OpenText™ IMaaS Tools for VS Code

User Guide

This guide describes how to use the OpenText™ IMaaS Tools VS Code extension pack for developing and deploying applications that consume the OpenText™ IMaaS APIs.

This documentation has been created for software version OpenText™ IMaaS Tools for VS Code 22.3.5.

It is also valid for subsequent software releases unless OpenText has made newer documentation available with the product, on an OpenText website, or by any other means.

Open Text Corporation

275 Frank Tompa Drive, Waterloo, Ontario, Canada, N2L 0A1

Tel: +1-519-888-7111

Toll Free Canada/USA: 1-800-499-6544 | International: +800-4996-5440

Fax: +1-519-888-0677

Support: https://support.opentext.com

For more information, visit https://www.opentext.com

Last updated: 09/09/2022

Contents

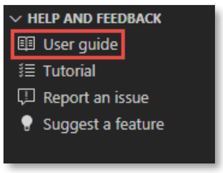
1 Introduction

This user guide provides instructions for pro-code developers on how to use the OpenText™ IMaaS Tools for VS Code 22.3.5 to build applications that consume the IMaaS (Information Management as a Service) APIs from the OpenText™ API Cloud.

It covers the topics of connecting to a developer organization in the OpenText API Cloud, creating an IMaaS application project with its different models (IMaaS application configuration artifacts), and deploying this application to the different IMaaS APIs through the integrated ALM (Application Lifecycle Management) deployment functionality.

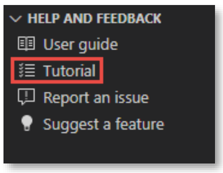
We recommend that you always use the link to this user guide from the OpenText™ IMaaS Tools **Help and Feedback** section in VS Code (see Fig. 1.1), so that you are certain to have the up-to-date user guide which corresponds with the IMaaS Tools for VS Code version you have installed in your Visual Studio Code IDE.

Fig. 1.1:



In addition to this user guide, the **Help and Feedback** section also contains the IMaaS developer tutorial (see Fig. 1.2), which guides you through a detailed journey on how to build an application with the OpenText™ IMaaS Tools for VS Code. It is highly recommended you follow this tutorial as a way of getting started, as not does it provide you with a thorough understanding of the use of the IMaaS functionality in VS Code, but it also explains where to find key developer documentation, and how to consume the IMaaS APIs of the OpenText™ API Cloud.

Fig. 1.2:



2 Overview

Installing the OpenText™ IMaaS Tools VS Code extension pack adds the IMaaS developer functionality to the Visual Studio Code IDE, referred to as VS Code from here.

The IMaaS developer functionality is available in VS Code from three different locations:

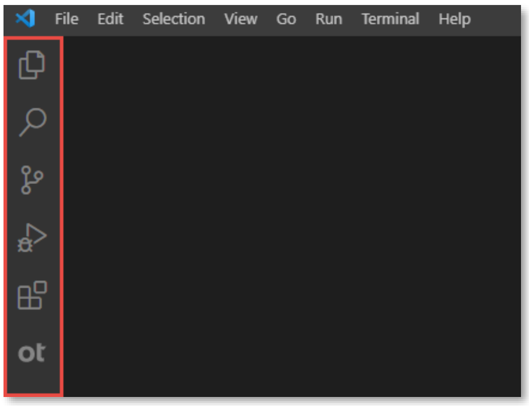
- The **OpenText IMaaS Tools** view in the VS Code Activity Bar
- The **Explorer** view in the VS Code Activity Bar
- The VS Code **Command Palette**

In this chapter we go over these three locations and describe the IMaaS developer capabilities they provide.

2.1 OpenText IMaaS Tools view

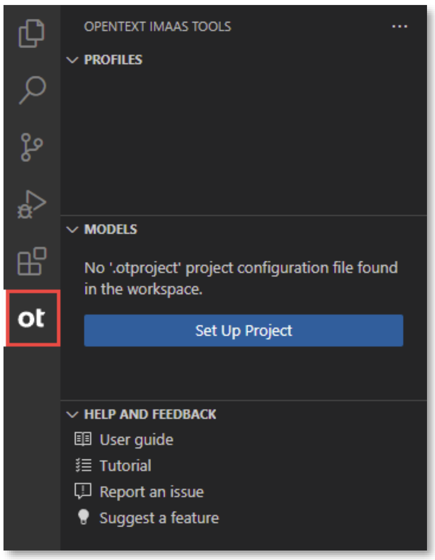
In VS Code, you can switch between different views through the Activity Bar (see Fig. 2.1).

Fig. 2.1:



When the OpenText™ IMaaS Tools VS Code extension pack is installed, you can access the OpenText IMaaS Tools view from the Activity Bar (see Fig 2.2).

Fig. 2.2:



No '.otproject' project configuration file found in the workspace.

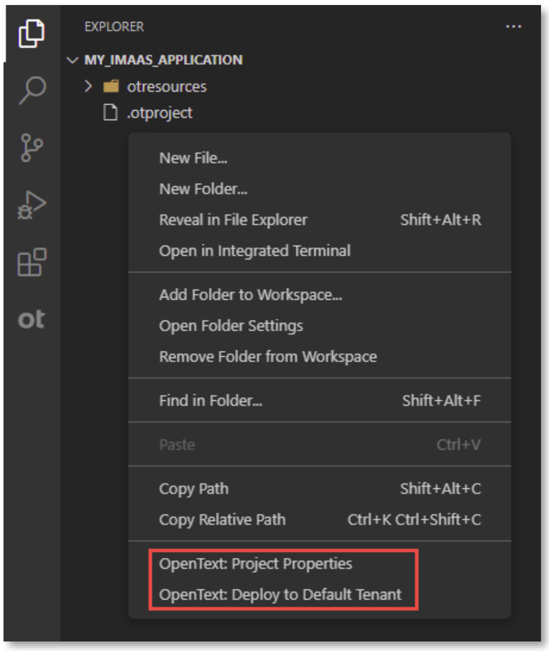
The OpenText IMaaS Tools view is divided into three sections:

- **Profiles:**
  This is where the developer can configure authentication profiles to allow connecting and deploying to multiple developer organizations in the OpenText™ API Cloud.
- **Models:**
  This is where the developer can configure the OpenText project and, once configured, explore the different models that exist in the project.
- **Help and Feedback:**
  This is where the developer can directly access the OpenText™ IMaaS Tools VS Code extension pack's user guide, the IMaaS developer tutorial (explains how to get started with developing IMaaS applications), and where they can report an issue or suggest a new feature.

2.2 Explorer view

In the standard VS Code Explorer view, installing the OpenText™ IMaaS Tools VS Code extension pack adds two menu entries to the contextual menu of the VS Code workspace root folder (see Fig. 2.3).

