One Identity Manager 9.0

API Development Guide

Legend

WARNING: A WARNING icon highlights a potential risk of bodily injury or property damage, for which industry-standard safety precautions are advised. This icon is often associated with electrical hazards related to hardware.

CAUTION: A CAUTION icon indicates potential damage to hardware or loss of data if instructions are not followed.

One Identity Manager API Development Guide
Updated - 01 August 2022, 14:23
Version - 9.0

Contents

Basic principles of API development

In this section, you will find information about API development basics.

Detailed information about this topic

API Server basics

In this chapter you will find basic information about the API Servers architecture, which is important for custom programming with your own API methods.

Detailed information about this topic

General information about the API Server

- The API Server deploys the API.
- The API Server is implemented using the Owin Platform (see http://owin.org/).
- URLs are case sensitive.

Calling the API Server web interface

From the API Server's web interface you can:

- Configure the API Server.
- Call the swagger documentation for your API.
- Open the Operations Support Web Portal.
- Call all installed web applications.

To call the API Server web interface

- In a web browser, open the webpage (URL) of your API Server.

Encryption

The API Server stores data securely encrypted on the client and in the database.

The certificate is configured when the API server is installed on the IIS.

For more information about configuring encryption, see the *One Identity Manager Web Application Configuration Guide*.

General notes on programming your own API methods

- Because the API Server is stateless, save the API methods without a client specific state.
  For example, you cannot, therefore, define global variables or store session object status data. When the API Server processes are restarted, these values are not restored.
- Access to current HTTP requirements over ASP.NET APIs is not supported.
- After enabling routes, you cannot change the definition objects anymore.
- Use asynchronous code for defining API methods. This supports more efficient usage of server resources and improves performance of the system under load. The methods of the API and the underlying object model convert this asynchronousity using the Task-based Asynchronous Pattern (TAP). For more information about TAP, see https://docs.microsoft.com/de-de/dotnet/standard/asynchronous-programming-patterns/task-based-asynchronous-pattern-tap.
- Do not use the **HttpContext.Current** method when you define API methods. You can query the current HTTP requirements with the **QBM.CompositionApi.ApiManager.Context.Current** static method.
- If you define API methods that modify data, do NOT use the **GET** method.

Guidelines and conventions

In this chapter, you will find general policies and conventions, which you must take into account when you create an API.

Detailed information about this topic

Handling of API Server queries

In this section, you will find information about handling queries that are sent to the API Server.

Authentication

When a query is sent to the API Server, there is a test to ascertain the success of the primary and, possibly, secondary login in the session for the current project (see Authentication on page 8).

> NOTE: This test is not done if the API method used by the query is marked as **AllowUnauthenticated**.

The **imx-session-<API project name>** cookie is evaluated to allocate the current session.

If a cookie is passed that cannot be associated with an active session in the current process, the security token in the cookie is used to set up a new session (see Session status and security tokens on page 10).

If there is no primary login, the API Server tries to establish a database connection through one of the enabled single sign-on authentication modules.

If login cannot be carried out, the process is canceled and the HTTP error code **500** is passed to the client (see Response codes on page 18).

Authorizing method access

The API Server checks whether the currently logged in user is authorized to run the method. If the user does not have the required permissions, the process is canceled and the HTTP error code **500** is passed to the client (see Response codes on page 18).

Validating the query

The API Server calls the validators stored with the API method one by one. If one fails, the process is canceled and the HTTP error code **400** is passed to the client (see Response codes on page 18).

Processing queries (for entity methods)

- GET (for loading entities)
  - Determines the WHERE clause with internal and external filters
  - Loads data from the database
  - Enriches entities with calculated columns
- Entities in delayed logic mode can be changed with a POST query or deleted with a DELETE query. Entities in this mode are stateless and do not occupy resources on the server after the query has been processed.
  Supported HTTP methods:
  - GET (for loading entities)
  - POST (for changing entities)
  - DELETE (for deleting entities)
- Interactive entities must be created once with a PUT query and after that they obtain their own ID. Use the ID in subsequent queries (POST or DELETE).

  Supported HTTP methods:
  - PUT (for creating interactive entities)
  - POST (for changing interactive entities)
  - DELETE (for deleting interactive entities)

Authentication

User authentication is carried out on the API Server for each API project.

