BiZZdesign Open API

This feature is only available in the BiZZdesign cloud solution.

[BETA]

A data enrichment open API is available in the HoriZZon server. With this data enrichment API, external data and attributes can be added to existing structural data created by Enterprise Studio. In order to use the open API a named "API client" must be created in HoriZZon.

Currently, the data enrichment API can only be used in the context of ArchiMate modeling.

Required roles

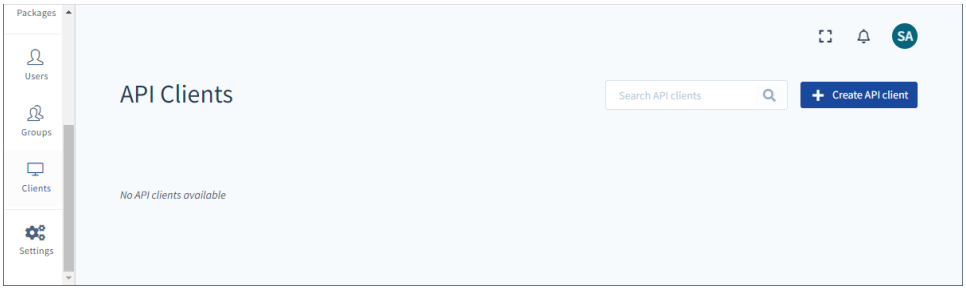
Administrator or System Administrator

On this page:

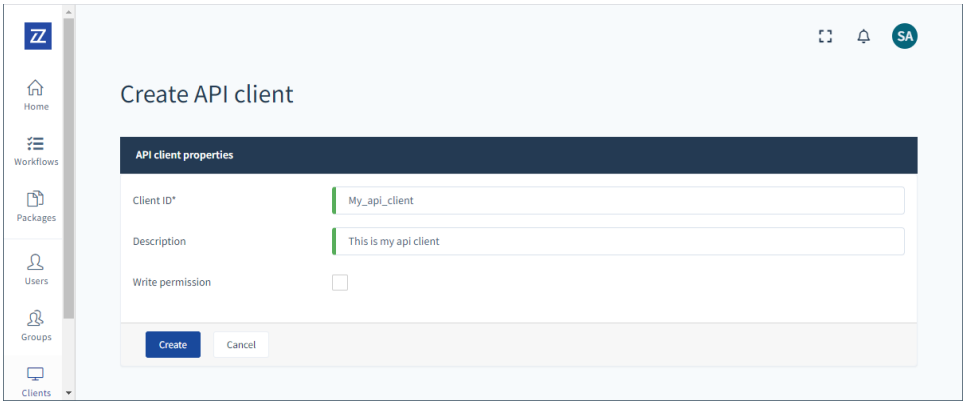
Creating a named API client

The first step for using the API is creating an API Client. Every API Client has a unique identifier (the Client ID) and an associated secret key that can be used to request access tokens to authenticate with the API.

