the failure in this field.

- `ErrorCode`: If the call fails, the return value will include the web server error code in this field.

Please note that trying to start an ADIF export scheme with the `adifimport` endpoint will fail. The error message does not inform about the wrong type of scheme but states that the ADIF scheme is not found. This is due to the fact that the REST API searches for the ADIF scheme name in the subset of ADIF import schemes only and therefore the ADIF export scheme is not found.

Checking ADIF Execution Result Status

Endpoint name: `adifsessionstatus`

HTTP method: `POST`

Service call:

> *ServerAdress*`/api/v2/adifsessionstatus`

Header Fields:

> `Authorization:`*TypeValue*

> (i) For information about the required Authorization header content, see [Authorization](#).

> `Content-Type: application/json; charset=utf-8`

Payload

The payload is a JSON object with the following structure:

> `{`
>
> `"SessionToken": "`*Session Token Returned via the adifimport call*`"`
>
> `}`

Field	Manda-tory/Op-tional	Required Value
`"SessionToken": " Session Token Returned via the adi-fimport call "`	Mandatory	The session token from the return value of the call to the `adifimport` endpoint that triggered the ADIF import execution.

Return value:

The return value is a JSON object in the following format:

```
{
"SessionID": "RPC_20200624182254131",
"SchemeName": "ADIF Scheme Name",
"Status": "SUCCESS",
"Start": "2020-06-24T18:22:54.183",
"EstimatedExecutionTime": 0,
"End": "2020-06-24T18:22:54.24",
"IDOCPath": "",
"IDOCFileName": "EXPORTTEST.ZIP",
"Log": true,
"Count": 0
}
```

The following information is returned:

- `SessionID`: The ID of the ADIF execution session.

- `SchemeName`: The name of the executed ADIF export scheme.

- `EstimatedExecutionTime`: Returns the estimated execution time in minutes defined in the ADIF import scheme. A call to the `adifsessionresult` or `adiflog` endpoints should be scheduled after the estimated execution time only.

- `Status`: Returns `SUCCESS` if the ADIF execution was successful, `FAILED` if an error occured during execution, `WARNING` if a warning message was written to the log file during execution, and `STARTED` during execution.

- `Start`: Returns the date and time at which the ADIF execution started.

- `End`: Returns the date and time at which the ADIF execution ended.

- `IDOCPath`: This field is not relevant.

- `IDOCFileName`: If data is imported from a file streamed in the RESTful service call, the file name is returned. If data is imported from a file in the **Internal Document Selector**, the file name including the path to the file in the **Internal Document Selector** is returned.

- `Log`: Returns `true` if a log file is available. The log file can be downloaded via a call to the `adiflog` endpoint.

- `Count`: This field is not relevant.

Downloading the Log File for ADIF Execution

Endpoint name: `adiflog`

HTTP method: `POST`

Service call:

    ServerAdress`/api/v2/adiflog`

Header Fields:

    `Authorization:`*TypeValue*

> For information about the required Authorization header content, see [Authorization](#).

    `Content-Type: application/json; charset=utf-8`

Payload

The payload is a JSON object with the following structure:

```
{
"SessionToken": "Session Token Returned via the adifimport call"
}
```

Field	Manda-tory/Op-tional	Required Value
`"SessionToken": " Session Token Returned via the adi-fimport call "`	Mandatory	The session token from the return value of the call to the `adifimport` endpoint that triggered the ADIF import execution.

Return value:

The return value is the content of the log file.

Starting an ADIF Export via RESTful Service Call

A set of endpoints can be used to start an ADIF export based on an ADIF export scheme stored in the Alfabet database targeted by the REST API call.

The following attribute settings are required on the ADIF export scheme to perform execution via the REST-ful services:

- **Applicable for REST API**: Set to `True`.

- **Estimated Execution Time**: Enter the estimated execution time for the ADIF job in minutes. The estimated execution time will be added to return value of the RESTful service call triggering the execution to inform the client side about the estimated wait time until a result for the triggered ADIF export can be expected and requested via RESTful service call.