Running an API method requires prior authentication on an API project. If the API method is marked as `AllowUnauthenticated`, authentication is not required (you can find an example in the SDK)

Authentication has two steps:

1. Required primary authentication: Default authentication through an authentication module
2. Optional secondary authentication: Multi-factor authentication (by OneLogin)

For more information about configuring authentication, see the *One Identity Manager Web Application Configuration Guide*.

Detailed information about this topic

- Authentication (primary) on page 9
- Logging out on page 9

Related topics

- Handling of API Server queries on page 7

Authentication (primary)

You can use the **imx/login/<API project name>** API method for primary authentication on the API project.

To do this, use the **POST** HTTP method to send a query containing the following:

```
{ "Module": "RoleBasedPerson", "User": "<user name>", "Password": "<password>" }
```

TIP: See the SDK for examples.

Security mechanisms

The API Server uses a security mechanism to prevent cross-site request forgery (XSRF) attacks. This randomly generates a token (**XSRF-TOKEN**) and sends it to the client in a cookie at login. The client must then transmit the value of this token in an HTTP header (**X-XSRF-TOKEN**) in each request sent to the server. If this header is missing, the request is terminated with error code **400**.

TIP: You can change the name of the cookie and HTTP header in the Administration Portal.

Logging out

You can use the **imx/logout/<API project name>** API method to log out of the API project.

To do this, use the **POST** HTTP method to send a query without content.

Session status and security tokens

The status a session is saved in a cookie. This cookie contains an encrypted security token which is used to restore a login to the API Server if the API Server was restarted in the mean time. The security token is cryptographically signed by the certificate selected on installation.

NOTE: If the API Server's current user restarts the browser, the cookie and its session information are reset.

Related topics

- Querying session status on page 10

Querying session status

You can use the **imx/sessions/<API project name>** API method to query the status of the session. The response contains the following information:

- Permitted authentication module and associated parameters of the respective API project.
- Type of secondary login

API methods

You can define the following types of API methods.

Entity methods

Entity methods work with small parts of the object model in order to read data from the database or write data to the database. When you create an entity method, you only need to enter the table and column name and, if required, a filter condition (WHERE clause). Internal processing is handled by the API Server. The data schema for the input and output also has a specific format.

For examples for the definition of entity methods, see the SDK under `Sdk01_Basics\01-BasicQueryMethod.cs`.

User-defined methods

User-defined methods are methods for which you fully define the processing, input, and output data in code. This type therefore offers the greatest flexibility.

For examples for the user-defined methods, see the SDK under `Sdk01_Basics\03-CustomMethod.cs`.

SQL methods

SQL methods are methods that provide data from a predefined SQL query through the API. Create the parameters of a query as SQL parameters.

For examples for the definition of SQL methods, see the SDK under `Sdk01_Basics\02-BasicSqlMethod.cs`.

Detailed information about this topic

- General notes and information about entity methods on page 11

General notes and information about entity methods

In this section, you will find advice and information for creating and implementing entity methods.

Limiting results

NOTE: Entity-based methods normally work with a limit to avoid unintentionally loading extremely large amounts of data.

The following query parameters help you to limit the amount of data that is returned by obtaining multiple data sets from sequential responses:

Query parameter	Default value	Description
PageSize	20	Specifies the maximum number of data sets that can be contained in the response. If you only determine the total number and do not want to obtains single data sets, use the value **-1**.
StartIndex	0	Specifies as from which data sets the results are returned in the response. This parameter is null-based (the first element is addressed with the value **0**).

Example

The following query returns 50 employees and starts with the 101st employee:

`https://<Host-Name>/ApiServer/portal/person?PageSize=50&StartIndex=100`

Sort order

Use the **Orderby** query parameter to sort the results returned in an response. This parameter allows you to sort the column names of the underlying database table.

