OpenText™ Thrust Studio

User Guide

This guide describes how to use the OpenText™ Thrust Studio VS Code extension pack for developing and deploying applications that consume the OpenText Cloud Platform APIs.

Contents

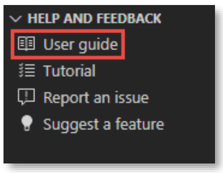
1 Introduction

This user guide provides instructions for pro-code developers on how to use OpenText™ Thrust Studio 25.2.5 to build applications that consume the IM (Information Management) APIs from the OpenText Cloud Platform.

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

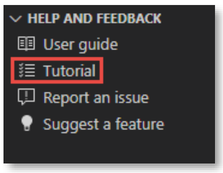
We recommend that you always use the link to this user guide from the OpenText Thrust Studio **Help and Feedback** section in VS Code (see Fig. 1.1), so that you are certain to have the up-to-date user guide which corresponds with the OpenText™ Thrust Studio version you have installed in your Visual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

Fig. 1.1:



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

Fig. 1.2:



2 Overview

Installing the OpenText™ Thrust Studio VS Code extension pack adds the OpenText Cloud Developer functionality to the Visual Studio Code IDE, referred to as VS Code from here on out.

OpenText™ Thrust Studio offers the OpenText Cloud Developer functionality in VS Code from three different locations:

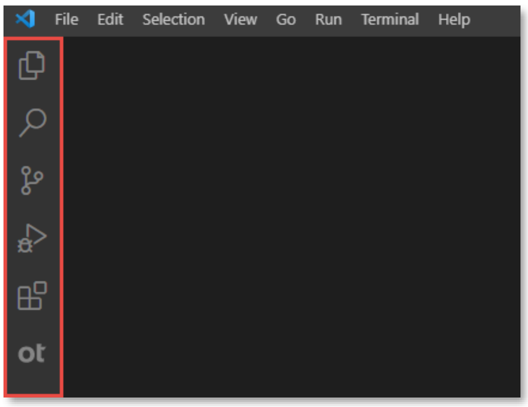
- The **OpenText Thrust Studio** view in the VS Code Activity Bar
- The **Explorer** view in the VS Code Activity Bar
- The VS Code **Command Palette**

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

2.1 OpenText Thrust Studio view

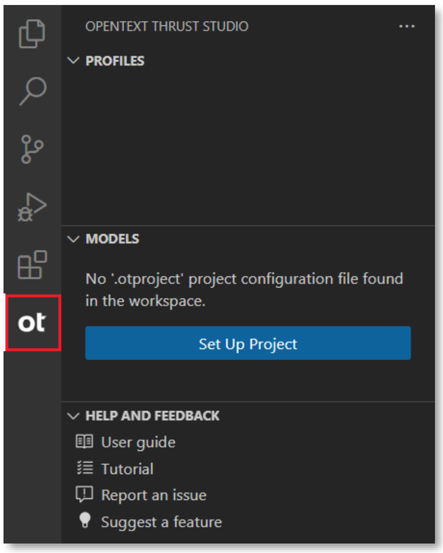
In VS Code, you can switch between different views through the Activity Bar (see Fig. 2.1).

Fig. 2.1:



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

Fig. 2.2:



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

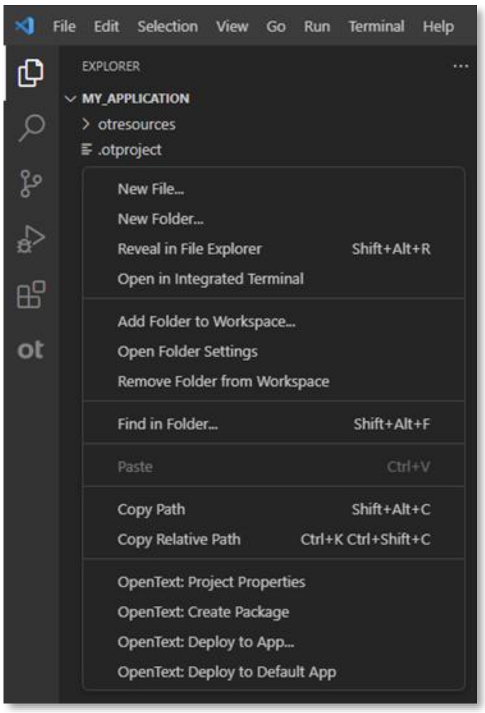
The OpenText Thrust Studio view is divided into three sections:

- **Profiles:**
  This is where the developer can configure authentication profiles to allow connecting and deploying to multiple developer organizations in the OpenText Developer Platform.
- **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™ Thrust Studio VS Code extension pack's user guide, the Cloud Developer tutorial (explains how to get started with developing applications for the OpenText Cloud Platform), 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™ Thrust Studio 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:


File
Edit
Selection
View
Go
Run
Terminal
Help
EXPLORER
MY_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: Create Package
OpenText: Deploy to App...
OpenText: Deploy to Default App

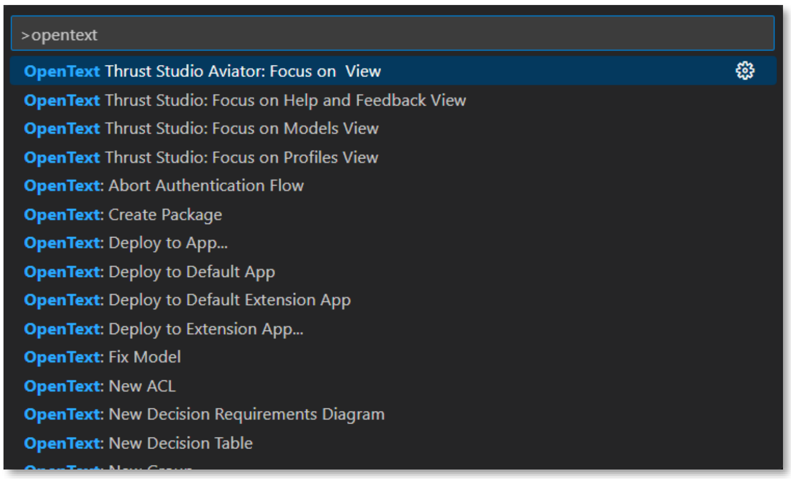
- Clicking the **OpenText: Project Properties** menu entry opens the OpenText Cloud application project properties screen, allowing you to edit the project and application properties.
- Clicking the **OpenText: Create Package** menu entry creates a local package of the project. The package can be stored for later deployment or to different organizations or tenants.
- Clicking the **OpenText: Deploy to App…** menu entry deploys your OpenText Cloud application project as a specific application to make it available in the tenants subscribed to that application.
- Clicking the **OpenText: Deploy to Default App** menu entry deploys your OpenText Cloud application project as the configured default application 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™ Thrust Studio 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™ Thrust Studio related commands.

Fig. 2.4:



>opentext

OpenText Thrust Studio Aviator: Focus on View

OpenText Thrust Studio: Focus on Help and Feedback View

OpenText Thrust Studio: Focus on Models View

OpenText Thrust Studio: Focus on Profiles View

OpenText: Abort Authentication Flow

OpenText: Create Package

OpenText: Deploy to App...

OpenText: Deploy to Default App

OpenText: Deploy to Default Extension App

OpenText: Deploy to Extension App...

OpenText: Fix Model

OpenText: New ACL

OpenText: New Decision Requirements Diagram

OpenText: New Decision Table

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™ Thrust Studio VS Code extension pack. The following chapters in this user guide describe how to use those features.

3 Setting up organization profiles

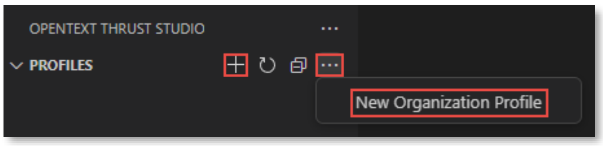
This chapter describes how to set up VS Code OpenText Cloud API organization profiles. More specifically, it details how to create a profile to your OpenText Cloud Platform organization(s), so that it is possible to use these profiles to authenticate and deploy your application project to the OpenText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

3.1 Adding an organization profile

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.

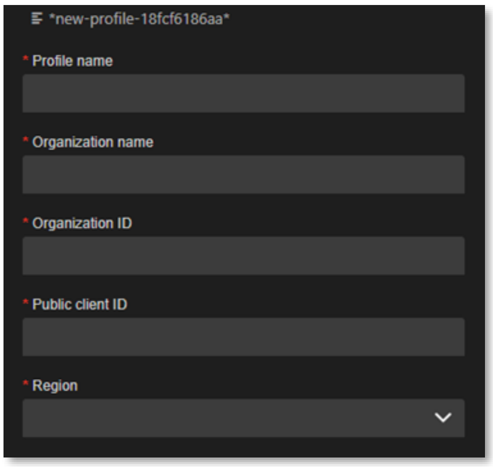
Adding a profile is done by using **New Organization Profile** from the contextual menu of the **Profiles** section (see Fig. 3.1).

Fig. 3.1:



Using this option opens the authentication profile configuration screen (see Fig. 3.2).

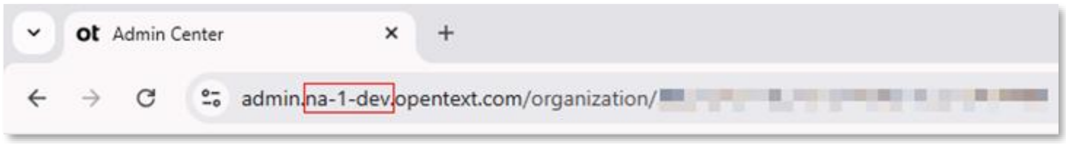
Fig. 3.2:



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`, `au`, `ca` and `us`. The region can be found by looking at the url that is used to open Admin Center (see Fig. 3.3).

Fig. 3.3:



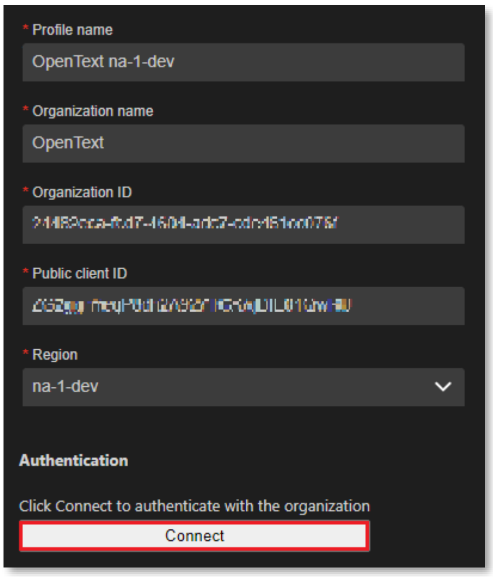
admin.na-1-dev.opentext.com/organization/[redacted]

Organization name, Organization ID and Public client ID are provided when creating an organization in **developer.opentext.com**, and they can also be retrieved from the organization 'Overview' and 'Service clients' page in Admin Center.

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. 3.4).

Fig. 3.4:



* Profile name	OpenText na-1-dev
* Organization name	OpenText
* Organization ID	[redacted]
* Public client ID	[redacted]
* Region	na-1-dev

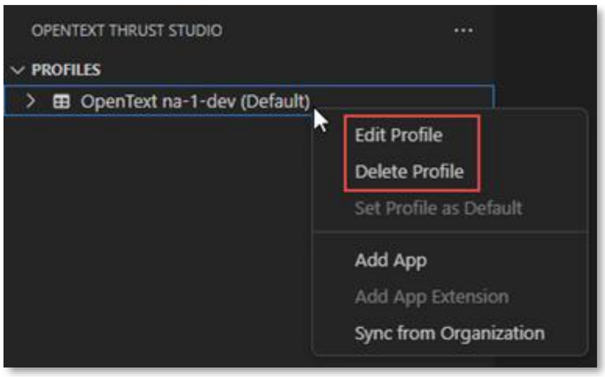
Authentication

Click Connect to authenticate with the organization

To change an existing organization profile, use the **Edit Profile** context menu option of the specific profile (see Fig. 3.5).

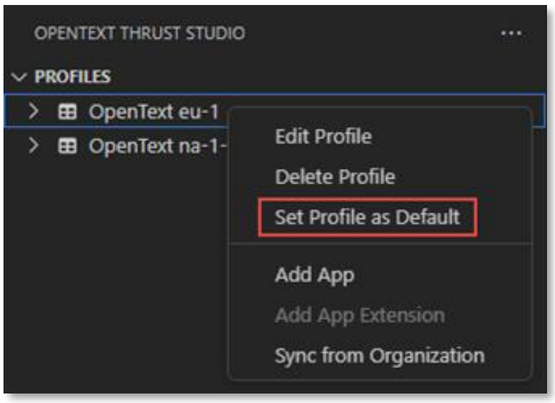
To delete an organization profile, use the **Delete Profile** context menu option of the specific profile. Note that in case you have multiple organization profiles that it is not possible to delete a profile that is marked as default. A default profile can only be deleted when it is the only existing profile (see Fig. 3.5).