1. In the sidebar menu, click **Clients**.
2. Click **Create API client**.

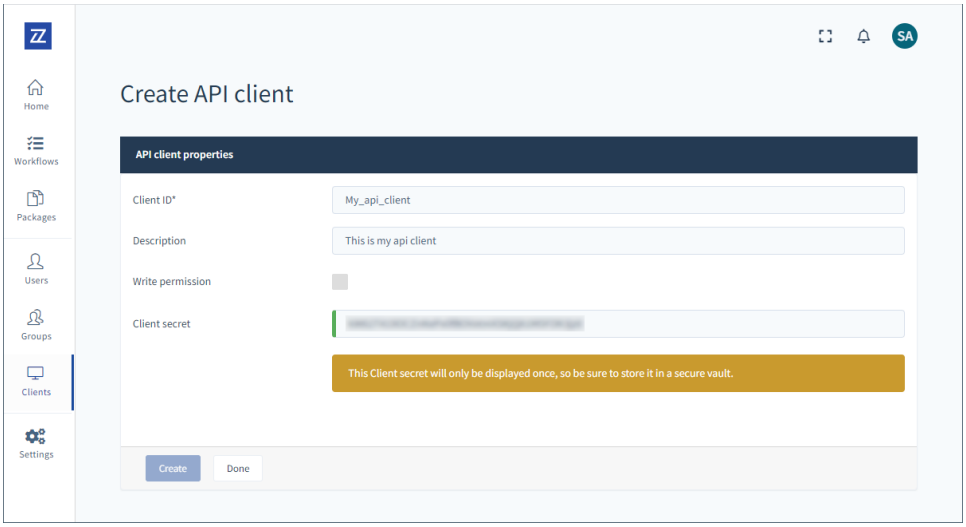


3. Set the API client properties. If you want to have the possibility to also write data into the API client, select **Write permission**. Click **Create** to create the client.



4. When the API client is successfully created, a generated client secret is shown. This client secret is needed to connect your application to the BiZZdesign open API. Save a copy of the client secret somewhere in a safe place. After that, click **Done**.

   Once you leave the page, the client secret is gone.



Requesting an API Access Token

The generated API Client credentials can be used to create a temporary access token to authenticate your API calls. For authentication OAuth 2 is used, a modern standard for securing access to APIs. The token is created by sending a POST request to `https://<orgname>.bizzdesign.cloud/oauth/token` with the following parameters :

Key	Value
grant_type	client_credentials
client_id	<generated Client ID>
client_secret	<generated secret>

The hostname for requesting API Access Tokens is the same hostname used in the Enterprise Studio clients

These parameters can either be set as query parameters in the URL or in the body using x-www-form-url encoded data. Possible return values are:

Response code	Body	Description
200	Access token in RFC 7519 format (JWT)	Relevant attributes are access_token (which contains the token used to authenticate to the API) and expires_in (lifetime of the access token, in seconds). The access token uses the JSON Web Token format defined in RFC 7519.
400	JSON-formatted error message	Invalid request
401	empty	The combination of client_id and client_secret is invalid
403	empty	Your current license level does not cover use of the Open API

Below is a simple example using Curl, assuming your HoriZZon environment is normally accessed via `https://<orgname>.horizzon.cloud/` .

```
curl -X POST -H "Content-Type: application/x-www-form-urlencoded" \
--data-urlencode 'grant_type=client_credentials' \
--data-urlencode 'client_id=CustomerClientID' \
--data-urlencode 'client_secret=CustomerClientSecret' \
https://<orgname>.bizzdesign.cloud/oauth/token
```

Submitting requests to the Open API

Once a valid access token is obtained requests can be submitted to the Open API using the following URL:

`https://<orgname>-api.horizzon.cloud/api/<version>/<API-call>`

Where `<version>` represents the API version identifier. It may change with every new version of the Open API.

The identifier associated with the current Open API version is: **beta2**

So, the API is now reachable via `https://<orgname>-api.horizzon.cloud/api/beta2/`

> The hostname used for OpenAPI requests is different from the hostname normally used to access HoriZZon, note the extra -api part in the name!

The access token is used as a bearer token, which means it should be sent in the Authorization: header, with the value of the header set to "Bearer <tokenvalue>". An example using Curl to request a list of all collaborations in JSON format:

```
curl -H "Accept: application/json" \
-H "Authorization: Bearer ReplaceThisWithTheBearerTokenRetrievedInThePreviousStep" \
https://<orgname>-api.horizzon.cloud/api/<version>/collaborations
```

Open API documentation

> Please carefully check which version of Enterprise Studio you are currently working with, and then choose the appropriate Open API documentation. Not sure which software version you are working with? Check What version of Enterprise Studio do I have?

