

OpenText™ Thrust Services Tutorial

Building a Contract Approval application

This tutorial has been created for software version OpenText™ Thrust Studio 24.4.1.

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.

Note that if you are using this tutorial with a later version of the OpenText™ Thrust Studio VS Code extension pack, the screenshots and usage might not always correspond.

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 <http://www.opentext.com>

Copyright © 2025 Open Text. All Rights Reserved.

Trademarks owned by Open Text.

One or more patents may cover this product. For more information, please visit <https://www.opentext.com/patents>

Disclaimer

No Warranties and Limitation of Liability

Every effort has been made to ensure the accuracy of the features and techniques presented in this publication. However, Open Text Corporation and its affiliates accept no responsibility and offer no warranty whether expressed or implied, for the accuracy of this publication. This includes the application code that is being provided with this tutorial, as this code is intended for educational purposes only and should not be used in a production setting.

Last updated: 01/21/2025

Contents

Introduction	5
1 [5'] Get started.....	8
1.1 Audience	8
1.2 Prerequisites	8
2 [20'] Set up the OpenText Thrust Studio IDE	9
2.1 Download and install VS Code	9
2.2 Add the OpenText Thrust Studio extension pack to VS Code	13
2.3 Install the LTS version of Node.js	16
2.4 Verify the Node.js installation from VS Code.....	17
3 [10'] Set up a connection to the developer organization	18
3.1 Add an organization profile	18
3.2 Test the connection	22
3.3 Add a tenant to the organization profile.....	24
4 [10'] Create an OpenText project	26
4.1 Create a file system folder for the Contract Approval application project	26
4.2 Set up an OpenText project for the Contract Approval application	28
4.3 Install Java	30
5 [5'] Create a namespace.....	31
5.1 Create the Contract Approval namespace	31
6 [10'] Create a trait.....	35
6.1 Create the Approval trait.....	35
7 [25'] Create types	40
7.1 Create the Contract type.....	41
7.2 Create the Loan Contract type	46
7.3 Create the Customer type.....	48
8 [10'] Create (user) groups	50
8.1 Create the administrators group	50
8.2 Create the line_managers group	52
8.3 Create the risk_managers group	54
8.4 Create the contract_approval_users group	56
9 [15'] Create ACLs	58
9.1 Create the created ACL	59
9.2 Create the pending_approval ACL	61
9.3 Create the completed ACL	63
10 [25'] Create a decision table	65
10.1 Create the Required Approvals decision table	66
11 [140'] Create workflows	74
11.1 Create the Solvency Check workflow	75

11.2 Create the Manager Approval workflow	90
11.3 Create the Contract Approval workflow	107
12 [25'] Deploy an application to the OpenText Thrust Services	169
12.1 Deploy the Contract Approval application and get the application credentials	169
12.2 Verify that the application is deployed	170
12.3 Add the redirect URL for the application authentication flow.....	172
12.4 Add the application users to the application groups.....	175
13 [25'] Work with the OpenText Thrust Services APIs	184
13.1 Download and install Postman	184
13.2 Download the Contract Approval application.....	185
13.3 Import the Postman collection and environment into Postman	187
13.4 Verify the deployment of the application models using the OpenText Thrust Services APIs 191	
14 [30'] Build the Contract Approval application	200
14.1 Import the Contract Approval App code into your project.....	201
14.2 Understand the main logic of the Contract Approval application	203
14.3 Authenticate and get authorized with the OpenText Thrust Services APIs.....	205
14.4 Set the environment variables	209
14.5 Use the content (CSS and CMS) APIs	210
14.6 Use the Workflow Service API.....	215
14.7 Use the Viewer Service API.....	217
14.8 Use the Risk Guard Service API	220
15 [60'] Test the Contract Approval application	222
15.1 Start the Contract Approval application and sign in	223
15.2 Approve a standard contract that only requires automatic approval (no additional required approvals)	229
15.3 Approve a loan contract that requires all additional approvals.....	242
15.4 Reject a manual contract approval task	257
15.5 Expire a manual contract approval task	262
16 [15'] Bonus exercise: Use the ocp command line interface.....	267
16.1 Install the ocp cli	268
16.2 Use the developer profile.....	269
16.3 Deploy the application project from command line.....	270
About OpenText	273

Introduction

This tutorial is intended as a comprehensive end-to-end course for pro-code developers. Throughout the different sections you will learn how to:

- Build an application that utilizes the OpenText Thrust Services APIs.
- Develop a React based application in OpenText Thrust Studio, the bespoke Integrated Development Environment (IDE) based on Microsoft Visual Studio Code (VS Code) to use when developing on OpenText Thrust Services.

The application you will be building is a simple Contract Approval application utilizing the OpenText Thrust Services. This application will allow you to do the following:

- Upload documents
- Store document related metadata for two types of contracts
- View documents
- Perform document analysis to detect PII (Personally Identifiable Information)
- Use a decision table to determine the required approvals
- Use a workflow to automate the different contract approval steps

The following OpenText Thrust Services APIs will be used to build the Contract Approval application:

Service	API	Purpose
Content Storage	Content Storage Service (CSS)	Uploading and storing of documents
Content Metadata	Content Metadata Service (CMS)	Defining and storing of document metadata
Viewing & Transformation	Viewer Service	Viewing documents
Risk Guard	Risk Guard Service	Document analysis
Decision	Decision Service	Defining and executing business rules (decision tables)
Workflow	Workflow Service	Defining, executing, and auditing business processes (workflows)

In this tutorial, the following exercise modules detail the different steps for building the Contract Approval application:

1. Get started
5 min
2. Set up the OpenText Thrust Studio IDE
20 min
3. Set up a connection to the developer organization
10 min
4. Create an OpenText project
10 min
5. Create a namespace
5 min
6. Create a trait
10 min
7. Create types
25 min
8. Create (user) groups
10 min
9. Create ACLs
15 min
10. Create a decision table
25 min
11. Create workflows
140 min
12. Deploy an application to the OpenText Thrust Services
25 min
13. Work with the OpenText Thrust Services APIs
25 min
14. Build the Contract Approval application
30 min
15. Test the Contract Approval application
60 min
16. Bonus exercise: Use the ocp command line interface
15 min

You can directly open the finished application in VS Code without going through all the exercise modules and familiarize yourself with the OpenText Thrust Services API example code, different models, and the Contract Approval sample application. The project sources for the completed Contract Approval application are available for download. To directly run the completed project from VS Code, see [Test the application](#).

1 [5'] Get started

The following software environment is used:

Software	Version
Operating System	Windows 11 Enterprise 22H2, 64-bit
Microsoft Visual Studio Code	1.96.2
Node.js	22.13.0 LTS
Java	21.0.5
OpenText™ Thrust Studio	24.4.1
Postman	11.27.3
Google Chrome	132.0.6834.84 (64-bit)



Note

The procedure steps, images, and the other references used in this tutorial are based on the above environment. If you are using any other environment or software versions, refer to the software documentation for that version.

1.1 Audience

This tutorial is intended for pro-code developers who want to learn how to build applications that utilize the OpenText Thrust Services APIs.

1.2 Prerequisites

Make sure you can sign in to the developer.opentext.com website and have an active trial or paid developer plan.

If you do not have an active plan, navigate to developer.opentext.com/plans to sign up for one.

For a detailed description on how to create a developer account and sign up for a developer trial, refer to: developer.opentext.com/imsservices/trial.

2 [20'] Set up the OpenText Thrust Studio IDE

Learn how to:

- Download and install VS Code
- Add the OpenText Thrust Studio extension pack to VS Code
- Install the LTS version of Node.js

2.1 Download and install VS Code

1. Go to <https://code.visualstudio.com/download> and download the Microsoft VS Code distribution that matches your system.



Note

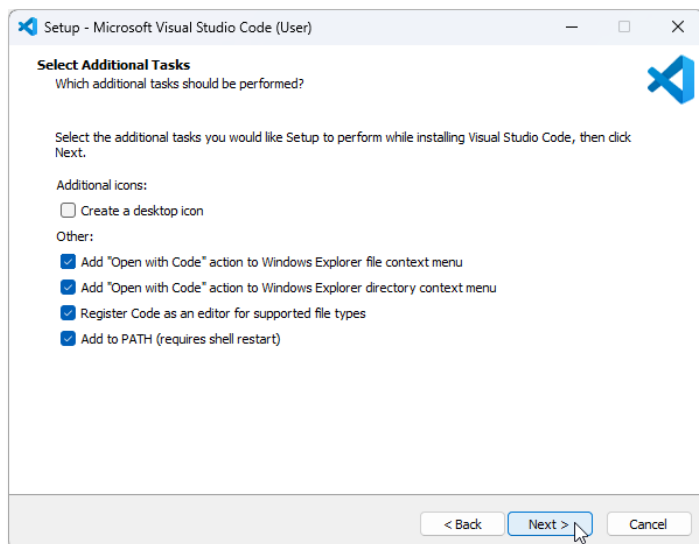
To install VS Code on a 64-bit Windows 11 system, choose to download the x64 User Installer for Windows and install the latest version of VS Code.



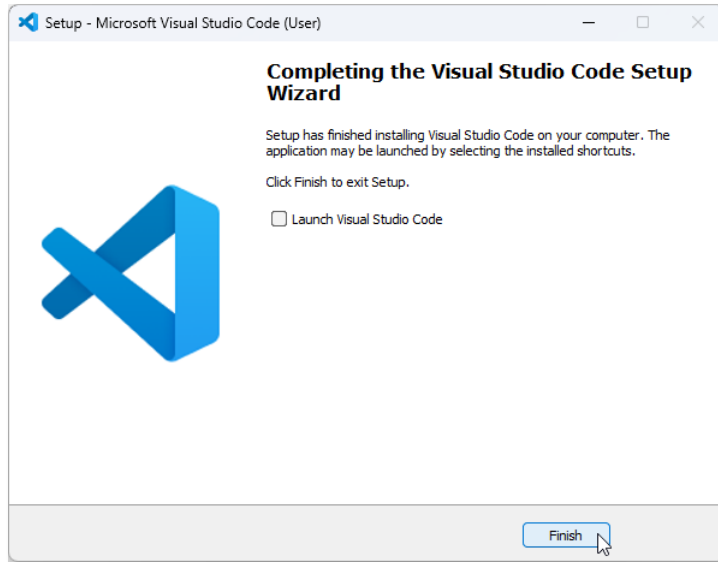
Important

OpenText Thrust Studio is tested with Windows and Mac Operating Systems. You can use Linux Operating System but if you run into problems, OpenText might not be able to provide a solution.

2. Save and run the installer.
3. Select **I accept the agreement** and click **Next** to continue.
4. Select the installation destination location and click **Next**. You can use the suggested default location.
5. Keep the default setting for the selecting of the start menu folder and click **Next**.
6. Select all the additional tasks under **Other** and optionally select **Create a desktop icon**. Click **Next** to continue.



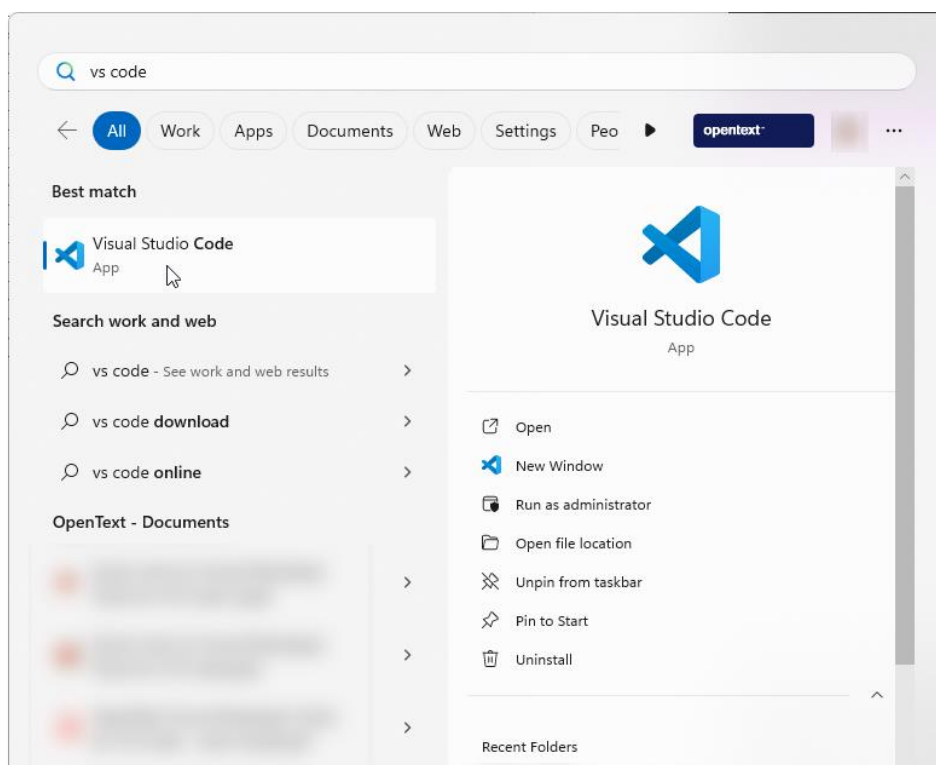
7. Verify your choices and click **Install** to start the VS Code installation.
8. After the installation is complete, uncheck “Launch Visual Studio Code” and click **Finish** to close the VS Code Setup Wizard.



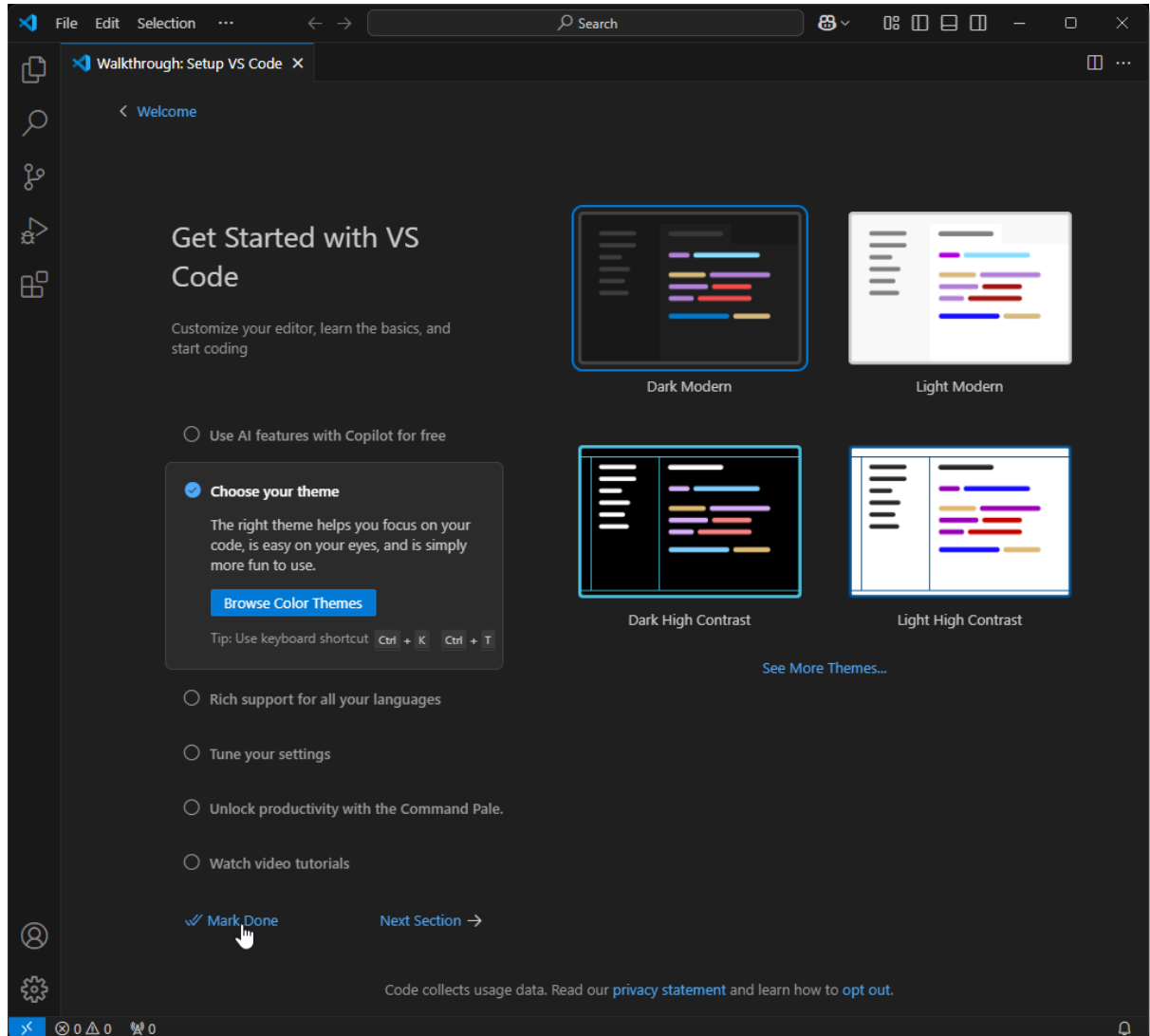
9. In the Windows **Start** menu, type `vs code` in the search box and select the **Visual Studio Code** application.

**Tip**

You can Pin to taskbar for easy access from the task bar.



10. The **Walkthrough: Setup VS Code** wizard is displayed when VS Code is launched for the first time.
11. Select a theme and click **Mark Done** to confirm your choice.

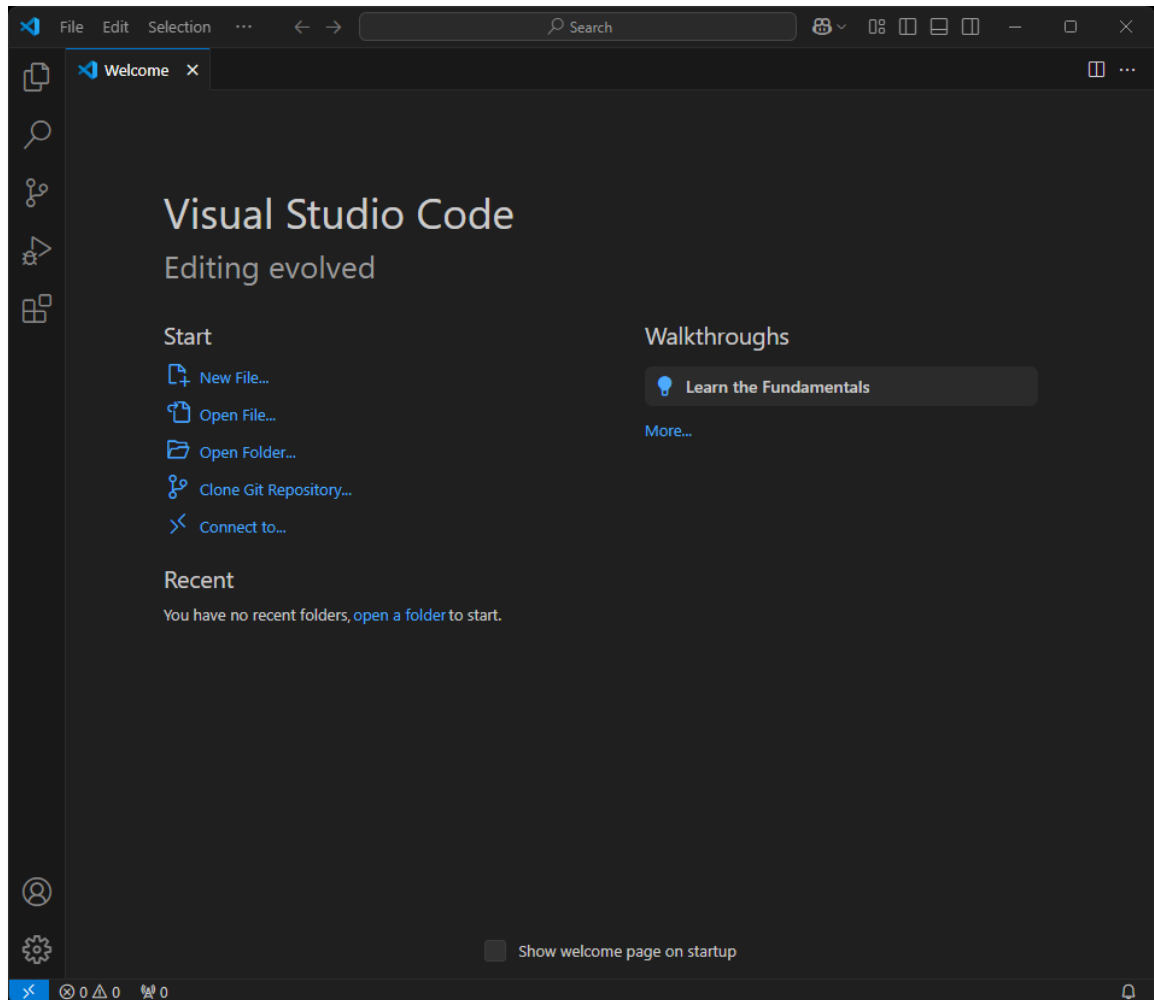


Note

For this tutorial the **Dark Modern** theme is selected. Take this into account when comparing your VS Code appearance with the VS Code screen shots throughout this document.

12. The standard VS Code welcome page is now displayed.

13. Make sure **Show welcome page on startup** is not selected so that the page does not appear when VS Code is launched every time.