Fig. 3.5:



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. 3.6).

Fig. 3.6:



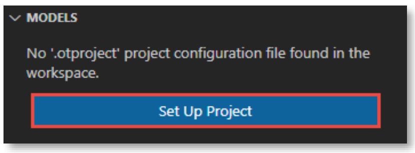
4 Setting up a project

This chapter describes how to set up an OpenText Cloud application project. Setting up (i.e.: configuring) a new project is essential, as it enables the OpenText Cloud 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 OpenText Cloud application project.

OpenText Thrust Studio supports two types of projects, one for standalone applications and the other to extend existing applications. Use a standalone application project when you develop an application that uses the OpenText Cloud Platform APIs on its own. An app extension project can be used to extend Core Content. For detailed information about that see the Core Content documentation.

Setting up a project is done via the **Set Up Project** button under the **Models** section of the **OpenText Thrust Studio** view (see Fig. 4.1). Note that this button is only available when a project has not yet been set up. When clicked, you are given the choice to select the project type (see Fig. 4.2). Select the project type for your project and the **OpenText Project Properties** configuration form opens (see Fig. 4.3).

Fig. 4.1:



MODELS

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

Fig. 4.2:

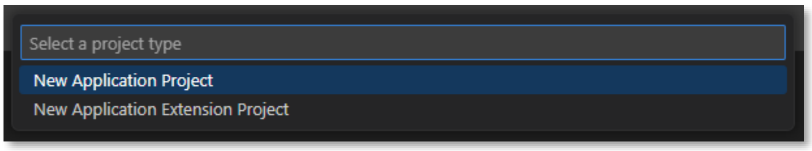
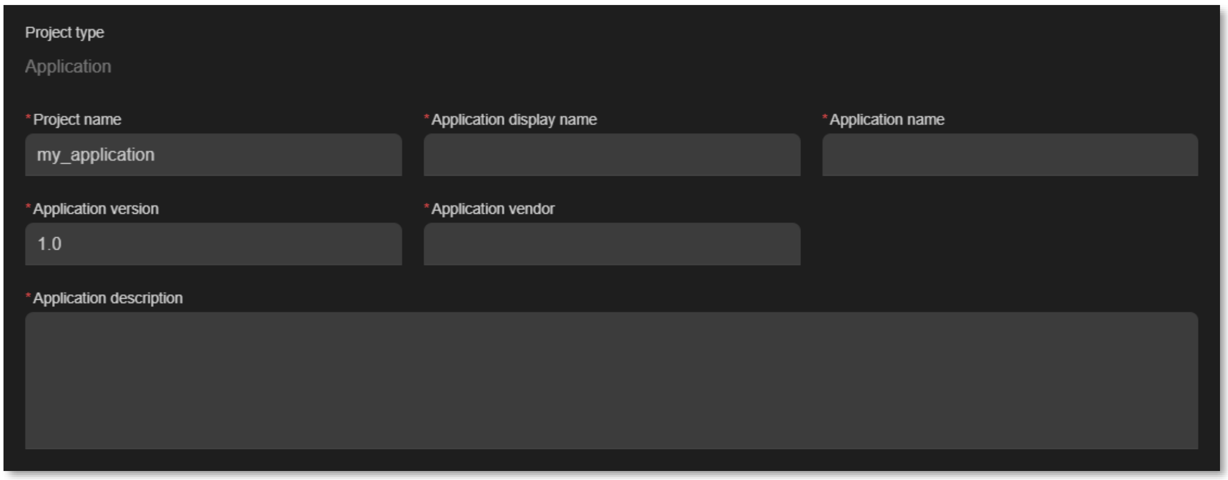


Fig. 4.3:



Project type
Application

*Project name: my_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:

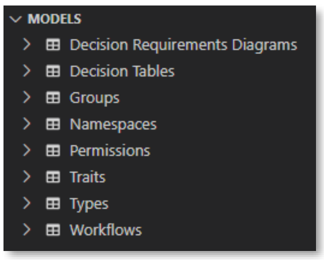
- **Project type:** the selected type of the project, either 'Application' or 'Application Extension' (read-only)

- **Project name:** the name of the 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 application
- **Application name:** the unique (technical) name of the application; a default value is automatically filled using the (previously entered) application display name
- **Application version:** the version label of the application, defaulted to "1.0"
- **Application vendor:** the name of the owner/vendor of the application
- **Application description:** the description 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. 4.4).

Fig. 4.4:



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. 4.5) 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. 4.6) or any of its subfolders, or by opening the **.otproject** file (see Fig. 4.7):

Fig. 4.5:

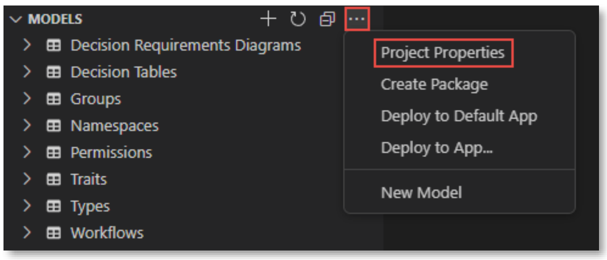


Fig. 4.6:

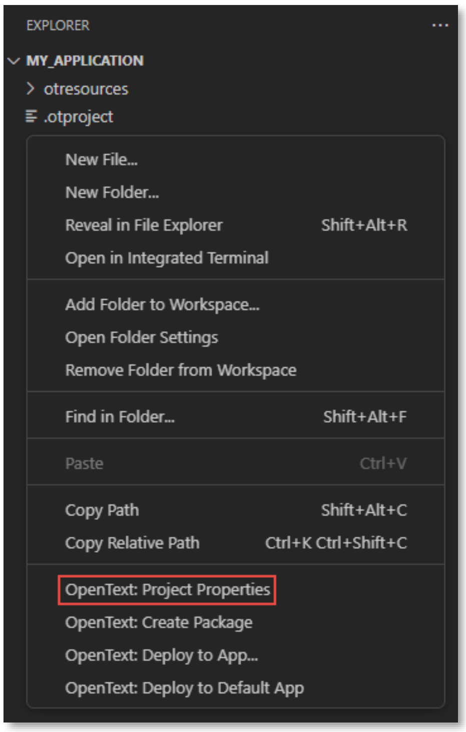
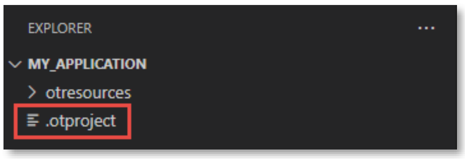

EXPLORER
MY_APPLICATION
otresources
.otproject
New File...
New Folder...
Reveal in File Explorer Shift+Alt+R
Open in Integrated Terminal
Add Folder to Workspace...
Open Folder Settings
Remove Folder from Workspace
Find in Folder... Shift+Alt+F
Paste Ctrl+V
Copy Path Shift+Alt+C
Copy Relative Path Ctrl+K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

Fig. 4.7:


EXPLORER
MY_APPLICATION
otresources
.otproject

5 Creating models

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

For each type of model, OpenText Thrust Studio provides 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) or the **New Model** menu entry of the **[…]** menu (see Fig. 5.2) of the **Model Explorer** (**Models** section of the **OpenText Thrust Studio** view).

   Fig. 5.1:

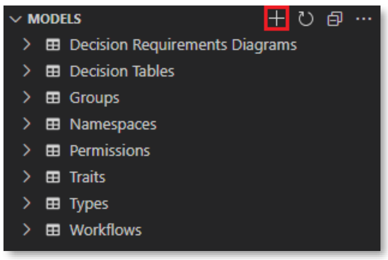
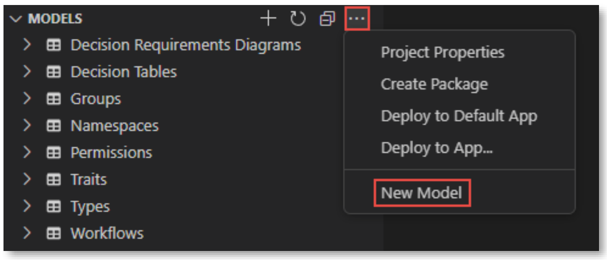
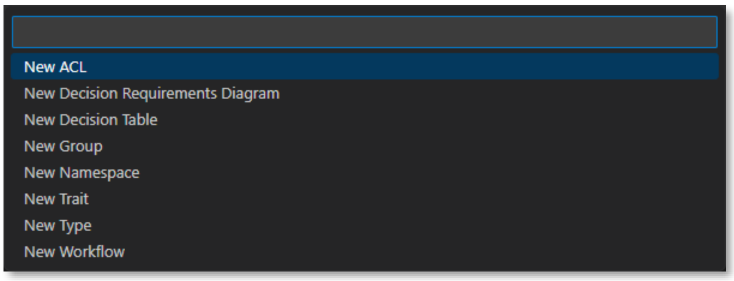
   

   Fig. 5.2:

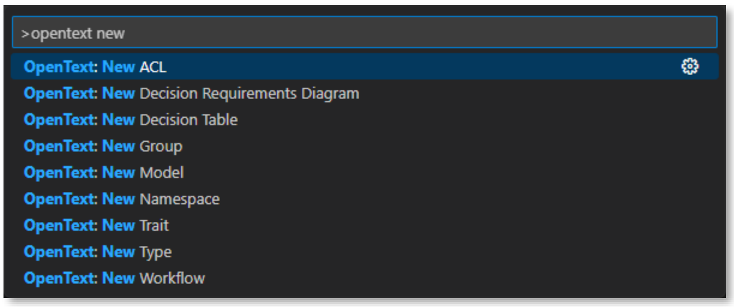
   This allows you to select the type of model you want to create (see Fig. 5.3). You are asked to provide a model name when clicking the chosen model type.

   Fig. 5.3:

2. Through the **Command Palette** (see Fig. 5.4). 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.

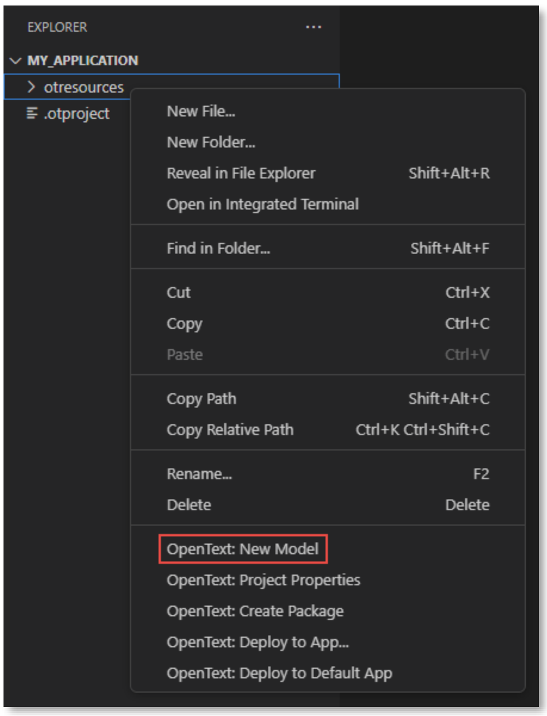
Fig. 5.4:



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. 5.5) from the contextual menu on a model folder (i.e.: the **otresources** folder or one of its subfolders).

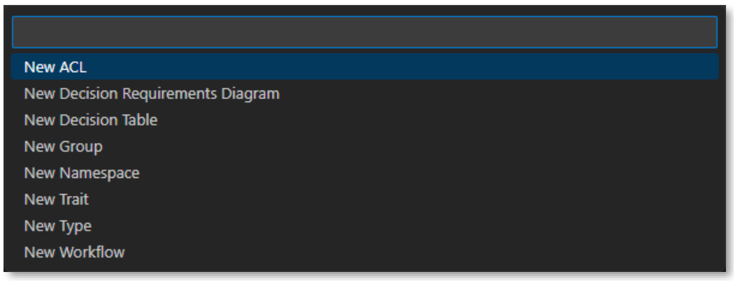
Fig. 5.5:



EXPLORER
MY_APPLICATION
otresources
.otproject
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: Create Package
OpenText: Deploy to App...
OpenText: Deploy to Default App

This allows you to select the type of model you want to create (see Fig. 5.6). You are asked to provide a model name when clicking the chosen model type.