If the ADIF export scheme is configured to export to file, the generated file or files are stored in the **Internal Document Selector** of the Alfabet database.

> (i) For information about the **Internal Document Selector**, see *Uploading Documents and Managing User Permissions to Document Folders in the Internal Document Selector* in the reference manual *User and Solution Administration* and *Making Documents and Files Available to the Alfabet User Community* in the reference manual *System Administration*.

ADIF export via RESTful service call is executed asynchronously and requires a number of consecutive calls to the following endpoints in the given order. Details about the endpoints are given in the following sections.

> ⚠ Please note that the Alfabet Web Application must be configured to connect to a running Alfabet Server to execute ADIF jobs via the RESTful services. For more information, see XXX in the reference manual *System Administration*.

1) Schedule asynchronous execution of the ADIF export via a call to the `adifexport` endpoint. A session token is returned.

2) Check the execution status for the ADIF export with the returned session token via the `adifsessionresult` endpoint. If ADIF export to file is executed, the return value of the call will provide information about the location of the generated file in the **Internal Document Selector**.

3) If the status returned by the call to the `adifsessionresult` endpoint is `FAILED`, check the log file of the ADIF execution via the `adiflog` endpoint.

4) If ADIF export to file has been successfully executed, you can download the file via the endpoint `idocdownload`.

The following sections inform about the required calls:

- [Triggering Asynchronous Execution of ADIF Export](#)

- [Checking ADIF Execution Result Status](#)

- [Downloading the Log File for ADIF Execution](#)

- [Downloading Documents from the Internal Document Selector](#)

Triggering Asynchronous Execution of ADIF Export

Endpoint name: `adifexport`

HTTP method: `POST`

Service call:

> *ServerAdress*/api/v2/adifexport

Header Fields:

> Authorization:*TypeValue*

(i) For information about the required Authorization header content, see Authorization.

> Content-Type: application/json; charset=utf-8

Payload

The payload is a JSON object with the following structure:

> {
>
> "Scheme": "ADIFExportSchemeName",
>
> "UserArgs": {"@AppRef":"76-2760-0"},
>
> "Verbose": false,
>
> }

It may have the following fields:

Field	Mandatory/Optional	Required Value
"Scheme":" ADIFExportSchemeName "	Mandatory	The name of the ADIF export scheme that shall be executed.
"UserArgs": {" arg1name ":" arg1value "," arg2name ": " arg2value "}	Optional, only required if the ADIF scheme uses variables.	When the ADIF scheme is configured to use variables, the variables can be specified in the field UserArgs as a JSON object with one field for each variable. The field name must be identical to the variable name and the field value defines the variable value for the current

Field	Mandatory/Optional	Required Value
		execution of the ADIF export scheme.
`"Verbose": true/false`	Optional, default is `false`	If this field is set to `true`, additional information about the running process is logged in the logging information in the `ALFA_ADIF_SESSION` table. **NOTE**: Verbose logging is in most cases not required and can lead to a decrease in performance.
`"FolderPath":"IDOC:\\FolderName"`	Optional, for export to file the default is `IDOC:\\ADIF_SYS`	If the executed ADIF scheme is exporting to file, the export result will be stored in the **Internal Document Selector** in the folder defined with this field with the file name defined in the `ExportFileName` field. The path must start with `IDOC:\` and backslashes must be used between folder names. Please note that backslashes must be escaped with another backslash in JSON. If you are including the JSON

Field	Mandatory/Optional	Required Value
		into code when implementing a RESTful client, further escaping of the backslashes required by JSON may be required for code execution. You might require setting four backslashes between folder names. It is not possible to upload documents directly into the IDOC:\ root.
"ExportFileName":"Filename.zip"	Optional, the default is <ADIFSchemeName>_<Timestamp>.ZIP	If the executed ADIF scheme is exporting to file, the export result will be stored in the **Internal Document Selector** in the folder defined in the `ExportFileName` field with the file name defined in this field. If the file already exists in the **Internal Document Selector**, it will be overwritten.