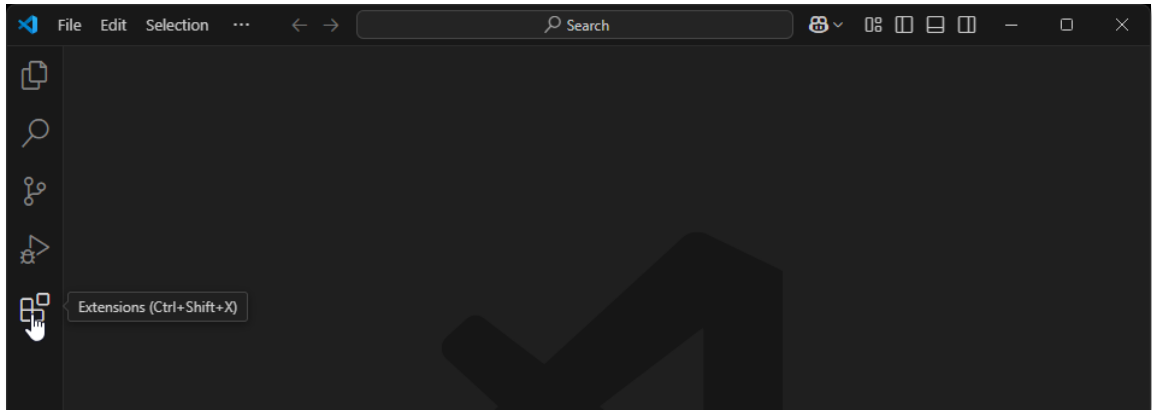
14. Close the welcome page.

Next step:

Add the OpenText Thrust Studio extension pack to VS Code.

2.2 Add the OpenText Thrust Studio extension pack to VS Code

1. On the VS Code Activity Bar, click **Extensions**.

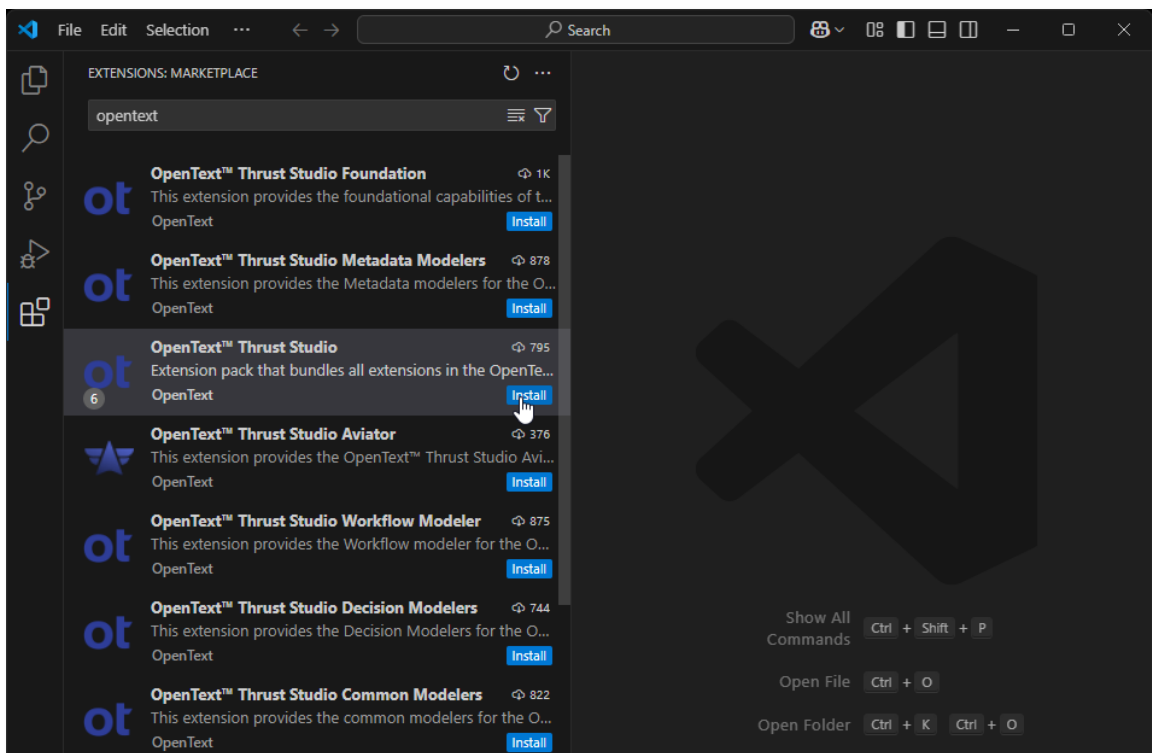


2. On the **Search Extensions in Marketplace** search bar, type `opentext` and choose to install the **OpenText™ Thrust Studio** extension pack.

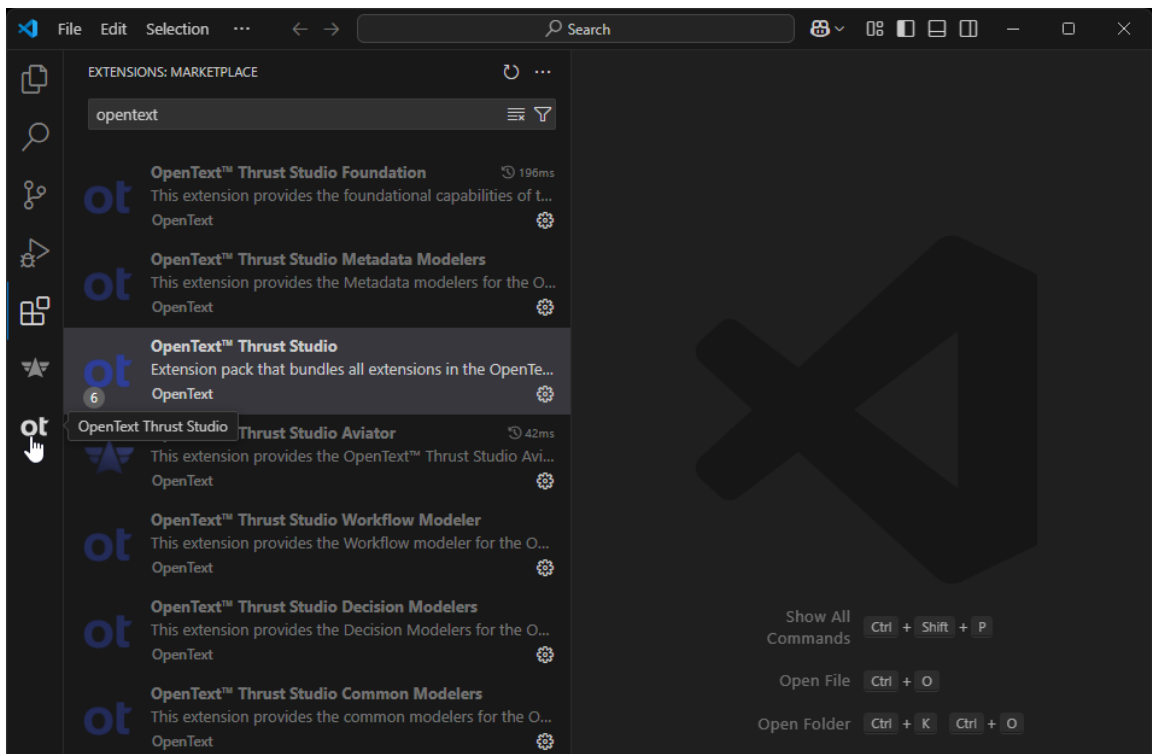


Note

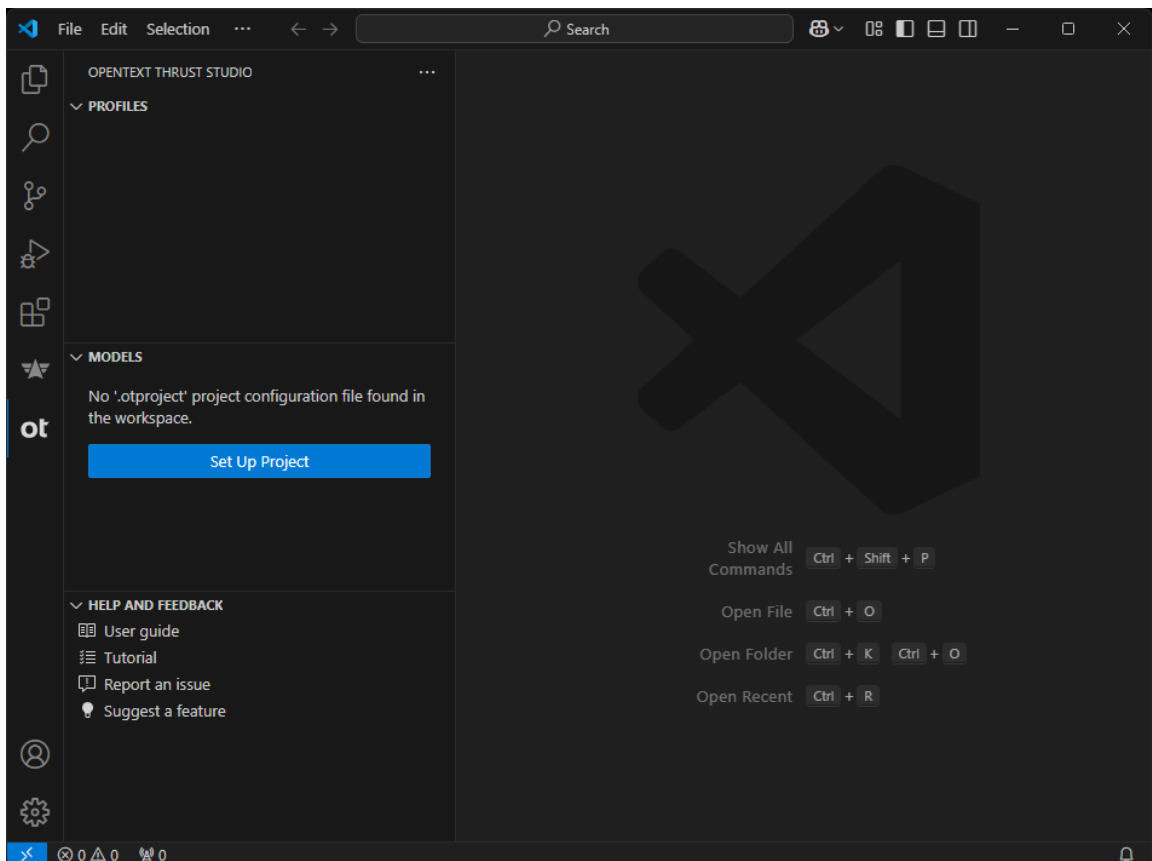
If prompted, click Reload to ensure that the installed VS Code extension pack is enabled.



3. On the **Activity Bar**, click the **OpenText Thrust Studio** button .



4. This opens the **OpenText Thrust Studio** view.



Next step:

Install the latest Long-Term Support (LTS) version of Node.js to support building and running the Contract Approval application.

2.3 Install the LTS version of Node.js

1. Go to <https://nodejs.org/en/download/>.
2. Locate the Node.js LTS version that corresponds with your Operating System and download the installer.

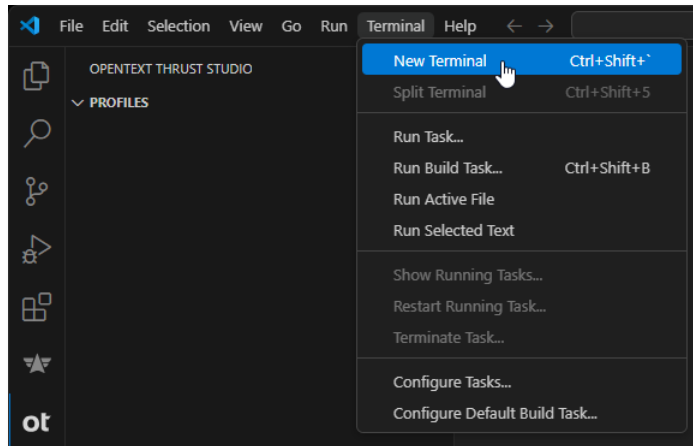


3. Run the installer. The **Node.js Setup Wizard** is displayed.
4. Click **Next**.
5. Select **I accept the terms in the License Agreement** and click **Next** to continue.
6. Select the installation destination location and click **Next**. You can use the suggested default location.
7. From the **Custom Setup** screen, you can accept the default and click **Next**.
8. From the **Tools for Native Modules** screen, you can accept the default and click **Next**.
9. Click **Install** to start the installation.
10. After the setup is complete, click **Finish** to close the **Node.js Setup Wizard**.

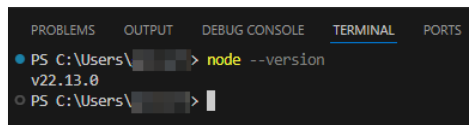
2.4 Verify the Node.js installation from VS Code

After Node.js is installed, verify that it is ready to be used when running Node.js code from VS Code.

1. Make sure VS Code is not still open (as you need to initialize it again) and open it.
2. In VS Code, select **Terminal > New Terminal**.



3. In the TERMINAL, type `node --version`. The installed version of Node.js is displayed (for example, 22.13.0).



Next exercise module:


Set up a connection to the developer organization.

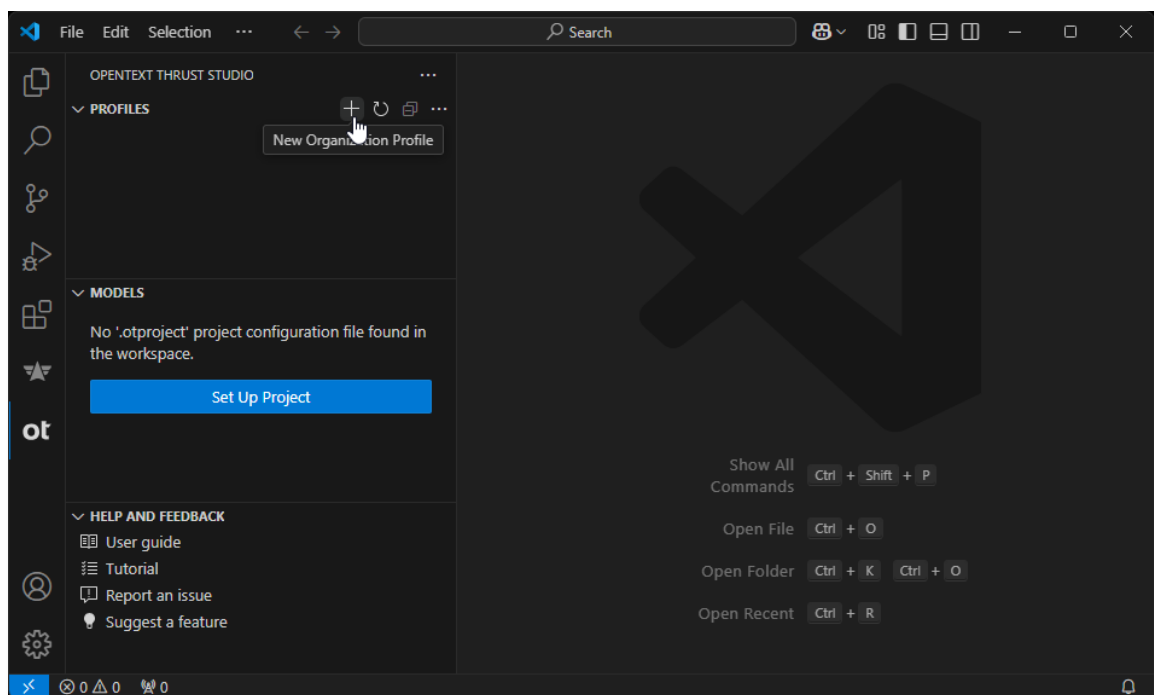
3 [10'] Set up a connection to the developer organization

Learn how to:

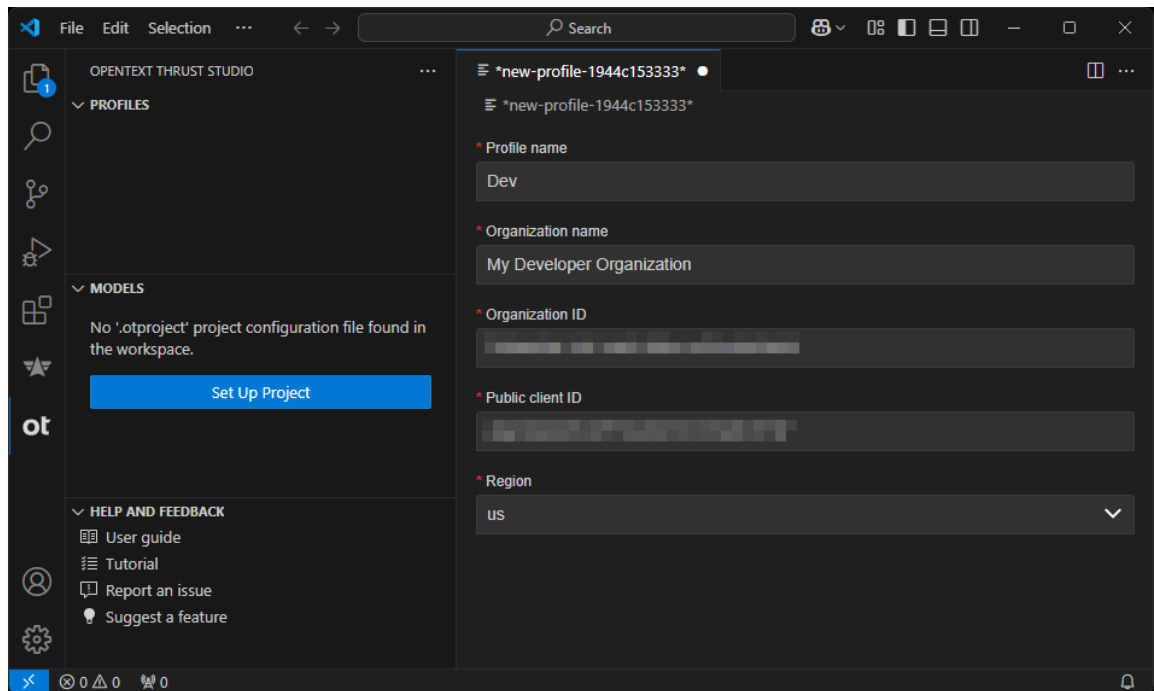
- Add an organization profile in VS Code to allow connecting to your developer organization
- Test the connection
- Add a tenant to the organization profile to allow deploying applications to that tenant

3.1 Add an organization profile

1. Open VS Code and on the Activity Bar, select the **OpenText Thrust Studio** view.
2. In the **PROFILES** section, click the **New Organization Profile** button  to create a new organization profile.