Fig. 5.6:



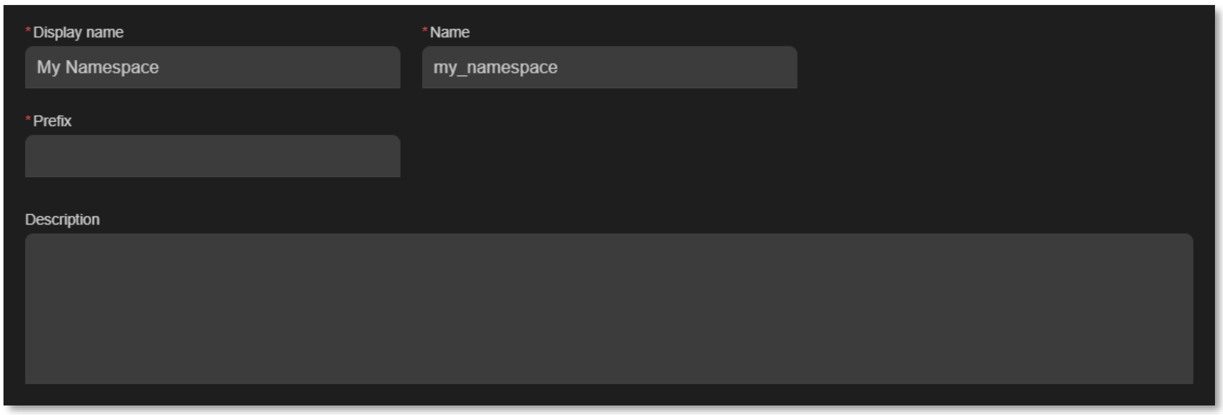
The remainder of this chapter describes the different modelers and how to use them to create the corresponding models.

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. 5.7).

Fig. 5.7:



*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 must 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 must be unique (within your developer tenant)
- **Description:** the description of the namespace

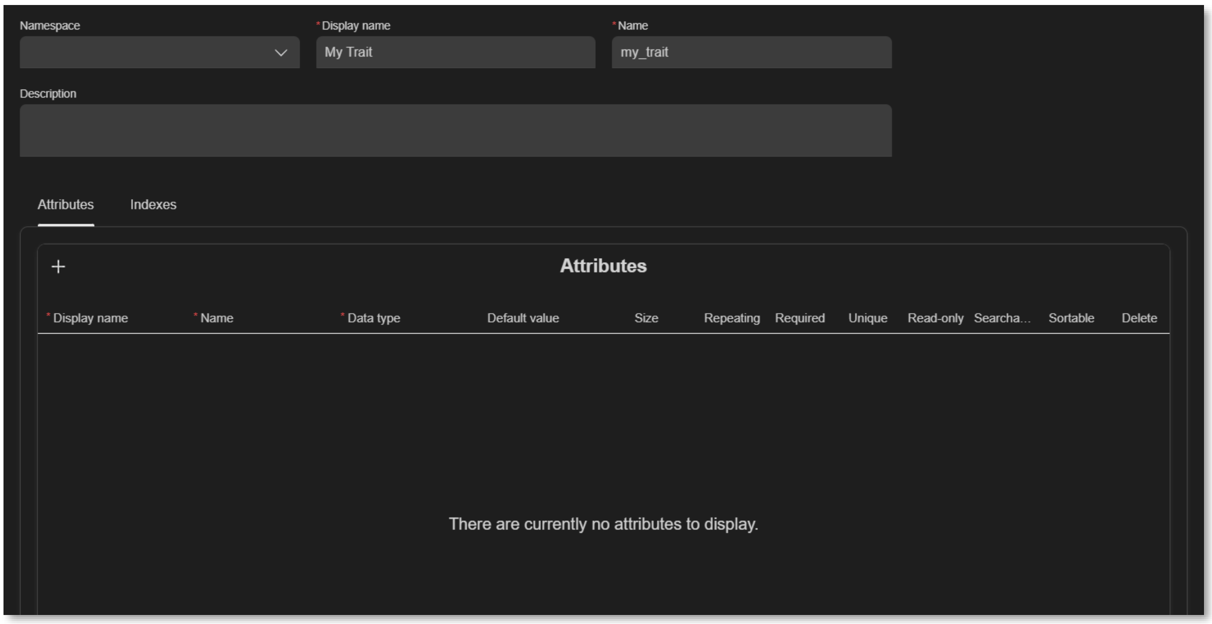
To save the namespace model, select **Save** from the **File** menu, or press **Ctrl+S** on your keyboard.

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. 5.8).

Fig. 5.8:



Namespace

*Display name: My Trait

*Name: my_trait

Description

Attributes | Indexes

Attributes

*Display name	*Name	*Data type	Default value	Size	Repeating	Required	Unique	Read-only	Searcha...	Sortable	Delete

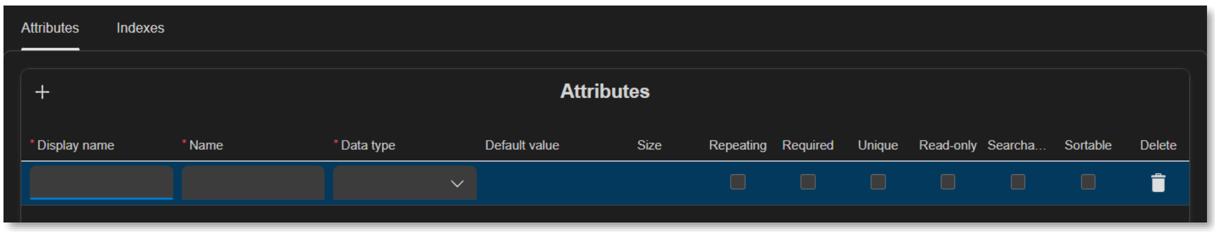
There are currently no attributes to display.

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 must 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. 5.9) 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 must 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, datetime, double, id, integer, string and user)
    - **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)
    - **Required:** whether or not the attribute must be filled upon creation
    - **Unique:** whether or not the attribute needs to be unique across all instances of the trait
    - **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

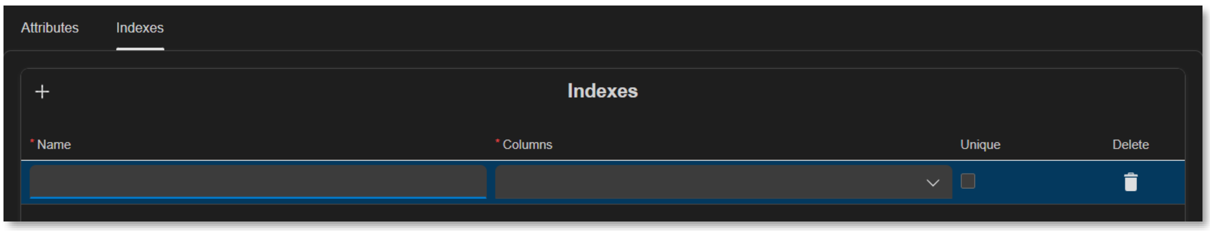
Fig. 5.9:



* Display name	* Name	* Data type	Default value	Size	Repeating	Required	Unique	Read-only	Searcha...	Sortable	Delete
					☐	☐	☐	☐	☐	☐	🗑

- **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. 5.10) 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. 5.10:



* Name	* Columns	Unique	Delete
		☐	🗑

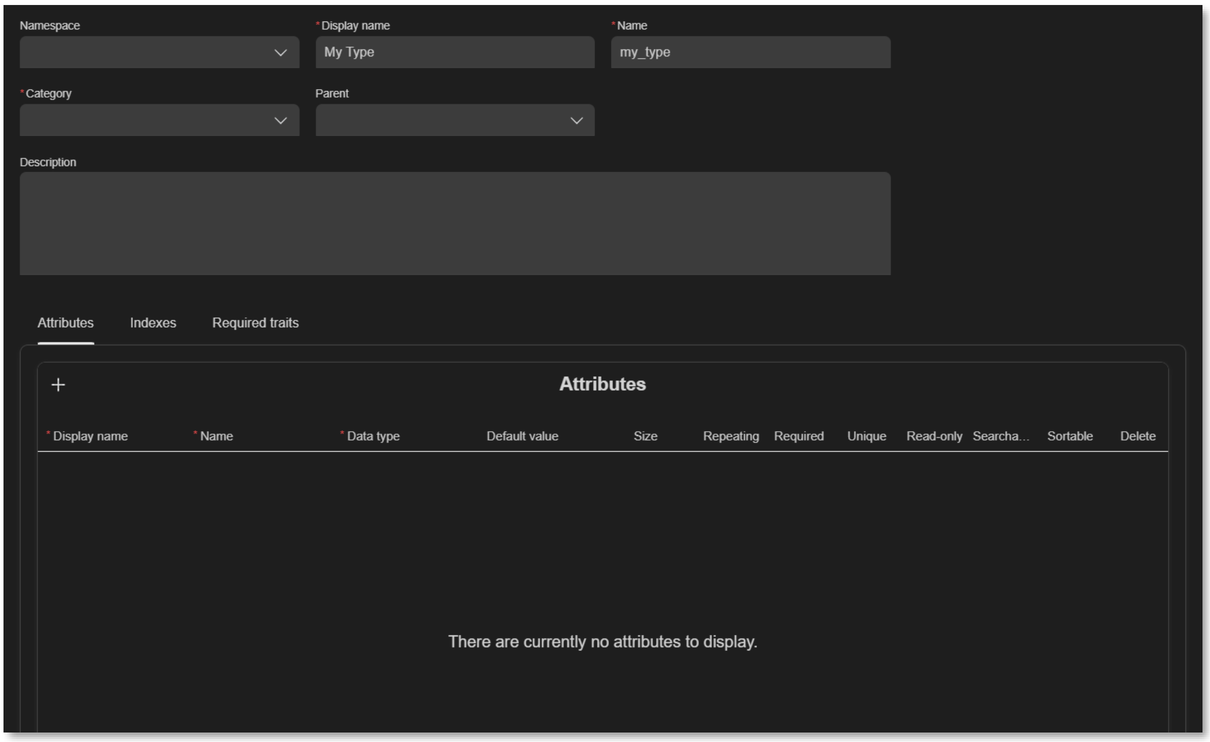
To save the trait definition model, select **Save** from the **File** menu, or press **Ctrl+S** on your keyboard.

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. 5.11).

Fig. 5.11:

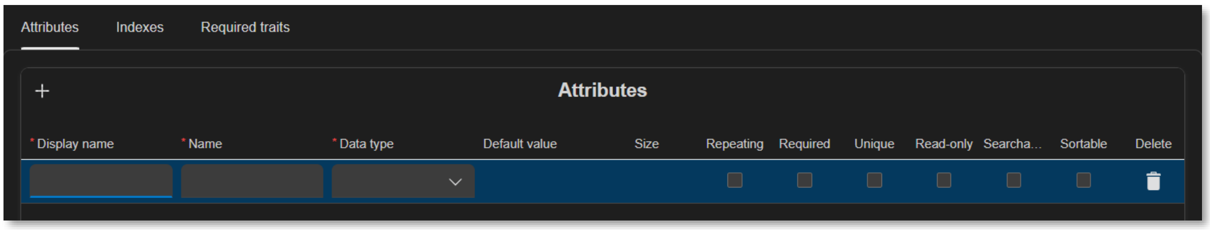


The different model properties on the type definition modeler screen are:

- **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 must 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

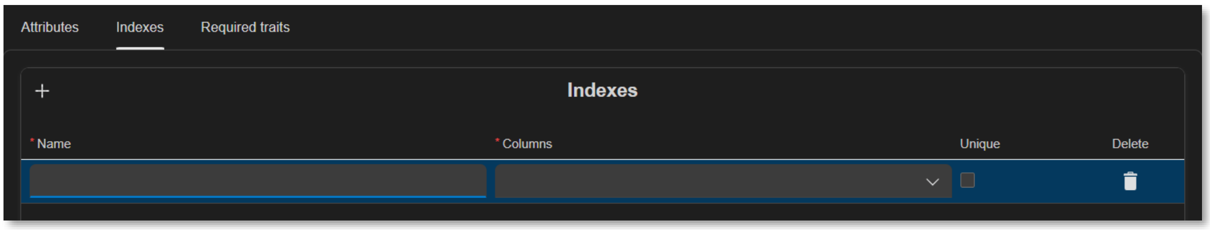
- **Description:** the description of the type
- **Attributes:** the attributes list defines the different attribute definitions of the type definition; to add an attribute definition to a type definition, you need to use the ⊞ on the top right of the attributes list; each attribute definition (see Fig. 5.12) 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 must 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, datetime, double, id, integer, string and user)
  - **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)
  - **Required:** whether or not the attribute must be filled upon creation
  - **Unique:** whether or not the attribute needs to be unique across all instances of the type
  - **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. 5.12:



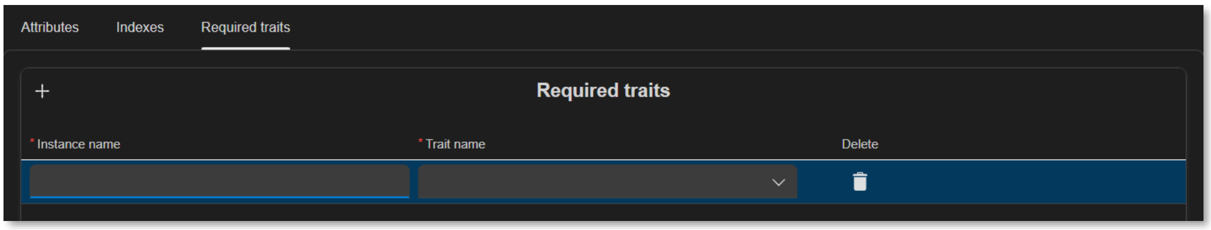
- **Indexes:** for performance and/or unique constraints reasons, it is possible to create indexes for certain attributes or a combination of attributes; the indexes list defines the different index definitions of the type definition; to add an index definition to a type definition, you need to use the ⊞ on the top right of the indexes list; each index definition (see Fig. 5.13) 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. 5.13:



- **Required traits:** the required traits list defines the different mandatory traits for the type definition; each required trait definition (see Fig. 5.14) 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

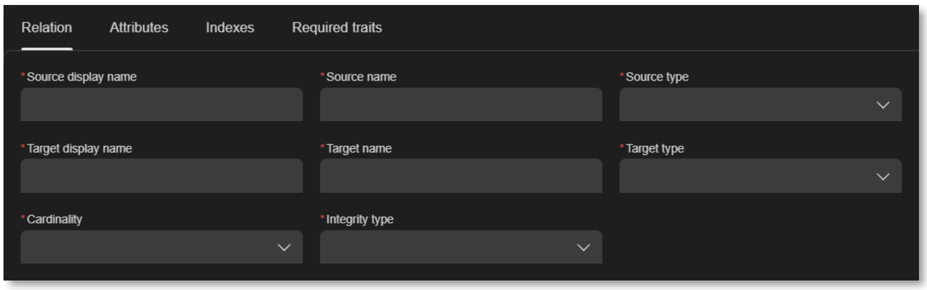
Fig. 5.14:



For type definitions of category **relation** (where the **Category** property is **relation**) an additional **Relation** tab is displayed (see Fig 5.15) 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 name:** the technical name of the relation source type
- **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 name:** the technical name of the relation target type
- **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”)
- **Integrity type:** what to do upon deletion of relation source or target

Fig. 5.15:



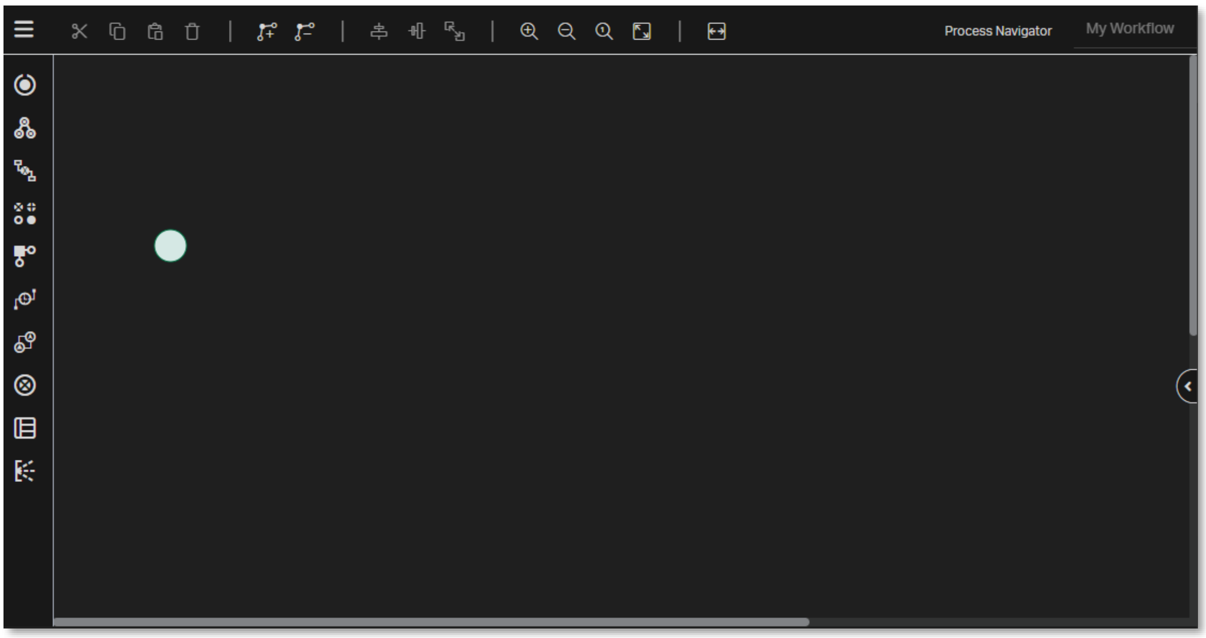
To save the type definition model, select **Save** from the **File** menu, or press **Ctrl+S** on your keyboard.

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. 5.16).

Fig. 5.16:



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

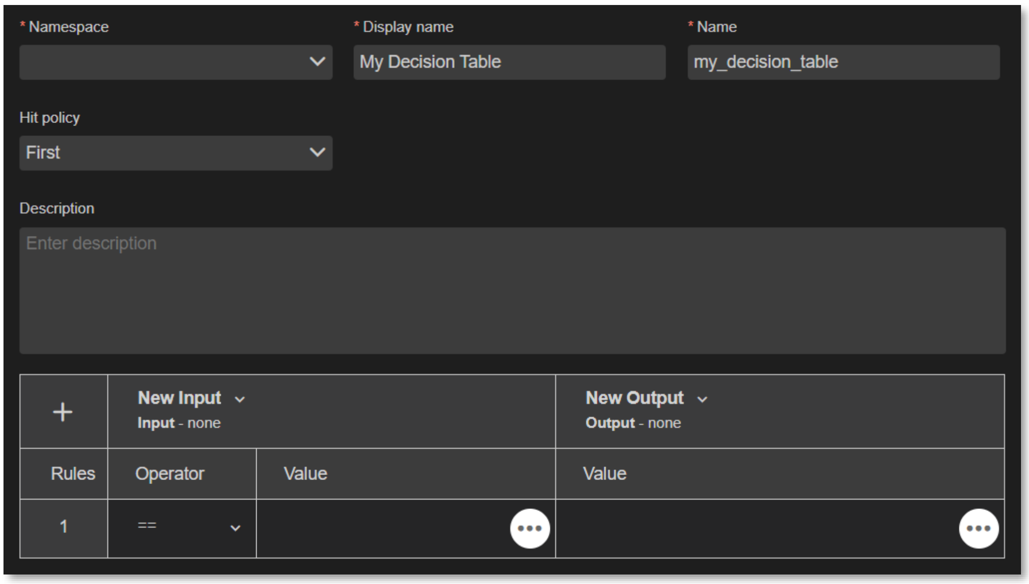
To save the workflow model, select **Save** from the **File** menu, or press **Ctrl+S** on your keyboard.

5.5 Creating a decision table definition

A decision table definition represents a set of defined business rules organized in an easy-to-understand tabular view. The stored decision table definition is compliant with the DMN (Decision Model and Notation) specification. For more information on Decision Service and the Decision Table Modeler, you can refer to the Decision Service product documentation, the Decision Table Modeler product documentation or the Decision Service API reference.

You can create a decision table via any of the three model creation methods. This opens the decision table modeler (see Fig. 5.17).

Fig. 5.17:



* Namespace:

* Display name: My Decision Table

* Name: my_decision_table

Hit policy: First

Description: Enter description

+	New Input Input - none		New Output Output - none
Rules	Operator	Value	Value
1	==		

For an exhaustive user guide of the decision table modeler, please refer to the Decision Table Modeler product documentation.

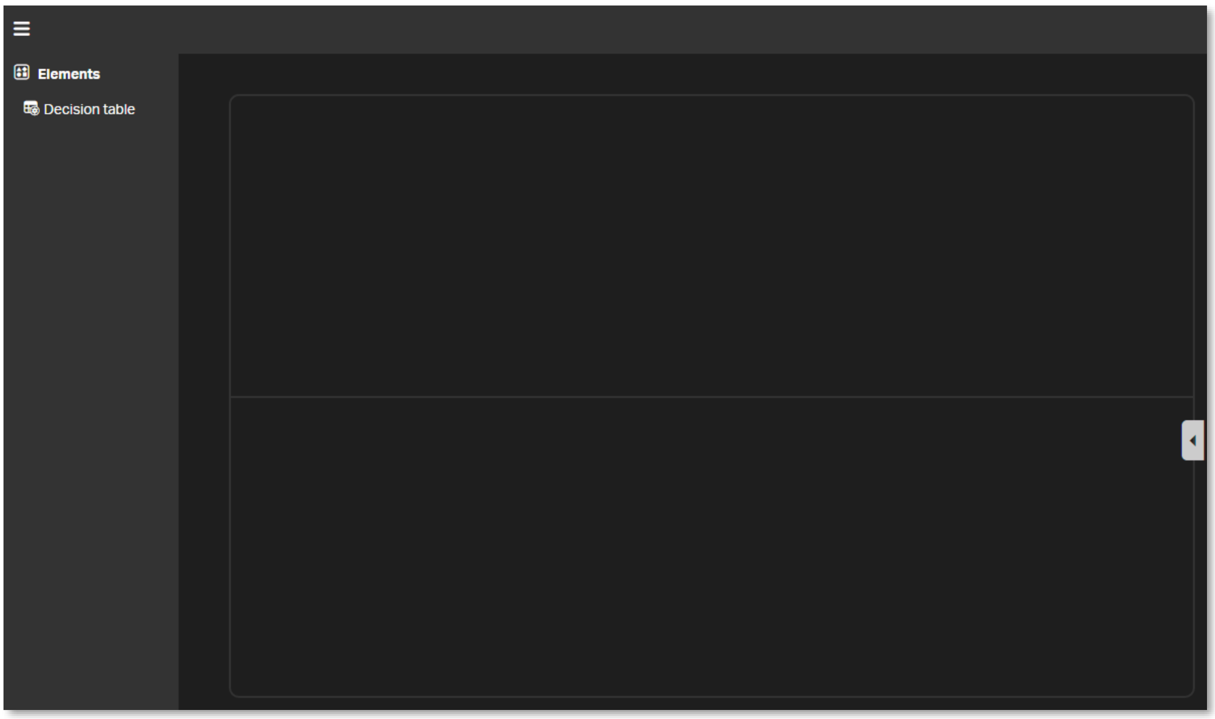
To save the decision table model, select **Save** from the **File** menu, or press **Ctrl+S** on your keyboard.

5.6 Creating a decision requirements diagram definition

A decision requirements diagram (DRD) definition allows chaining and executing several decision tables to achieve a business outcome. For more information on Decision Service and the Decision Requirements Diagram Modeler, you can refer to the Decision Service product documentation, the Decision Requirements Diagram Modeler product documentation or the Decision Service API reference.

You can create a decision requirements diagram via any of the three model creation methods. This opens the decision requirements diagram modeler (see Fig. 5.18).

Fig. 5.18:



For an exhaustive user guide of the decision requirements diagram modeler, please refer to the Decision Requirements Diagram Modeler product documentation.