Return value:

The return value is a JSON object in the following format:

```
{
"ExportFile": "",
"Successful": true,
"SessionToken": "BA7557F7581F46609BCC338E9DBBA96E",
```

```
    "EstimatedExecutionTime": 0,

    "ResultMessage": "ADIF scheme has been sent to the Alfabet Server for
    asynchronous execution."

    }
```

The following information is returned:

- `SessionToken`: Copy this token and use it in the calls to the `adifsessionresult` and `adiflog` endpoints to request information about the success of the ADIF export execution.

- `EstimatedExecutionTime`: Returns the estimated execution time in minutes defined in the ADIF scheme. A call to the `adifsessionresult` or `adiflog` endpoints should be scheduled after the estimated execution time only.

- `Successful`: Returns `true` if the ADIF export has been successfully scheduled for asynchronous execution. For information about the success of the ADIF export execution itself you must send a separate request to the `adifresult` endpoint.

- `ResultMessage`: Returns a description of the action that has been executed successfully.

- `ExportFile`: This field is not relevant.

- `ErrorMessage`: If the call fails, the return value will include information about the reason for the failure in this field.

- `ErrorCode`: If the call fails, the return value will include the web server error code in this field.

Please note that trying to start an ADIF import scheme with the `adifexport` endpoint will fail. The error message does not inform about the wrong type of scheme but states that the ADIF scheme is not found. This is due to the fact that the REST API searches for the ADIF scheme name in the subset of ADIF export schemes only and therefore the ADIF import scheme is not found.

Checking ADIF Execution Result Status

Endpoint name: `adifsessionstatus`

HTTP method: `POST`

Service call:

> *ServerAdress*`/api/v2/adifsessionstatus`

Header Fields:

> `Authorization:`*TypeValue*

(i) For information about the required Authorization header content, see Authorization.

> `Content-Type: application/json; charset=utf-8`

Payload

The payload is a JSON object with the following structure:

```
    {
```

```
"SessionToken": "Session Token Returned via the adifexport call"

}
```

Field	Manda- tory/Op- tional	Required Value
`"SessionToken": " Session Token Returned via the adifexport call "`	Mandatory	The session token from the return value of the call to the `adifexport` endpoint that triggered the ADIF export execution.

Return value:

The return value is a JSON object in the following format:

```
{
"SessionID": "RPC_20200624182254131",
"SchemeName": "ADIFExportSchemeName",
"Status": "SUCCESS",
"Start": "2020-06-24T18:22:54.183",
"EstimatedExecutionTime": 0,
"End": "2020-06-24T18:22:54.24",
"IDOCPath": "IDOC:\\DOCUMENTS",
"IDOCFileName": "EXPORTTEST.ZIP",
"Log": true,
"Count": 0
}
```

The following information is provided:

- `SessionID`: The ID of the ADIF execution session.

- `SchemeName`: The name of the executed ADIF export scheme.

- `EstimatedExecutionTime`: Returns the estimated execution time in minutes defined in the ADIF scheme. A call to the `adifsessionresult` or `adiflog` endpoints should be scheduled after the estimated execution time only.

- `Status`: Returns `SUCCESS` if the ADIF execution was successful, `FAILED` if an error occured during execution, `WARNING` if a warning message was written to the log file during execution, and `STARTED` during execution.

- `Start`: Returns the date and time at which the ADIF execution started.

- `End`: Returns the date and time at which the ADIF execution ended.

- `IDOCPath`: If the ADIF scheme is configured to export data to a file, the file is stored in the **Internal Document Selector** of the Alfabet database. The path within the **Internal Document Selector** will then be returned in this field. The information can be used together with the information returned in the `IDOCFileName` field to download the results via a call to the `idocdownload` endpoint. For more

information about the download of the document, see Downloading Documents from the Internal Document Selector.