3. In the organization profile screen, fill the property fields:



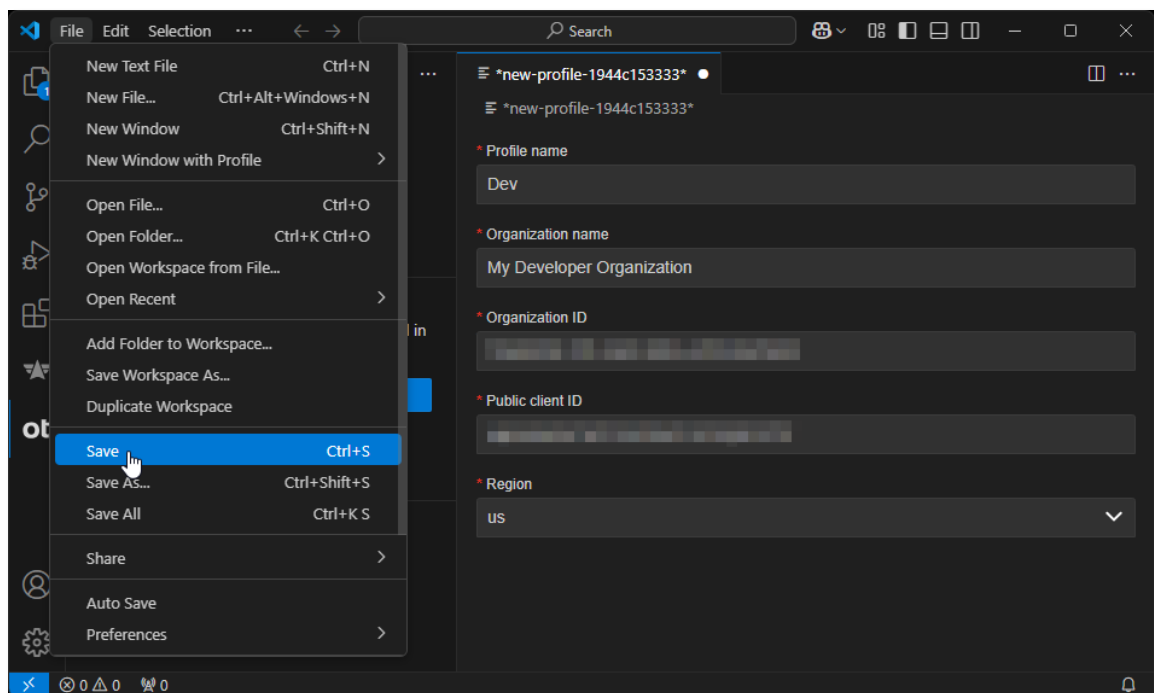
The following table describes the organization profile property fields:

Property	Description
Profile name	The profile name is used as a display name in the PROFILES section.
Organization name	<p>The name of the organization. It is recommended to use the value from the organization overview page in Admin Center.</p> <p>For more information on accessing the organization overview page in Admin Center, see Application Administration.</p>
Organization ID	<p>The unique identifier for the organization.</p> <p>For more information about how to view/copy your organization ID, see Application Administration.</p>
Public client ID	<p>The unique identifier of the public OAuth service client for the organization. Make sure to use the client id of the public service client.</p> <p>For more information about service clients and how to view/copy a client ID, see Application Administration.</p>
Region	<p>The region where the organization is made available. For this tutorial we select the us region, but if your developer subscription (trial or other) is in a different region, you must select that region.</p> <p>For more information about regions and how to see what is your region from Admin Center, see Application Administration.</p>

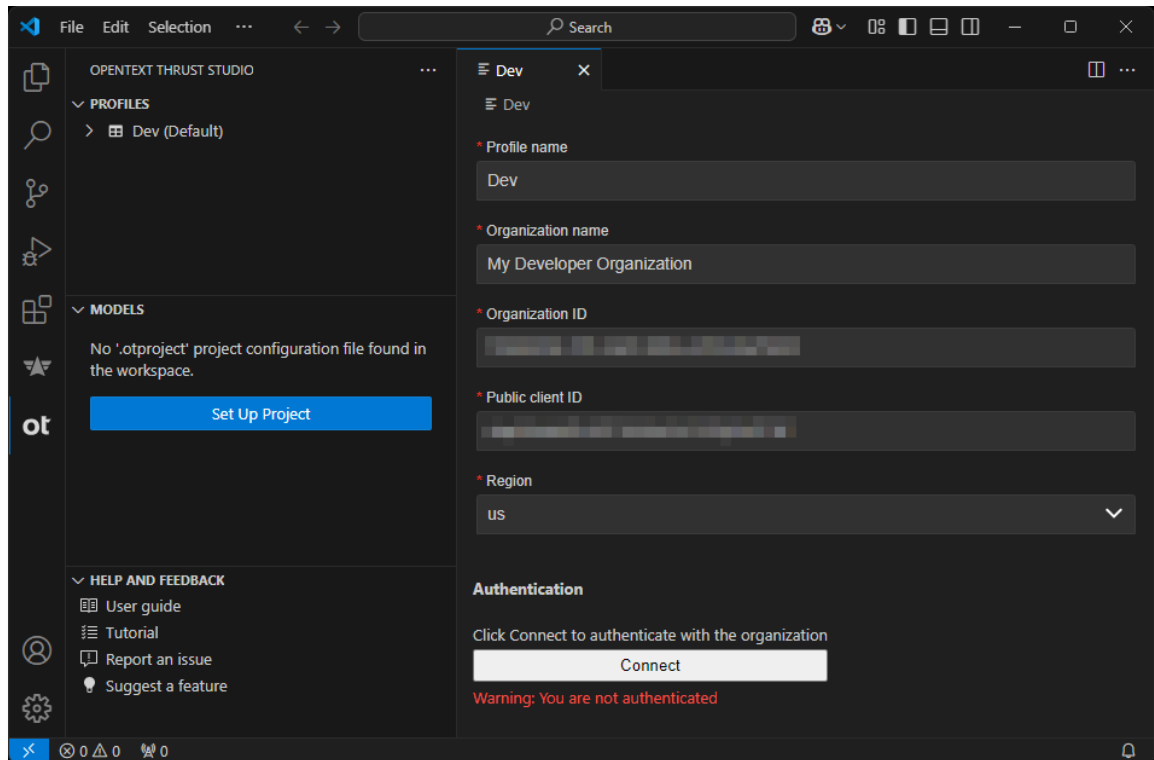
4. Select **File > Save** to save the organization profile.

**Note**

OpenText Thrust Studio configuration artifacts such as organization connection set up, project set up, and different model configurations use the standard VS Code file saving functionality. To save your changes to any configuration artifact, press **Ctrl+S** (for Windows systems) or select **File > Save**.



- After saving, the organization profile is listed as the default profile in the **PROFILES** section and the tab above the organization profile is renamed to the profile name. On the organization profile screen an **Authentication** section with the **Connect** button appears.

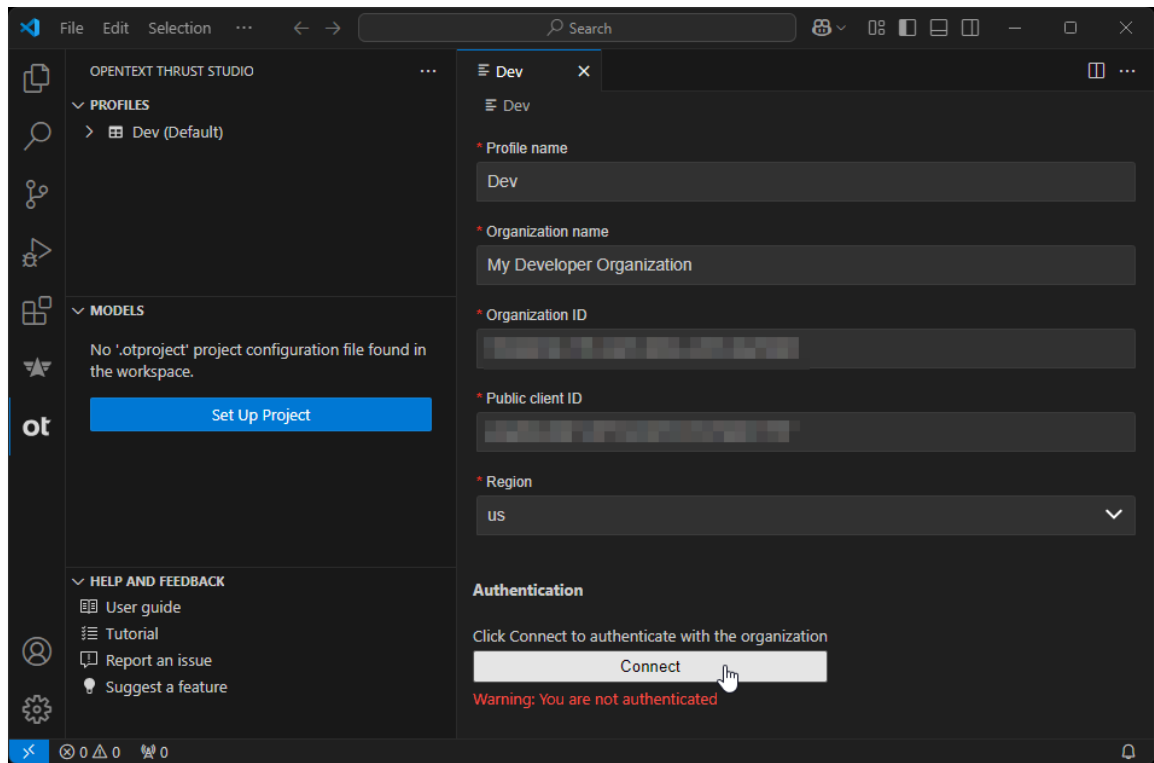


Next step:

Test the connection.

3.2 Test the connection

1. Click **Connect** to test the newly configured connection to your developer organization.

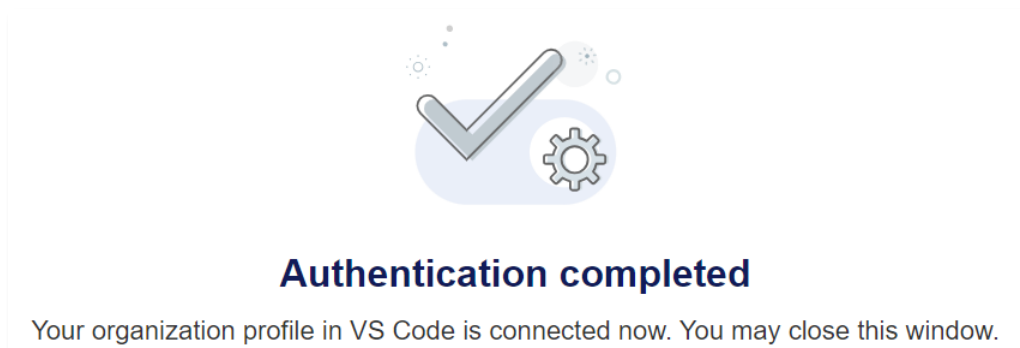


2. On the sign in page, authenticate with your developer.opentext.com username and password.

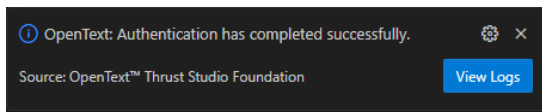


Note

If you are already signed in, the browser will immediately show the **Authentication completed** page.



3. After successful authentication, a confirmation message is displayed in VS Code.



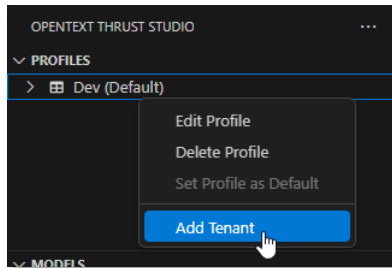
4. Close the organization profile screen.

Next step:

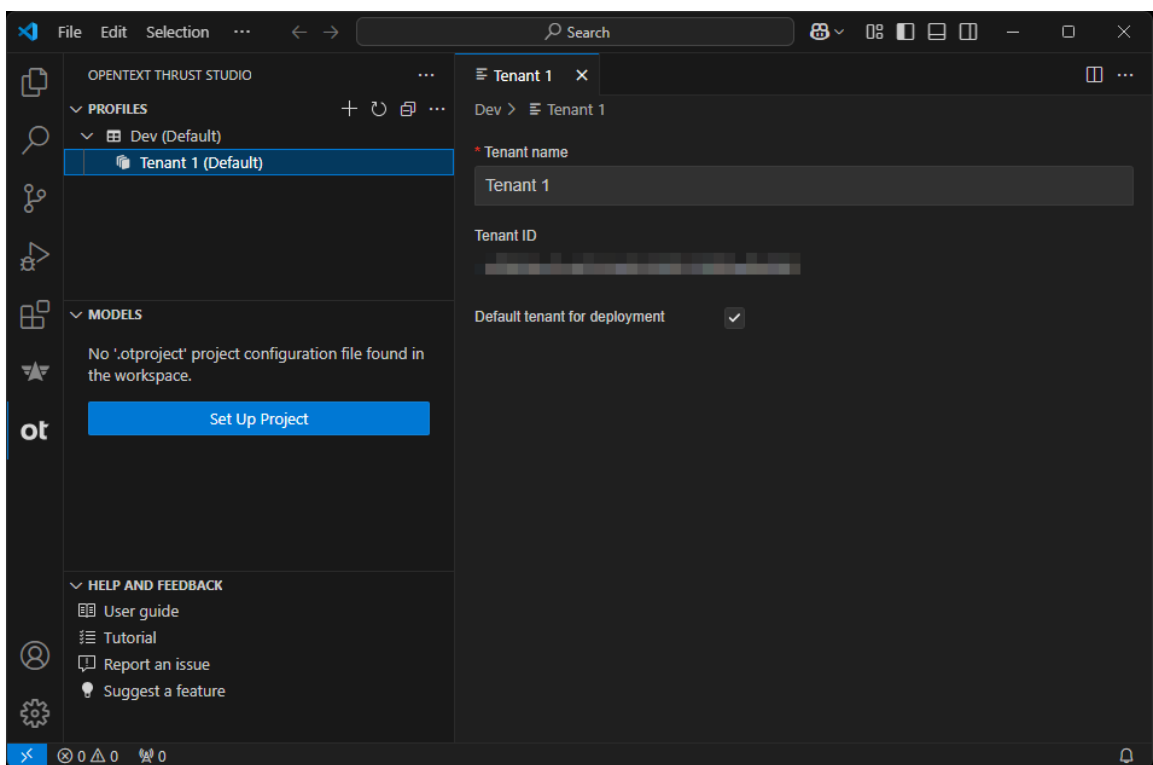
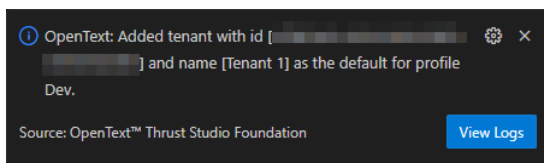
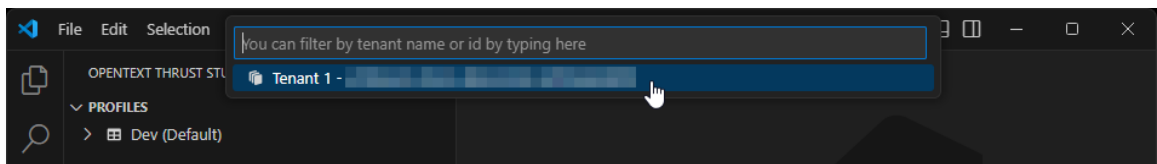
Add a tenant to the organization profile.

3.3 Add a tenant to the organization profile

1. In the **PROFILES** section, right-click the profile and select **Add Tenant**.



2. Choose the correct tenant from the tenant selection dropdown (for developer trials there is only one with **Tenant 1** as the default name, but for other developer subscriptions there can be multiple).



The following table describes the tenant property fields:

Property	Description
Tenant name	The name of the tenant. It is recommended to use the proposed value as it matches the tenant name from the tenant page in Admin Center. For more information on accessing the tenant page in Admin Center, see Application Administration .
Tenant ID	The unique identifier for the tenant.
Default tenant for deployment	Whether the tenant is the default tenant for deployment. That is, if upon deployment the user does not explicitly choose to deploy to a specific tenant in a specific organization, the default tenant of the default organization will be used as deployment target.

**Note**

The first tenant added to the organization profile is made the default tenant for deployment.

Next exercise module:

Create an OpenText project.

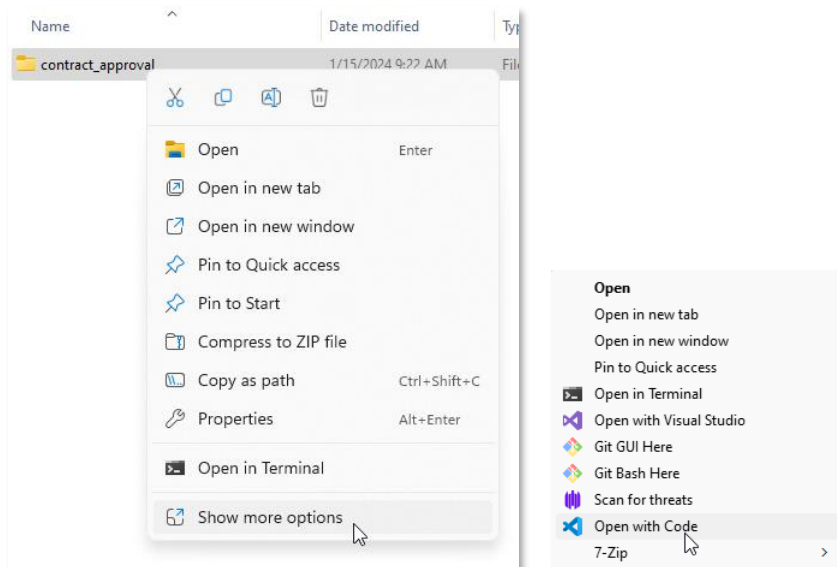
4 [10'] Create an OpenText project

Learn how to:

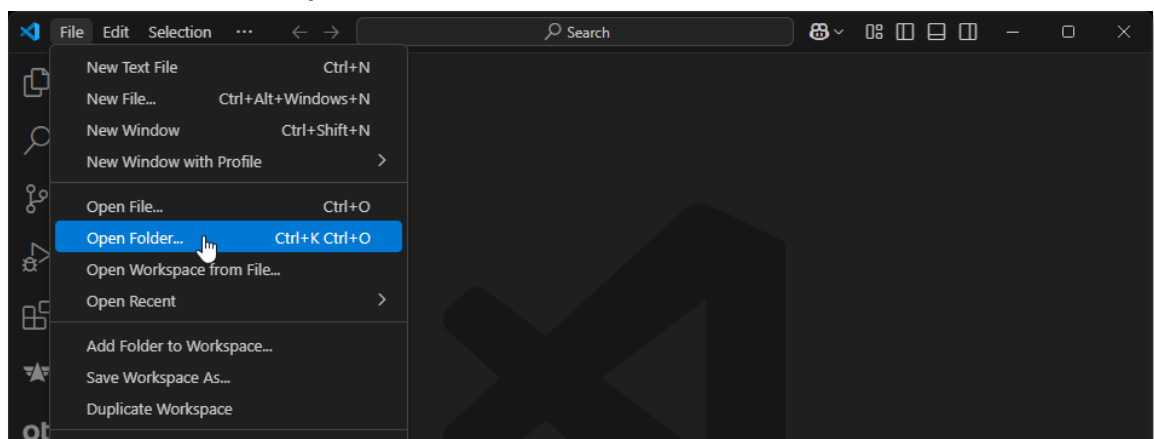
- Create a folder in your file system for building the Contract Approval application
- Set up an OpenText project for the Contract Approval application models

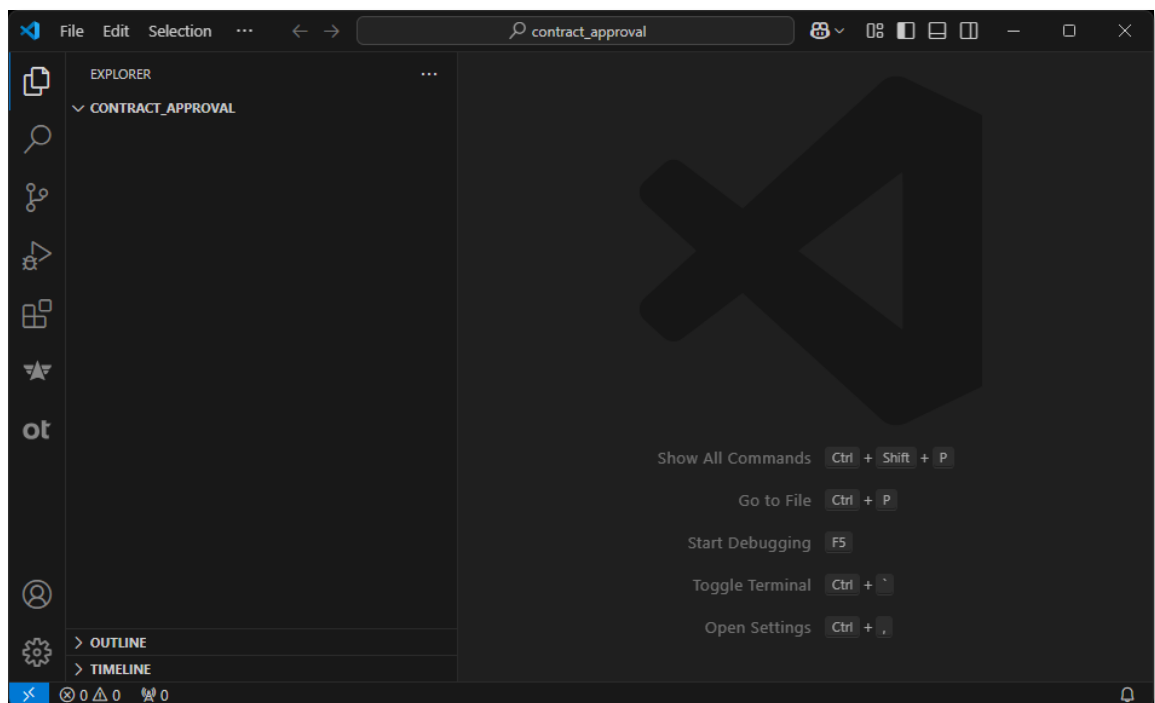
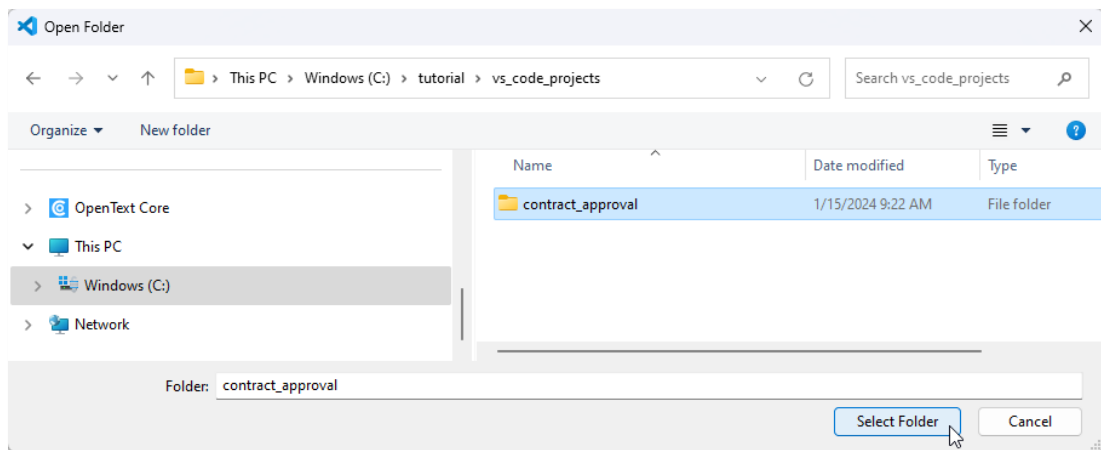
4.1 Create a file system folder for the Contract Approval application project

1. In your system's file explorer (for example, Windows File Explorer) create a new folder. Name the folder **contract_approval**.
2. Do any one of the following to open the Contract Approval application project folder in VS Code:
 - In the Windows File Explorer, right-click the newly created **contract_approval** folder and select **Show more options** and then click **Open with Code**.