Fig. 2.3:


EXPLORER
MY_IMAAS_APPLICATION
otresources
.otproject
New File...
New Folder...
Reveal in File Explorer Shift+Alt+R
Open in Integrated Terminal
Add Folder to Workspace...
Open Folder Settings
Remove Folder from Workspace
Find in Folder... Shift+Alt+F
Paste Ctrl+V
Copy Path Shift+Alt+C
Copy Relative Path Ctrl+K Ctrl+Shift+C
OpenText: Project Properties
OpenText: Deploy to Default Tenant

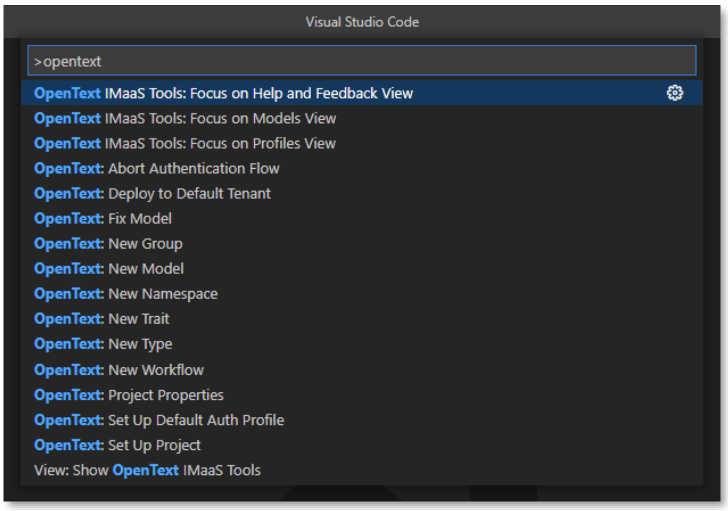
- Clicking the **OpenText: Project Properties** menu entry opens the IMaaS project properties screen, allowing you to edit the project and application properties.
- Clicking the **OpenText: Deploy to Default Tenant** menu entry deploys your IMaaS application project to the configured default tenant in the default authentication profile, see Setting up organization profiles, for details.

2.3 Command Palette

The VS Code Command Palette (available by pressing **Ctrl+Shift+P** or **F1**, depending on your system) allows you to access all the functionality of VS Code. So, the functionality that has been added to VS Code by the OpenText™ IMaaS Tools VS Code extension pack is also directly available from the Command Palette (see Fig. 2.4), i.e.: as the different commands.

Since the Command Palette allows filtering commands, typing "**opentext**" in the filter box ensures you see all available OpenText™ IMaaS Tools related commands.

Fig. 2.4:



Visual Studio Code

```
>opentext
OpenText IMaaS Tools: Focus on Help and Feedback View
OpenText IMaaS Tools: Focus on Models View
OpenText IMaaS Tools: Focus on Profiles View
OpenText: Abort Authentication Flow
OpenText: Deploy to Default Tenant
OpenText: Fix Model
OpenText: New Group
OpenText: New Model
OpenText: New Namespace
OpenText: New Trait
OpenText: New Type
OpenText: New Workflow
OpenText: Project Properties
OpenText: Set Up Default Auth Profile
OpenText: Set Up Project
View: Show OpenText IMaaS Tools
```

The commands from the Command Palette that allow creating new models are described further in the Creating models chapter of this user guide.

You now have an understanding of the different locations in VS Code where you can access the features of the OpenText™ IMaaS Tools extension pack. The following chapters in this user guide describe how to use those features.

3 Setting up organization profiles

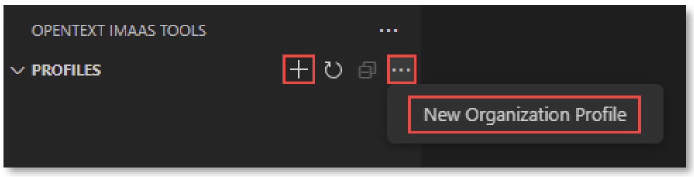
This chapter describes how to set up VS Code OpenText Cloud API organization profiles. More specifically, it details how to create a profile to your OpenText Cloud Platform organization(s) and tenants, so that it is possible to use these profiles to authenticate and deploy your application project to the API Cloud.

3.1 Adding organization profiles

When working with the OpenText Cloud APIs you can work with multiple organizations. You may, for example, have an organization in different regions. Adding an organization profile allows you to connect to these different organizations.

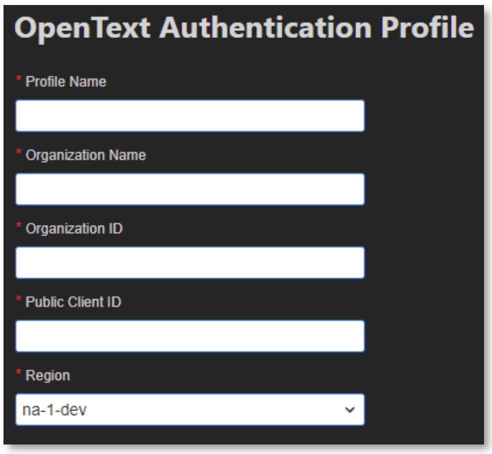
Adding a profile is done by using **New Organization Profile** from the contextual menu of the **Profiles** section (see Fig. 3.1.1).

Fig. 3.1.1:



Using this option opens the authentication profile configuration screen (see Fig. 3.1.2). By default, only the Region is prefilled.

Fig. 3 1.2:



OpenText Authentication Profile

* Profile Name

* Organization Name

* Organization ID

* Public Client ID

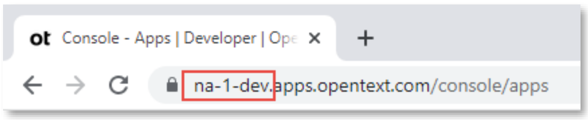
* Region

na-1-dev

The different authentication profile properties on the **OpenText Authentication Profile** configuration screen are:

- **Profile Name:** the name of the authentication profile
- **Organization Name:** the name of the developer organization
- **Organization ID:** the (unique) ID of the developer organization
- **Public Client ID:** the public client ID of the developer organization
- **Region:** the region in which the organization is located. Possible options are `na-1-dev`, `na-1`, `eu-1`. This field is defaulted to `na-1-dev`. The region can be found by looking at the url that is used to open the Developer Console (see Fig. 3.1.3).

Fig. 3.1.3:

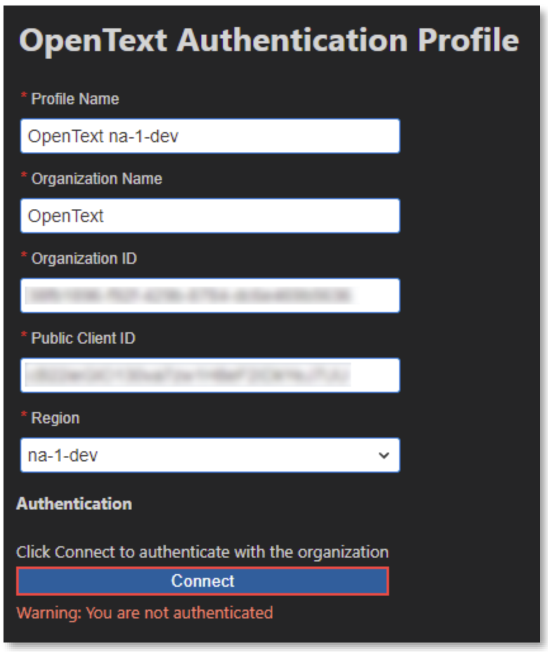

ot Console - Apps | Developer | Ope
na-1-dev.apps.opentext.com/console/apps

Organization name, Organization ID and Public Client ID are provided when creating an organization in **developer.opentext.com**, and they can also be retrieved through the organization Console in your **developer.opentext.com** subscription.

To save the profile configuration, select **Save** from the **File** menu, or press **Ctrl+S** on your keyboard.

To test the configured organization profile, click **Connect** from the **OpenText Authentication Profile** form (see Fig. 3.1.4).

Fig. 3.1.4:



OpenText Authentication Profile

* Profile Name
OpenText na-1-dev

* Organization Name
OpenText

* Organization ID
[redacted]

* Public Client ID
[redacted]

* Region
na-1-dev

Authentication

Click Connect to authenticate with the organization

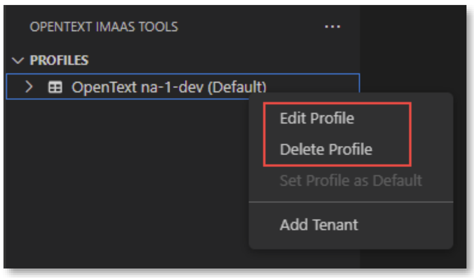
Connect

Warning: You are not authenticated

To change an existing organization profile, use the **Edit Profile** context menu option of the specific profile (see Fig. 3.1.5).