- `IDOCFileName`: If the ADIF scheme is configured to export data to a file, the file is stored in the **Internal Document Selector** of the Alfabet database. The name of the file will then be returned in this field. The information can be used together with the information returned in the `IDOCPath` field to download the results via a call to the `idocdownload` endpoint. For more information about the download of the document, see Downloading Documents from the Internal Document Selector.

- `Log`: Returns `true` if a log file is available. The log file can be downloaded via a call to the `adiflog` endpoint.

- `Count`: This field is not relevant.

Downloading the Log File for ADIF Execution

Endpoint name: `adiflog`

HTTP method: `POST`

Service call:

> *ServerAdress*/api/v2/adiflog

Header Fields:

> Authorization:*TypeValue*

> (i)   For information about the required Authorization header content, see Authorization.

> Content-Type: application/json; charset=utf-8

Payload

The payload is a JSON object with the following structure:

> {
>
> "SessionToken": "*Session Token Returned via the adifexport call*"
>
> }

Field	Manda-tory/Op-tional	Required Value
"SessionToken": " Session Token Returned via the adifexport call "	Mandatory	The session token from the return value of the call to the `adifexport` endpoint that triggered the ADIF export execution.

Return value:

The return value is the content of the log file.

Exporting Information about the Content of the Internal Document Selector

This endpoint can be used to export information about the file content of folders in the **Internal Document Selector** of the Alfabet database. Only files in one selected folder can be listed via a call to the endpoint. Sub-folders of the selected folder and the file content thereof are not included in the list.

> ⓘ For information about the **Internal Document Selector**, see *Uploading Documents and Managing User Permissions to Document Folders in the Internal Document Selector* in the reference manual *User and Solution Administration* and *Making Documents and Files Available to the Alfabet User Community* in the reference manual *System Administration*.

Endpoint name: `idocfilelist`

HTTP method: `POST`

Service call:

> *ServerAdress*`/api/v2/idocfilelist`

Header Fields:

> `Authorization:`*TypeValue*

> ⓘ For information about the required Authorization header content, see [Authorization](#).

> `Content-Type: application/json; charset=utf-8`

Payload:

The payload is a JSON object with the following structure:

```
{
"Path": "IDOC:\\Folder\\Subfolder"
}
```

It may have the following fields:

Field	Manda-tory/Op-tional	Required Value
`"Path":" Path to folder in IDOC "`	Mandatory	The path to the IDOC folder the list shall be created for. The path must start with `IDOC:\` and backslashes must be used between folder names.
		Please note that backslashes must be escaped with another backslash in JSON.
		If you are including the JSON into code when implementing a RESTful client, further escaping of the backslashes required by

Field	Manda-tory/Op-tional	Required Value
		JSON may be required for code execution. You might require setting four backslashes between folder names.

Return value:

The return value is a JSON list of objects with one object per document that informs about the document with the following fields:

- `Path`: The complete path of the document starting from the root of the Internal Document Selector.

- `Name`: The name of the document.

- `CreationDate`: The date and time the document was uploaded into the Internal Document Selector.

- `DateModified`: The date and time the document was last changed in the Internal Document Selector.

- `Size`: The size of the document in byte.

```
{
  "Files": [
    {
      "Path": "IDOC:\\Documents\\CC_Corporate_FI-CO.pdf",
      "Name": "CC_Corporate_FI-CO.pdf",
      "CreationDate": "2008-10-03T13:45:48.5",
      "DateModified": "2010-04-04T13:45:48",
      "Size": 24117
    },
    {
      "Path": "IDOC:\\Documents\\CC_TradeWeb.pdf",
      "Name": "CC_TradeWeb.pdf",
      "CreationDate": "2007-08-30T13:46:09.14",
      "DateModified": "2009-02-28T13:46:09",
      "Size": 24892
    }
  ]
}
```

Downloading Documents from the Internal Document Selector

This endpoint can be used to download files from the Internal Document Selector of the Alfabet database or one of its sub-folders.