- In VS Code, click **File > Open folder** and select the folder.



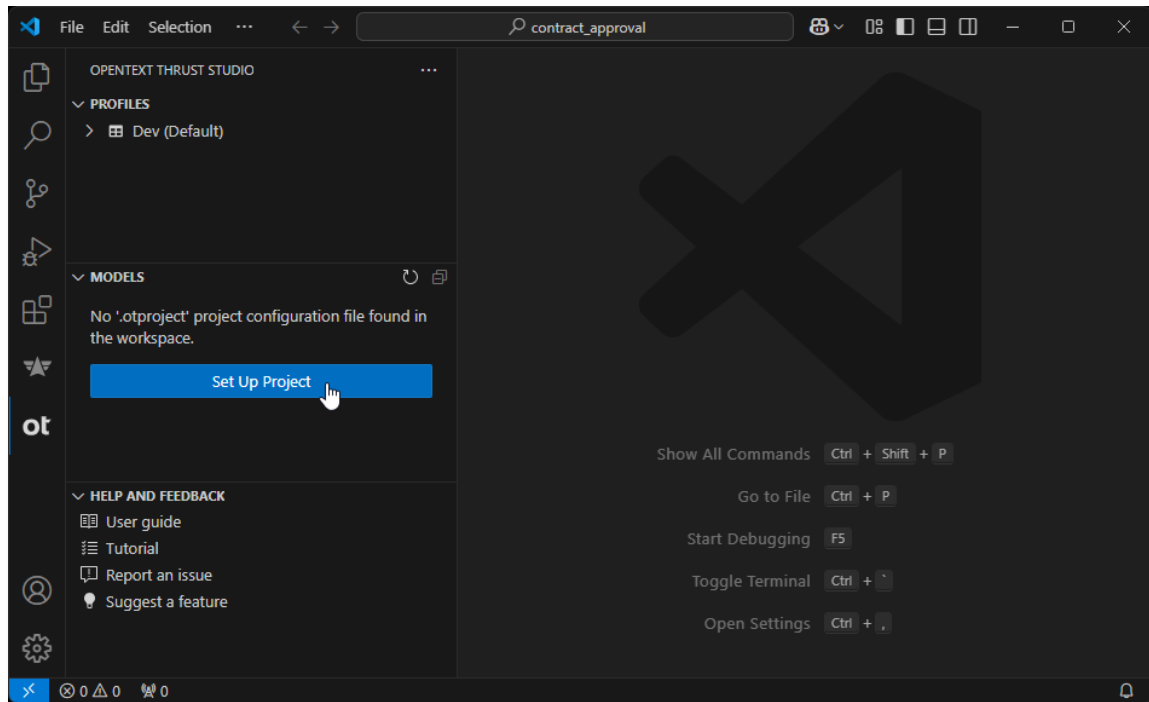


Next step:

Set up the OpenText project to allow building the Contract Approval application models.

4.2 Set up an OpenText project for the Contract Approval application

1. Go to the OpenText Thrust Studio view and click **Set Up Project**.

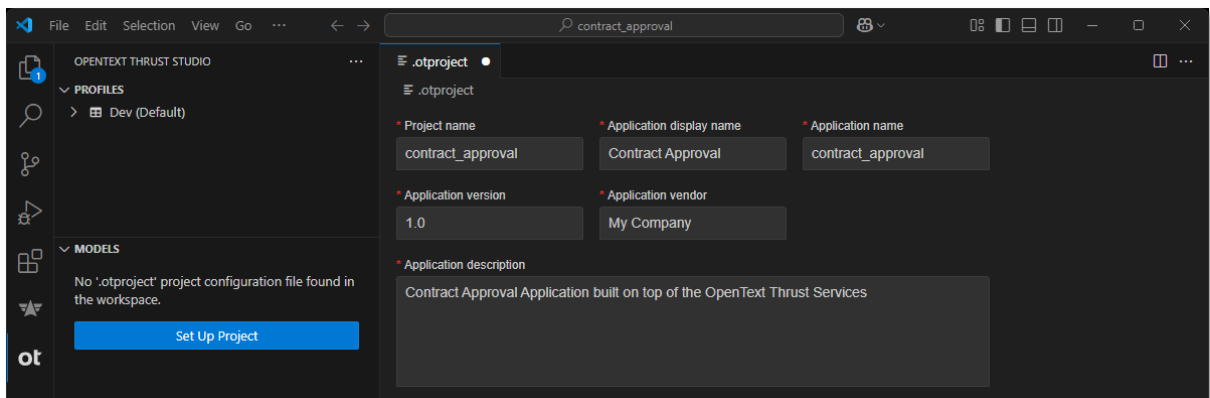


2. Fill the OpenText project properties:

Field	Value
Project name	The project name has been automatically populated from the project folder name and does not need to be changed.
Application display name	Contract Approval
Application name	The system automatically populates the application name from the display name. Leave the application name to the generated <code>contract_approval</code> value.
Application version	1.0
Application vendor	Name of the organization that owns the application. In this example, My Company is used.
Application description	Contract Approval Application built on top of the OpenText Thrust Services

**Note**

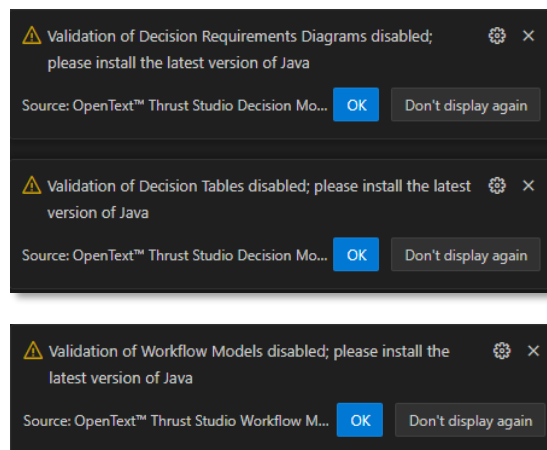
You can always view and modify the OpenText project properties from the VS Code Explorer view by choosing **OpenText: Project Properties** from the contextual menu of your project (root) folder or any of its subfolders, or by clicking/opening the **.otproject** file.



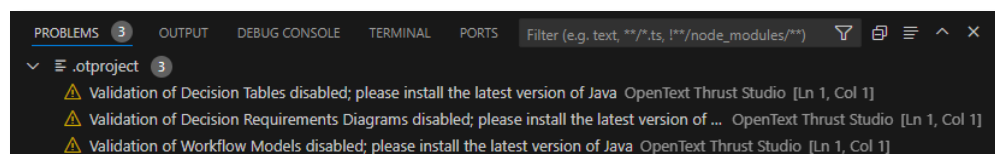
3. Save and close the OpenText project properties form.

**Note**

If Java is not installed on your system, VS Code will display the following three warning messages:



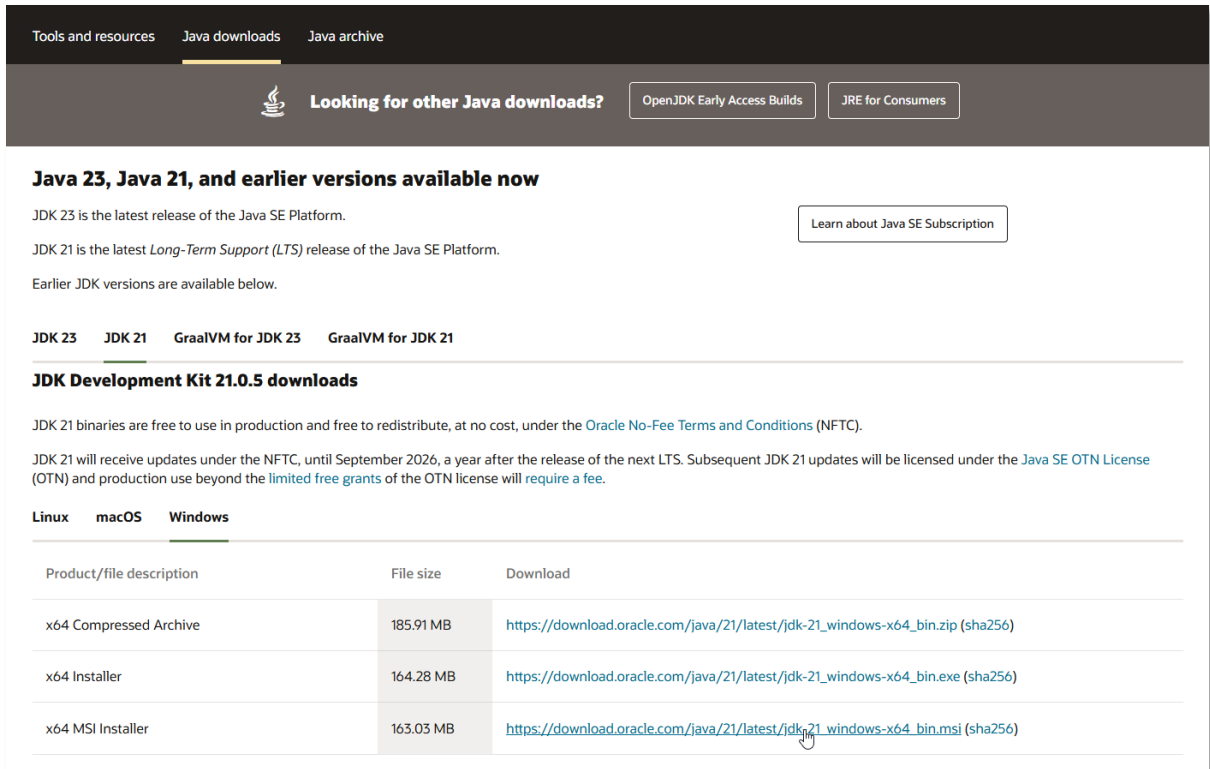
You can click **Don't display again** to stop these warning messages from popping up. However, in addition to the pop ups, the error message will show under the **PROBLEMS** tab until Java is installed. To open the **PROBLEMS** tab, select **View > Problems**.

**Tip**

Install the latest LTS Java version. To install the LTS Java version, see [Install Java](#).

4.3 Install Java

1. Go to <https://www.oracle.com/java/technologies/downloads>.
2. Download and install the latest Java LTS version for your operating system.



The screenshot shows the Oracle Java Downloads page. The navigation bar includes 'Tools and resources', 'Java downloads' (selected), and 'Java archive'. Below the navigation bar, there's a section titled 'Looking for other Java downloads?' with buttons for 'OpenJDK Early Access Builds' and 'JRE for Consumers'. The main content area is titled 'Java 23, Java 21, and earlier versions available now'. It states that JDK 23 is the latest release of the Java SE Platform and JDK 21 is the latest Long-Term Support (LTS) release. A button 'Learn about Java SE Subscription' is visible. Below this, there's a section for 'JDK Development Kit 21.0.5 downloads'. It mentions that JDK 21 binaries are free to use in production and free to redistribute under the Oracle No-Fee Terms and Conditions (NFTC). It also states that JDK 21 will receive updates under the NFTC until September 2026, after which updates will be licensed under the Java SE OTN License (OTN) and production use beyond the limited free grants of the OTN license will require a fee. At the bottom, there's a table with tabs for 'Linux', 'macOS', and 'Windows' (selected). The table has three columns: 'Product/file description', 'File size', and 'Download'. It lists three download options for Windows: 'x64 Compressed Archive' (185.91 MB), 'x64 Installer' (164.28 MB), and 'x64 MSI Installer' (163.03 MB). The 'x64 MSI Installer' link is highlighted with a mouse cursor.

Product/file description	File size	Download
x64 Compressed Archive	185.91 MB	https://download.oracle.com/java/21/latest/jdk-21_windows-x64_bin.zip (sha256)
x64 Installer	164.28 MB	https://download.oracle.com/java/21/latest/jdk-21_windows-x64_bin.exe (sha256)
x64 MSI Installer	163.03 MB	https://download.oracle.com/java/21/latest/jdk-21_windows-x64_bin.msi (sha256)



Note

For this tutorial, the latest LTS JDK version is **21.0.5**, and the **x64 MSI Installer** for Windows is used.

Next exercise module:

Create a namespace.

5 [5'] Create a namespace

Learn how to:

- Create a namespace model

A namespace allows grouping different type, trait, workflow, and decision table definitions together. For example, within the context of an application.

For more information on namespaces, see [Define a namespace, trait and "FILE" document type](#) section in the Content Metadata Service product documentation or the **Namespace** resource documentation in the [Content Metadata Service API reference](#).

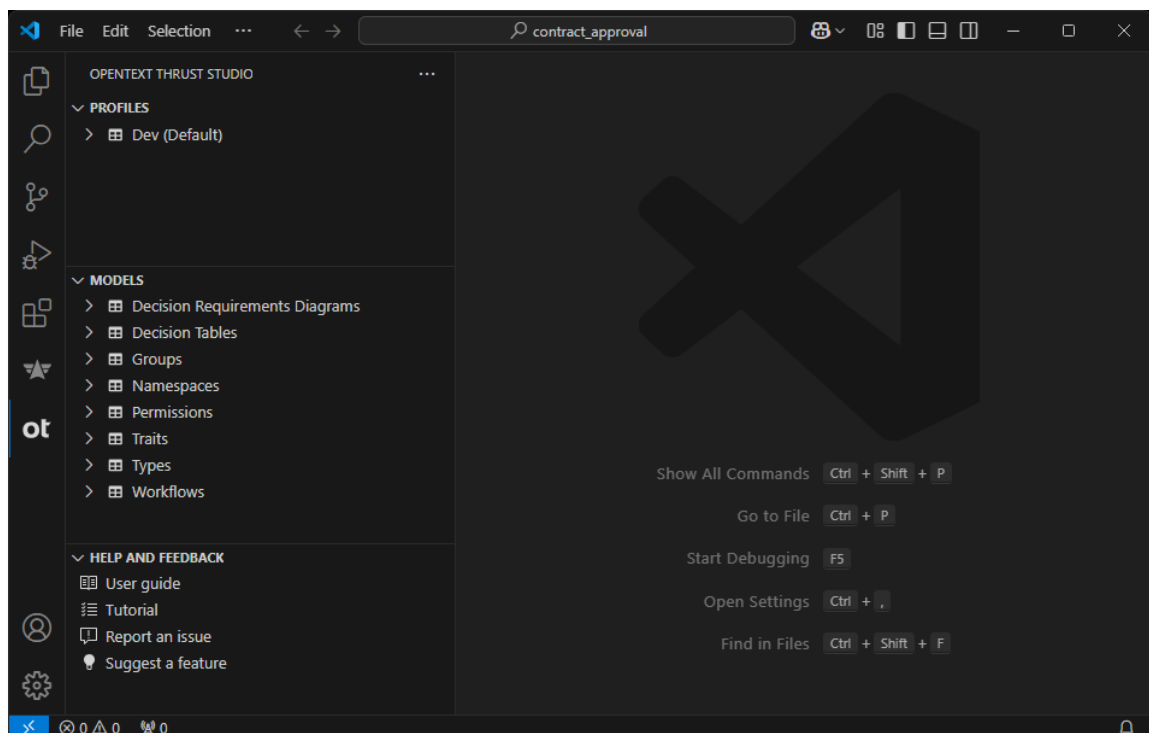
5.1 Create the Contract Approval namespace

1. In VS Code, switch to the **OpenText Thrust Studio** view.



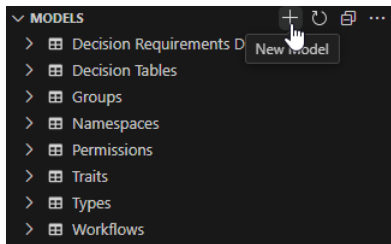
Note

After setting up the OpenText project, the **MODELS** section displays a tree view and allows you to explore the different models in your application project.

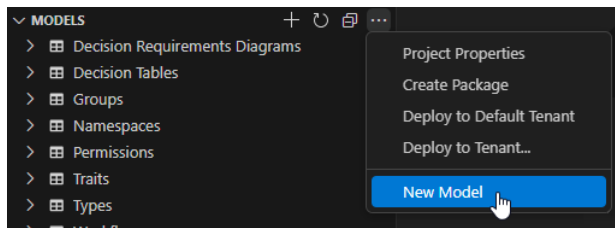


2. To create a new namespace, in the **MODELS** section, do any one of the following:

- Click the **New Model** button .



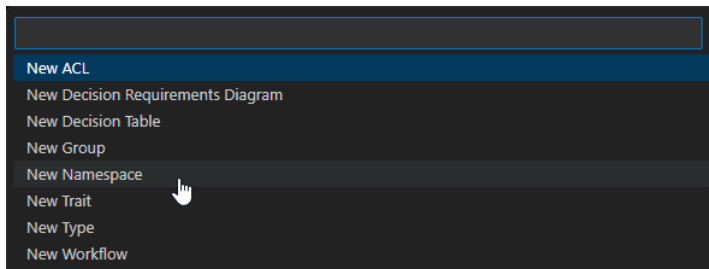
- Click the **More Actions** button  and select **New Model**.



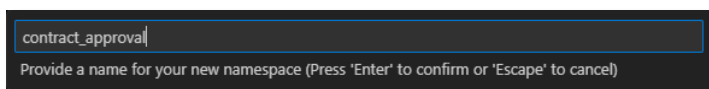
Note

There are different methods to create models. This is the first method. To understand the other methods for creating models, see [Create the Approval trait](#) and [Create the Contract type](#).

3. From the list, select **New Namespace**.



4. In the input box, type `contract_approval` for the (file) name of the namespace and press **Enter**.



5. Fill the Approval namespace properties using the following details:

Field	Value
Display name	The display name for the namespace. This is automatically populated based on the previously chosen namespace name (the model file name). Leave the value to be <code>Contract Approval</code> .
Name	The technical name for the namespace. This name must be unique within the tenant, and it is automatically populated based on the previously chosen namespace name (the model file name). Leave the value to be <code>contract_approval</code> .

Field	Value
Prefix	<p>The prefix representing the namespace. It is used for the (internal) system names of traits and types that belong to the namespace. The prefix must be unique within the tenant.</p> <p>Enter <code>ca</code> as value.</p>
Description	<code>Contract Approval Namespace</code>

contract_approval.otsns

* Display name

Contract Approval

* Name

contract_approval

* Prefix

ca

Description

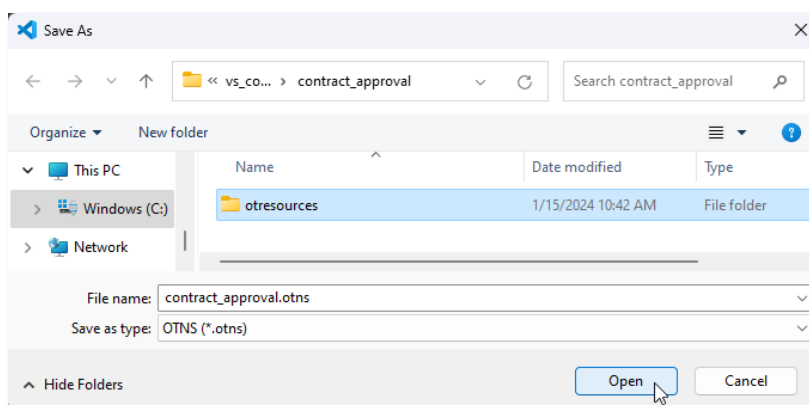
Contract Approval Namespace

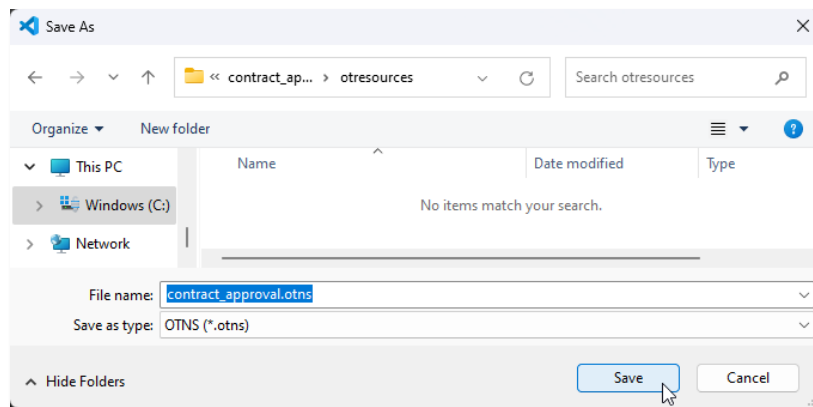
6. Save the Contract Approval namespace model.
7. In the **Save As** dialog box that opens when saving the model, select the **otresources** folder as target folder and make sure the file name is `contract_approval.otns`.



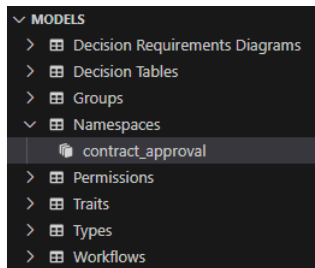
Note

The **otresources** folder is the model folder and it gets generated automatically during the project setup. You must save all the models inside this folder or one of its subfolders (as you can create sub folders to organize your work).





8. Click **Save** and close the namespace model.
9. The model explorer displays the new **contract_approval** namespace under **Namespaces**. The model explorer shows the different models according to their unique key, which in context of a namespace is the name property.



Next exercise module:

Create a trait.