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

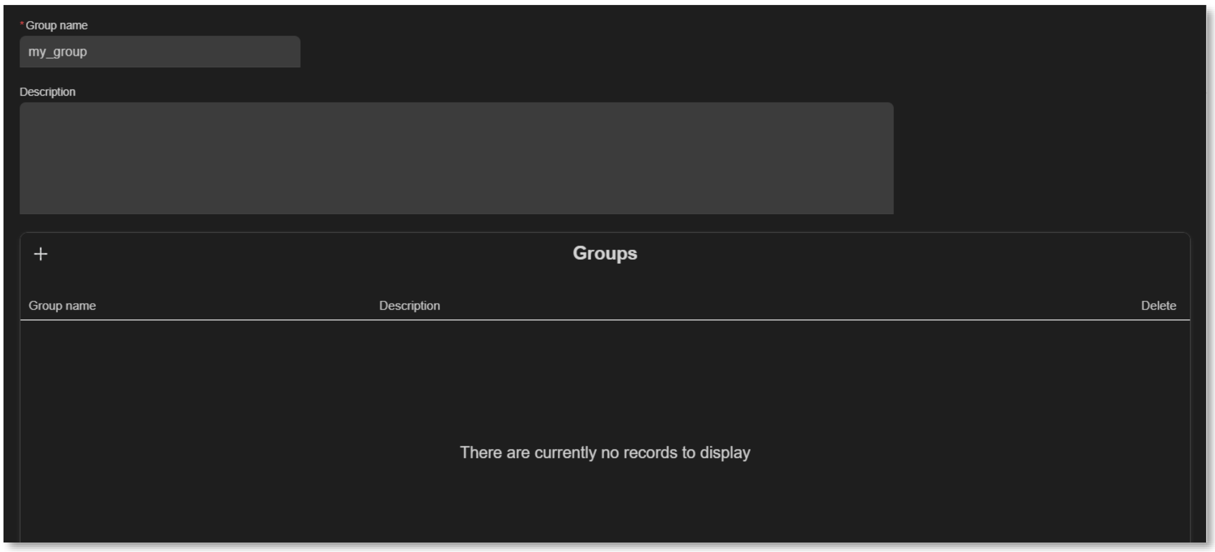
5.7 Creating a group

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. 5.19).

Fig. 5.19:



The different model properties on the group modeler screen are:

- **Group name:** the name of the group; this must 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.

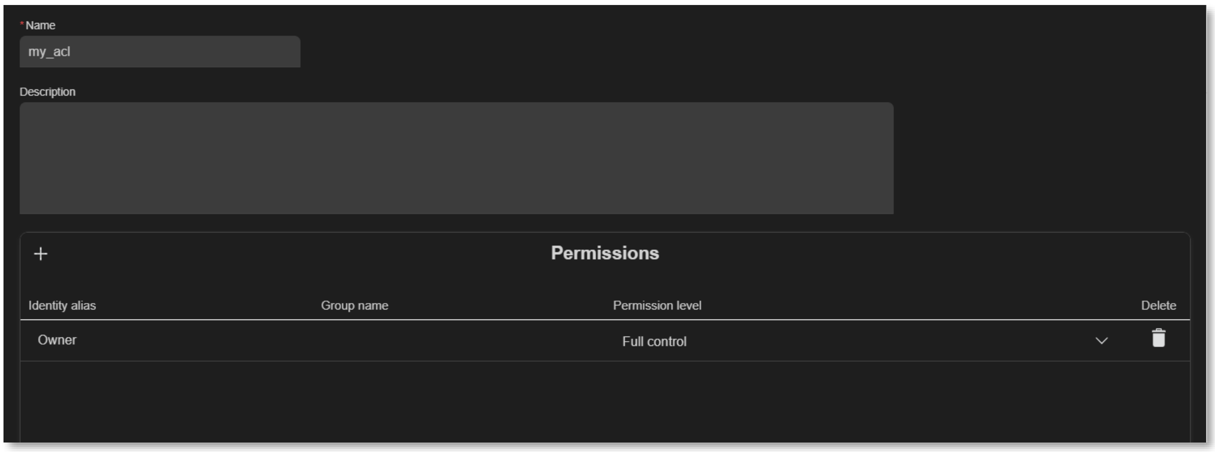
5.8 Creating an ACL

An ACL (Access Control List) determines the security for the object (type instance) it is applied to. Users are granted the appropriate authorization on the object, based on the group (cf. Creating a group) or alias ('owner' or 'everyone') permissions defined in the ACL. A user's authorization corresponds with the highest accessor permission in the ACL that matches the user.

It is not possible to add an individual user to an ACL as the ACL modeler is not directly connected to the user management system. Directly including a user in an ACL is also not recommended as it would not allow for easy portability (across environments) or changes in the users database.

You can create an ACL via any of the three model creation methods. This opens the ACL modeler (see Fig. 5.20).

Fig. 5.20:



* Name
my_acl

Description

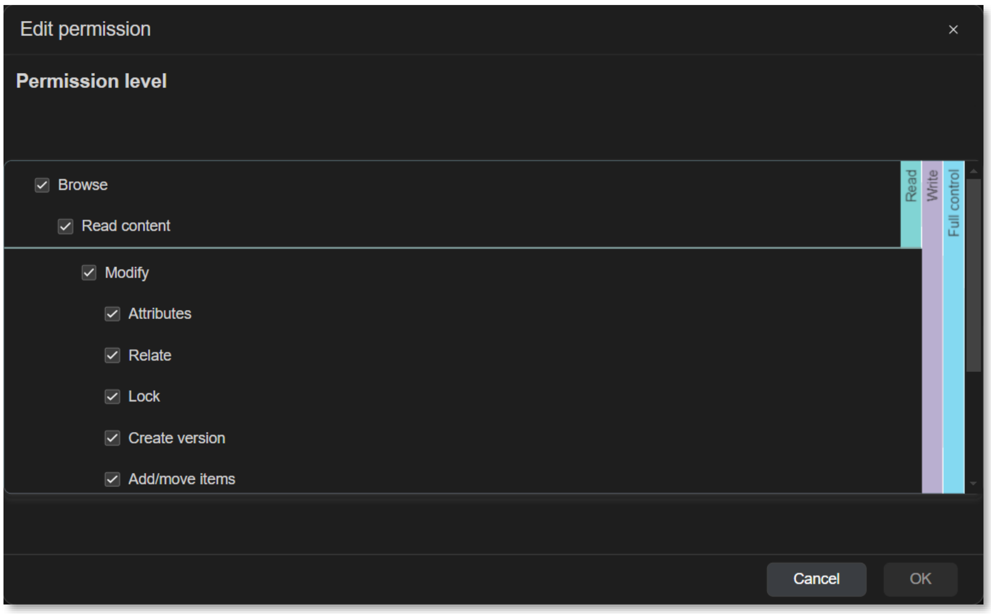
Permissions

Identity alias	Group name	Permission level	Delete
Owner		Full control	

The different model properties on the ACL modeler screen are:

- **Name:** the name of the ACL; this must be unique (within your subscription)
- **Description:** the description of the ACL
- **Permissions:** the list of permission definitions that define the permissions granted by the ACL; to add a permission definition to the ACL, you need to use the ⊞ on the top right of the permissions list; each permission definition (see Fig. 5.21) has the following properties:
  - **Identity alias:** the first part of the accessor identity; it defines whether the permission definition refers to the owner of the object, a specific user group or all users in the system; possible values are "Owner", "Group" or "Everyone"
  - **Group name:** the second part of the accessor identity; when the identity alias is "Group", it defines the group name
  - **Permission level:** the permission level that is granted to the accessor identity; possible values are "Read", "Write", "Full control" or "Custom"; the "Custom" permission level (see Fig. 5.21) allows for a more granular definition of the permission level (beyond "Read", "Write" and "Full control")

Fig. 5.21:


Edit permission

Permission level

- ✓ Browse
  - ✓ Read content
    - ✓ Modify
      - ✓ Attributes
      - ✓ Relate
      - ✓ Lock
      - ✓ Create version
      - ✓ Add/move items

Note that by default (when creating a new ACL) the owner of the object is granted “Full control” permissions. This can be modified if required.

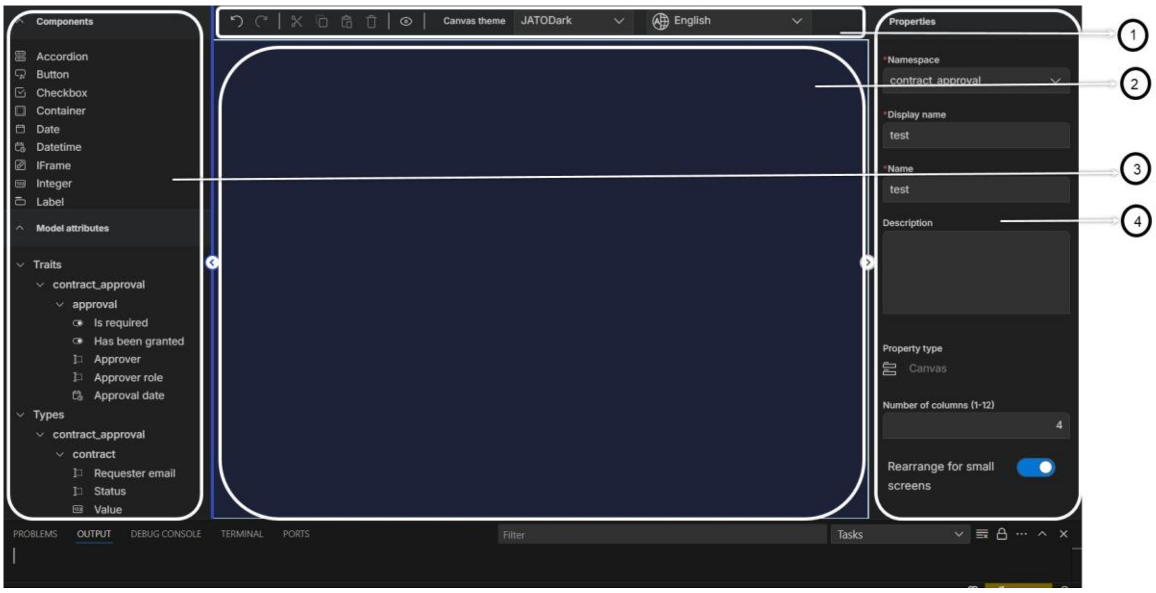
To save the ACL model, select **Save** from the **File** menu, or press **Ctrl+S** on your keyboard.

5.9 Creating UI components

The UI Modeler is the integration of the Forms API UI Designer in Thrust Studio. UI Modeler is a what you see is what you get (WYSIWYG) design tool that allows you to visually create forms and UI elements without writing code.

Following are the different elements in the Thrust Studio UI Modeler:

Fig. 5.22:



*Namespace: contract_approval

*Display name: test

*Name: test

Description

Property type: Canvas

Number of columns (1-12): 4

Rearrange for small screens

Following table provides details of the elements marked in the preceding layout:

Number	Name
1	Toolbar
2	Canvas
3	Control palette
4	Properties

The Control palette contains different controls that can be clicked and dragged to build a form or UI component. You can build a form or UI component using the Components from the Controls palette. Controls palette contains the following sections:

- Components: Lists the production ready UI controls. The property values must be configured using the Properties panel. For information about the controls and their properties, see Components.
- Model attributes: Lists the bound controls that are predefined property fields from Traits, Types, and Workflows. These data models get injected programmatically from the different models existing in your project. The property values for the controls can be added or modified using the Properties panel. When possible to change them, the values provided on the form override the preconfigured values from the models.

To understand the different elements in UI Modeler and their functionality, see UI Designer.

It is required to provide the Name and Namespace attributes during the UI component model creation as their combination uniquely represents the UI component in the subscription.

Make sure to complete all mandatory UI component attributes in the **Properties** section.

Following are the UI component attributes:

- **Namespace**: The namespace to which the UI component belongs. The Namespace property field lists the namespaces that exist within the project. The namespace must be selected. In combination with the Name property, the Namespace allows to uniquely identify the UI component in the subscription.
  Note that if the namespace is not selected, an error appears in the Thrust Studio Problems tab.
- **Display name**: Display name for the UI component. A default value is automatically filled using the model (file) name you initially chose.
- **Name**: The technical name of the UI component. A default value is automatically filled using the display name. In combination with the Namespace property, the Name allows to uniquely identify the UI component in the subscription.
- **Description**: Description for the form created.
- **Control type**: This is a read-only property that displays the type of the selected control. At UI component canvas/root level, this is "Canvas".
- **Number of columns (1-12)**: Allows selecting the number of grid columns in the container for configuring the form layout.
- **Rearrange for small screens**: Rearranging the UI elements for smaller screens. This ensures a user-friendly experience on smaller devices while maintaining the functionality and aesthetics.

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

See Components section for information about the properties displayed for the selected control.

6 Deploying a project

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

To deploy a standalone application or an application extension, an App or App Extension is required under an organization profile. An App and App Extension are the link between the application project and the app subscription in a tenant. It is possible to have multiple apps in your organization, all based on the same project, but with a different lifecycle. As an example, there can be an app used by the development team, another app used by the QA team, and a production version of the app used by end-users. These apps have their own API keys (service clients) and are managed separately, by different people. An app can have one or more subscribed tenants, where a tenant only can be subscribed to a single app. All tenants subscribed to an app use the same app version and are updated together when a new version of the project is deployed to that app.