> (i) For information about the Internal Document Selector, see *Uploading Documents and Managing User Permissions to Document Folders in the Internal Document Selector* in the reference manual *User and Solution Administration* and *Making Documents and Files Available to the Alfabet User Community* in the reference manual *System Administration*.

Endpoint name: `idocdownload`

HTTP method: `POST`

Service call:

> *ServerAdress*`/api/v2/idocdownload`

Header Fields:

> `Authorization:`*TypeValue*

> (i) For information about the required Authorization header content, see Authorization.

> `Content-Type: application/json; charset=utf-8`

Payload:

The payload is a JSON object with the following structure:

> `{`
>
> `"Path": "IDOC:\\Folder\\Subfolder\\FileName.FileExtension"`
>
> `}`

It may have the following fields:

Field	Manda-tory/Op-tional	Required Value
`"Path":" Path to file in IDOC "`	Mandatory	The path to the IDOC file that shall be downloaded including the file name. The path must start with `IDOC:\` and backslashes must be used between folder names. Please note that backslashes must be excaped with another backslash in JSON. If you are including the JSON into code when implementing a RESTful client, further escaping of the backslashes required by JSON may be required for code execution. You might require setting four backslashes between folder names.

Return value:

The return value of the successful service call provides the downloaded file in a stream.

Uploading Documents to the Internal Document Selector

This endpoint can be used to upload a file to the **Internal Document Selector** of the Alfabet database or one of its sub-folders.

> For information about the **Internal Document Selector**, see *Uploading Documents and Managing User Permissions to Document Folders in the Internal Document Selector* in the reference manual *User and Solution Administration* and *Making Documents and Files Available to the Alfabet User Community* in the reference manual *System Administration*.

> Upload of files into the **Internal Document Selector** can also be performed directly for ADIF export to file executed via the RESTful service endpoint `adifexport`. The exported content is then not provided for download, but directly stored in the **Internal Document Selector**. For more information, see Starting an ADIF Export via RESTful Service Call.

Endpoint name: `idocupload`

HTTP method: `POST`

Service call:

> *ServerAdress*`/api/v2/idocupload`

Header Fields:

> `Authorization:`*TypeValue*

> For information about the required Authorization header content, see Authorization.

> `Content-Type: multipart/form-data; charset=utf-8;`

Payload:

The payload consists of two parts in the given order:

1) A JSON object providing details about the storage of the file in the Internal Document Selector.

2) The file to be uploaded.

> The way the payload is added depends on the type of client or client code.
>
> For RESTful service client applications and object oriented programming languages, the content is added as two separate files: a JSON file containing the JSON object and the ZIP file containing the files for importfile for upload.
>
> For clients implemented in string oriented programming languages, the payload needs to start and end with a delimiter defined in the Content-Type header field (`Content-Type: multipart/form-data; charset=utf-8; delimiter: DelimiterStringNotPartOfAnyContent`), and the delimiter must also be used to separate the two parts of the payload with the JSON directly defined between the delimiters and the import data provided as file stream. The two parts of the payload must be the following, defined in the given order:

```
---DelimiterStringNotPartOfAnyContent--

{

"FolderPath": "IDOC:\\ImportFolder",

"Name": "UploadFile.pdf",

""OverwriteExistingFile": true

}

---DelimiterStringNotPartOfAnyContent--

file content stream data

---DelimiterStringNotPartOfAnyContent--
```

The JSON may have the following fields:

Field	Manda-tory/Op-tional	Required Value
`"FolderPath":" Path to folder in IDOC "`	Mandatory	The path to the IDOC folder the file should be uploaded to. The path must start with `IDOC:\` and backslashes must be used between folder names. Please note the following:that • Backslashes must be excaped with another backslash in JSON. If you are including the JSON into code when implementing a RESTful client, further escaping of the backslashes required by JSON may be required for code execution. You might require setting four backslashes between folder names. • It is not possible to upload files to the `IDOC:` root. • The file name in the internal document selector does not have to be identical with the name of the file that is uploaded.
`"Name":" File-Name.FileExten-tion "`	Mandatory	The name of the file that shall be created in IDOC. The name does not have to be identical to the name of the file providing the content for upload. The file extension must match the up-loaded content. Please note that the upload will fail if a file with the same name already exists in the defined Internal Document Selector folder and the field `OverwriteExistingFile` is missing or set to `false`.
`"OverwriteExisting-File":true|false`	Optional	If this field is set to `true`, an existing file with the same name in the same folder of the Internal Document Selector will be over-written by the uploaded file. If set to `false`, an existing file can-not be overwritten and upload will fail if a file with the same