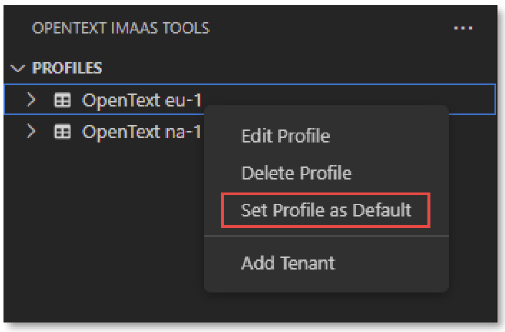
To delete an organization profile, use the **Delete profile** context menu option of the specific profile. Note that in case you have multiple organization profiles that it is not possible to delete a profile that is marked as default. A default profile can only be deleted when it is the only existing profile (see Fig. 3.1.5).

Fig. 3.1.5:



The first organization profile that is added is set as the default profile. In case multiple organization profiles are available then one of these can be set as the default profile for deployment. Use the **Set Profile as Default** context menu option of the specific profile (see Fig. 3.1.6).

Fig. 3.1.6:



3.2 Adding tenants to a profile

An organization can have multiple tenants. When no specific tenants are added to the organization authentication profile then, when deploying your project, the application is deployed to all tenants in that organization. It is possible to add specific tenants and mark one of these tenants as the default tenant to deploy to. This way it is possible to deploy to a specific tenant only.

To add a specific tenant to an organization profile, use the **Add Tenant** context menu option of the specific profile (see Fig. 3.2.1). In the top middle of the VS Code UI input fields are shown, use these to enter the **Tenant Name** and **Tenant ID** (see Fig. 3.2.2 and see Fig. 3.2.3).

Fig. 3.2.1:

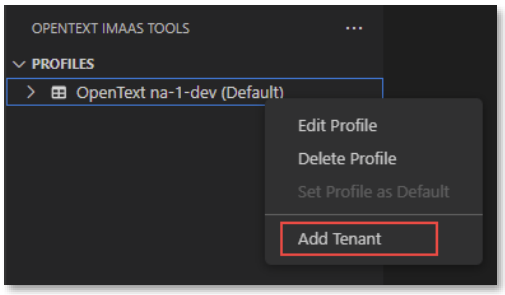
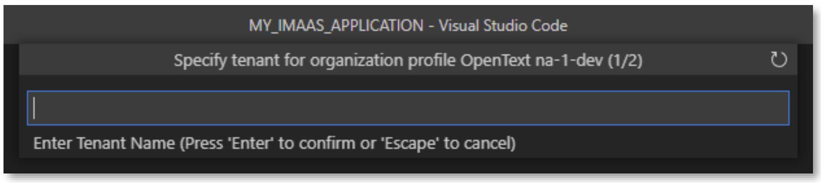


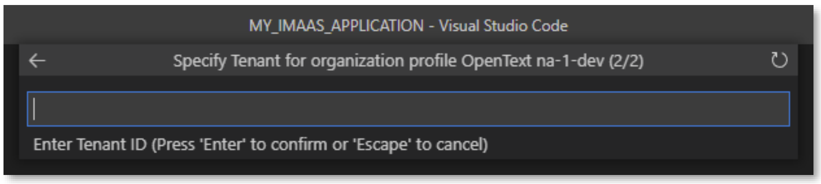
Fig. 3.2.2:


MY_IMAAS_APPLICATION - Visual Studio Code

Specify tenant for organization profile OpenText na-1-dev (1/2)

Enter Tenant Name (Press 'Enter' to confirm or 'Escape' to cancel)

Fig. 3.2.3:


MY_IMAAS_APPLICATION - Visual Studio Code

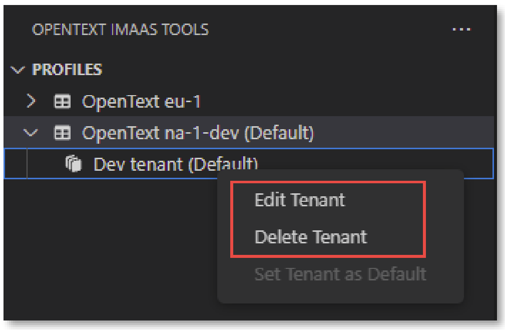
Specify Tenant for organization profile OpenText na-1-dev (2/2)

Enter Tenant ID (Press 'Enter' to confirm or 'Escape' to cancel)

To change an existing tenant, use the **Edit Tenant** context menu option of the specific tenant (see Fig. 3.2.4).

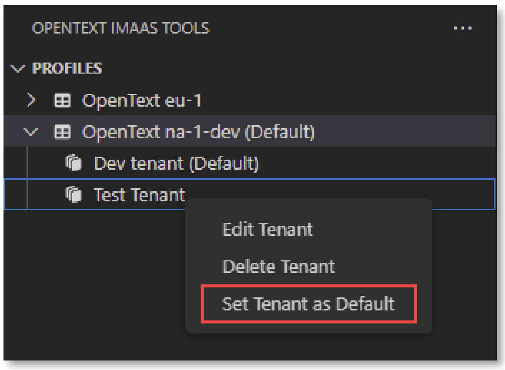
To delete a tenant, use the **Delete Tenant** context menu option of the specific tenant (see Fig. 3.2.4).

Fig. 3.2.4:



The first tenant that is added to an organization profile is set as the default tenant. In case multiple tenants are available then one of these can be set as the default tenant for deployment. Use the **Set Tenant as Default** context menu option of the specific tenant (see Fig. 3.2.5).

Fig. 3.2.5:

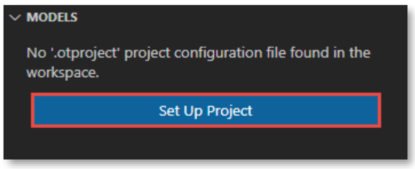


4 Setting up a project

This chapter describes how to set up a (IMaaS application) project. Setting up (i.e.: configuring) a new project is essential, as it enables the IMaaS developer modeling capabilities within the context of the current VS Code workspace folder. Without a project configuration, you are not able to create models for your IMaaS application project.

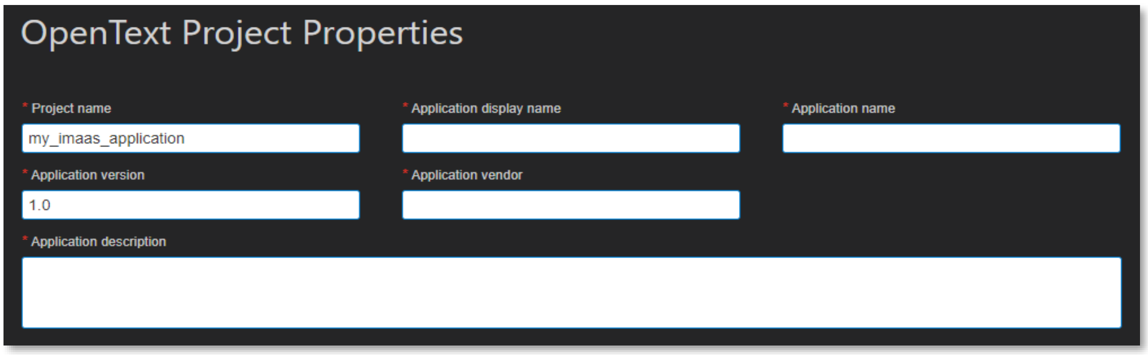
Setting up a project is done via the **Set Up Project** button under the **Models** section of the **OpenText IMaaS Tools** view (see Fig. 4.1). Note that this button is only available when a project has not yet been set up. When clicked, the **OpenText Project Properties** configuration form opens (see Fig. 4.2).