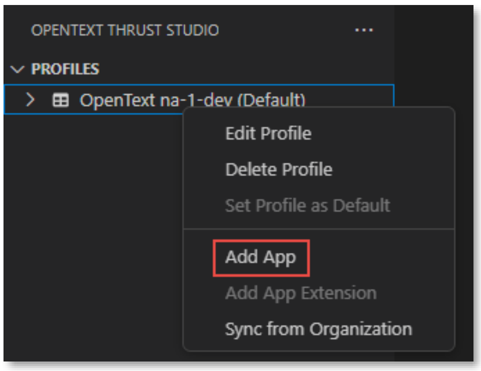
Depending on the project type you must either follow the steps to deploy a standalone application project or to deploy an application extension project.

6.1 Deploying a standalone application project

6.1.1 Creating a standalone application under a profile

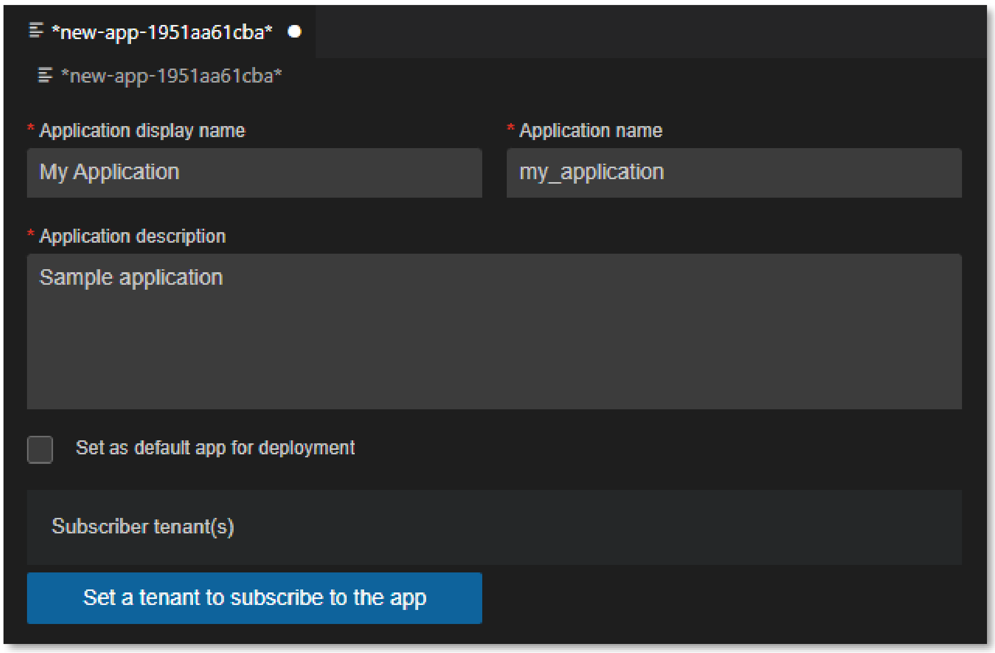
To add an app under an organization profile, use the **Add App** context menu option of that specific profile (see Fig. 6.1).

Fig. 6.1:



Using this option opens the new app configuration screen (see Fig. 6.2).

Fig. 6.2:



new-app-1951aa61cba

new-app-1951aa61cba

* Application display name: My Application

* Application name: my_application

* Application description: Sample application

☐ Set as default app for deployment

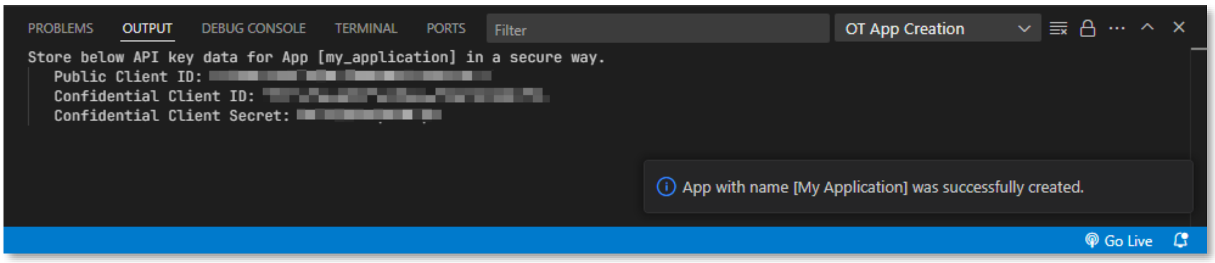
Subscriber tenant(s)

The different properties on the **New App** configuration screen are:

- **Application display name:** the display name of the app
- **Application name:** the name of the app
- **Application description:** the description of the app
- **Set as default app for deployment:** whether this app is used as the default app for deployment of the project
- **Subscriber tenant(s):** list of tenants that are subscribed to this app.
  For new apps use the **Set a tenant to subscribe to the app** button to select a tenant to subscribe to this app. When deploying the project, it will be deployed to this tenant.

Save the new app. Thrust Studio creates the app in Admin Center and displays the service client related to the new app (see Fig. 6.3). The **Output** view automatically opens and displays the **OT App Creation** output. It contains the application API key data. This API key together with the tenant id of the subscribed tenant needs to be kept securely, as this is to be used in your project code to authenticate/communicate with the OpenText Cloud Platform APIs in context of your application.

Fig. 6.3:



```
Store below API key data for App [my_application] in a secure way.
   Public Client ID: [redacted]
   Confidential Client ID: [redacted]
   Confidential Client Secret: [redacted]
```

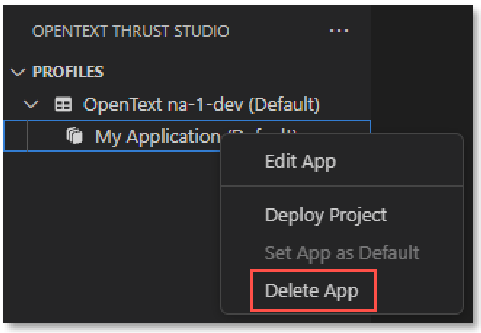
App with name [My Application] was successfully created.

Note that only the app entry and service clients are created. The project and its models are not yet deployed.

6.1.2 Deleting an App

To delete an app from an organization profile use the **Delete App** option (see Fig. 6.4) in the contextual menu of the specific app.
Note: this will not only delete the app from your local profile, but the app will also be deleted from Admin Center, and with that it will be deleted for everyone.

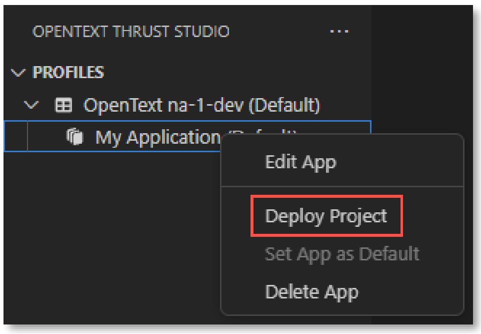
Fig. 6.4:



6.1.3 Deploying the project

Next step is to deploy your project to the app. This can be done via the **Deploy Project** (see Fig. 6.5) option in the app's contextual menu. Using this option will deploy the models as part of your project to the app that was created in the specific organization.

Fig. 6.5:



To see the detailed deployment steps open the ALM Output channel in the **Output** view. The view contains the progress and status of the deployment. In case of an error this view also shows more details about the error.

6.1.4 Deploying to a default or specific app

Deploying a project can also be done via the **Deploy to Default App** option in the **More Actions** menu (see Fig. 6.6) or the workspace (root) folder's contextual menu (see Fig. 6.7) in the **Explorer** view, or via the **OpenText: Deploy to Default App** command in the **Command Palette** (see Fig. 6.8), or via the **More Actions** menu of the model explorer in the **OpenText Thrust Studio** view (see Fig 6.9).

Using the **Deploy to Default App** option deploys your project to the default app in the default organization.

To deploy your project to another organization or app either mark that as default before triggering deployment or use the **Deploy to App...** option. Using this option allows you to select a specific app, from all apps that are related to the project, to be used for deployment.

Fig. 6.6:

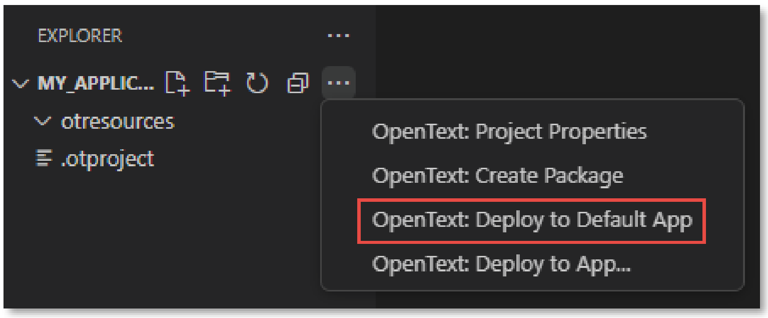

EXPLORER
MY_APPLIC...
otresources
.otproject
OpenText: Project Properties
OpenText: Create Package
OpenText: Deploy to Default App
OpenText: Deploy to App...

Fig. 6.7:

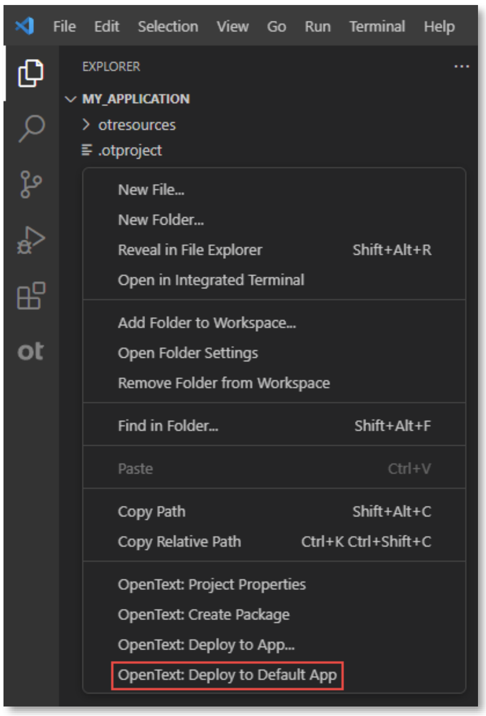

File Edit Selection View Go Run Terminal Help
EXPLORER
MY_APPLICATION
otresources
.otproject
New File...
New Folder...
Reveal in File Explorer Shift+Alt+R
Open in Integrated Terminal
Add Folder to Workspace...
Open Folder Settings
Remove Folder from Workspace
Find in Folder... Shift+Alt+F
Paste Ctrl+V
Copy Path Shift+Alt+C
Copy Relative Path Ctrl+K Ctrl+Shift+C
OpenText: Project Properties
OpenText: Create Package
OpenText: Deploy to App...
OpenText: Deploy to Default App

Fig. 6.8:

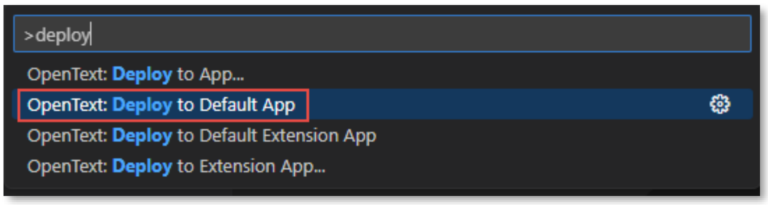
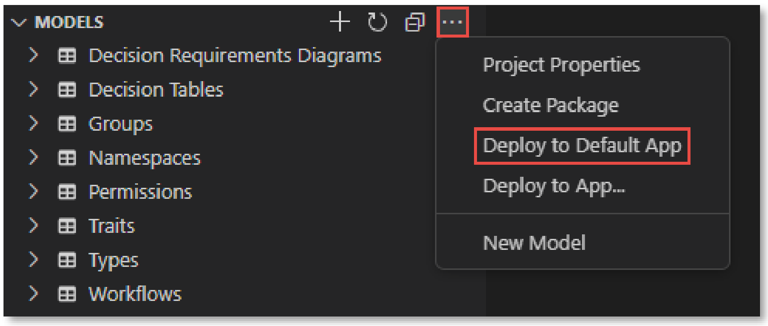

>deploy
OpenText: Deploy to App...
OpenText: Deploy to Default App
OpenText: Deploy to Default Extension App
OpenText: Deploy to Extension App...