Field	Manda-tory/Op-tional	Required Value
		name already exists in the defined Internal Document Selector folder. The default value is `false`.

Return value:

If the upload is successful, the return value will be a JSON object returning the IDOC ID if the uploaded document in a field `IDocID`.

Checking Whether the Alfabet components are Running

This endpoint can be used to check whether the Alfabet components that might be involved in the execution of a RESTful service request are running and can be accessed. The service call checks the availability of the following components:

- The Alfabet Web Application.

- The database server hosting the Alfabet database,

- If the server alias of the Alfabet Web Application is configured to connect to an Alfabet Server, the availability of that Alfabet Server is checked.

Endpoint name: `monitor`

HTTP method: `GET`

Service call:

> *ServerAdress*`/api/monitor`

Header Fields:

> `Authorization:`*TypeValue*

> For information about the required Authorization header content, see Authorization.

> `Content-Type: application/json; charset=utf-8`

Payload

No payload required.

Return value:

The return value is a JSON object that informs about the availability of the components.

> For example:

> `{`

```
"Name": "Alfabet WebApplication",

"State": "Error",

"Reason": "Alfabet WebApplication is running Database is running
Alfabet Application Server error: Unable to connect to the
application server. Check the network connections and resources.
Contact your administrator."

}
```

Updating the Meta-Model

This endpoint can be used to update the Alfabet meta-model in the target database with a meta-model configuration stored in an AMM file.

> ⓘ For information about the creation of AMM files and the update of the meta-model, see *Applying Configuration Changes to Other Databases* in the reference manual *Configuring Alfabet with Alfabet Expand*.

> ⚠ Please note that the Alfabet Web Application must be configured to connect to a running Alfabet Server to execute ADIF jobs via the RESTful services. For more information, see XXX in the reference manual *System Administration*.

Endpoint name: updateMM

HTTP method: POST

Service call:

> *ServerAdress*/api/v2/updateMM

Header Fields:

> Authorization:*TypeValue*

> ⓘ For information about the required Authorization header content, see Authorization.

> Content-Type: multipart/form-data; charset=utf-8;

Payload:

The content of the AMM file.

> The way the payload is added depends on the type of client or client code.
>
> For RESTful service client applications and object oriented programming languages, the content is added as two separate files: a JSON file containing the JSON object and the AMM file.
>
> For clients implemented in string oriented programming languages, the payload needs to start and end with a delimiter defined in the Content-Type header field (Content-Type: multipart/form-data; charset=utf-8; delimiter: DelimiterStringNotPartOfAnyContent), and the delimiter must also be used to separate the two parts of the payload with the

JSON directly defined between the delimiters and the AMM file provided as file stream. The two parts of the payload must be the following, defined in the given order:

```
---DelimiterStringNotPartOfAnyContent--

{

???

"Name": "FileName.amm",

"Verbose": true

}

---DelimiterStringNotPartOfAnyContent--

file content stream data

---DelimiterStringNotPartOfAnyContent--
```

Return value:

The return value is a JSON object in the following format:

```
{

"ResultMessage": "Configuration File has been sent to the Alfabet Server for
update.",

"Log": "",

}
```

The following information is returned:

- `ResultMessage`: Informs about the success of the service call. The service call hands over the AMM file to the Alfabet Server. The update of the meta-model is then performed by the Alfabet Server following the RESTful service call.

- `Log`: This field is not relevant.