Fig. 4.1:



No '.otproject' project configuration file found in the workspace.

Fig. 4.2:



OpenText Project Properties

* Project name: my_imaas_application	* Application display name:	* Application name:
* Application version: 1.0	* Application vendor:	
* Application description:		

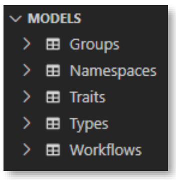
The different project properties on the **OpenText Project Properties** configuration screen are:

- **Project name:** the name of the IMaaS application project; a default value is automatically filled using the VS Code workspace folder name (this can be different from the application name itself)
- **Application display name:** the user-friendly name (i.e.: label) of the IMaaS application
- **Application name:** the unique (technical) name of the IMaaS application; a default value is automatically filled using the (previously entered) application display name
- **Application description:** the description of the IMaaS application
- **Application version:** the version label of the IMaaS application, defaulted to “1.0”
- **Application vendor:** the name of the owner/vendor of the application

To save the project configuration, select **Save** from the **File** menu, or press **Ctrl+S** on your keyboard.

Once a project has been set up for the VS Code workspace (folder), the **Models** section shows the model tree instead of the **Set Up Project** button. The model tree allows exploring and editing the models that exist within the project (see Fig. 4.3).

Fig. 4.3:



When a project has been set up, you can always view or modify the OpenText project properties from the **Model Explorer** by choosing **Project Properties** from the contextual menu (see Fig. 4.4) and in the VS Code **Explorer** view by choosing **OpenText: Project Properties** from the contextual menu of your VS Code workspace (root) folder (see Fig. 4.5) or any of its subfolders, or by opening the **.otproject** file (see Fig. 4.6):

Fig. 4.4:

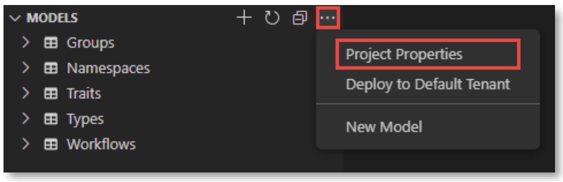


Fig. 4.5:

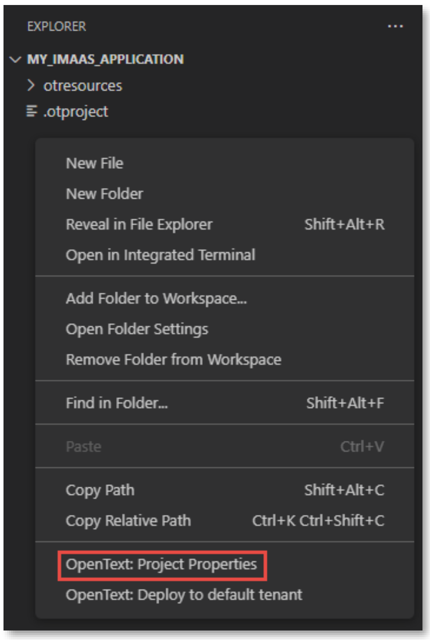
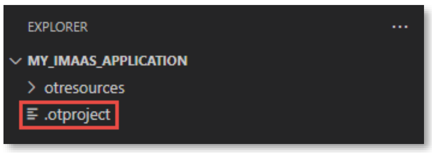

EXPLORER
MY_IMAAS_APPLICATION
otresources
.otproject
New File
New Folder
Reveal in File Explorer Shift+Alt+R
Open in Integrated Terminal
Add Folder to Workspace...
Open Folder Settings
Remove Folder from Workspace
Find in Folder... Shift+Alt+F
Paste Ctrl+V
Copy Path Shift+Alt+C
Copy Relative Path Ctrl+K Ctrl+Shift+C
OpenText: Project Properties
OpenText: Deploy to default tenant

Fig. 4.6:



5 Creating models

This chapter describes how to create models. Models are the configuration artifacts that define your IMaaS application itself. Deploying an application effectively amounts to deploying the different models to their respective IMaaS API endpoints in the OpenText™ API Cloud.

For each type of model, the OpenText™ IMaaS Tools provide a bespoke editor or modeler. To create a new model and launch the corresponding modeler, there are three methods:

1. Through the ⊞ button (see Fig. 5.1.1) or the **New Model** menu entry of the **[…]** menu (see Fig. 5.1.2) of the **Model Explorer** (**Models** section of the **OpenText IMaaS Tools**).

   Fig. 5.1.1:

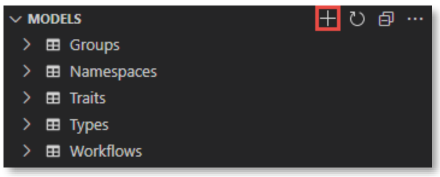
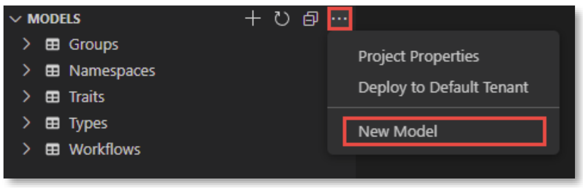
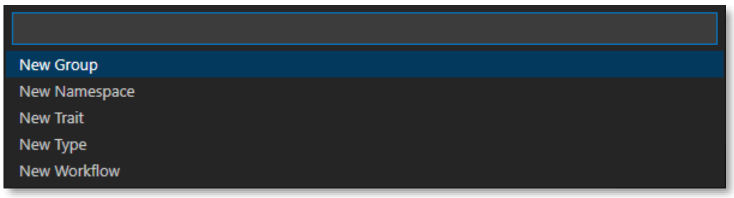
   

   Fig. 5.1.2:

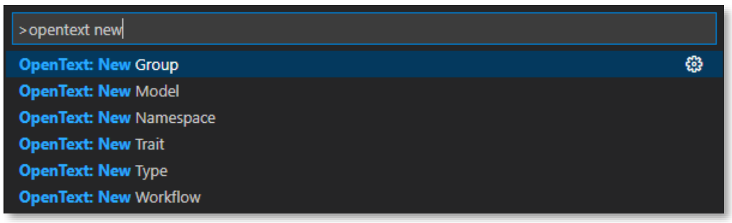
   This allows you to select the type of model you want to create (see Fig. 5.1.3). You are asked to provide a model name when clicking the chosen model type.

   Fig. 5.1.3:

2. Through the **Command Palette** (see Fig. 5.2). You are asked to provide a model name when choosing one of the commands. Note that you can either use the **OpenText: New Model** command (which behaves the same way as the previously explained method) or any of the commands for direct creation of specific types of models.

   Fig. 5.2:

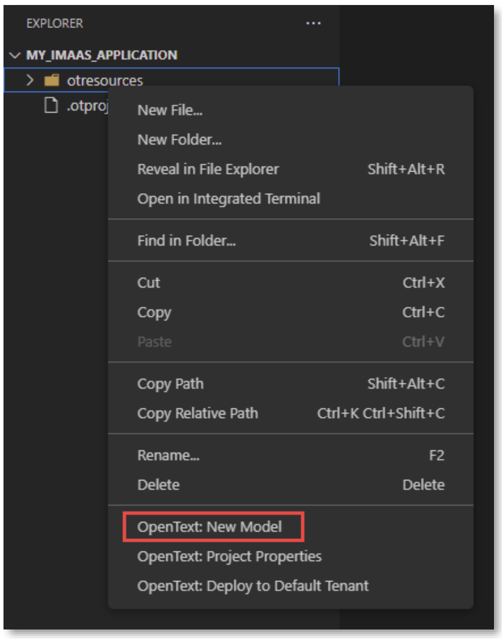
   

   ```
   >opentext new
   ```

Note that it is only allowed to create models in a model folder (the **otresources** folder that was automatically generated when setting up the project, or one of its subfolders). This means that for the above two methods, you need to save the model inside the **otresources** folder.