Examples

The following query returns employees sorted by first name in ascending order:

`https://<Host-Name>/ApiServer/portal/person?OrderBy=FirstName`

Employees sorted in descending order by first name:
`https://<host name>/ApiServer/portal/person?OrderBy=FirstName%20DESC`

Filtering

Use the **filter** query parameter to filter the results returned in an response. A filter like this consists of a JSON formatted string that must contain the following:

- **ColumnName**: Name of the column used to filter
- **CompareOp**: The operator for comparing the contents of the selected column with the expected value

  The following comparison operators are permitted:

  - **Equal**: The results only include data sets with column data that matches the comparison value.
  - **NotEqual**: The results only include data sets with column data that does NOT match the comparison value.
  - **LowerThan**: The results only include data sets with column data less than the comparison value.
  - **LowerOrEqual**: The results only include data sets with column data less than or equal to the comparison value.
  - **GreaterOrEqual**: The results only include data sets with column data greater than or equal to the comparison value.
  - **Like**: Requires the use of a percent sign (%) as a placeholder. You can enter up to two percent signs in this value. The results only include data sets with column data that matches the comparison value pattern.
  - **NotLike**: Requires the use of a percent sign (%) as a placeholder. You can enter up to two percent signs in this value. The results only include data sets with column data that does NOT match the comparison value pattern.
  - **BitsSet**: The value is compared to the comparison value using the AND (&) logical operator. The result must not be equal to 0.
  - **BitsNotSet**: The value is compared to the comparison value using the AND (&) logical operator. The result must be equal to 0.
- **Value1**: Comparison value for comparing the contents of the column

- **Value 2**: If this second comparison value is passed down, the value of **CompareOp** is ignored and all the values that are greater or equal to **Value1** and less or equal to **Value2** are determined.

Example

The following query returns all employees with the last name "User1":

```
https://<Host-Name>/ApiServer/portal/person/all?filter=[{ColumnName:
'LastName', CompareOp: 'Equal', Value1: 'User1'}]
```

Grouping

You can use the **group** path parameter to group the results returned in a response. You can use the **by** query parameter to specify which attribute to use for grouping. Furthermore, you can use the **withcount** query parameter to specify (values: **true** or **false**) whether to calculate the number of objects for each group. This may increase the runtime.

NOTE: The API method must support grouping (by using the **EnableGrouping** parameter).

The result of the query contains a filter condition that you can pass to the URL parameter as filter.

Example

The following queries determine the number of identities grouped by primary location:

```
https://<host name>/ApiServer/portal/person/all/group?by=UID_
Locality&withcount=true
```

Response:

```
{
    {
        "Display": "(No value: Primary location)",
        "Filters": [
            {
                "ColumnName": "UID_Locality",
                "CompareOp": 0
            }
        ],
        "Count": 42
    },
    {
        "Display": "Berlin",
        "Filters": [
            {
```

```
                "ColumnName": "UID_Locality",
                "CompareOp": 0,
                "Value1": "c644f672-566b-4ab0-bac0-2ad07b6cf457"
            }
        ],
        "Count": 12
    }
}
```


Hierarchical data structure

Some data model tables are defined as hierarchical structures (**Department** for example). Data from such tables is loaded from a specific hierarchy level.

You can use the **parentKey** query parameter of the parent object to specify the hierarchy level.

Example

The following query determines the service categories directly below the **Access Lifecycle** service category:

`https://<host name>/ApiServer/portal/servicecategories?parentKey=QER-f33d9f6ec3e744a3ab69a474c10f6ff4`

The following query determines the service categories that do not have a parent service category:

`https://<Host-Name>/ApiServer/portal/servicecategories?parentKey=`

The following query determines all service categories regardless of their hierarchy:

`https://<Host-Name>/ApiServer/portal/servicecategories`

You can use the **noRecursive** path parameter to specify whether the data is queried as a flat list (values: **true** or **false**).

Example

`https://<Host-Name>/ApiServer/portal/servicecategories?noRecursive=true`

Additional query parameters

You can use the **withProperties** query parameter to specify whether additional information from specific tables columns are returned in the response.

NOTE: To enable table columns for these queries, set the **Show in wizards** option in the column properties of the relevant columns in the Designer.

TIP: You can delimit the names of multiple columns with commas.

Example

The following query determines the number of all identities and also returns their preferred name and title:

```
https://<host
name>/ApiServer/portal/person/all?withProperties=PreferredName,Title
```

Response:


```
{
    "TotalCount": 105950,
    "TableName": "Person",
    "Entities": [
        {
            "Display": "100, User (USER1)",
            "LongDisplay": "100, User (USER1)",
            "Keys": [
                "bbf3f8e6-b719-4ec7-be35-cbd6383ef370"
            ],
            "Columns": {
                "DefaultEmailAddress": {
                    "Value": "USER1@qs.ber",
                    "IsReadOnly": true
                },
                "IdentityType": {
                    "Value": "Primary",
                    "IsReadOnly": true,
                    "DisplayValue": "Primary identity"
                },
                "PreferredName": {
                    "Value": "Johnny",
                    "IsReadOnly": true
                },
                "Title": {
                    "Value": "Dr.",
                    "IsReadOnly": true
                },
                "XObjectKey": {
                    "Value": "<Key><T>Person</T><P>bbf3f8e6-b719-4ec7-be35-
cbd6383ef370</P></Key>",
                    "IsReadOnly": true
                }
            }
}
```