Chapter 7:   Accessing the Alfabet User Interface From the External Application

The Alfabet RESTful API provides access to data in the Alfabet database to external applications processing the data. If the external application should also provide links to the Alfabet user interface to its users, links opening defined views of the Alfabet user interface can be defined using a special link syntax. The links point to either an object profile, object cockpit, or graphic view for a specified object. In the link, the `REFSTR` value of the object for that the view is opened must be provided. The `REFSTR` value of a relevant object can be requested via the Alfabet RESTful API in order to build the link. The endpoint `objects` can deliver a list of `REFSTR` values for all objects found via a query in a configured Alfabet report.

The required link syntax and the access rights that apply for links to the Alfabet user interface from external applications are described in detail in the section *Links to Alfabet Views from External Applications* in the reference manual *System Administration*.

Chapter 8:   Testing the Alfabet RESTful API

For first tests of the Alfabet RESTful API, requests can be sent via a commercial REST client like Google's Advanced REST client or a Swagger editor.

For both the tests via a commercial REST client or tests of customer developed REST clients, tests for service calls for the endpoint update should only be performed on Alfabet test databases and not on the productive Alfabet database.

This chapter provides information about how to perform a simple test of a service call and how to interpret error codes that you might see during testing:

- Testing the Alfabet RESTful API

    - Configurations Required to Use a Swagger Editor for Testing

Testing the Alfabet RESTful API

You can use commercially available REST clients for simple tests of service calls to the Alfabet RESTful API. This does not require any addition configurations except for Swagger editors.

Configurations Required to Use a Swagger Editor for Testing

A Swagger editor can be used for testing. To view all calls in ready-to-use Swagger format, the Alfabet RESTful services and the Web browser must be configured as follows:

1) Open the Alfabet Administrator.

2) Click the **Alfabet Aliases** node in the explorer. A workspace with a toolbar opens.

3) In the toolbar, click **Tools** > **Configure alfabet.config**. An editor opens.

4) Click the Browse ▦ button on the right of the **Web Folder** field and select the main directory of the Alfabet Web Application from the directory browser. The `alfabet.config` file in the subdirectory **config** of the selected directory opens in the editor.

5) Add the following code as child element of the `alfaSection` XML element, substituting `WebApplicationFolderPath` with the absolute path to the virtual directory of the Alfabet Web Application:

```
<add key="SwaggerSpecFileName"
value="WebApplicationFolderPath\SwaggerSpec\AlfabetWeb5_SwaggerSpec.json"
/>
```

6) Click **Save**. The change is saved and the editor is closed.

7) Close the Alfabet Administrator.

8) Go to the URL `http:// URLOfTheALfabetWebApplication /swagger/docs/v2` and copy the JSON code that is displayed.

9) Open your Swagger editor and paste the JSON code into the editor code field.

10) Make sure that **Cross Origin Resource Sharing** is enabled on your browser.

The header: Access-Control-Allow-Origin: * must be included in the calls generated by the Sever. Enabling Cross Origin Resource Sharing in the browser is a simple way of achieving this if not otherwise implemented in the system.

Chapter 9:   Checking Success of Service Calls to the Alfabet RESTful Api

The return value for service call of the RESTful services informs about the success of the service call. If an error occures, the error message informs about the reason for the failure of the call. The error messages are handed over in a JSON object with two fields:

- `ErrorMessage`: Information about the reason for the failure.

- `ErrorCode`: The web server error code.

For security reasons, the error messages delivered back to the REST client are not very detailed. In additions, service calls that trigger activity like ADIF execution asynchronously will only inform about the successful triggering of the process without giving information about the success of the execution of the ADIF job. The execution of processes triggered via the RESTful services can also be logged by the Alfabet Web Application for synchronous process or by the Alfabet Server for asynchronous processes.

The configuration of the Alfabet components to write log information to a central log file or to send it to an external log server is described in the section *Central Logging of Functionality for Alfabet Components* in the reference manual *System Administration*.

INDEX