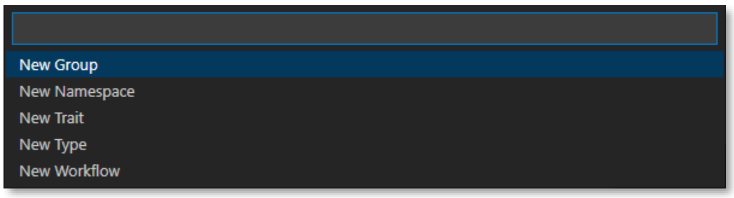
3. From the **Explorer** view, through the **OpenText: New Model** menu entry (see Fig. 5.3.1) from the contextual menu on a model folder (i.e.: the **otresources** folder or one of its subfolders).

Fig. 5.3.1:


EXPLORER
MY_IMAAS_APPLICATION
otresources
.otproj
New File...
New Folder...
Reveal in File Explorer Shift+Alt+R
Open in Integrated Terminal
Find in Folder... Shift+Alt+F
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Copy Path Shift+Alt+C
Copy Relative Path Ctrl+K Ctrl+Shift+C
Rename... F2
Delete Delete
OpenText: New Model
OpenText: Project Properties
OpenText: Deploy to Default Tenant

This allows you to select the type of model you want to create (see Fig. 5.3.2). You are asked to provide a model name when clicking the chosen model type.

Fig. 5.3.2:



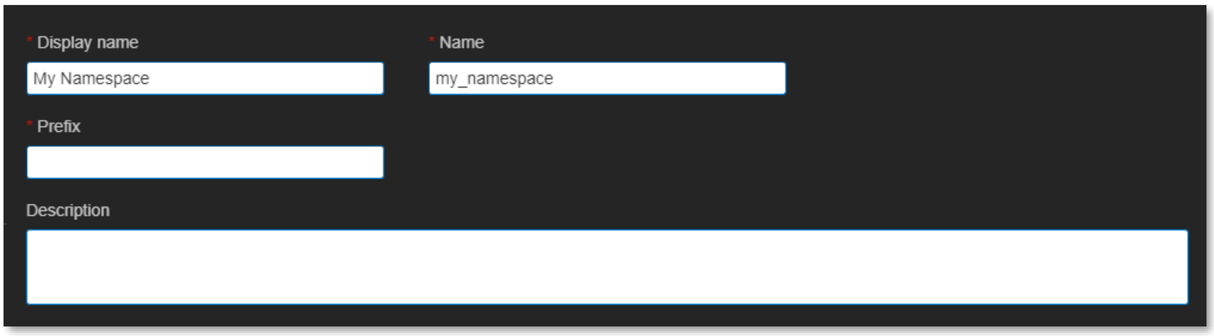
The remainder of this chapter describes the different modelers and how to use them to create the corresponding models.

5.1 Creating a namespace

A namespace allows grouping the different types, traits, and workflows together (e.g.: within the context of an application). For more information on namespaces, you can refer to the Define a namespace, trait and "FILE" document type section in the Content Metadata Service (CMS) product documentation or the Namespace resource documentation in the Content Metadata Service API reference.

You can create a namespace via any of the three model creation methods. This opens the namespace modeler (see Fig. 5.4).

Fig. 5.4:



* Display name	* Name
My Namespace	my_namespace
* Prefix	
Description	

The different model properties on the namespace modeler screen are:

- **Display name:** the user-friendly name (i.e.: label) of the namespace; this does not have to be unique, and a default value is automatically filled using the model name you initially chose
- **Name:** the (technical) name of the namespace; this has to be unique (within your developer tenant), and a default value is automatically filled using the display name
- **Prefix:** the prefix representing the namespace (used in system naming of traits and types that are within that namespace); this has to be unique (within your developer tenant)
- **Description:** the description of the namespace

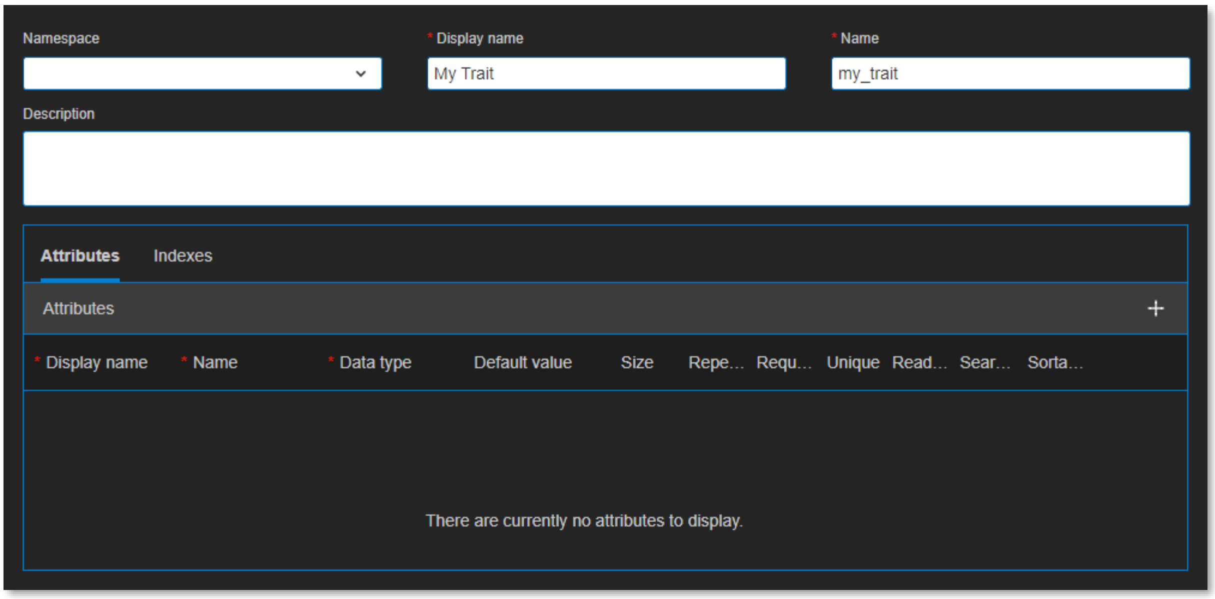
To save the namespace model, select **Save** from the **File** menu, or press **Ctrl+S** on your keyboard.

5.2 Creating a trait definition

A trait definition allows grouping several attributes into one more complex multi-attribute property. Trait instances can be dynamically added to a type instance as part of the business process when using the application, but they can also be made mandatory as a required trait in a type definition, so that they must always be added when creating a new type instance. For more information on traits (definitions and instances), you can refer to the Define a namespace, trait and "FILE" document type and Create instances using custom type with trait sections in the Content Metadata Service (CMS) product documentation or the Trait resource documentation in the Content Metadata Service API reference.

You can create a trait (i.e.: a trait definition) via any of the three model creation methods. This opens the trait modeler (see Fig. 5.5).