HTTP methods

HTTP requests can apply the following HTTP methods:

- **GET**: This method requests data from the application server.
- **PUT**: This method changes data on the application server.
- **POST**: This method creates data on the application server.
- **DELETE**: This method deletes data on the application server.

Date formats

Date values in requests to change or add objects must be specified in ISO 8601 format in the client's local time zone.

Example

```
2016-03-19T13:09:08.123Z
```

Related topics

- Parameter formats on page 16

Parameter formats

HTTP requests use the following types of parameters:

- Path parameters
- Query parameters

Related topics

- Date formats on page 16

Path parameters

Path parameters extend the URL path. A forward slash is used as the delimiter.

If a query uses a path parameters, they are given in URI format.

Example

`https://<host name>/ApiServer/imx/sessions/exampleparameter`

Query parameters

Query parameter are appended to the URL with a question mark (?) or an ampersand (&).

The first query parameter must be prefixed by a question mark. In this case, you must use the following format:

`?parameter name=parameter value` (for example, `?orderBy=LastName`)

Subsequent query parameters must be prefixed by an ampersand. In this case, you must use the following format:

`&parameter name=parameter value` (for example, `?sortOrder=ascending`)

NOTE: Unknown query parameters are rejected by the server with error code **400**. This also affects query parameters with incorrect upper and lower case spelling.

Example

`https://<host name>/AppServer/portal/person?orderBy=LastName`

Response formats

Most API methods return results in JSON format (application/json). Furthermore, there is support for results in CSV and PDF format as long as the result of the respective API method is declared as exportable (with the **AllowExport** flag). Basically, an API method can return results in any format compatible with HTTP.

To obtain results in CSV format

- In the query, set **Accept header** to **text/csv**.

To obtain results in PDF format

- In the query, set **Accept header** to **applciation/pdf**.

  NOTE: To obtain results in PDF format, the **RPS** module must be installed on your system.

Related topics

- Response codes on page 18

Response codes

Responses that are sent from the REST API use the following codes. If queries fail, an explanatory error message is displayed.

Response codes	Description
200	Query successful.
204	Query successful. Response has no content.
401	Access not authorized. The session must be authorized first.
404	The given resource could not be found.
405	The HTTP method used is not allowed for this query.
500	A server error occurred. The error message is sent with the response. On the ground of security, a detailed error message is not included in the response. For more information, see the application log file on the server.

Related topics

- Response formats on page 17

Avoiding deadlocks

API development includes a lot of asynchronous code with async/await constructs. To avoid deadlocks, use the **ConfigureAwait(false)** method for every **await** keyword.

For more information, see https://blog.stephencleary.com/2012/07/dont-block-on-async-code.html and https://devblogs.microsoft.com/dotnet/configureawait-faq/.

2

Examples and help – Software Development Kit

To make it easier for you to start developing your API, One Identity provides a Software Development Kit (SDK) with lots of commented code example.

The SDK can be found on the installation medium in the directory `QBM\dvd\AddOn\ApiSamples`.

3

Implementing your own APIs

To implement your own APIs, you can create API plugins.

The API Server loads all DLLs matching the `*.CompositionApi.Server.PlugIn.dll` naming scheme and deploys the API definitions contained therein.

To implement your own API

1. Create an API plugin (see Creating and editing API plugins on page 20).
2. Compile the appropriate TypeScript API client (see Compiling TypeScript API clients on page 21).

Detailed information about this topic

- Creating and editing API plugins on page 20
- Compiling TypeScript API clients on page 21

Creating and editing API plugins

With the help of API plugins, you can implement and use your customized APIs.

Prerequisites:

- You use a version management system (for example, Git).
- You use an Integrated Development Environment (IDE).