6 [10'] Create a trait

Learn how to:

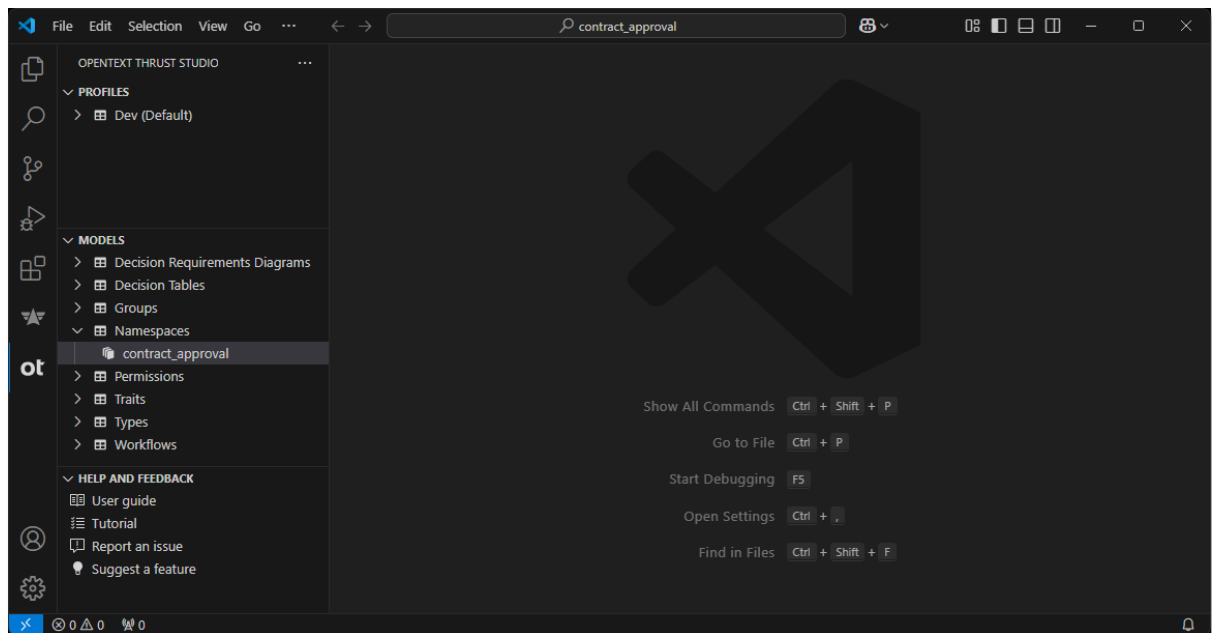
- Create a trait model

A trait allows grouping several attributes into one complex multi-attribute property. Trait instances can be dynamically added to a type instance as part of the business process when using the application. Traits can also be made mandatory in a type so that they must always be added when creating a new type instance. In this tutorial, the concept of mandatory traits is used to represent the different approval steps on a contract.

For more information on traits, see [Define a namespace, trait and "FILE" document type](#) and [Create instances using custom type with trait](#) sections in the Content Metadata Service product documentation or the **Trait** resource documentation in the [Content Metadata Service API reference](#).

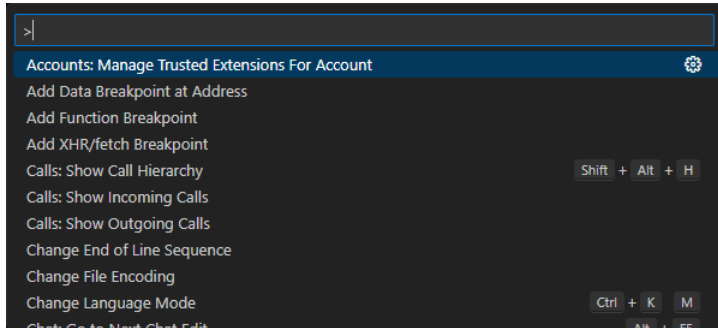
6.1 Create the Approval trait

1. In VS Code, switch to the **OpenText Thrust Studio** view.



- To create a new trait, press **F1** or **Ctrl+Shift+P**.

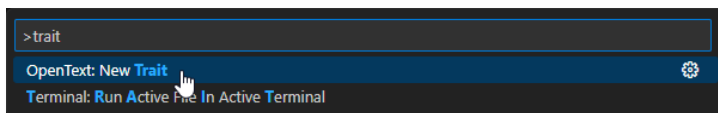
The command palette is displayed.



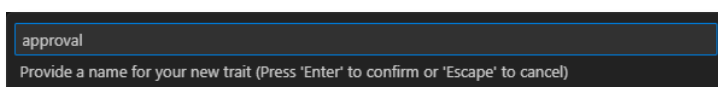
Note

There are different methods to create models. This is the second method. To understand the other methods to create models, see [Create the Contract Approval namespace](#) and [Create the Contract type](#).

- In the **command palette**, type `trait`. The command list is filtered and the entries containing trait are displayed.
- From the list, select **OpenText: New Trait**.




- In the **Provide a name for your new trait** input box, type `approval` for the (file) name of the trait and press **Enter**.



Note

The chosen name of “approval” reflects the purpose of this specific trait.

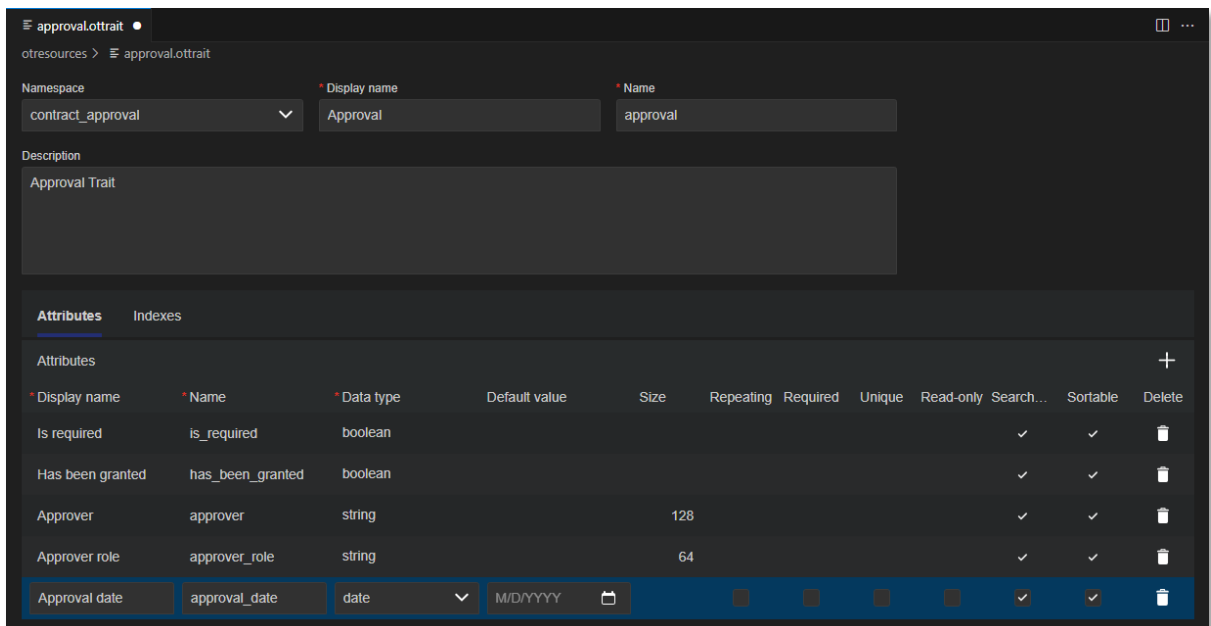
6. Fill the Approval trait properties using the following details:

Field	Value
Namespace	<p>The namespace to which the trait belongs.</p> <p>Select the contract_approval namespace as the namespace.</p> <div>  <p>Note</p> <p>OpenText Thrust Studio dynamically updates the different model reference lists. For a trait, the list of available namespaces is dynamically updated based on the namespaces that exist in the project.</p> </div>
Display name	<p>The display name is the user-friendly name for the trait. This does not have to be unique, and it is automatically populated based on the previously chosen trait name (the model file name).</p> <p>Leave the value to be <code>Approval</code>.</p>
Name	<p>The technical name for the trait. This name must be unique within the context of the tenant and selected namespace. It is automatically populated based on the previously chosen trait name (the model file name).</p> <p>Leave the value to be <code>approval</code>.</p>
Description	<code>Approval Trait</code>
Attributes	<p>The attributes list defines the different attributes of the trait.</p> <p>The following are the attribute properties:</p> <ul style="list-style-type: none"> • Display name: The attribute display name. OpenText recommends that the display name is unique. • Name: The attribute technical name. The value must be unique within the trait, and it is automatically populated based on the specified display name. • Data type: The data type of the attribute. The data type can be bigint, boolean, date, datetime, double, id, integer, string, or user. • Default value: The default value for the attribute, which is automatically assigned 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: Indicates the maximum length constraint for the string attribute. Applicable only to the string data type. • Repeating: Indicates if the attribute can have multiple values. • Unique: Indicates if the attribute needs to be unique across all instances of the trait. • Required: Indicates if the attribute must have a value when creating an instance of the trait. • Read-only: Indicates if the attribute can be modified after creation. • Searchable: Indicates if the attribute can be filtered on when performing a search. • Sortable: Indicates if the attribute can be used to sort a search result. <p>You will add the attributes for the Approval trait in the next step.</p>

7. In the **Attributes** list, click the **Add** button  to add the different attributes.

The Approval trait represents an approval step, and the following table describes each attribute and the property values to assign:

Attribute description	Display name	Name	Data type	Default value	Size	Boolean properties
Whether or not the approval is required	Is required	is_required	boolean			searchable, sortable
Whether or not the approval has been granted	Has been granted	has_been_granted	boolean			searchable, sortable
The email address of the approver	Approver	approver	string		128	searchable, sortable
The role of the approver	Approver role	approver_role	string		64	searchable, sortable
The exact date and time at which the approval request has been approved or at which it has been rejected	Approval date	approval_date	date			searchable, sortable



approval.ottrait

otresources > approval.ottrait

Namespace: contract_approval

Display name: Approval

Name: approval

Description: Approval Trait

Attributes

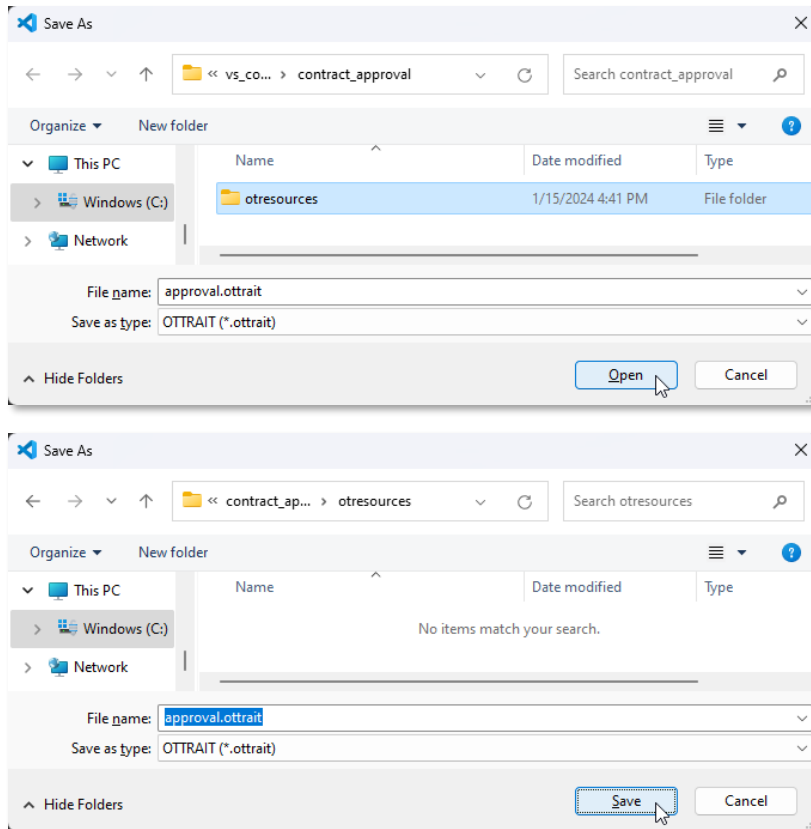
* Display name	* Name	* Data type	Default value	Size	Repeating	Required	Unique	Read-only	Searchable	Sortable	Delete
Is required	is_required	boolean							✓	✓	✖
Has been granted	has_been_granted	boolean							✓	✓	✖
Approver	approver	string		128					✓	✓	✖
Approver role	approver_role	string		64					✓	✓	✖
Approval date	approval_date	date	M/D/YYYY						✓	✓	✖

8. Save the Approval trait model.
9. In the **Save As** dialog box that opens when saving the model, select the **otresources** folder as target folder and make sure the file name is `approval.ottrait`.

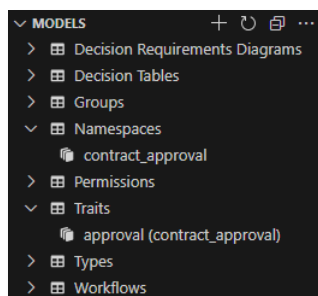


Note

The **otresources** folder is the model folder and it gets generated automatically during the project setup. You must save all the models inside this folder or one of its subfolders (as you can create sub folders to organize your work).



10. Click **Save** and close the trait model.
11. The model explorer displays the new **approval (contract_approval)** trait under **Traits**. The model explorer shows the different models according to their unique key, which in context of a trait is the combination of the namespace and name properties.



Next exercise module:

Create types.

7 [25'] Create types

Learn how to:

- Create a file type model
- Create a file type model that is a subtype
- Create a folder type model

A type is the main component for building an application's (custom) data model. A type has its own attributes and required traits that are added to the type instances when they are created.

A type can be of the following categories:

- file
- folder
- object
- relation

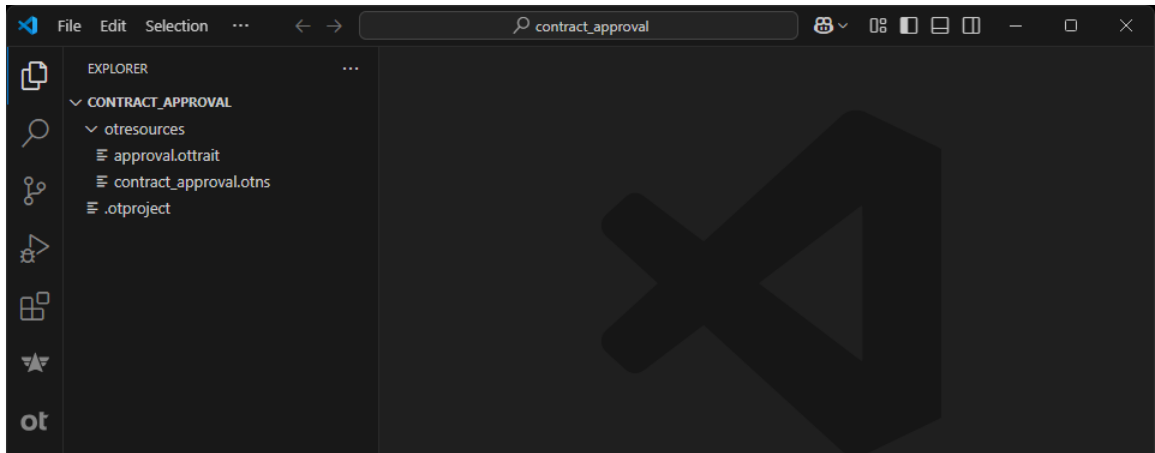
The following types will be created in this exercise module:

Type	Description
Contract	The Contract type represents a standard/base contract. It is the first out of two file types.
Loan Contract	The Loan Contract type is a specialization/subtype of the standard contract. It is the second file type.
Customer	The Customer type is the folder type. It is intended to contain all contracts (standard contracts and loan contracts) related to a specific customer.

For more information about types, see [Define a namespace, trait and "FILE" document type](#) and [Create instances using custom type with trait](#) sections in the Content Metadata Service product documentation or the **Type** resource documentation in the [Content Metadata Service API reference](#).

7.1 Create the Contract type

1. In VS Code, switch to the Explorer view.
2. From the **contract_approval** application root folder, expand the **otresources** (model) folder. You can see the previously created **approval** trait and **contract_approval** namespace models.

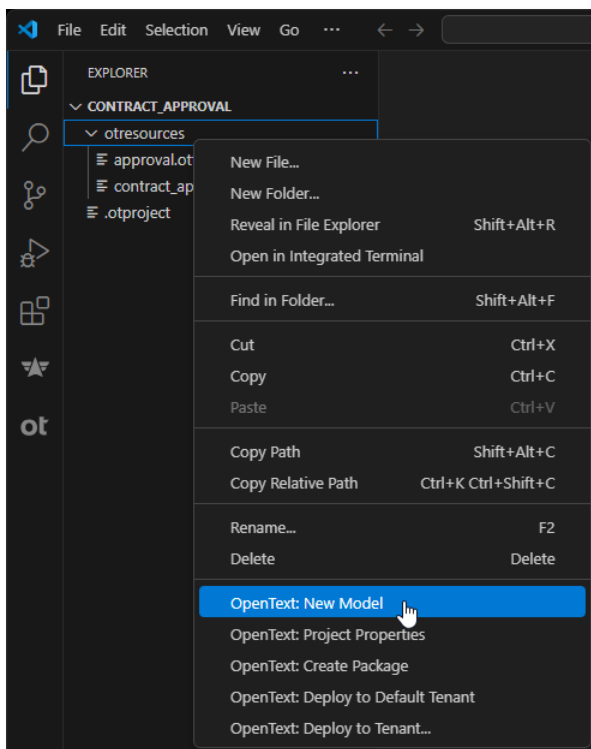


3. Right-click the **otresources** (model) folder and select **OpenText: New Model** to create a new type.

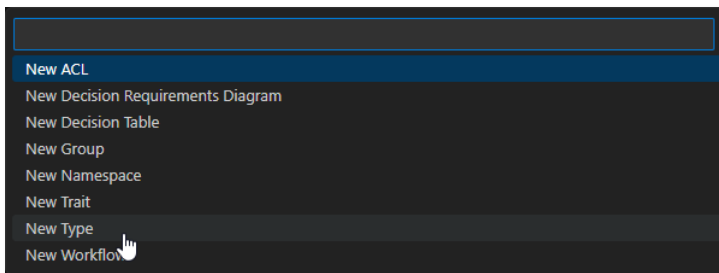


Note

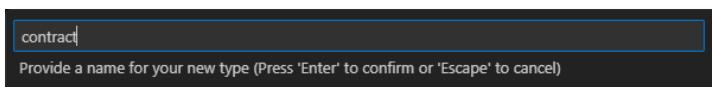
There are different methods to create models. This is the third method. To understand the other methods to create models, see [Create the Contract Approval namespace](#) and [Create the Approval trait](#).



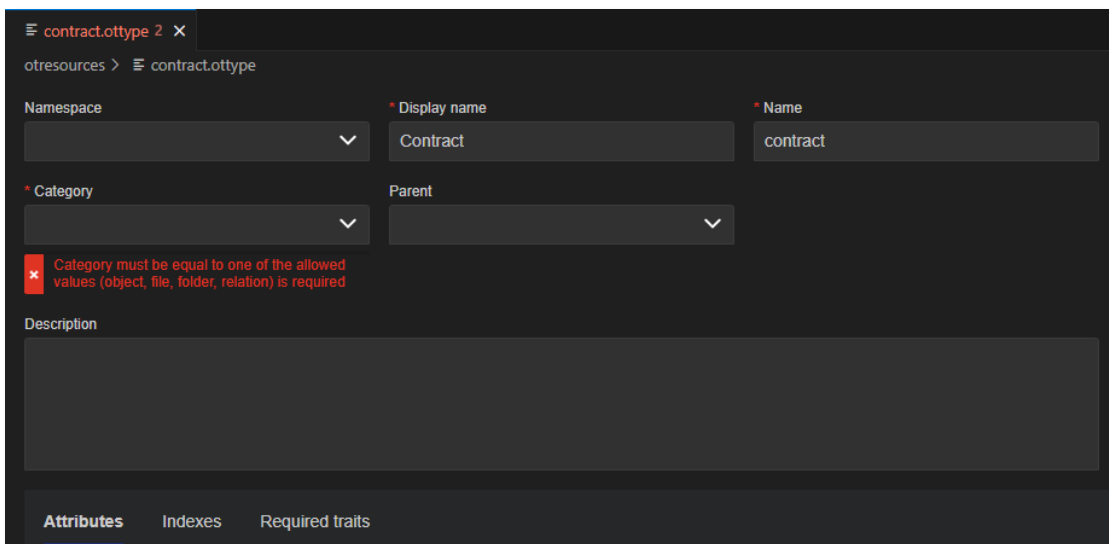
4. From the list, select **New Type**.



5. In the input box, type `contract` for the (file) name of the type and press **Enter**.

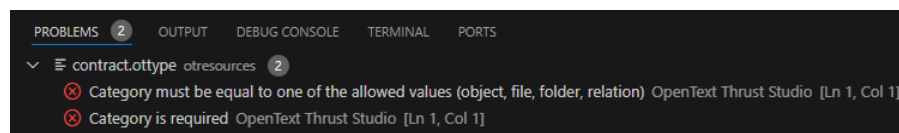


The value for the **Category** property is required and because the model is created directly in the file system (that is, it has already been saved), an error message is displayed for not selecting one of the allowed values.