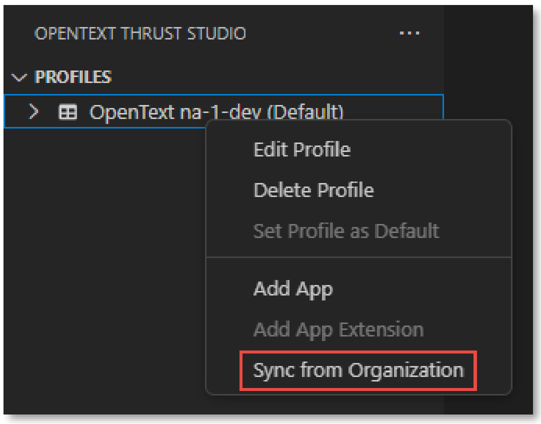
Fig. 6.9:



6.1.5 Synchronize apps for the project from Admin Center

When working in a team it can be that someone else created one or more apps for the project already. These apps are available in Admin Center and can be synchronized to your local Thrust Studio configuration. To synchronize apps from Admin Center use the **Sync from Organization** option (see Fig. 6.10). This option adds the apps available in Admin Center to the organization profile. Note that only those apps that are related to the open project are added.

Fig. 6.10:



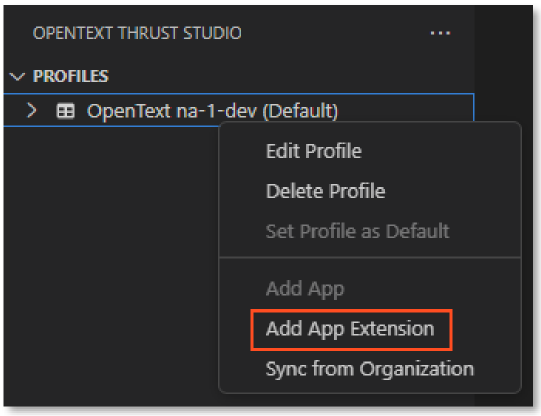
6.2 Deploying an application extension project

To deploy an application extension project an app extension is used. The app extension is the link between the project and the target (i.e., extended) Core app subscription in a tenant. Deploying the project creates the project models, for example the workflows, in the target subscription. This mechanism can be used to extend Core Content, as it allows to deploy an advanced workflow in a Core Content subscription from where it can be used in a business workflow. That way it is possible to add more extensive and complex integrations to Core Content. See the Core Content documentation for more information.

6.2.1 Creating an application extension under a profile

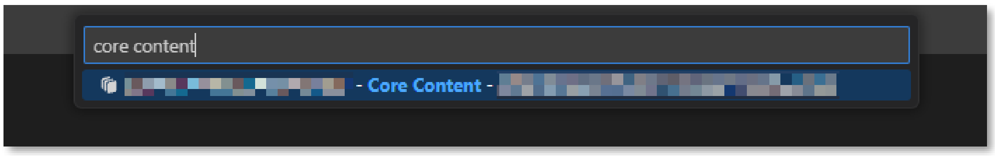
To add an app extension under an organization profile, use the **Add App Extension** context menu option of that specific profile (see Fig. 6.11).

Fig. 6.11:



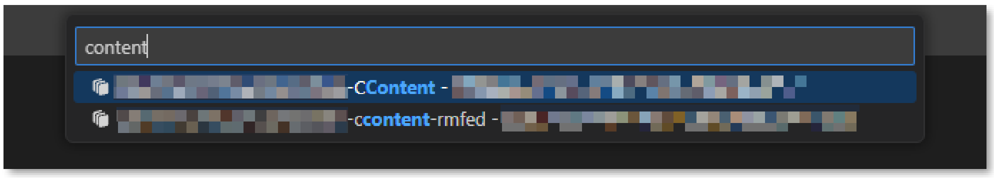
Using this option shows the list of available tenants in your organization. Select the tenant that contains your Core Content subscription. You can filter the tenants by typing a part of the name or id to search for the correct tenant (see Fig. 6.12).

Fig. 6.12:



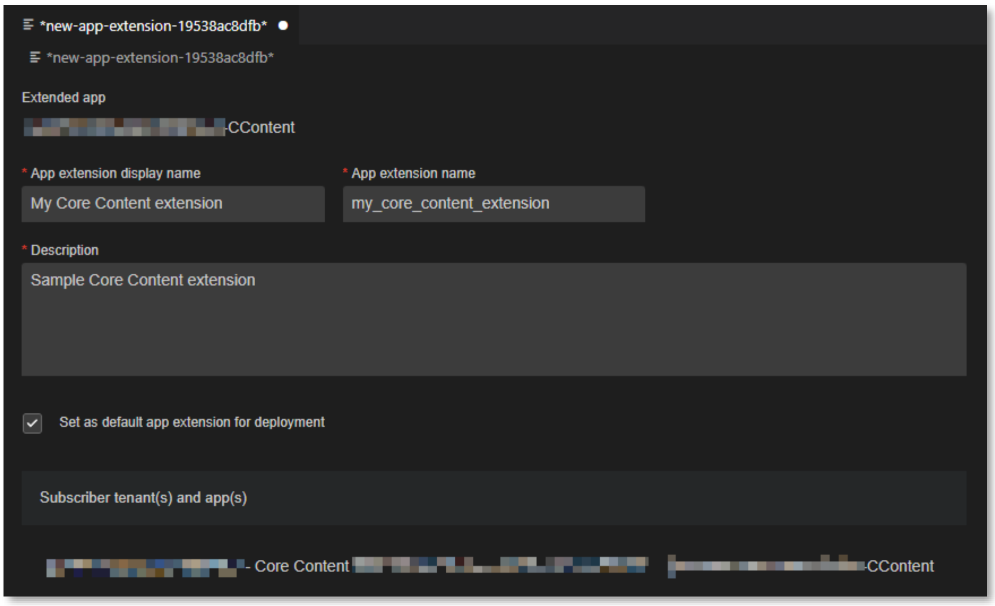
Next the list of application subscriptions available in the tenant is shown. Select the Core Content subscription you want to extend. Again, use the filter mechanism to quickly find the right subscription (see Fig. 6.13).

Fig. 6.13:


content
[redacted]-CContent - [redacted]
[redacted]-ccontent-rmfed - [redacted]

Next the new app extension configuration screen opens (see Fig. 6.14). The different fields are prepopulated based on the project properties, and the selected tenant is mentioned as subscribed tenant.

Fig. 6.14:



new-app-extension-19538ac8dfb ●

new-app-extension-19538ac8dfb

Extended app

[redacted]CContent

* App extension display name: My Core Content extension

* App extension name: my_core_content_extension

* Description: Sample Core Content extension

☑ Set as default app extension for deployment

Subscriber tenant(s) and app(s)

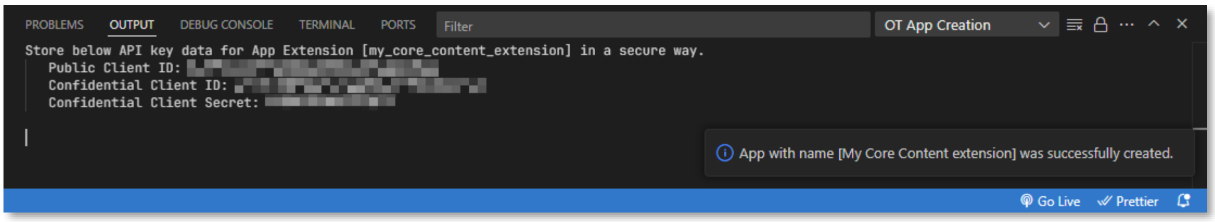
[redacted]- Core Content [redacted] [redacted]CContent

The different properties on the **New App Extension** configuration screen are:

- **App extension display name:** the display name of the app extension
- **App extension name:** the name of the app extension
- **Description:** the description of the app extension
- **Set as default app extension for deployment:** whether this app extension is used as the default app extension for deployment of the project
- **Subscriber tenant(s) and app(s):** list of (tenant) application subscriptions that are extended by this app extension.

Save the new app extension. Thrust Studio creates the app extension in Admin Center and displays the service client related to the new app extension (see Fig. 6.15). The **Output** view automatically opens and displays the **OT App Creation** output. It contains the application API key data. This API key together with the tenant id of the subscribed tenant needs to be kept securely in case you want to use this from your project code to authenticate/communicate with the OpenText Cloud Platform APIs in context of your Core Content application extension.

Fig. 6.15:



```
Store below API key data for App Extension [my_core_content_extension] in a secure way.
   Public Client ID: [redacted]
   Confidential Client ID: [redacted]
   Confidential Client Secret: [redacted]
```

App with name [My Core Content extension] was successfully created.

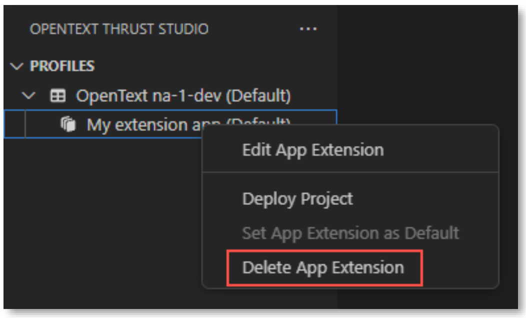
Note that only the app extension entry and service clients are created. The project and its models are not yet deployed.

6.2.2 Deleting an App extension

To delete an app extension from an organization profile use the **Delete App Extension** option (see Fig. 6.16) in the contextual menu of the specific app.

Note: this will not only delete the app extension from your local profile, but the app extension will also be deleted from Admin Center, and with that it will be deleted for everyone.

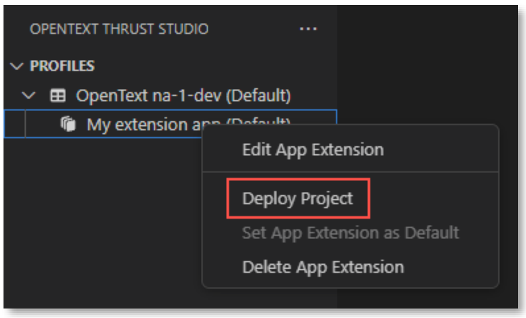
Fig. 6.16:



6.2.3 Deploying the project

Next step is to deploy your project to the app extension. This can be done via the **Deploy Project** (see Fig. 6.17) option in the app extension's contextual menu. Using this option will deploy the models as part of your project to the app extension that was created in the specific organization.

Fig. 6.17:



To see the detailed deployment steps open the ALM Output channel in the **Output** view. The view contains the progress and status of the deployment. In case of an error this view also shows more details about the error.

6.2.4 Deploying to a default or specific app extension

Deploying an extension project can be done via the **More Actions** menu (see Fig. 6.18) or the workspace (root) folder's contextual menu (see Fig. 6.19) in the **Explorer** view, or via the **OpenText: Deploy to Default Extension App** command in the **Command Palette** (see Fig. 6.20), or via the **More Actions** menu of the model explorer in the **OpenText Thrust Studio** view (see Fig 6.21).

Using the **Deploy to Default Extension App** option deploys your project to the default app in the default organization.

To deploy your project to another organization or app either mark that as default before triggering deployment or use the **Deploy to Extension App...** option. Using this option allows you to select a specific extension app, from all extension apps that are related to the project, to be used for deployment.

Fig. 6.18:

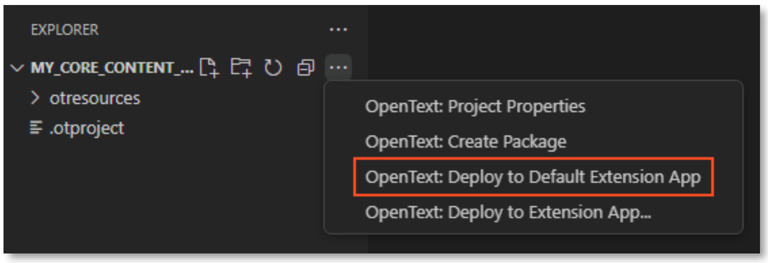

EXPLORER
MY_CORE_CONTENT_...
otresources
.otproject
OpenText: Project Properties
OpenText: Create Package
OpenText: Deploy to Default Extension App
OpenText: Deploy to Extension App...