To create an API plugin

1. Start your IDE (such as Visual Studio).
2. Create a new .NET Framework 4.8 project named `CCC.CompositionApi.Server.Plugin`.
3. Add references to the following DLL files from the One Identity Manager installation directory:

- `QBM.CompositionApi.Server.dll`
- `VI.Base.dll`
- `VI.DB.dll`

4. Create the API code.
5. Compile the DLL file in your IDE.
6. Import the DLL file into your One Identity Manager database using the Software Loader and assign it to the **Business API Server** machine role. For more information on importing files using the Software Loader, see the *One Identity Manager Operational Guide*.
7. Restart the API Server and ensure that the `CCC.CompositionApi.Server.Plugin.dll` file is present.

To edit an existing API plugin

1. Start your IDE (such as Visual Studio).
2. Open the existing .NET Framework 4.8 project.
3. Edit the API code.
4. Compile the DLL file in your IDE.
5. Import the DLL file into your One Identity Manager database using the Software Loader and assign it to the **Business API Server** machine role. For more information on importing files using the Software Loader, see the *One Identity Manager Operational Guide*.
6. Restart the API Server and ensure that the `CCC.CompositionApi.Server.Plugin.dll` file is present.

Compiling TypeScript API clients

After you create an API plugin, you need to compile a corresponding TypeScript API client.

To compile a TypeScript API client

1. Open a command line prompt.
2. Run the following command:

   ```
   imxclient compile-api -N -W /copyapi imx-api-ccc.tgz /packagename imx-api-ccc
   ```

   The dialog to select the database connection is opened.
3. In the dialog, perform one of the following actions:
   - to use an existing connection to the One Identity Manager database, select it in the **Select a database connection** menu.

- OR -

- to create a new connection to the One Identity Manager database, click **Add new connection** and enter a new connection.

4. Select the authentication method and, under **Authentication method**, enter the login data for the database.
5. Click **Log in**.
6. Import the `imx-api-ccc` npm package into your TypeScript application.

ImxClient command line program

You can use the ImxClient command line tool to run different functions for managing the API Server and files on the command line.

Detailed information about this topic

- Starting the ImxClient command line program on page 23
- ImxClient command overview on page 23

Starting the ImxClient command line program

You can start the ImxClient command line tool at any time using any command line interface.

To start the ImxClient command line program

1. Open a command line interface (for example, Windows Powershell).
2. In the command line program, go to the One Identity Manager installation directory.
3. Run the `ImxClient.exe` application.

ImxClient command overview

The following chapters contain a list of all ImxClient commands that you can run.

Detailed information about this topic

- help on page 24
- compile-app on page 24
- compile-api on page 25
- repl on page 26
- connect on page 26
- install-apiserver on page 27
- run-apiserver on page 28
- fetch-files on page 29

- push-files on page 30
- get-apistate on page 31
- get-filestate on page 32
- workspace-info on page 33
- check-translations on page 33
- version on page 34

help

Displays a list of available commands.

Parameters

To view help for a specific command, add the command as a parameter.

Example: `help fetch-files`

Related topics

- ImxClient command line program on page 23
- ImxClient command overview on page 23

compile-app

Runs HTML5 package compilation.

This command performs the following steps:

1. Runs the **npm install** command in the application folder.
2. Runs the **npm run build** command in the package folder.
3. Creates the output in subdirectory `dist`
   .The output is stored as a zip file in the database.

Parameters

Login parameters:

- `/conn <database connection>`: Specifies the database to connect to.
- `/dialog <dialog authentication>`: Specifies the dialog authentication.

Optional parameter:

- `/conndialog <option>`: Specifies whether a login window is displayed for the database connection. The following options are possible:
  - `off`: The login window is not shown. If the database is not connected, an attempt is made to establish a connection.
  - `show`: The login window is shown (even is a database is already connected) and the new connection replaces the old one.
  - `fallback` (default): The current database connection is used. If the database is not connected, an attempt is made to establish a connection.
- `/factory <target system>`: Specifies the target system for the connection. Enter this parameter if you want to establish a connection to the application server.
- `/workspace <path to working directory>`: Specifies the working directory. This folder contains the application to be compiled. This folder normally contains the `package.json` file of the application. If you do not enter anything here, the current directory is used.
- `/app <application project name>`: Specifies which application project to compile. If you do not specify anything here, all application projects are compiled.
- `-D`: Runs debug compilation.
- `/copyto <file path>`: Saves the result of the compilation as ZIP files in a folder.
- `/exclude <module name>`: Omits packages of a module at compile time (for example, **AOB**).