Note

Validation errors for saved models are displayed under the **PROBLEMS** tab until the issues are resolved. To open the **PROBLEMS** tab, select **View > Problems**.



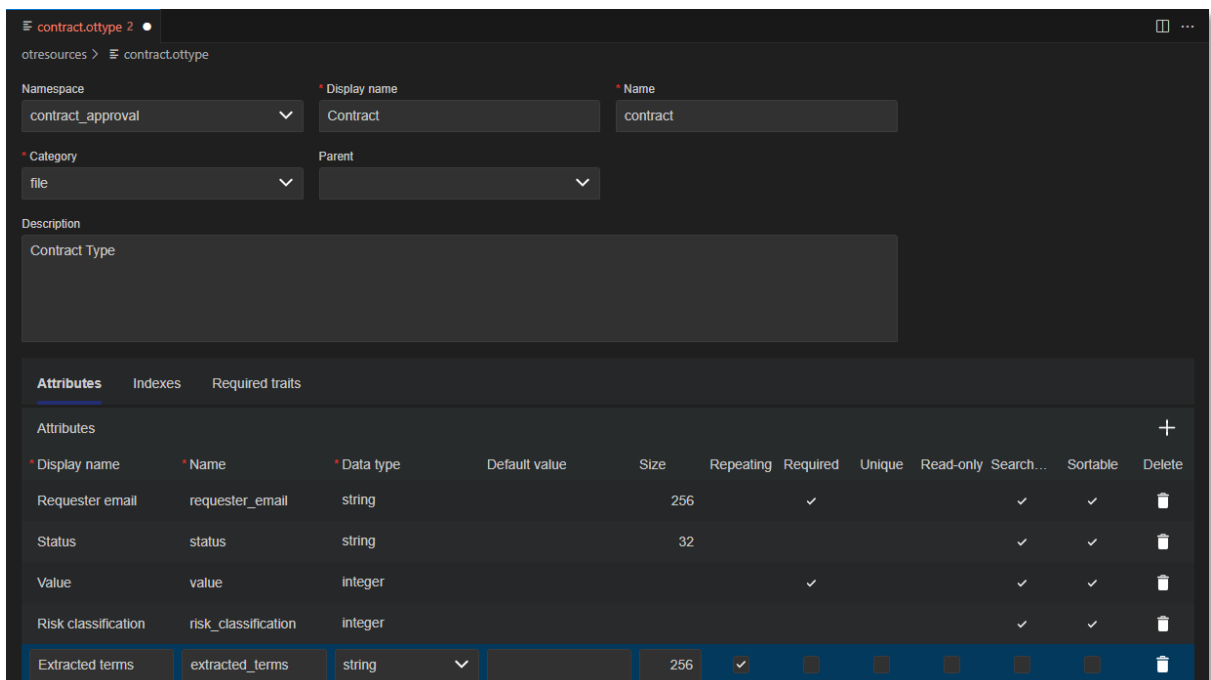
6. Fill the Contract type properties using the following details:

Field	Value
Namespace	<p>The namespace to which the type belongs.</p> <p>Select the contract_approval namespace.</p> <div>  <p>Note</p> <p>OpenText Thrust Studio dynamically updates the different model reference lists. For a type, the lists of available namespaces, traits and parent types are dynamically updated based on the namespaces, traits and types that exist in the project.</p> </div>
Display name	<p>The display name is the user-friendly name for the type. This does not have to be unique, and it is automatically populated based on the previously chosen type name (the model file name).</p> <p>Leave the value to be <code>Contract</code>.</p>
Name	<p>The technical name for the type. This name must be unique within the context of the tenant and selected namespace. It is automatically populated based on the previously chosen type name (the model file name).</p> <p>Leave the value to be <code>contract</code>.</p>
Category	<p>The category to which the type belongs. It can be a file, folder, object or relation.</p> <p>Select the file category.</p>
Parent	<p>The parent type for the type you are creating.</p> <p>The Contract type does not have a parent type as it is the base type for contracts.</p>
Description	<code>Contract Type</code>
Attributes	<p>The attributes list defines the different attributes of the type.</p> <p>For more information about attribute properties, see Create the Approval trait.</p> <p>You will add the attributes for the Contract type in step 7.</p>
Required Traits	<p>The required traits list defines the different mandatory traits for the type.</p> <p>The following are the required trait properties:</p> <ul style="list-style-type: none"> Instance name: The name of the required trait. This must be unique across the type's required traits. In this tutorial, the required traits are all Approval traits and the instance name will represent the type of approval. Trait name: The unique key (trait name + namespace) representing the selected trait. In this tutorial, all required traits will be approval (contract_approval) traits. <p>You will add the required traits for the Contract type in step 9.</p>

7. In the **Attributes** list, click the **Add** button  to add the different attributes.

The following table describes each attribute and the property values to assign:

Attribute description	Display name	Name	Data type	Default value	Size	Boolean properties
The email address of the person requesting the approval of the contract	Requester email	requester_email	string		256	required, searchable, sortable
The current (approval) status of the contract	Status	status	string		32	searchable, sortable
The (monetary) value of the contract	Value	value	integer			required, searchable, sortable
The risk classification of the contract in context of the personal data it contains	Risk classification	risk_classification	integer			searchable, sortable
The personal data related terms that were found in the contract	Extracted terms	extracted_terms	string		256	repeating



contract.ottype 2

otresources > contract.ottype

Namespace: contract_approval

Display name: Contract

Name: contract


Category: file

Parent:

Description: Contract Type

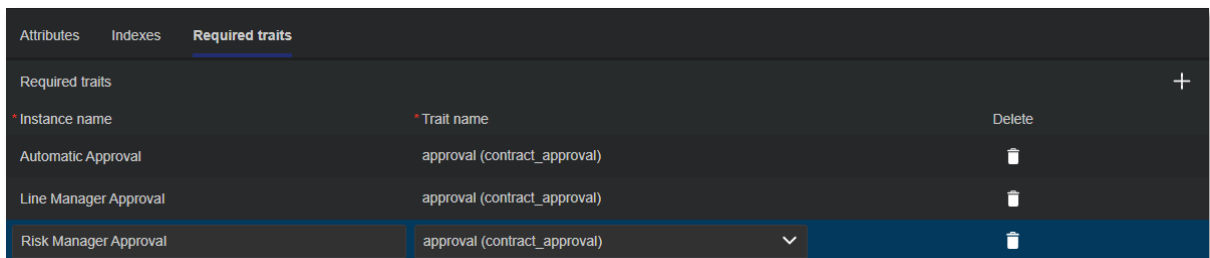
Attributes




Display name	Name	Data type	Default value	Size	Repeating	Required	Unique	Read-only	Searchable	Sortable	Delete
Requester email	requester_email	string		256		✓			✓	✓	✕
Status	status	string		32					✓	✓	✕
Value	value	integer				✓			✓	✓	✕
Risk classification	risk_classification	integer							✓	✓	✕
Extracted terms	extracted_terms	string		256	✓				✓	✓	✕

8. Select the **Required traits** tab and click the **Add** button  to add the required traits.

The following table describes each required trait and the property values to assign:

Required trait	Instance name	Trait Name
The automatic (by the system) approval, which is always required	Automatic Approval	approval (contract_approval)
The approval by the Line Manager, which is only required when the contract value is above 1000	Line Manager Approval	approval (contract_approval)
The approval by the Risk Manager, which is only required when the risk classification is above 3. For example, 4: HIGH or 5: VERY HIGH.	Risk Manager Approval	approval (contract_approval)



Required traits		
Instance name	Trait name	Delete
Automatic Approval	approval (contract_approval)	
Line Manager Approval	approval (contract_approval)	
Risk Manager Approval	approval (contract_approval)	

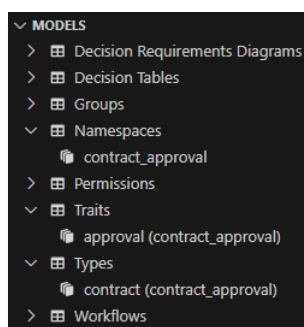
9. Save and close the Contract type model.



Note

Since you correctly filled the model properties, there no longer are validation errors under the **PROBLEMS** tab.

10. The model explorer displays the new **contract (contract_approval)** type under **Types**. The model explorer shows the different models according to their unique key, which in context of a type is the combination of the namespace and name properties.




Next step:

Create the Loan Contract type.

7.2 Create the Loan Contract type

1. In VS Code, create a new type model using any of the three model creation methods explained in the following exercises:
 - [Create the Contract Approval namespace](#)
 - [Create the Approval trait](#)
 - [Create the Contract type](#)
2. Type `loan_contract` for the (file) name of the type.
3. Fill the Loan Contract type properties using the following details:

Field	Value
Namespace	Select the contract_approval namespace.
Display name	Loan Contract
Name	<code>loan_contract</code>
Category	To allow subtyping the Contract type, and thus inheriting all its attributes and required traits, the Loan Contract type must be of the file category. Select the file category.
Parent	The parent type for the type you are creating. Select the contract (contract_approval) type.  Note The contract (contract_approval) parent type is only available to be selected from the Parent list when file is selected as category.
Description	Loan Contract Type
Attributes	See Attributes .
Required Traits	See Required Traits .

Attributes:

Attribute description	Display name	Name	Data type	Default value	Boolean properties
The total count of monthly payments required/chosen to reimburse the loan contract value	Monthly installments	<code>monthly_installments</code>	integer	12	required
The yearly income, which will be used together with the monthly payments and the loan contract value to determine solvency of the customer	Yearly income	<code>yearly_income</code>	integer		required

loan_contract.ottype

Namespace: contract_approval

Display name: Loan Contract

Name: loan_contract

Category: file

Parent: contract (contract_approval)

Description: Loan Contract Type

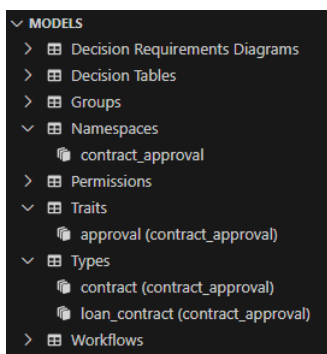
* Display name	* Name	* Data type	Default value	Size	Repeating	Required	Unique	Read-only	Search...	Sortable	Delete
Monthly installments	monthly_installments	integer		12		✓					
Yearly income	yearly_income	integer				✓					

Required traits:

Required trait description	Instance name	Trait name
The automated approval step checks if the customer is solvent by checking the monthly cost. The monthly cost must not exceed 25% of the monthly income (calculated from the loan contract cost, the total count of monthly payments and the yearly income).	Solvency Check	approval (contract_approval)

* Instance name	* Trait name	Delete
Solvency Check	approval (contract_approval)	

- Save and close the Loan Contract type model. The model explorer displays the new **loan_contract (contract_approval)** type under **Types**.



Next step:

Create the Customer type.

7.3 Create the Customer type

1. In VS Code, create a new type model using any of the three model creation methods explained in the following exercises:
 - [Create the Contract Approval namespace](#)
 - [Create the Approval trait](#)
 - [Create the Contract type](#)
2. Type `customer` for the (file) name of the type.
3. Fill the Customer type properties using the following details:

Field	Value
Namespace	Select the contract_approval namespace.
Display name	Customer
Name	customer
Category	Select the folder category.
Parent	The Customer type does not have a parent type.
Description	Customer Type
Attributes	See Attributes .

Attributes:

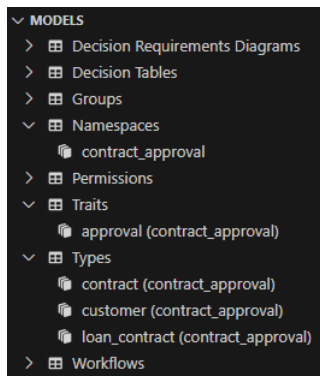
Attribute description	Display name	Name	Data type	Default value	Size	Boolean properties
The email address of the customer	Customer email	customer_email	string		256	required, searchable, sortable

The screenshot shows the OpenText Thrust Services VS Code extension interface. The 'customer.ottype' file is open. The configuration for the 'customer' type is as follows:

- Namespace:** contract_approval
- Display name:** Customer
- Name:** customer
- Category:** folder
- Parent:** (empty)
- Description:** Customer Type
- Attributes:**

* Display name	* Name	* Data type	Default value	Size	Repeating	Required	Unique	Read-only	Search...	Sortable	Delete
Customer email	customer_email	string		256	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

4. Save and close the Customer type model. The model explorer displays the new **customer (contract_approval)** type under **Types**.



Next exercise module:

Create (user) groups.

8 [10'] Create (user) groups

Learn how to:

- Create group models
- Add subgroups

After deploying the application into a tenant, the deployed groups can be populated with users to allow managing role-based behavior and security (through ACLs).

The following user groups will be created in this exercise module:

User group	User privilege
administrators	Has full access on all contracts created within the application to allow administering them.
line_managers	Can approve contracts as line manager.
risk_managers	Can approve contracts as risk manager.
contract_approval_users	Regular user of the Contract Approval application. Will not have administrator access or be allowed to approve contracts.

8.1 Create the administrators group

1. In VS Code, create a new group model using any of the three model creation methods explained in the following exercises:
 - [Create the Contract Approval namespace](#)
 - [Create the Approval trait](#)
 - [Create the Contract type](#)
2. Type `administrators` for the (file) name of the group.
3. Fill the administrator group properties using the following details:

Field	Value
Group name	The technical name for the group. This name must be unique within the context of the tenant. It is automatically populated based on the previously chosen group name (the model file name). Leave the value to be <code>administrators</code> .
Description	Group containing the <code>administrators</code> for the Contract Approval application
Groups	The administrators group has no (sub) groups.

administrators.otgrp

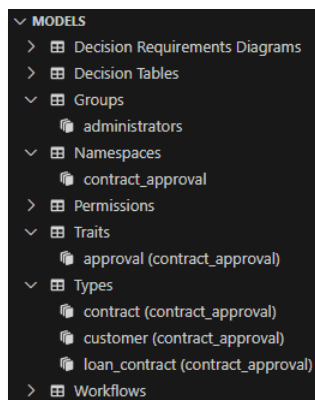
Group name
administrators

Description
Group containing the administrators for the Contract Approval application

Groups	
Group name	Description
Delete	

There are currently no records to display

- Save and close the administrators group model. The model explorer displays the new **administrators** group under **Groups**.



Next step:

Create the line_managers group.

8.2 Create the line_managers group

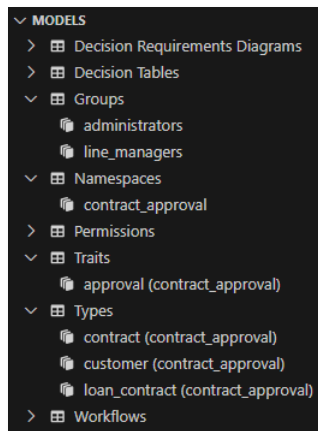
1. In VS Code, create a new group model using any of the three model creation methods explained in the following exercises:
 - [Create the Contract Approval namespace](#)
 - [Create the Approval trait](#)
 - [Create the Contract type](#)
2. Type `line_managers` for the (file) name of the group.
3. Fill the line_managers group properties using the following details:

Field	Value
Group name	<code>line_managers</code>
Description	Group containing the users that have the Line Manager role
Groups	The line_managers group has no (sub) groups.

The screenshot shows the VS Code interface for editing a group model named 'line_managers.otgrp'. The 'Group name' field is set to 'line_managers'. The 'Description' field contains the text 'Group containing the users that have the Line Manager role'. Below these fields is a table for 'Groups' with columns 'Group name', 'Description', and 'Delete'. The table is currently empty, and a message at the bottom states 'There are currently no records to display'.

Groups		
Group name	Description	Delete
There are currently no records to display		

4. Save and close the line_managers group model. The model explorer displays the new **line_managers** group under **Groups**.



Next step:

Create the risk_managers group.

8.3 Create the risk_managers group

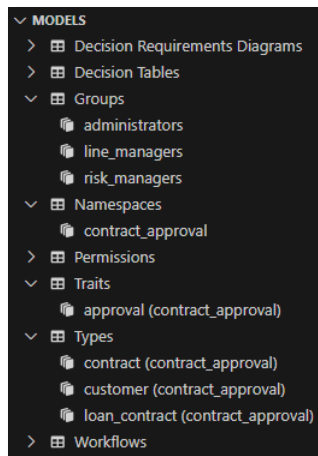
1. In VS Code, create a new group model using any of the three model creation methods explained in the following exercises:
 - [Create the Contract Approval namespace](#)
 - [Create the Approval trait](#)
 - [Create the Contract type](#)
2. Type `risk_managers` for the (file) name of the group.
3. Fill the risk_managers group properties using the following details:

Field	Value
Group name	<code>risk_managers</code>
Description	Group containing the users that have the Risk Manager role
Groups	The risk_managers group has no (sub) groups.

The screenshot shows the VS Code editor with a file named `risk_managers.otgrp`. The interface displays the following fields and values:

- Group name:** `risk_managers`
- Description:** Group containing the users that have the Risk Manager role
- Groups:** A table with columns `Group name`, `Description`, and `Delete`. The table is currently empty, with a message at the bottom stating "There are currently no records to display".

4. Save and close the risk_managers group model. The model explorer displays the new **risk_managers** group under **Groups**.



Next step:

Create the contract_approval_users group.

8.4 Create the contract_approval_users group

1. In VS Code, create a new group model using any of the three model creation methods explained in the following exercises:
 - [Create the Contract Approval namespace](#)
 - [Create the Approval trait](#)
 - [Create the Contract type](#)
2. Type `contract_approval_users` for the (file) name of the group.
3. Fill the contract_approval_users group properties using the following details:

Field	Value
Group name	<code>contract_approval_users</code>
Description	Group containing the users that are allowed to use the Contract Approval application
Groups	Select both the line_managers and risk_managers groups using the + in the Groups list.

contract_approval_users.otgrp

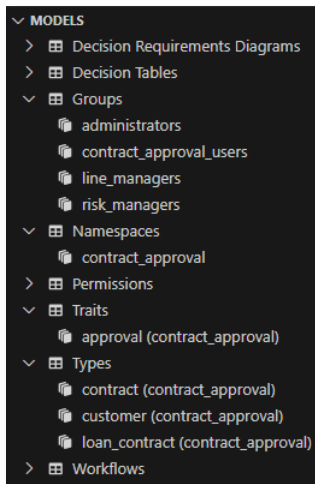
* Group name
contract_approval_users

Description
Group containing the users that are allowed to use the Contract Approval application

Groups

Group name	Description	Delete
line_managers	Group containing the users that have the Line Manager role	
risk_managers	Group containing the users that have the Risk Manager role	

4. Save and close the contract_approval_users group model. The model explorer displays the new **contract_approval_users** group under **Groups**.



Next exercise module:

Create ACLs.

9 [15'] Create ACLs

Learn how to:

- Create Access Control List (ACL) models that correspond with the different permissions for your Contract Approval application
- Assign custom permissions

The following ACLs will be created in this exercise module:

ACL	Description
created	<p>Applied to newly created contracts, granting the following permissions:</p> <ul style="list-style-type: none"> • Full control for the owner or creator of the new contract • Read access for the regular application users • Full control for the administrators
pending_approval	<p>Applied to contracts that are waiting to be approved by a line manager, granting the following permissions:</p> <ul style="list-style-type: none"> • Custom read access allowing to change the status and security of the contract (that is, approve it) for the owner of the contract, removing full control • Read access for the regular application users • Custom read access allowing to change the status and security of the contract (that is, approve it) for the line managers • Custom read access allowing to change the status and security of the contract (that is, approve it) for the risk managers • Full control for the administrators
completed	<p>Applied to completed (approved, rejected, or expired) contracts, granting the following permissions:</p> <ul style="list-style-type: none"> • Read access for the owner, removing custom approval permissions • Read access for the regular application users • Full control for the administrators

9.1 Create the created ACL

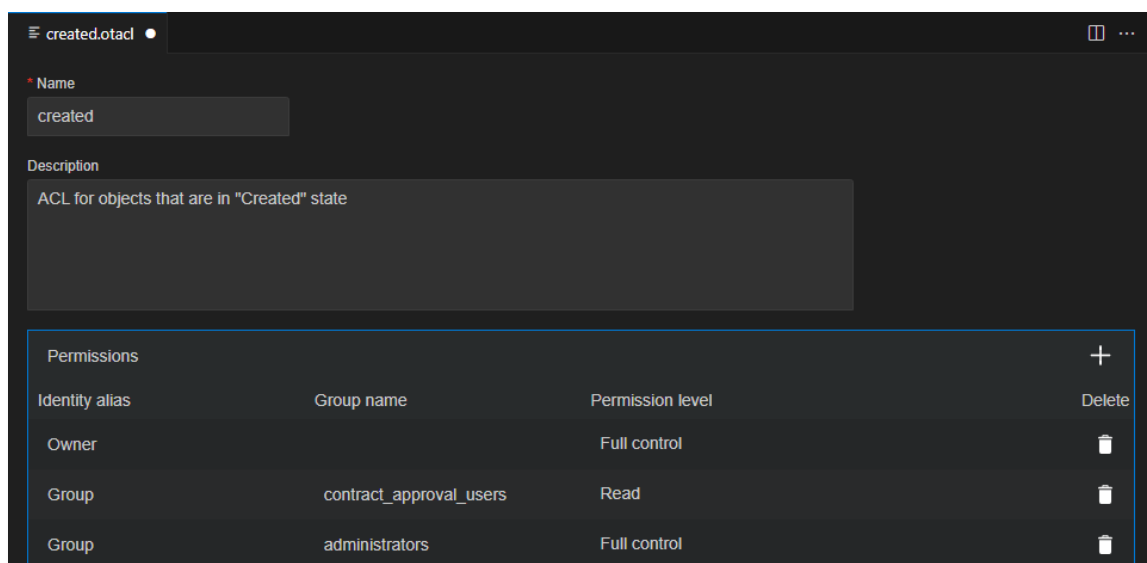
1. In VS Code, create a new ACL model using any of the three model creation methods explained in the following exercises:
 - [Create the Contract Approval namespace](#)
 - [Create the Approval trait](#)
 - [Create the Contract type](#)
2. Type `created` for the (file) name of the ACL.
3. Fill the created ACL properties using the following details:

Field	Value
Name	<code>created</code>
Description	ACL for objects that are in "Created" state
Permissions	<p>The permissions list defines the different permissions for each of the accessors. When creating an ACL, this list already contains one entry granting the owner full control.</p> <p>You will add the permissions for the created ACL in the next step.</p>

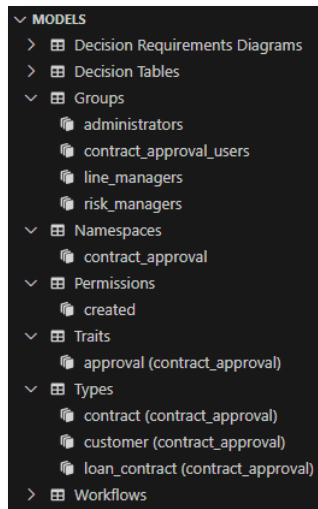
4. In the **Permissions** list, click  to add the different permissions.