Fig. 5.5:

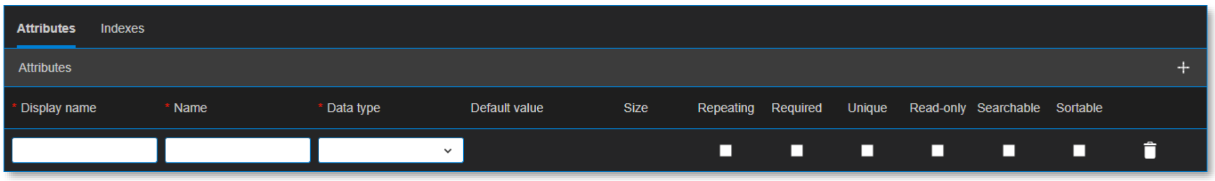


The different model properties on the trait definition modeler screen are:

- **Namespace:** the namespace to which the trait belongs; the namespace dropdown list is populated with the namespaces that exist within the project, and you can opt not to select any namespace
- **Display name:** the user-friendly name (i.e.: label) of the trait; this does not have to be unique, and a default value is automatically filled using the model name you initially chose
- **Name:** the (technical) name of the trait; this has to be unique (within your developer tenant), and a default value is automatically filled using the display name
- **Description:** the description of the trait

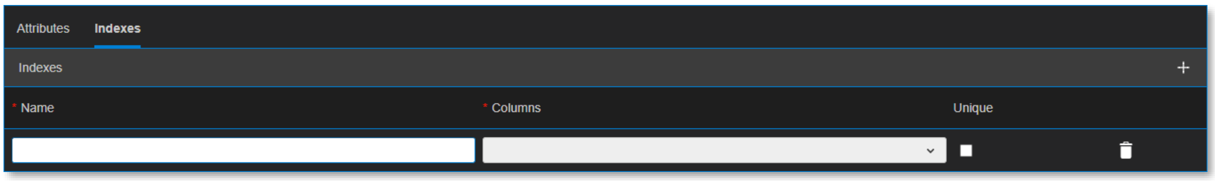
- **Attributes:** the attributes list defines the different attribute definitions of the trait definition; to add an attribute definition to a trait definition, you need to use the ⊞ on the top right of the attributes list; each attribute definition (see Fig. 5.6) has the following properties:
  - **Display name:** the user-friendly name of the attribute; this does not have to be unique, but it is recommended (to avoid confusion)
  - **Name:** the technical name of the attribute; this has to be unique within a trait definition, and it gets automatically populated for your convenience based on the display name you fill
  - **Data type:** the data type of the attribute; this is a pick list (bigint, boolean, date, double, integer, string and uuid)
  - **Default value:** the default value for the attribute (i.e.: the value that gets automatically assigned to the attribute when creating a new instance of the trait); whether it is possible to assign a default value and how to assign it depends on the chosen data type
  - **Size:** the size property only applies to the string data type and can thus only be chosen when picking the string data type; it represents the maximum length constraint for the string attribute
  - **Repeating:** whether or not the attribute is multi-valued (can have multiple values)
  - **Unique:** whether or not the attribute needs to be unique across all instances of the trait
  - **Required:** whether or not the attribute must be filled upon creation
  - **Read-only:** whether or not the attribute can be modified after creation
  - **Searchable:** whether or not the attribute can be filtered against when performing a search
  - **Sortable:** whether or not the attribute can be used to sort a search result

Fig. 5.6:



- **Indexes:** for performance and/or unique constraints reasons, it is possible to create indexes for certain attributes or a combination of attributes; the indexes list defines the different index definitions of the trait definition; to add an index definition to a trait definition, you need to use the ⊞ on the top right of the indexes list; each index definition (see Fig. 5.7) has the following properties:
  - **Name:** the name of the index
  - **Columns:** the different columns (i.e.: attributes) to which the index applies
  - **Unique:** whether or not a unique constraint needs to be enforced

Fig. 5.7:



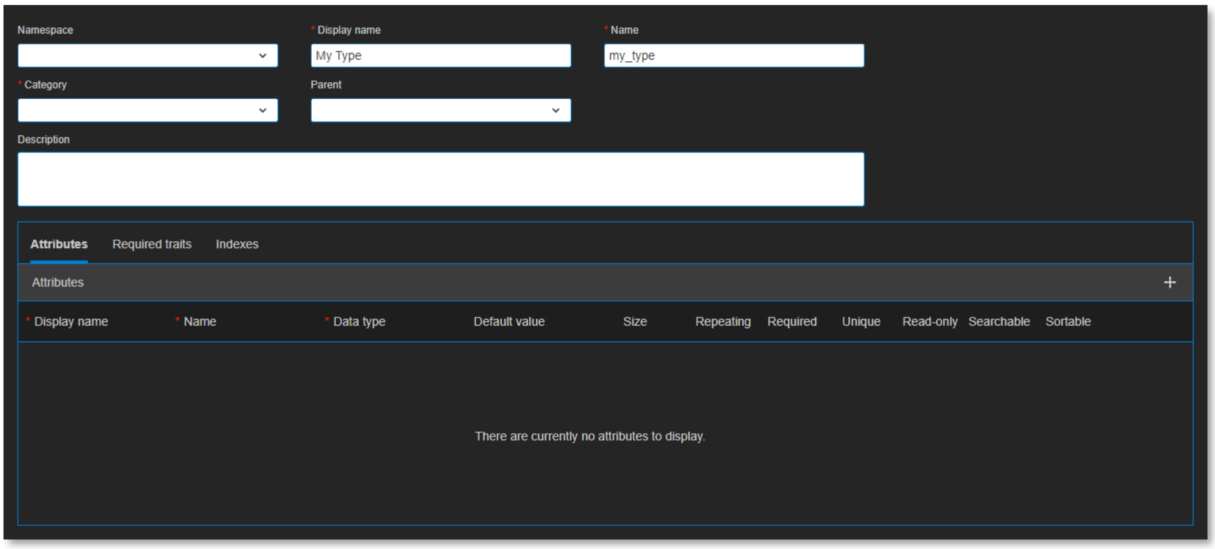
To save the trait definition model, select **Save** from the **File** menu, or press **Ctrl+S** on your keyboard.

5.3 Creating a type definition

A type definition is the main component for building your application's (custom) data model. A type definition has its own attributes and required traits (i.e.: traits that are always added to the type instance upon creation). A type definition can be of four categories: **object** (i.e.: object with metadata, but no content), **file** (i.e.: document with metadata and content), **folder** (i.e.: container for subfolders, objects and files) or **relation** (i.e.: relates/links type instances with each other). Type definitions also allow for inheritance within the same category (i.e.: base type). For more information on types (definitions and instances), you can refer to the Define a namespace, trait and "FILE" document type and Create instances using custom type with trait sections in the Content Metadata Service (CMS) product documentation or the Type resource documentation in the Content Metadata Service API reference.

You can create a type (i.e.: a type definition) via any of the three model creation methods. This opens the type modeler (see Fig. 5.8).

Fig. 5.8:

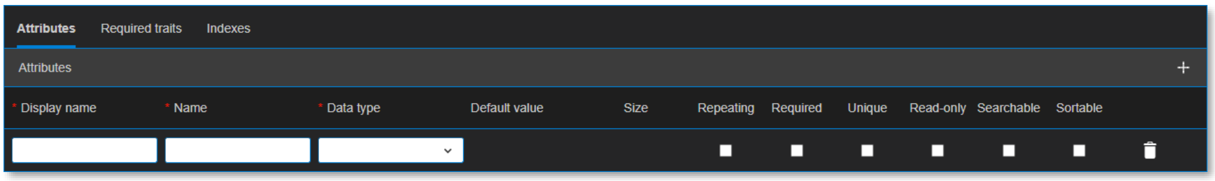


The different model properties on the type definition modeler screen are:

- **Namespace:** the namespace to which the type belongs; the namespace dropdown list is populated with the namespaces that exist within the project, and you can opt not to select any namespace
- **Display name:** the user-friendly name (i.e.: label) of the type; this does not have to be unique, and a default value is automatically filled using the file name you initially chose
- **Name:** the (technical) name of the type; this has to be unique (within your developer tenant), and a default value is automatically filled using the display name
- **Category:** the type category to which the type belongs; this can be **object**, **file**, **folder** or **relation**
- **Parent:** the parent type for the type, if it is a subtype of another; the parent dropdown list is populated with the types of the same category that exist within the project
- **Description:** the description of the type