Related topics

- ImxClient command line program on page 23
- ImxClient command overview on page 23

compile-api

Compiles the API and saves the result to the database.

Parameters

Login parameters:

- `/conn <database connection>`: Specifies the database to connect to.
- `/dialog <dialog authentication>`: Specifies the dialog authentication.

Optional parameter:

- `/conndialog <option>`: Specifies whether a login window is displayed for the database connection. The following options are possible:

    - `off`: The login window is not shown. If the database is not connected, an attempt is made to establish a connection.
    - `show`: The login window is shown (even is a database is already connected) and the new connection replaces the old one.
    - `fallback` (default): The current database connection is used. If the database is not connected, an attempt is made to establish a connection.
- `/factory <target system>`: Specifies the target system for the connection. Enter this parameter if you want to establish a connection to the application server. Example: `QBM.AppServer.Client`.
- `/copyapi <folder path>`: Specifies where to copy the `imx-api.tgz` to.
- `/copyapidll <API DLL path>`: Specifies which API DLL file to use. The `/solution` and `/branch` parameters are ignored if you use this parameter.
- `/nowarn <error1,error2,...>`: Specifies which errors are ignored during compilation. Enter the codes for the warnings, separated by commas.
- `/warnaserror <error1,error2,...>`: Specifies which warnings are displayed as errors during compilation. Enter the codes for the warnings, separated by commas.

Related topics

- ImxClient command line program on page 23
- ImxClient command overview on page 23

repl

Starts the ImxClient command line tool in REPL mode.

In this mode, the following actions are performed in an infinite loop:

- Read commands from **stdin.**
- Forward commands to the relevant plugin.
- Output the results of processing to **stdout.**

Related topics

- ImxClient command line program on page 23
- ImxClient command overview on page 23

connect

Establishes a database connection.

If a connection to a database has already been established, this is closed and a new connection is then established.

Parameters

Login parameters:

- `/conn <database connection>`: Specifies the database to connect to.
- `/dialog <dialog authentication>`: Specifies the dialog authentication.

Optional parameter:

- `/conndialog <option>`: Specifies whether a login window is displayed for the database connection. The following options are possible:
  - `off`: The login window is not shown. If the database is not connected, an attempt is made to establish a connection.
  - `show`: The login window is shown (even is a database is already connected) and the new connection replaces the old one.
  - `fallback` (default): The current database connection is used. If the database is not connected, an attempt is made to establish a connection.
- `/factory <target system>`: Specifies the target system for the connection. Enter this parameter if you want to establish a connection to the application server.
  Example: `QBM.AppServer.Client`.

Related topics

- ImxClient command line program on page 23
- ImxClient command overview on page 23

install-apiserver

Installs an API Server on the local Internet Information Services (IIS).

Parameters

Login parameters:

- `/conn <database connection>`: Specifies the database to connect to.
- `/dialog <dialog authentication>`: Specifies the dialog authentication.

Required parameters:

- `/app <application name>`: Specifies which name is used for the application (for example, in the brower's titlebar).
- `/sessioncert <certificate thumbprint>`: Specifies which (installed) certificate is used for creating and verifying session tokens.

> TIP: For example, to obtain a certificate thumbprint, you can use the **Manage computer certificates** Windows function and find the thumbprint through the certificate's detailed information.

Optional parameter:

- `/conndialog <option>`: Specifies whether a login window is displayed for the database connection. The following options are possible:
  - `off`: The login window is not shown. If the database is not connected, an attempt is made to establish a connection.
  - `show`: The login window is shown (even is a database is already connected) and the new connection replaces the old one.
  - `fallback` (default): The current database connection is used. If the database is not connected, an attempt is made to establish a connection.
- `-u`: Allows insecure HTTP connections to the API Server website. By default, the API Server website can only be opened over an encrypted connection.
- `/site <site name>`: Specifies the website on the IIS under which the web application will be installed. If you do not enter anything, the website is found automatically (normally **Default website**).
- `/searchservice <URL>`: Specifies the application server's URL that the search service you want to use is hosted on.

  > NOTE: If you would like to use the full text search, then you must specify an application server. You can enter the application server in the configuration file at a later date.

Related topics

- ImxClient command line program on page 23
- ImxClient command overview on page 23

run-apiserver

Starts or stops a self-hosted API Server.