The following table describes each permission and the property values to assign:

Identity alias	Group name	Permission level
Owner		Full control
Group	<code>contract_approval_users</code>	Read
Group	<code>administrators</code>	Full control



5. Save and close the created ACL model.
6. The model explorer displays the new **created** ACL under **Permissions**.



Next step:

Create the pending_approval ACL.

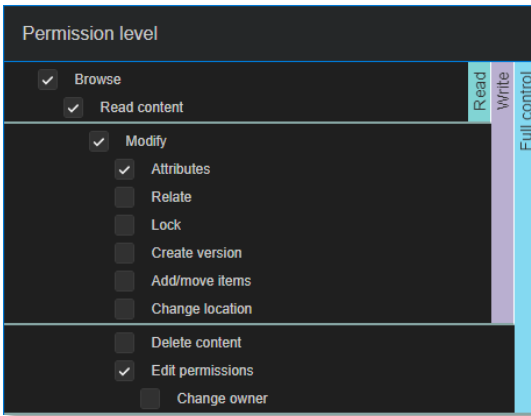
9.2 Create the pending_approval ACL

1. In VS Code, create a new ACL model using any of the three model creation methods explained in the following exercises:
 - [Create the Contract Approval namespace](#)
 - [Create the Approval trait](#)
 - [Create the Contract type](#)
2. Type `pending_approval` for the (file) name of the ACL.
3. Fill the pending_approval ACL properties using the following details:

Field	Value
Name	<code>pending_approval</code>
Description	ACL for objects that are in "Pending Approval" state
Permissions	You will add the permissions for the pending_approval ACL in the next step.

4. In the **Permissions** list, click  to add the different permissions.

The following table describes each permission and the property values to assign:

Identity alias	Group name	Permission level
Owner		Custom: Browse, Read content, Modify, Attributes, Edit permissions 
Group	contract_approval_users	Read
Group	line_managers	Custom: Browse, Read content, Modify, Attributes, Edit permissions
Group	risk_managers	Custom: Browse, Read content, Modify, Attributes, Edit permissions
Group	administrators	Full control

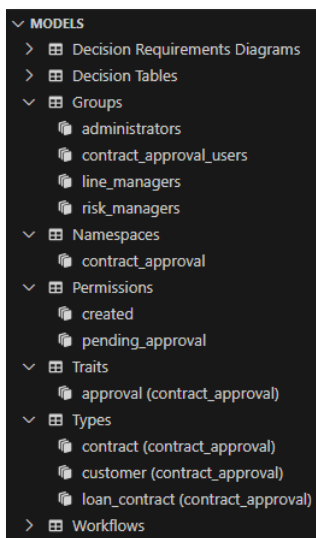
pending_approval.otad

Name
pending_approval

Description
ACL for objects that are in "Pending Approval" state

Permissions			+
Identity alias	Group name	Permission level	Delete
Owner		Custom	
Group	contract_approval_users	Read	
Group	line_managers	Custom	
Group	risk_managers	Custom	
Group	administrators	Full control	

5. Save and close the pending_approval ACL model. The model explorer displays the new **pending_approval** ACL under **Permissions**.




Next step:

Create the completed ACL.

9.3 Create the completed ACL

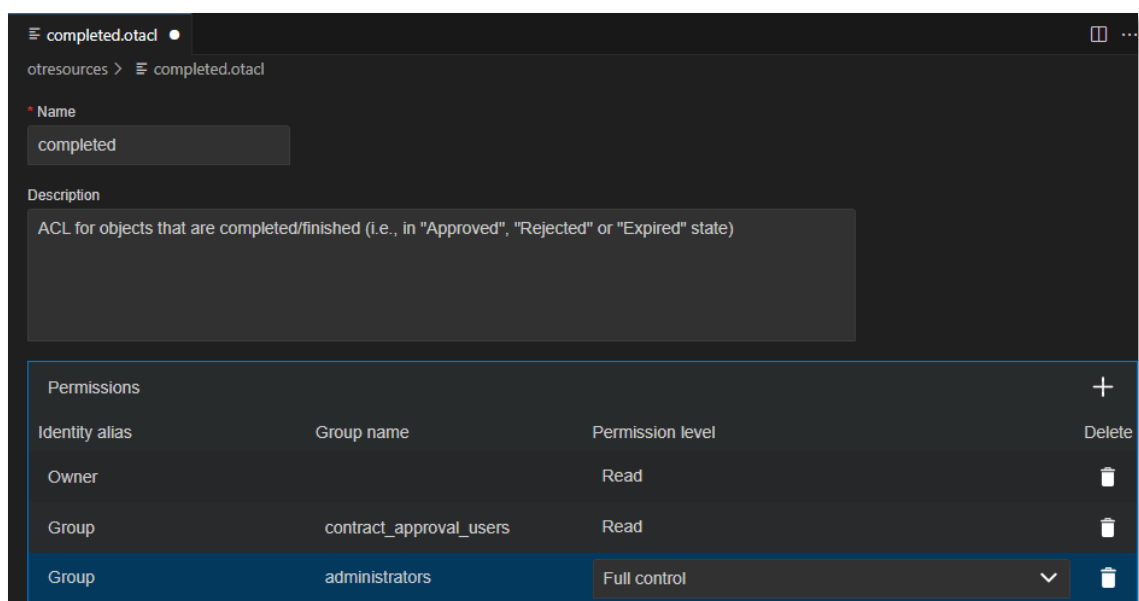
1. In VS Code, create a new ACL model using any of the three model creation methods explained in the following exercises:
 - [Create the Contract Approval namespace](#)
 - [Create the Approval trait](#)
 - [Create the Contract type](#)
2. Type `completed` for the (file) name of the ACL.
3. Fill the completed ACL properties using the following details:

Field	Value
Name	<code>completed</code>
Description	ACL for objects that are completed/finished (i.e., in "Approved", "Rejected" or "Expired" state)
Permissions	You will add the permissions for the completed ACL in the next step.

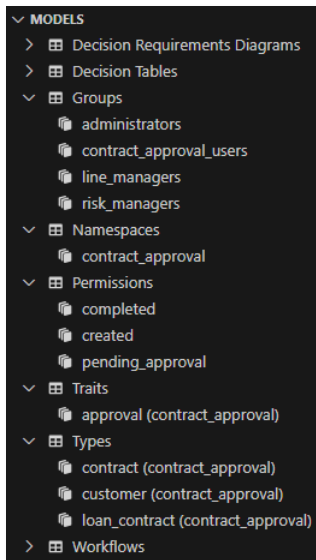
4. In the **Permissions** list, click the **Add** button  to add the different permissions.

The following table describes each permission and the property values to assign:

Identity alias	Group name	Permission level
Owner		Read
Group	<code>contract_approval_users</code>	Read
Group	<code>administrators</code>	Full control



5. Save and close the completed ACL model. The model explorer displays the new **completed** ACL under **Permissions**.



Next exercise module:

Create a decision table.

10 [25'] Create a decision table

Learn how to:

- Create a decision table model
- Define input and output variables
- Determine the right hit policy for the output variables

A decision table allows configuring business rules to be used when building the application logic. These configured business rules can be consumed directly from the application code or from a workflow.

The following decision table will be created in this exercise module:

Decision table	Description
Contract Approvals	<p>Defines the business logic to determine the necessary approvals for a given contract.</p> <p>Depending on the following criteria, the decision table decides the required approvals for the contract:</p> <ul style="list-style-type: none"> • Loan contract: An automated solvency checks to ensure the loan applicant can pay back the loan according to the agreed conditions. • Standard contract (that is, not a loan contract) and its value is over 1000: A manual approval from line manager. • Loan contract and its value is over 5000: A manual approval from the line manager. The loan contract value requiring line manager approval is higher because there is already an automated solvency check for loan contracts. • Contract risk level is higher than 3 (medium risk): A manual approval from risk manager.

For more information on Decision Service decision tables, see the [Decision Service Reference Guide](#), the [Decision Table Modeler Reference Guide](#), or the [Decision Service API Reference](#).

10.1 Create the Required Approvals decision table

1. In VS Code, create a new decision table model using any of the three model creation methods explained in the following exercises:
 - [Create the Contract Approval namespace](#)
 - [Create the Approval trait](#)
 - [Create the Contract type](#)
2. Type `required_approvals` for the (file) name of the decision table.
3. Fill the created decision table properties using the following details:

Field	Value
Namespace	Select the contract_approval namespace.
Display name	Required Approvals
Name	<code>required_approvals</code>
Hit policy	<p>The hit policy provides instructions on how the business rules in the decision table must be executed, evaluated, and matched.</p> <p>Following are the options in the Hit policy list:</p> <p><u>Single hit policies</u></p> <ul style="list-style-type: none"> • First: Multiple (overlapping) rules can match, with different output entries. The first hit by rule order is returned (and evaluation can halt). • Any: There can be an overlap, but all the matching rules show equal output entries for each output, so any match can be used. If the output entries are not equal, the hit policy is incorrect, and the result is undefined. • Unique: No overlap is possible, and all rules are disjoint. Only a single rule can be matched. • Priority: Multiple rules can match, with different output entries. This policy returns the matching rule with the highest output priority. Output priorities are specified in the ordered list of allowed output values, in decreasing order of priority. Priorities are independent from the rule order. <p><u>Multiple hit policies</u></p> <ul style="list-style-type: none"> • Rule Order: Returns all hits in rule order. • Output Order: Returns all hits in decreasing output priority order. Output priorities are specified in the ordered list of allowed output values in decreasing order of priority. Priorities are independent from the rule order. • Collect: Returns all hits in arbitrary order. • Collect (Sum): Returns the sum of the values of all hits. • Collect (Min): Returns the smallest value of all hits. • Collect (Max): Returns the largest value of all hits. • Collect (Count): Returns the number of hits. <p>By default, the “First” hit policy is selected.</p> <p>Select the Rule Order hit policy to instruct the Decision Service to collect and return the output (required approval) of all business rules it hits in the same order as defined in the decision table.</p>
Description	Decision table to determine the required approval steps for the contract

required_approvals.otdt

* Namespace: contract_approval


* Display name: Required Approvals


* Name: required_approvals

Hit policy: Rule Order

Description: Decision table to determine the required approval steps for the contract

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

- To configure the New input column, in the decision table, from the **New Input** column, click the **Edit Action** button  and select **Edit**.
- Fill the **Edit input column** dialog box using the following details:

Field	Value
Column label	The column label is the label that is shown at the top of the input column in the decision table. Enter <code>Contract Type</code> for the column label.
Variable name	The variable name is the name of the input variable. Enter <code>contract_type</code> for the variable name.
Variable type	The variable type is the data type of the input variable. The data type can be string, number, boolean, date, dateTime, json, duration, or collection. Select the string variable type.
Allowed values (optional)	The allowed values is the optional ordered list (from highest to lowest priority) of the different possible/allowed values for the input variable. Add the following two allowed values by typing them in and clicking  : <ul style="list-style-type: none"> <code>ca_contract</code> <code>ca_loan_contract</code>

Edit input column

Column label
Contract Type

* Variable name
contract_type

* Variable type
string


Allowed values (optional)
Enter allowed values +

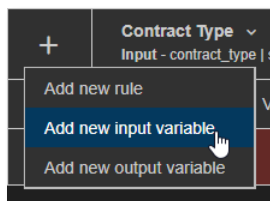
ca_contract x ca_loan_contract x

Submit Close

6. Click **Submit** to confirm the input column properties.

+	Contract Type ▼ Input - contract_type string [ca_contract,ca_loan_contract]	New Output ▼ Output - none
Rules	Operator	Value
1	== ▼	▼

7. To create the second input variable, in the decision table, from the first column, click the **Add Action** button  and select **Add new input variable**.



8. Fill the New input column dialog box using the following details:

Field	Value
Column label	Contract Value
Variable name	contract_value
Variable type	Select the number variable type.
Allowed values (optional)	Allowed values list has no entries.

New input column

Column label
Contract Value

* Variable name
contract_value


* Variable type
number

Allowed values (optional)
Enter allowed values +

Submit Close

9. Click **Submit** to confirm the input column properties.

Contract Value	Operator	Value	New Output
Input - contract_value number	==		Output - none

10. To create the third input variable, from the decision table first column, click the **Add Action** button  and select **Add new input variable**.

+ Contract Type

Input - contract_type | s

Add new rule


Add new input variable

Add new output variable

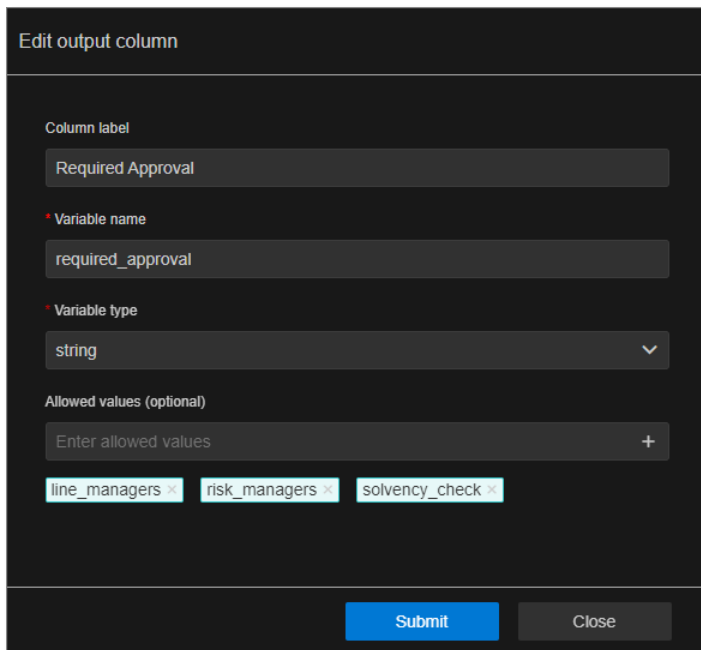
11. In the **New input column** dialog box, fill the input column properties for the new input variable using the following details:

Field	Value
Column label	Risk Level
Variable name	risk_level
Variable type	Select the number variable type.
Allowed values (optional)	<ul style="list-style-type: none"> • 1 • 2 • 3 • 4 • 5

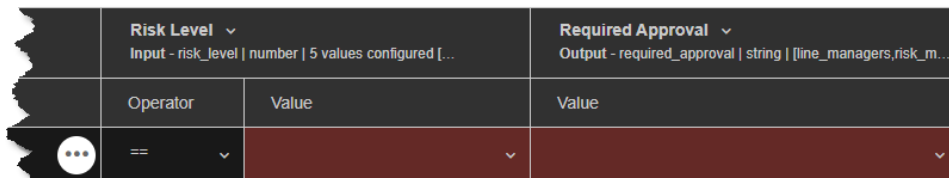
12. Click **Submit** to confirm the input column properties.

13. In the decision table, scroll and locate the **New Output** column and click the **Edit Action** button  and select **Edit** to configure the output column.
14. Fill the **Edit output column** dialog box properties using the following details:


Field	Value
Column label	Required Approval
Variable name	required_approval
Variable type	Select the string variable type.
Allowed values (optional)	<ul style="list-style-type: none"> line_managers risk_managers solvency_check



15. Click **Submit** to confirm the output column properties.



Risk Level	Required Approval
Input - risk_level number 5 values configured [...]	Output - required_approval string [line_managers,risk_m...]
Operator	Value
Value	Value

16. Add three more rules to the decision table. To do this, from the first column, click the **Add Action** button  and select **Add new rule** three times.



17. Fill all four rules according to the following table:



Note

Selecting a **value** of “-” will set the corresponding **operator** to “--NA--” as there is no value to perform a comparison operation on.

	Contract Type		Contract value		Risk Level		Required Approval
Rules	Operator	Value	Operator	Value	Operator	Value	Value
1	==	ca_loan_contract	-- NA --	-	-- NA --	-	solvency_check
2	==	ca_contract	>	1000	-- NA --	-	line_managers
3	==	ca_loan_contract	>	5000	-- NA --	-	line_managers
4	-- NA --	-	-- NA --	-	>	3	risk_managers

required_approvals.otut

Namespace: contract_approval | Display name: Required Approvals | Name: required_approvals

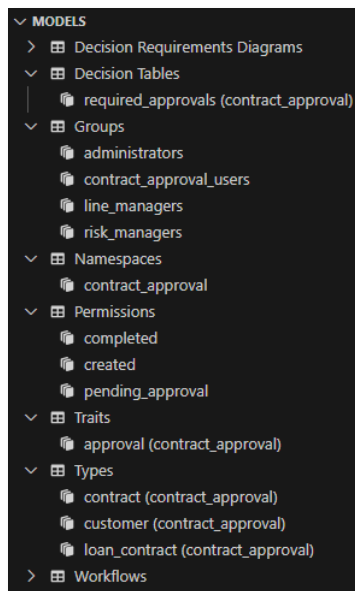
Hit policy: Rule Order

Description: Decision table to determine the required approval steps for the contract

	Contract Type		Contract Value		Risk Level		Required Approval
Rules	Operator	Value	Operator	Value	Operator	Value	Value
1	==	ca_loan_contract	-- NA --	-	-- NA --	-	solvency_check
2	==	ca_contract	>	1000	-- NA --	-	line_managers
3	==	ca_loan_contract	>	5000	-- NA --	-	line_managers
4	-- NA --	-	-- NA --	-	>	3	risk_managers

18. Save and close the Required Approvals decision table model.

19. The model explorer displays the new **required_approvals (contract_approval)** decision table under **Decision Tables**. The model explorer shows the different models according to their unique key, which in context of a decision table is the combination of the namespace and name properties.



Next exercise module:

Create workflows.

11 [140'] Create workflows

Learn how to:

- Create a workflow model
- Perform different types of REST API calls in a workflow
- Use a decision table in a workflow
- Use another workflow in a workflow
- Use JavaScript code in a workflow
- Define a user/manual task in a workflow
- Send an email from a workflow
- Define workflow execution paths (decide which paths to activate/follow, parallel paths, joining execution paths)

A workflow model represents an executable (business) process model from which process instances can be created. The executable process model is stored as BPMN 2.0 encoded JSON.

The following workflows will be created in this exercise module:

Workflow	Description
Solvency Check	The Solvency Check workflow performs the automated solvency check for loan contracts. It determines whether the requester has enough cash flow to pay back the loan contract based on the yearly income, monthly payments, and contract value.
Manager Approval	The Manager Approval workflow orchestrates the manual approval of a contract by a user that belongs to a specific manager group. In the context of the Contract Approval application this is either the line managers group or the risk managers group.
Contract Approval	The Contract Approval workflow represents the overall business process of approving a contract and is the main workflow for the Contract Approval application. As such, it calls the decision table and the other workflows. It consists of automated and manual approval tasks and not all approval tasks are always required. They are conditional, based on the rules defined in the decision table. At the end an email is sent to inform the requester about the outcome of the approval process.

For more information on Workflow Service process models and process instances, see the [Workflow Service product documentation](#), the [Workflow Modeler product documentation](#), or the [Workflow Service API reference](#).