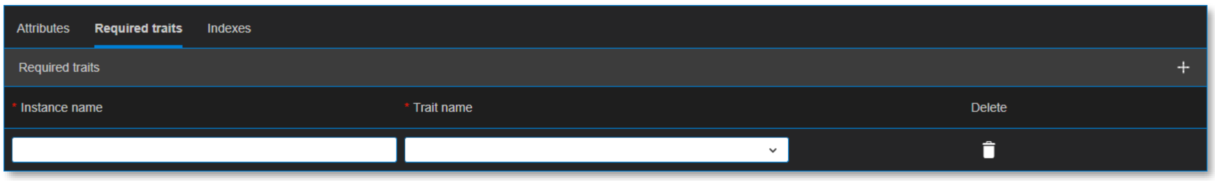
- **Attributes:** the attributes list defines the different attribute definitions of the type definition; to add an attribute definition to a type definition, you need to use the ⊞ on the top right of the attributes list; each attribute definition (see Fig. 5.9) has the following properties:
  - **Display name:** the user-friendly name of the attribute; this does not have to be unique, but this is recommended (to avoid confusion)
  - **Name:** the technical name of the attribute; this has to be unique within a type definition, and it gets automatically populated for your convenience based on the display name you fill
  - **Data type:** the data type of the attribute; this is a pick list (bigint, boolean, date, double, integer, string and uuid)
  - **Default value:** the default value for the attribute (i.e.: the value that gets automatically assigned to the attribute when creating a new instance of the type); whether it is possible to assign a default value and how to assign it depends on the chosen data type
  - **Size:** the size property only applies to the string data type and can thus only be chosen when picking the string data type; it represents the maximum length constraint for the string attribute
  - **Repeating:** whether or not the attribute is multi-valued (can have multiple values)
  - **Unique:** whether or not the attribute needs to be unique across all instances of the type
  - **Required:** whether or not the attribute must be filled upon creation
  - **Read-only:** whether or not the attribute can be modified after creation
  - **Searchable:** whether or not the attribute can be filtered against when performing a search
  - **Sortable:** whether or not the attribute can be used to sort a search result

Fig. 5.9:



- **Required traits:** the required traits list defines the different mandatory traits for the type definition; each required trait definition (see Fig. 5.10) has the following properties:
  - **Instance name:** the name of the required trait instance; this must be unique across the type definition's required traits
  - **Trait name:** the selected trait definition for the required trait instance; the trait name dropdown list is populated with the trait definitions that exist within the project

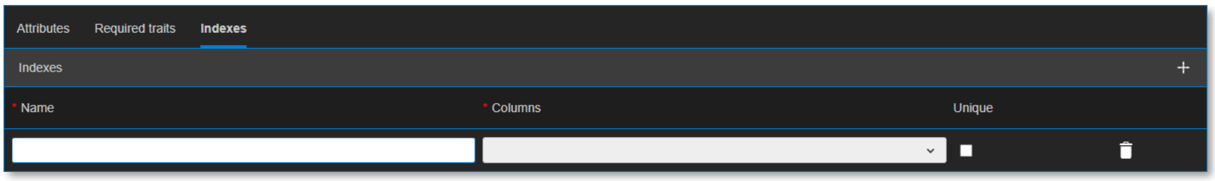
Fig. 5.10:



- **Indexes:** for performance and/or unique constraints reasons, it is possible to create indexes for certain attributes or a combination of attributes; the indexes list defines the different index definitions of the type definition; to add an index definition to a type definition, you need to use the ⊞ on the top right of the indexes list; each index definition (see Fig. 5.11) has the following properties:

- **Name:** the name of the index
- **Columns:** the different columns (i.e.: attributes) to which the index applies
- **Unique:** whether or not a unique constraint needs to be enforced

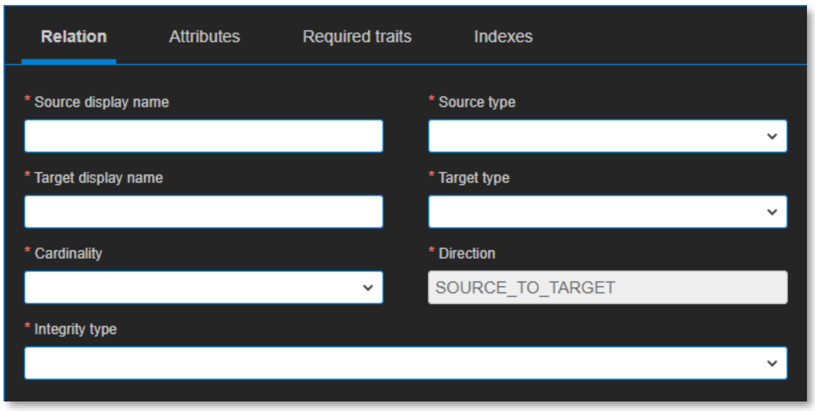
Fig. 5.11:



For type definitions of category **relation** (where the **Category** property is **relation**) an additional **Relation** tab is displayed (see Fig 5.12) to allow filling the different relation specific properties:

- **Source display name:** a user-friendly name for the relation source type (typically the meaning of the source type in the relation)
- **Source type:** the type definition for the source type
- **Target display name:** a user-friendly name for the relation target type (typically the meaning of the target type in the relation)
- **Target type:** the type definition for the target type
- **Cardinality:** the relation cardinality (can by “One to one”, “One to many” or “Many to many”)
- **Direction:** the direction of the relation (currently this can only be “SOURCE_TO_TARGET”)
- **Integrity type:** what to do upon deletion of relation source or target

Fig. 5.12:



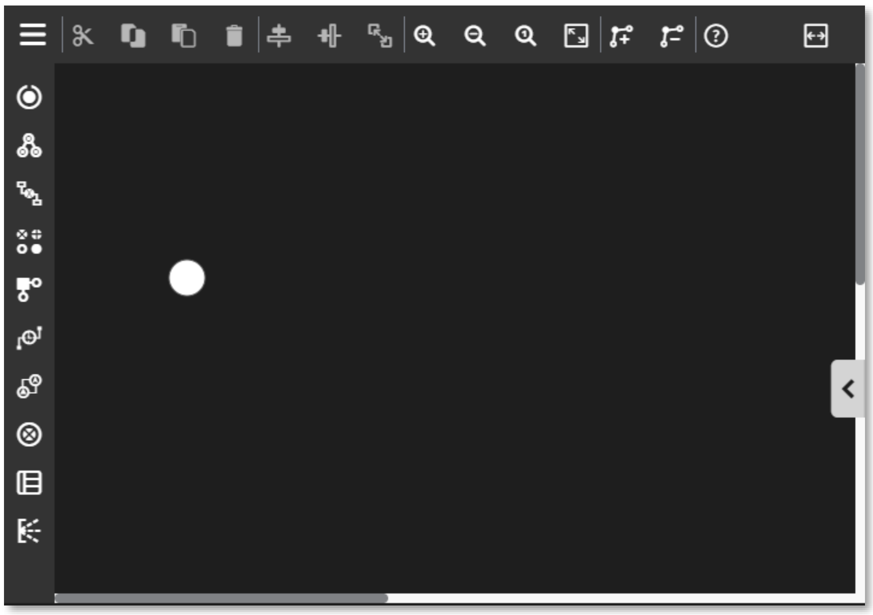
To save the type definition model, select **Save** from the **File** menu, or press **Ctrl+S** on your keyboard.

5.4 Creating a workflow definition

A workflow definition represents an executable process model from which process instances are created. The executable process model is stored as BPMN 2.0 encoded JSON. For more information on Workflow Service process models and process instances, you can refer to the Workflow Service product documentation , the Workflow Modeler product documentation or the Workflow Service API reference.