This command requires a database connection.

Parameters

Login parameters:

- `/conn <database connection>`: Specifies the database to connect to.
- `/dialog <dialog authentication>`: Specifies the dialog authentication.

Optional parameter:

- `/conndialog <option>`: Specifies whether a login window is displayed for the database connection. The following options are possible:
  - `off`: The login window is not shown. If the database is not connected, an attempt is made to establish a connection.
  - `show`: The login window is shown (even is a database is already connected) and the new connection replaces the old one.
  - `fallback` (default): The current database connection is used. If the database is not connected, an attempt is made to establish a connection.
- `/factory <target system>`: Specifies the target system for the connection. Enter this parameter if you want to establish a connection to the application server.
  Example: `QBM.AppServer.Client`.
- `-S`: Stops the API Server.
- `/baseaddress <URL with port>`: Specifies the web application's root URL and port.
- `/baseurl <root URL>`: Specifies the web application's URL.
- `-T`: Queries the status of the current API Server.
- `-B`: Locks the console.

Related topics

- ImxClient command line program on page 23
- ImxClient command overview on page 23

fetch-files

Loads a specific machine role from the database and saves it in a local folder.

Parameters

Login parameters:

- `/conn <database connection>`: Specifies the database to connect to.
- `/dialog <dialog authentication>`: Specifies the dialog authentication.

Optional parameter:

- `/conndialog <option>`: Specifies whether a login window is displayed for the database connection. The following options are possible:
  - `off`: The login window is not shown. If the database is not connected, an attempt is made to establish a connection.
  - `show`: The login window is shown (even is a database is already connected) and the new connection replaces the old one.

    - `fallback` (default): The current database connection is used. If the database is not connected, an attempt is made to establish a connection.
- `/factory <target system>`: Specifies the target system for the connection. Enter this parameter if you want to establish a connection to the application server.
  Example: `QBM.AppServer.Client`.
- `/workspace <working directory path>`: Specifies the working directory where the files should be placed. If you do not enter anything here, the current directory is used.
- `/targets <target1;target2;...>`: Specifies which machine roles you want to use.

Related topics

- ImxClient command line program on page 23
- ImxClient command overview on page 23

push-files

Saves files that you have changed locally back to the database.

Parameters

Login parameters:

- `/conn <database connection>`: Specifies the database to connect to.
- `/dialog <dialog authentication>`: Specifies the dialog authentication.

Optional parameter:

- `/conndialog <option>`: Specifies whether a login window is displayed for the database connection. The following options are possible:
    - `off`: The login window is not shown. If the database is not connected, an attempt is made to establish a connection.
    - `show`: The login window is shown (even is a database is already connected) and the new connection replaces the old one.
    - `fallback` (default): The current database connection is used. If the database is not connected, an attempt is made to establish a connection.
- `/factory <target system>`: Specifies the target system for the connection. Enter this parameter if you want to establish a connection to the application server.
  Example: `QBM.AppServer.Client`.
- `/targets <target1;target2;...>`: Specifies which machine roles you want to use.
- `/workspace <folder path>`: Specifies the working directory where the files are located that have been modified and are now to be stored in the database.
- `/tag <uid>`: Specifies the UID of a change tag.

- `/add <file1;file2;...>`: Specifies which new database files are added. Use relative paths.
- `/del <file1;file2;...>`: Specifies which database files are deleted. Use relative paths.
- `-C`: Prevents the saving of changed files and saves only new files, and deletes files from the database.

Related topics

- ImxClient command line program on page 23
- ImxClient command overview on page 23

get-apistate

Queries the compilation status of the API in the database.

Parameters

Login parameters:

- `/conn <database connection>`: Specifies the database to connect to.
- `/dialog <dialog authentication>`: Specifies the dialog authentication.

Optional parameter:

- `/conndialog <option>`: Specifies whether a login window is displayed for the database connection. The following options are possible:
  - `off`: The login window is not shown. If the database is not connected, an attempt is made to establish a connection.
  - `show`: The login window is shown (even is a database is already connected) and the new connection replaces the old one.
  - `fallback` (default): The current database connection is used. If the database is not connected, an attempt is made to establish a connection.
- `/factory <target system>`: Specifies the target system for the connection. Enter this parameter if you want to establish a connection to the application server. Example: `QBM.AppServer.Client`.
- `/branch <compilation branch ID>`: Queries the compilation status of the API saved under this compilation branch.
- `/htmlapp <name of the HTML package>`: Returns data for the specified HTML package.
- `-D`: Returns data for debug assemblies.
- `-R`: Returns data for release assemblies.