The following API calls are available for use:

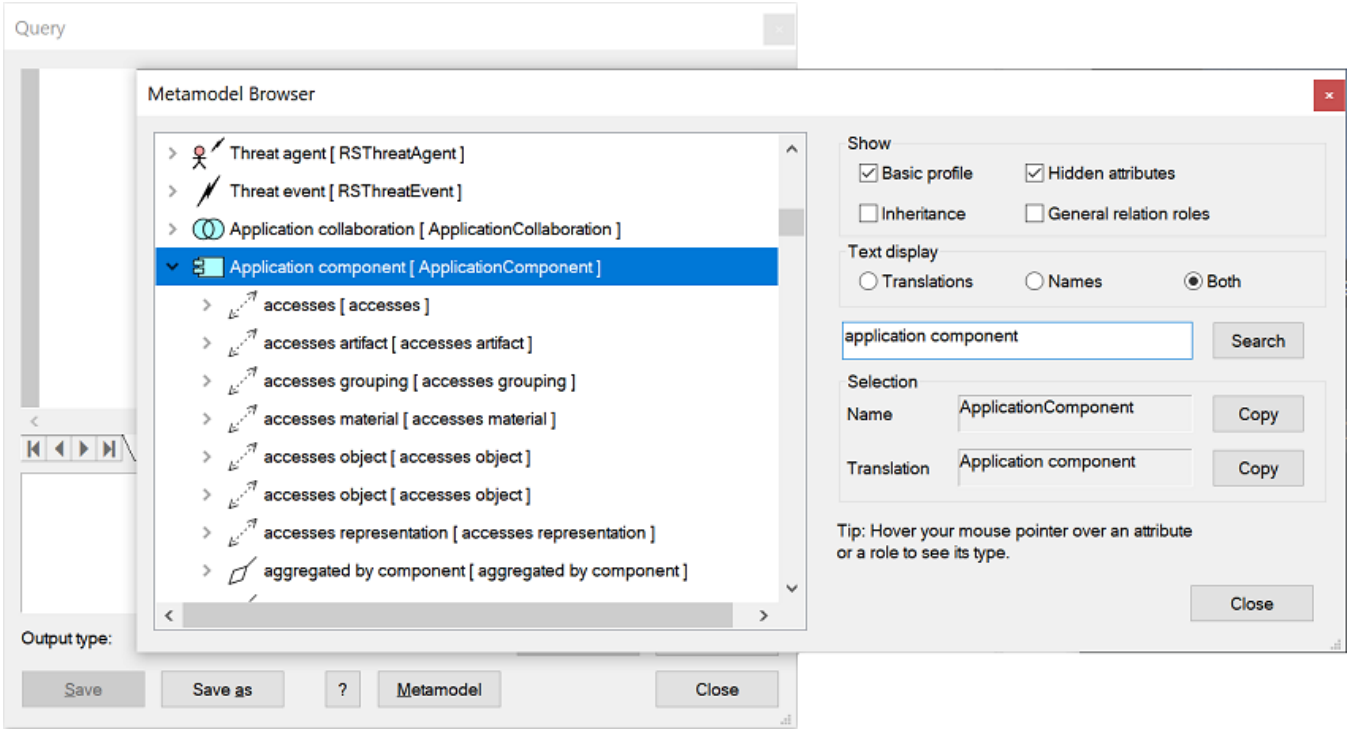
Finding type names

When using the Open API for requesting elements, their type names are needed to access the correct information. These type names can be found using the Metamodel Browser or you can use a script to generate a list of available type names in your model.

Using the Metamodel Browser

The Metamodel Browser can be accessed via the Query tool script editor. The Metamodel Browser contains the names of all elements, attributes and enumeration values of all the elements available in the metamodel of a model, including any customizations that have been made to the configuration of the metamodel.

Clicking the **Metamodel** button in the script editor will open the **Metamodel Browser** window. Type the name of the element you are looking for in the search box, and click **Search**. The result in **Name** will show the type name.