You can create a workflow (i.e.: a workflow definition) via any of the three model creation methods. This opens the workflow modeler (see Fig. 5.13).

Fig. 5.13:



For an exhaustive user guide of the workflow modeler, please refer to the Workflow Modeler product documentation.

To save the workflow model, select **Save** from the **File** menu, or press **Ctrl+S** on your keyboard.

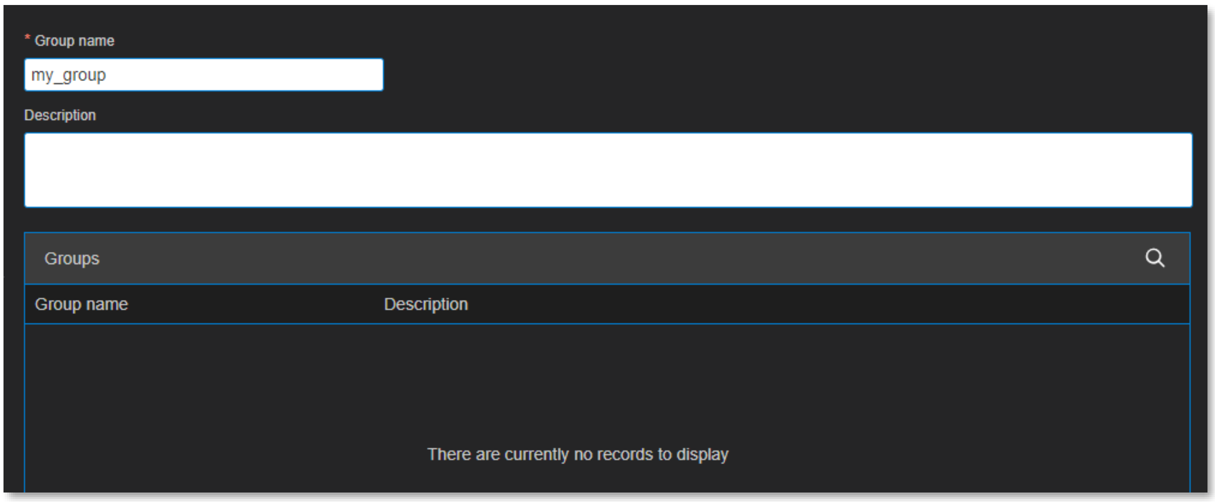
5.5 Creating a group

A (user) group can be used in the context of security (e.g.: as an accessor in an ACL), but also to represent a group of users that have a certain role for the application you are building. Groups can be nested (i.e.: one group can contain one or more other groups).

Adding users to a group is only possible once the group has been deployed (i.e.: this is a runtime and not a design-time activity). It can be done through the Admin Center, which can be opened from the Console view for your developer organization on **developer.opentext.com**. More specifically, you have to add users to the (subscription) groups that have been created inside of the application subscriptions. As there can be multiple application subscriptions in your organization (even multiple subscriptions in one tenant), you need to make sure that you add the users within the context of a specific subscription to the deployed application. This allows using multiple instances/subscriptions of the same application within the same organization or even tenant, each with a different set of users.

You can create a group via any of the three model creation methods. This opens the group modeler (see Fig. 5.14).

Fig. 5.14:



my_group

There are currently no records to display

The different model properties on the group modeler screen are:

- **Group name:** the name of the group; this has to be unique (within your subscription)
- **Description:** the description of the group
- **Groups:** the list of groups contained within the group (you can look up and select any group existing in the project); the list of contained groups shows the **Group name** and **Description** properties of the contained/nested groups

To save the group model, select **Save** from the **File** menu, or press **Ctrl+S** on your keyboard.

6 Deploying a project

This chapter describes how to deploy a project (i.e.: the application with its models) to the IMaaS API endpoints in the OpenText™ API Cloud. To be able to deploy a project, you must have set up an organization profile.

Deploying a project can be done via the **More Actions** menu (see Fig. 6.1) or the workspace (root) folder's contextual menu (see Fig. 6.2) in the **Explorer** view, or via the **Deploy to Default Tenant** command in the **Command Palette** (see Fig. 6.3), or via the **More Actions** menu of the model explorer in the **OpenText IMaaS Tools** view (see Fig 6.4).

Fig. 6.1:

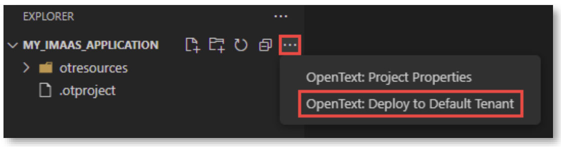

EXPLORER
MY_IMAAS_APPLICATION
otresources
.otproject
OpenText: Project Properties
OpenText: Deploy to Default Tenant

Fig. 6.2:

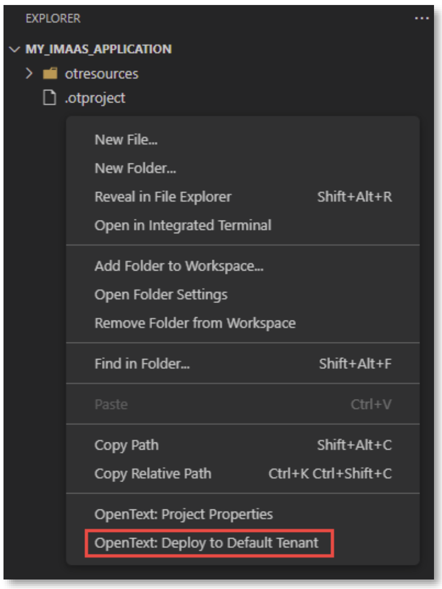

EXPLORER
MY_IMAAS_APPLICATION
otresources
.otproject
New File...
New Folder...
Reveal in File Explorer Shift+Alt+R
Open in Integrated Terminal
Add Folder to Workspace...
Open Folder Settings
Remove Folder from Workspace
Find in Folder... Shift+Alt+F
Paste Ctrl+V
Copy Path Shift+Alt+C
Copy Relative Path Ctrl+K Ctrl+Shift+C
OpenText: Project Properties
OpenText: Deploy to Default Tenant

Fig. 6.3:

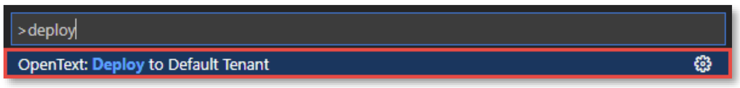
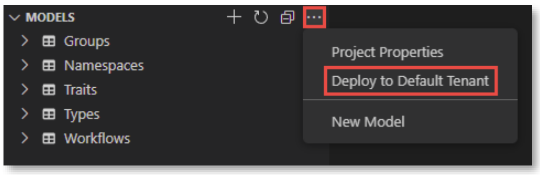


Fig. 6.4:



When deploying your application for the first time, the **Output** view automatically opens and displays the **OT Deployment** output. It contains the tenant ID and (application) API key data. These need to be kept securely, as they are to be used in your project code to authenticate/communicate with the IMaaS APIs in context of your application.

Once deployed, an application corresponding with your IMaaS project can be found in the Console view of your developer organization on **developer.opentext.com**.

Using the **Deploy to Default Tenant** option deploys your project to the default tenant in the default organization. In case an organization profile does not have any tenants configured for it, then deployment of the project is done to all tenants in corresponding organization.

To deploy your project to another organization or tenant mark that as default before triggering deployment.

In case you somehow lost the API key data for your application in a specific tenant then you can recreate this from the Developer Console.

7 Using the otcloud Command Line Interface

COMING SOON

8 Contact information

OpenText Corporation
275 Frank Tompa Drive
Waterloo, Ontario
Canada, N2L 0A1

For more information, visit www.opentext.com