Related topics

- ImxClient command line program on page 23
- ImxClient command overview on page 23

get-filestate

Compares the local file structure with the file structure in the database.

Using the **QBM | ImxClient | get-filestate | NewFilesExcludePatterns** configuration parameter, you can define which files are excluded from the synchronization. This prevents excessive load during synchronization. The `node_modules` and `imx-modules` folders are excluded from the synchronization by default.
You can adjust the configuration parameters in the Designer. Use the following formats when defining the rules:
https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.filesystemglobbing.matcher
Use the **|** character to delimit multiple entries.

NOTE: This configuration parameter is generally only used to exclude new files from the synchronization. Files that already exist in the database are not taken into account.

Parameters

Login parameters:

- `/conn <database connection>`: Specifies the database to connect to.
- `/dialog <dialog authentication>`: Specifies the dialog authentication.

Optional parameter:

- `/conndialog <option>`: Specifies whether a login window is displayed for the database connection. The following options are possible:
  - `off`: The login window is not shown. If the database is not connected, an attempt is made to establish a connection.
  - `show`: The login window is shown (even is a database is already connected) and the new connection replaces the old one.
  - `fallback` (default): The current database connection is used. If the database is not connected, an attempt is made to establish a connection.
- `/factory <target system>`: Specifies the target system for the connection. Enter this parameter if you want to establish a connection to the application server.
  Example: `QBM.AppServer.Client`.
- `/targets <target1;target2;...>`: Specifies which machine roles you want to use.
- `/workspace <directory path>`: Specifies the working directory where the files you want to match are located. If you do not enter anything here, the current directory is used.

Related topics

- ImxClient command line program on page 23
- ImxClient command overview on page 23

workspace-info

Queries the state of the Angular working directory (existing applications and last API client update).

Parameters

Optional parameter:

- `/workspace`: Specifies which working directory to query. If you do not enter anything here, the current directory is used.

Related topics

- ImxClient command line program on page 23
- ImxClient command overview on page 23

check-translations

Searches for captions (multilingual text) with missing translations in a particular folder and its subfolders.

Parameters

Login parameters:

- `/conn <database connection>`: Specifies the database to connect to.
- `/dialog <dialog authentication>`: Specifies the dialog authentication.

Required parameters:

- `/path <path to folder>`: Specifies the path to the folder you want to check.

Optional parameter:

- `/conndialog <option>`: Specifies whether a login window is displayed for the database connection. The following options are possible:
  - `off`: The login window is not shown. If the database is not connected, an attempt is made to establish a connection.

    - `show`: The login window is shown (even is a database is already connected) and the new connection replaces the old one.
    - `fallback` (default): The current database connection is used. If the database is not connected, an attempt is made to establish a connection.
- `/factory <target system>`: Specifies the target system for the connection. Enter this parameter if you want to establish a connection to the application server.
  Example: `QBM.AppServer.Client`.

Related topics

- ImxClient command line program on page 23
- ImxClient command overview on page 23

version

Shows the version of the ImxClient command line program in use.

Related topics

- ImxClient command line program on page 23
- ImxClient command overview on page 23

About us

One Identity solutions eliminate the complexities and time-consuming processes often required to govern identities, manage privileged accounts and control access. Our solutions enhance business agility while addressing your IAM challenges with on-premises, cloud and hybrid environments.

Contacting us

For sales and other inquiries, such as licensing, support, and renewals, visit https://www.oneidentity.com/company/contact-us.aspx.

Technical support resources

Technical support is available to One Identity customers with a valid maintenance contract and customers who have trial versions. You can access the Support Portal at https://support.oneidentity.com/.

The Support Portal provides self-help tools you can use to solve problems quickly and independently, 24 hours a day, 365 days a year. The Support Portal enables you to:

- Submit and manage a Service Request
- View Knowledge Base articles
- Sign up for product notifications
- Download software and technical documentation
- View how-to videos at www.YouTube.com/OneIdentity
- Engage in community discussions
- Chat with support engineers online
- View services to assist you with your product

Index

L

M

P

Q

R

S

T

U