Fig. 6.19:

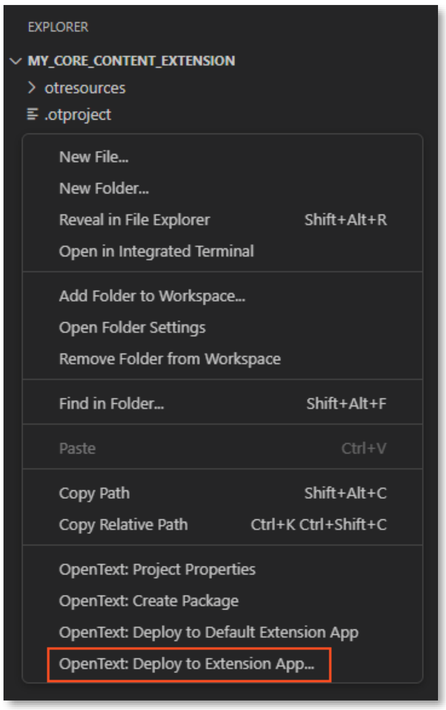

EXPLORER
MY_CORE_CONTENT_EXTENSION
otresources
.otproject
New File...
New Folder...
Reveal in File Explorer Shift+Alt+R
Open in Integrated Terminal
Add Folder to Workspace...
Open Folder Settings
Remove Folder from Workspace
Find in Folder... Shift+Alt+F
Paste Ctrl+V
Copy Path Shift+Alt+C
Copy Relative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

Fig. 6.20:

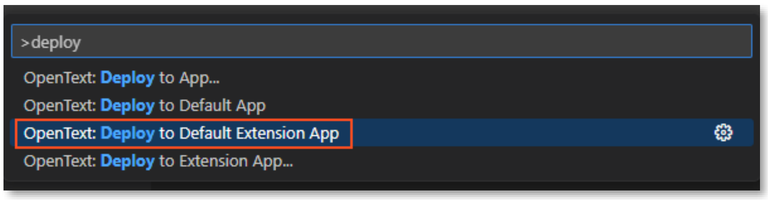
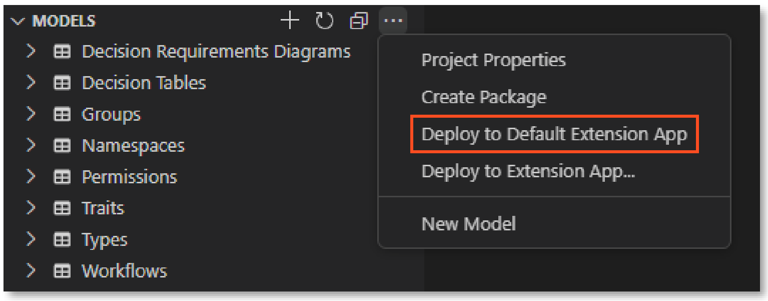

>deploy
OpenText: Deploy to App...
OpenText: Deploy to Default App
OpenText: Deploy to Default Extension App
OpenText: Deploy to Extension App...

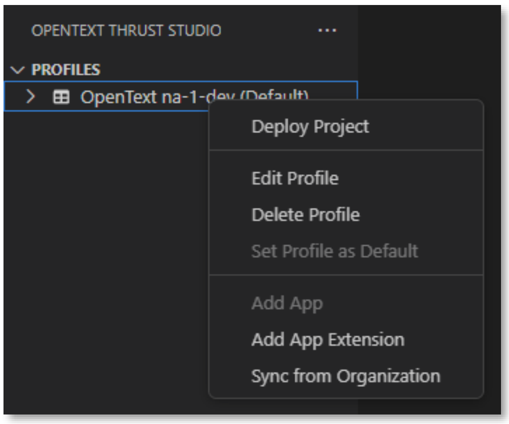
Fig. 6.21:



6.2.5 Synchronize app extensions for the project from Admin Center

When working in a team it can be that someone else created one or more app extensions for the project already. These app extensions are available in Admin Center and can be synchronized to your local Thrust Studio configuration. To synchronize app extensions from Admin Center use the **Sync from Organization** option (see Fig. 6.22). This option adds the app extensions that are available in Admin Center to the organization profile. Note that only those app extensions that are related to the open project are added.

Fig. 6.22:



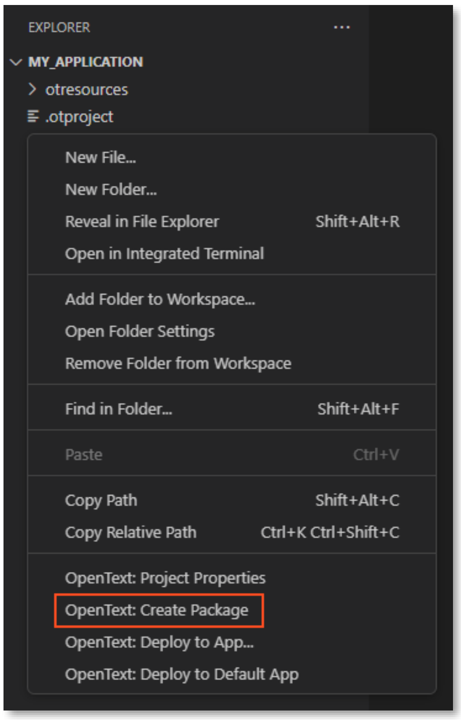
7 Creating a local package

This chapter describes how to create a local package of the project that can be stored and deployed later.

When deploying to the default tenant then a temporary package is created which then is uploaded to and deployed by the ALM deployment service. Sometimes a developer wants to store this package and deploy it to another environment. That is possible with the Create Package command.

Creating a local package can be done via the **More Actions** menu or the workspace (root) folder's contextual menu (see Fig. 7.1) in the **Explorer** view, or via the **OpenText: Create Packag**e command in the **Command Palette**, or via the **More Actions** menu of the model explorer in the **OpenText Thrust Studio** view.

Fig. 7.1:


EXPLORER
MY_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: Create Package
OpenText: Deploy to App...
OpenText: Deploy to Default App

When creating a package, the package name must be specified. The name is defaulted to the project name. The package file extension must be '.otpp'. On pressing 'Enter' the package will be created and stored in the root of the project folder.

Local packages can be deployed with ocp cli, see chapter **8.3 Command line deployments** below.

8 Using the ocp command line interface

This chapter describes the OpenText Cloud Platform command line interface, or in short ocp cli. This cli is the companion tool for OpenText™ Thrust Studio. The main purpose of this tool is to automate the deployment related tasks of application development for the OpenText Developer Cloud, but it can also be used to manage the organization profiles for deployments.

Below information is a summary of the functionality provided by the ocp cli. For a complete and extensive overview of all options and functionality see the NPM page of the cli at *https://www.npmjs.com/package/@opentext/ocp*

8.1 Installing the ocp cli

The OpenText Cloud Platform cli is available in the NPM nodejs package manager registry and can be installed with npm as follows

```
npm install -g @opentext/ocp
```

This makes the latest version of the ocp cli available to be used on the command line.

To get usage and help information use

```
ocp --help
```

To show the usage information for a specific action use

```
ocp <action> --help
```

8.2 Profile management

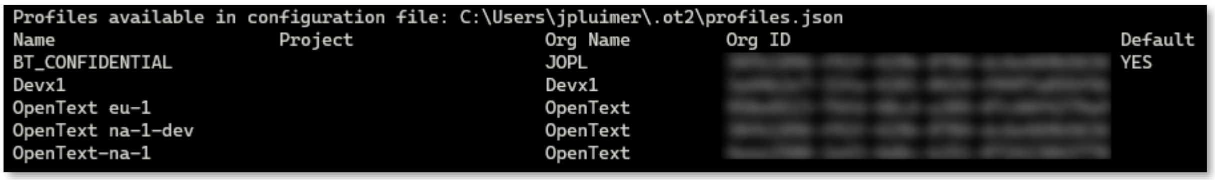
The ocp cli and OpenText™ Thrust Studio share the same profile configuration file. This means that both tools can be used to manage the developer profiles.

To list the configured profiles use

```
ocp list-profiles
```

This shows the profiles and some of their properties, like Name, Organization id and whether it is the default profile (see Fig. 8.1).

Fig. 8.1:



```
Profiles available in configuration file: C:\Users\jpluimer\.ot2\profiles.json
Name                    Project                 Org Name        Org ID                                  Default
BT_CONFIDENTIAL                                 JOPL            [redacted]                              YES
Devx1                                           Devx1
OpenText eu-1                                   OpenText
OpenText na-1-dev                               OpenText
OpenText-na-1                                   OpenText
```

The default organization profile is used by the cli in case no specific profile is specified when an action is requested.

To create a new organization profiles use

```
ocp add-profile
```

This action can be used to add a new organization profile. Use the `update-profile` action to update an existing profile.

To update an organization profiles use

```
ocp -profile
```

This action can be used to add a new organization profile or update an existing one.

To add a specific tenant to an organization profile use

```
ocp add-tenant
```

To validate that a profile is correct and can be used to authenticate the authenticate action can be used. To authenticate by using a specific organization profile use

```
ocp authenticate --profile QA
```

When the profile option is omitted then the default organization profile will be used to authenticate.

Two types of profiles are supported for authentication. There is the public client type and the confidential client type of profile.

Public client authentication

When using an organization profile with public client then the ocp cli and OpenText™ Thrust Studio trigger an authentication flow in the browser, requesting the developer to log in. This means that authenticating via a public client always needs developer involvement at some point, as they need to login at the identity provider. A profile is considered to use a public client in case only the client_id is configured and the client-secret is omitted.

Confidential client authentication

The other option of authentication is done by using the confidential client and secret. This option does not involve user interaction but is done completely standalone between the ocp cli and the OpenText Developer Cloud. Because this option does not need user involvement it is a good option for usage in a CI/CD pipeline, to run automated tests. A profile is considered to use a confidential client in case both the client_id and client_secret are configured for the profile.

For a full list of profile management examples see
https://www.npmjs.com/package/@opentext/ocp#profile-usage-examples

8.3 Command line deployments

OpenText™ Thrust Studio allows developers to deploy their projects to the OpenText Cloud Platform. The ocp cli can be used to deploy these same projects from the command line.

To deploy the project available in the current folder, via the default organization profile, to the default configured tenant, use

```
ocp deploy
```

The folder where the cli is executed must contain the .otproject file. The workspace option can be used to specify another folder for containing the application project to be deployed

```
ocp deploy --workspace ./my_projects/contract_approval
```

While deploying the application project the deployment service will create an application definition, if that doesn't exist yet, in the Developer Console. With that application definition also API key data, app credentials, are created and returned to the client. This API key data is displayed as output.
To save these app credentials to a file use

```
ocp deploy --app_credentials_output <file path>
```

This will add the following properties to the given file:

- `APPLICATION_ID`=<Unique application ID>
- `PUBLIC_CLIENT_ID`=<application public client secret>
- `CLIENT_ID`=< application confidential client ID>
- `CLIENT_SECRET`=< application confidential client secret>

Note, the app credentials only are created when the application definition does not exist in Developer Console.

The cli will first create a temporary package of the project and then upload this package to the OpenText Developer Cloud. It is possible to not upload the package directly but save it to a local file instead, such that it can be stored in a repository and made available for later use. To create a local package, use the target option

```
ocp deploy --target contract_approval.ot2p
```

This generates a deployable package without deploying. The package is saved in the current folder and always has an .ot2p file extension.

To deploy an earlier created package, use the source option

```
ocp deploy --source contract_approval.ot2p
```

8.4 Considerations when using ocp in a CI/CD pipeline

One part of application development is running automated tests from a CI/CD pipeline, in for example Jenkins or Gitlab. This allows the developer to automatically run a predefined set of tests to validate that the changes done to the application don't break functionality.

When developing an application that uses one or more OpenText Cloud Platform services, before running the actual tests, the application model definitions as part of the application must be deployed into the tenant against which the automated tests are run. This normally is one of the steps in a CI/CD pipeline.
A CI/CD pipeline typically runs without user interaction. This means that the pipeline step that deploys the application via the ocp cli best uses the confidential client approach.

There are a couple of different ways to pass the profile information to the ocp cli:

- via the .ot2/profiles.json file in the user's home directory,
- via a specifically specified configuration file in the config_file option,
- via a set of options directly passed when calling the ocp cli,
- via environment variables.

Depending on your needs, use any of the above options in your ocp deploy pipeline step.

Example: passing a local configuration file, instead of using the default ~/.ot2/profiles.json file and using the workspace option to specify the application project location

```
 ocp deploy --config_file ./projects/profiles/qa_profiles.json
--workspace ./projects/contract_approval --app_credentials_output
./ca_app_creds.properties
```

Example: where no predefined profile is used but all information is passed directly from the command line

```
 ocp deploy --org_id 6dd45f25-e88b-453b-9eab-a1eb157da9a1 --client_id
Z7JFAKE363a67joplga56g8IhaDrX1 --client_secret fk195kf94lkdd9 --region na-
1-dev --app_credentials_output ./ca_app_creds.properties --tenant_id
a52460e6-ce3a-47c1-9bbc-6bb595a7adaa
```

See https://www.npmjs.com/package/@opentext/ocp#environment-variables for a list of available environment variables that can be used.

9 Contact information

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

For more information, visit www.opentext.com