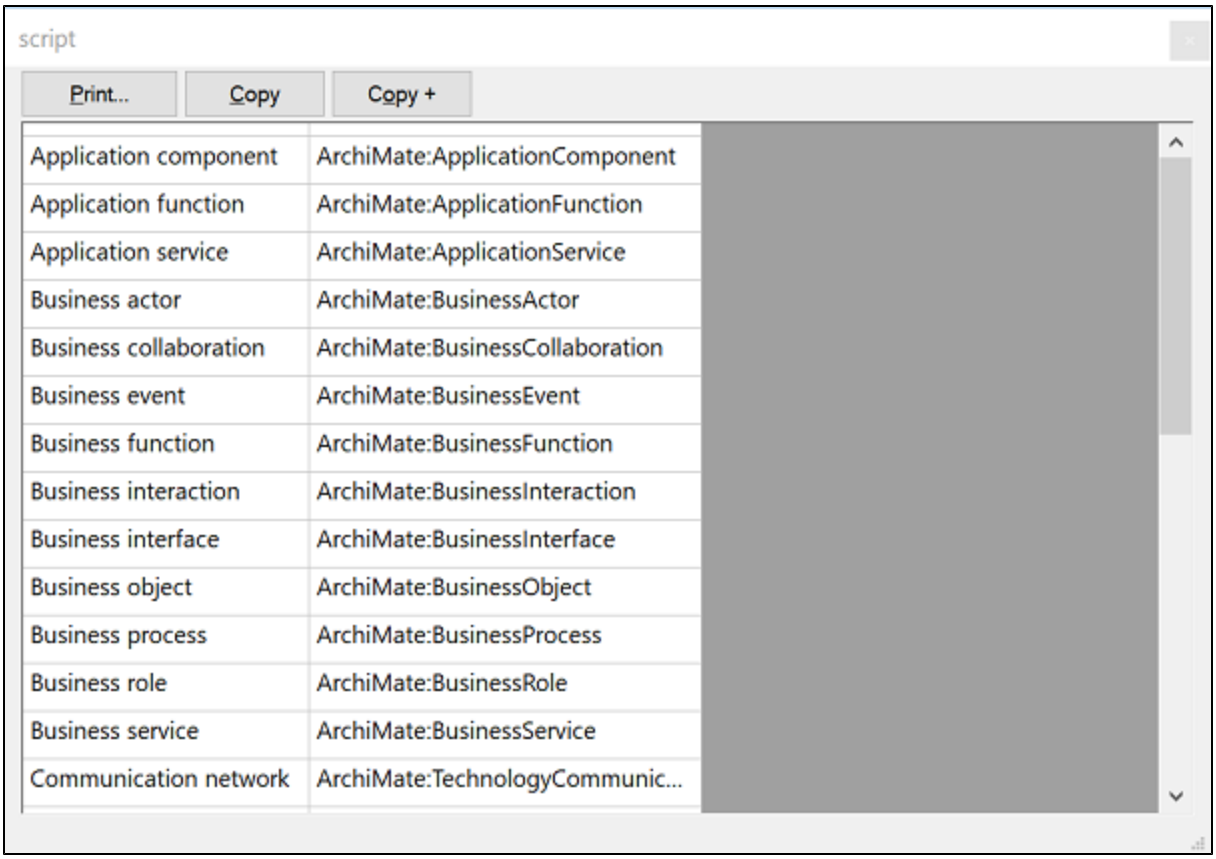
Using a script

Lists of type names can be generated using a script. The script can be executed in the script editor and needs the "Table" output type. Following scripts are available:

```
// Find the type of 1 object, selected in the model browser.

output selection[1].type().name();
```

```
// Find the type of all objects in the current open model package.

nameType = Index();
forall List("AbstractCompound", "AbstractElement") obj in modelpackage {
      nameType.add( obj.type().toString(), obj.type().name() );
}
forall name, type in nameType {
      output name, type;
}
```

```
// Find all possible types in a metamodel (whether present in the model package or not).

metamodel = "ArchiMate";
nameType = Index();
forall c in InternalObject("configuration").context(modelpackage).metaModel(metamodel).concepts(true) {
      if ( c.derivesFrom("AbstractCompound") || c.derivesFrom("AbstractElement") ) {
           nameType.add( c.label( modelpackage.activeLanguage() ), format("%s:%s", c.metaModel().name(), c.
name()) );
      }
}
forall name, type in nameType {
      output name, type;
}
```



Example of a list generated with a script

Related articles

- BiZZdesign Open API
- Including external data in model packages