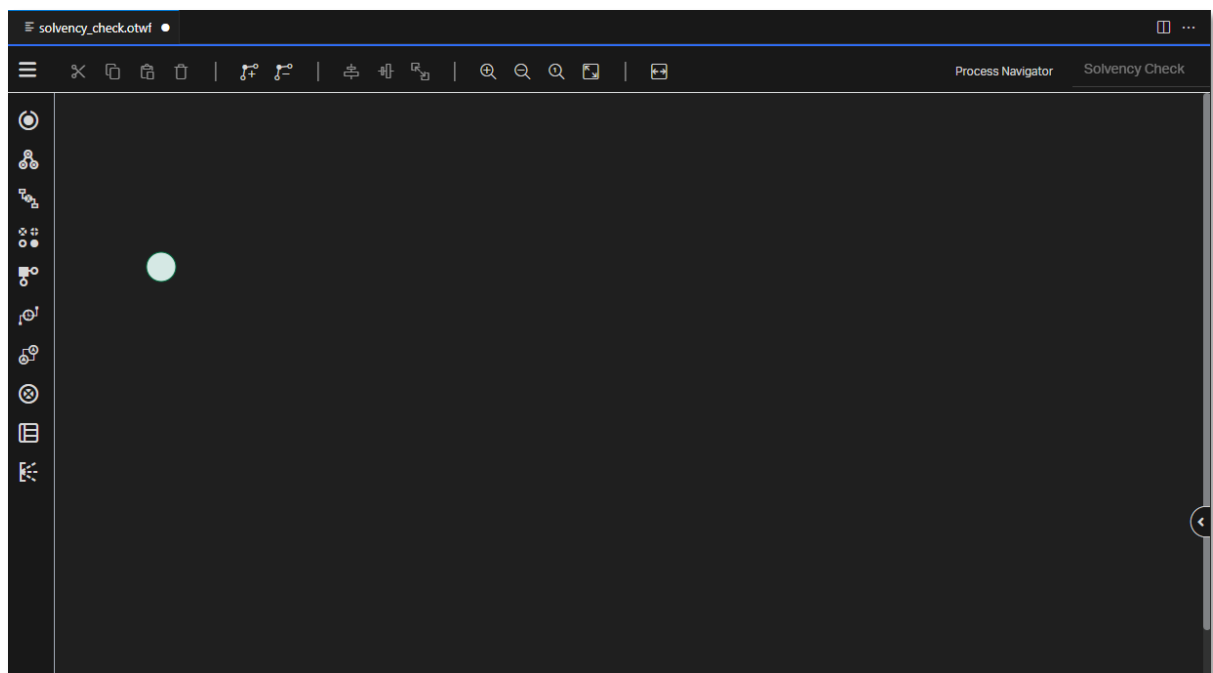
11.1 Create the Solvency Check workflow

For convenience, the creation of the Solvency Check workflow is split into several sub chapters:

- Create the workflow model
- Understand the workflow editor user interface
- Set the workflow attributes
- Build the process definition

11.1.1 Create the workflow model

1. In VS Code, create a new workflow model using any of the three model creation methods explained in the following exercises:
 - [Create the Contract Approval namespace](#)
 - [Create the Approval trait](#)
 - [Create the Contract type](#)
2. Type `solvency_check` for the (file) name of the workflow.



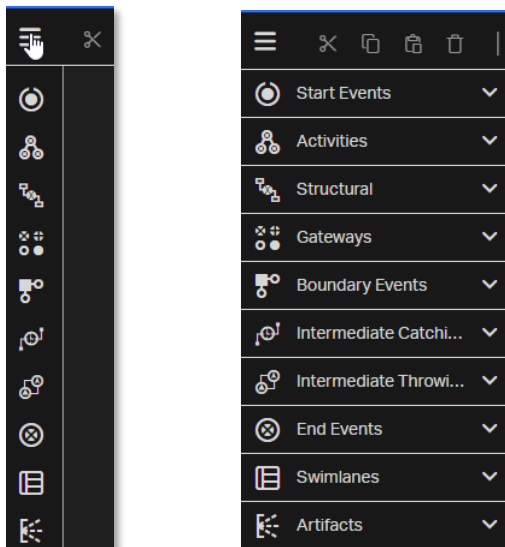
11.1.2 Understand the workflow editor user interface


1. In the workflow model editor, click the **Menu** button  to expand the palette.



Note

The workflow model editor is the most elaborate editor of the OpenText Thrust Studio extension pack. It is intended to build entire business process definitions in all its complexity.

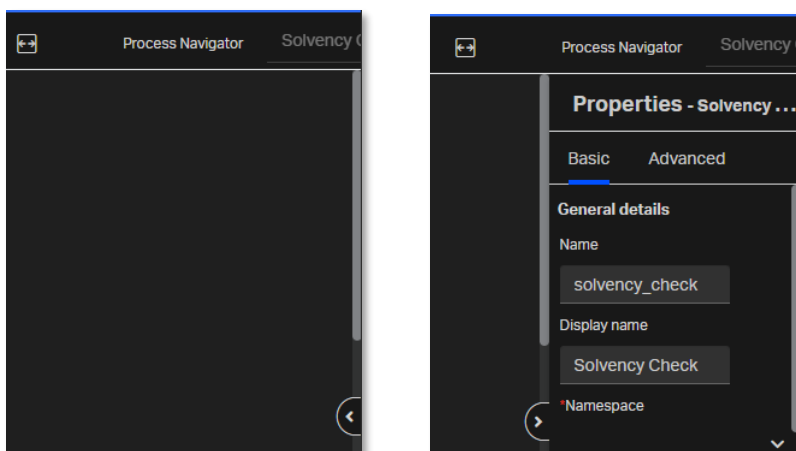


2. Click the  in the workflow model editor to expand the attribute bar.



Tip

Use the **Toggle side panels** button  to expand or collapse both side panes.




The following image illustrates the different user interface components of the workflow editor:



The following table explains the workflow editor user interface components from the preceding illustration:

Number	Component	Description
1	Menu bar	Contains buttons for the generic capabilities like copy, paste, delete, align, zoom, help, and other buttons.
2	Palette	Contains the different workflow elements which you can drag and drop on the canvas to build the workflow model.
3	Canvas	Area to build or draw the workflow model.
4	Attribute bar	Displays the attributes of the currently selected element. For example, in the preceding illustration, the canvas is selected and the attributes for the canvas are displayed.

11.1.3 Set the workflow attributes

1. Click the empty space in the **canvas** (that is, do not select the start event represented by .
2. Expand the attribute bar and fill the workflow attributes using the following details:

Attribute	Value
Name	The technical name for the workflow. This name must be unique within the context of the tenant and selected namespace. It is automatically populated based on the previously chosen workflow name (the model file name). Leave the value to be <code>solvency_check</code> .
Display name	The display name is the user-friendly name for the workflow. It is automatically populated based on the previously chosen workflow name (the model file name). Leave the value to be <code>Solvency Check</code> .
Namespace	The namespace to which this workflow belongs. Select the contract_approval namespace as the namespace.

Properties - solvency Check

Basic Advanced

General details

Name

Display name

*Namespace
 ▼

Start definitions

Signal definitions
+ Add

Message definitions
+ Add

Escalation definitions
+ Add

Context mapping

Data objects
+ Add

Execution details

Due date

Execution listeners
+ Add

**Note**

When building workflows, only those attributes that are relevant in context of the tutorial are explained. For more information on the different attributes of the different workflow elements, see [Workflow Modeler product documentation](#).

11.1.4 Build the process definition

1. Select the **Start Event**  that is already on the **canvas**.



Note

A start event indicates where a process begins. The type of start event defines how the process starts. The (standard) **start event** is used when a process instance is started through an API call (that is, there is no specific trigger).

2. Fill the start event attributes using the following details:

Attribute	Value
Display name	Enter <u>Start</u>

Properties - start

General details

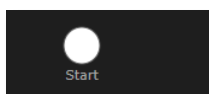
Id

Display name

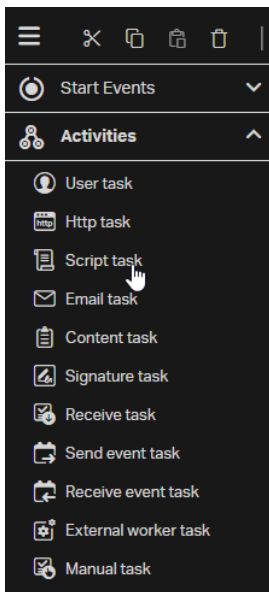
Initiator

Execution details

Execution listeners

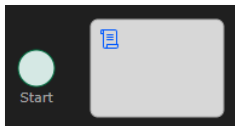


3. In the **palette**, find **Activities > Script task**.

**Note**

The script task is used to execute JavaScript scripts. Script tasks are mainly used to perform simple calculations or operations.

4. Drag and drop the **Script task** to the **canvas** next to the **Start event**.



5. Fill the Script task attributes using the following details:

Attribute	Value
Display name	Calculate solvency
Script	<p>Script that executes when the task executes.</p> <p>This task sets the solvent boolean (process) execution variable based on whether the monthlyBudget is larger or equal to the monthlyPayments.</p> <p>Enter the following script:</p> <pre>const contractDetails = JSON.parse(execution.getVariable("contract")); const monthlyPayments = contractDetails.properties.value / contractDetails.properties.monthly_installments; const monthlyBudget = contractDetails.properties.yearly_income / 12 / 4; execution.setVariable("solvent", monthlyBudget >= monthlyPayments);</pre>

Properties - Calculate solvency

Basic

Advanced

General details

Id

Enter value

Display name

Calculate solvency

Script details

*Script ⓘ

const contractDetails = JSON.p...

Persist script variables

☐

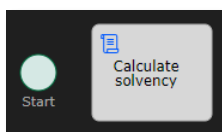
×

Execution details

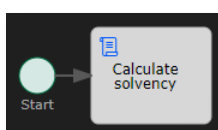
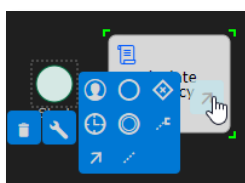
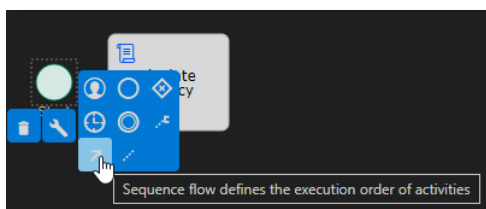
Execution listeners

+

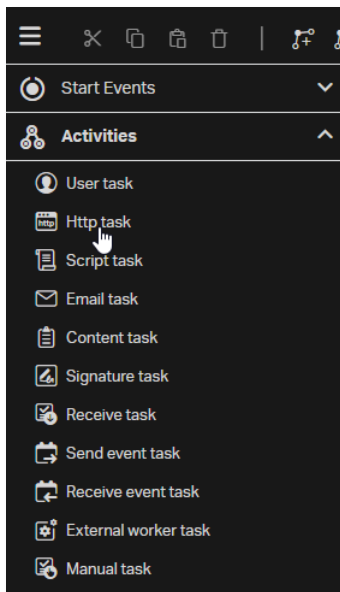
 Add



- Select the **Start** event and drag the **sequence flow** (arrow connector) to the **Calculate solvency** script task.



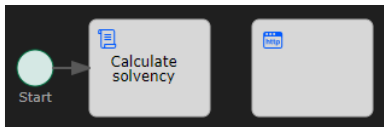
7. In the palette, find Activities > Http task.



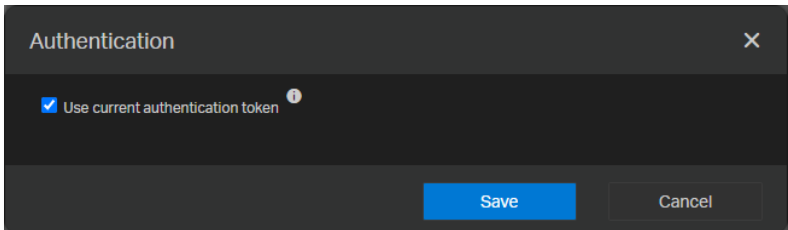
Note

The HTTP task allows to submit and store the result of an HTTP call.

8. Drag and drop the **Http task** from the **palette** to the **canvas** next to the **Calculate solvency script task**.



9. Fill the Http task attributes using the following details:

Attribute	Value
Display name	Update Solvency Check trait
Authentication	<p>How to authenticate with the REST API.</p> <p>For the Http tasks in this workflow, you will use the authentication credentials of the user performing the task.</p> <p>Select Use current authentication token and click Save to confirm.</p> 

Attribute	Value
Request method	<p>The request method to use in the HTTP call: GET, POST, PUT, DELETE or PATCH.</p> <p>This task updates the solvency check trait of the given contract, so this implies using the PATCH method.</p> <p>Select the PATCH request method.</p>
Request URL	<p>The request URL of the HTTP call. The request URL can contain expressions, such as <code>\${some_variable}</code>.</p> <p>To perform the request against the correct API base URL and to ensure updating the contract that was passed to the workflow through the contract_id variable, the request URL includes <code>\${base_url}</code> and <code>\${contract_id}</code>.</p> <p>Enter the following request URL:</p> <pre><code>\${base_url}/cms/instances/file/ca_contract/\${contract_id}</code></pre>
Request headers	<p>The line-separated HTTP request headers.</p> <p>Enter the following request headers:</p> <pre><code>Content-Type: application/json</code></pre>
Request body	<p>The request body to send. For example, a JSON file. Like the request URL, the request body can also contain expressions.</p> <p>This task sets the is_required, has_been_granted, approver, approver_role, and approval_date properties for the Solvency Check ca_approval trait of the contract.</p> <p>Enter the following request body:</p> <pre><code>{ "traits": { "ca_approval": { "Solvency Check": { "is_required": true, "has_been_granted": \${solvent}, "approver": "SYSTEM", "approver_role": "Solvency Check", "approval_date": "\${contract.update_time}" } } } }</code></pre>
Response variable name	<p>The variable name in which the HTTP response is stored.</p> <p>Enter the following response variable name:</p> <pre><code>contract</code></pre>
Save response as JSON	<p>Whether the response variable is stored as a JSON variable instead of a string.</p> <p>Choose to save the response as JSON.</p>

Properties - Update Solvency Check trait

Basic

Advanced

General details

Id

Enter value

Display name

Update Solvency Check trait

REST service details

Authentication

Authentication configured

*Request method

PATCH▼

*Request URL

\${base_url}/cms/instances/file/...

Request headers

Content-Type: application/json

Request body

{ "traits": { "ca_approval": { "...

Response variable name

contract

Save response as JSON

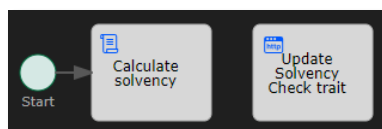
☒

☐

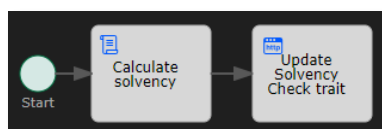
Save response as a transient variable

☐

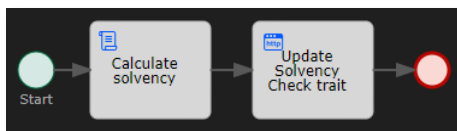
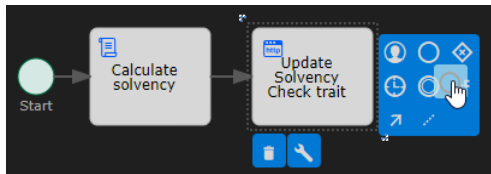
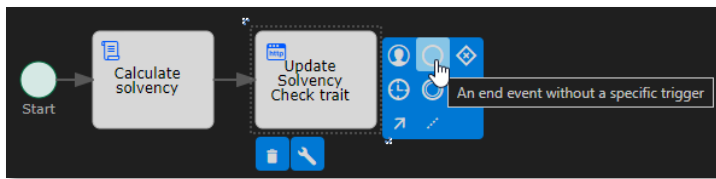
☒



10. Select the **Calculate solvency script task** and drag a **sequence flow** to the **Update Solvency Check trait http task**.



11. Select the **Update Solvency Check trait http task** and drag and drop an **End event** next to it.



12. Select the **End event**.



Note

An end event signifies the end of a path in a process or sub-process. When the process execution arrives at an end event, a result is always thrown. The type of end event defines the type of result that is thrown. The (standard) **end event** means that an unspecified result is thrown upon reaching it. As such, the business process engine will not perform anything besides ending the current path of execution.

13. Fill the End event attributes using the following details:

Attribute	Value
Display name	End

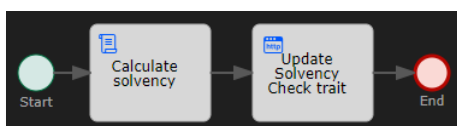
Properties - End

General details

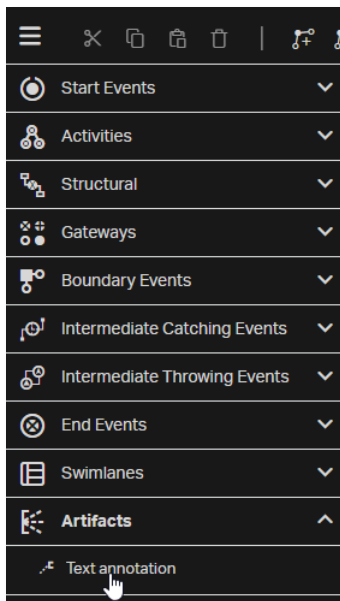
Id
Enter value

Display name
End

Execution details



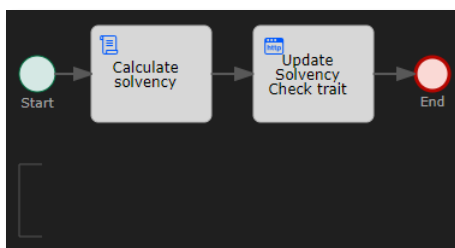
14. In the **palette**, find **Artifacts > Text annotation**.



Note

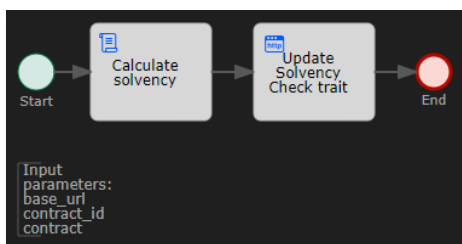
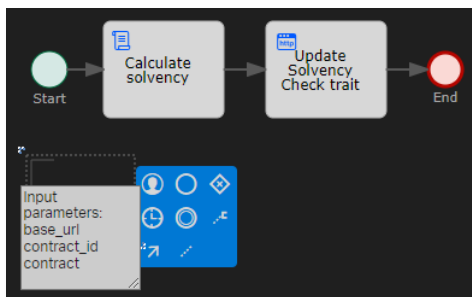
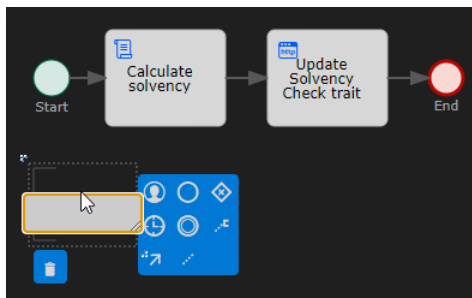
The text annotation allows you to describe the business process and flow objects in more detail. Add annotations to make your BPMN process more readable and further increase understanding of your process.

15. Drag and drop the **Text annotation** to the **canvas** under the **Start event**.

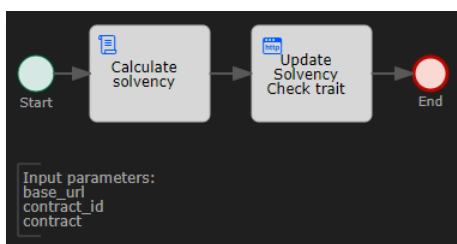


16. Double-click the **Text annotation** on the **canvas** to edit it inline.
17. Set the following text for the text annotation:

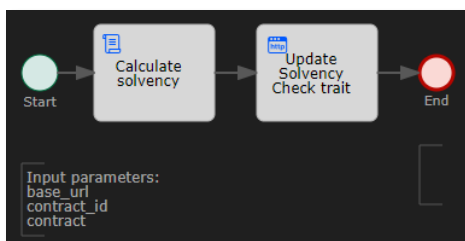
Text
Input parameters: base_url contract_id contract



18. Resize the **text annotation** so that the text displays correctly and fits the square brackets.



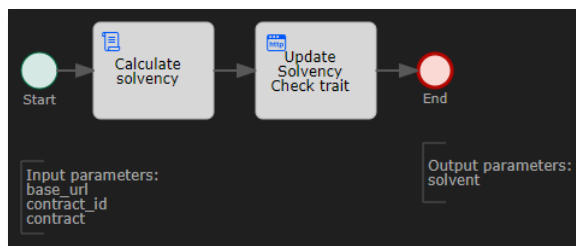
19. Drag and drop another **Text annotation** from the **palette** to the **canvas** under the **End event**.



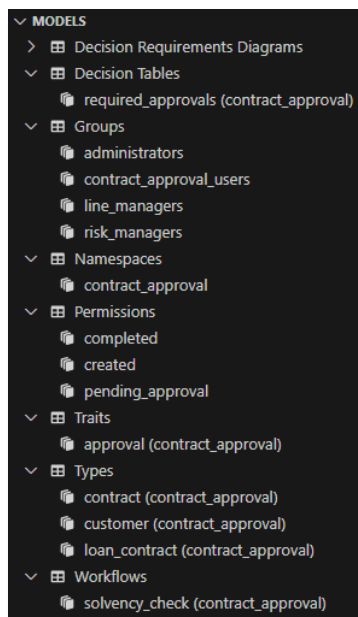
20. Double-click the **Text annotation** on the **canvas** to edit it inline and set the following text for the text annotation:

Text
Output parameters: solvent

21. Resize the **Text annotation** so that the text displays correctly and fits the square brackets.



22. Save and close the Solvency Check workflow model.
23. The model explorer displays the new **solvency_check (contract_approval)** workflow under **Workflows**. The model explorer shows the different models according to their unique key, which in context of a workflow is the combination of the namespace and name attributes.



Next step:

Create the Manager Approval workflow.

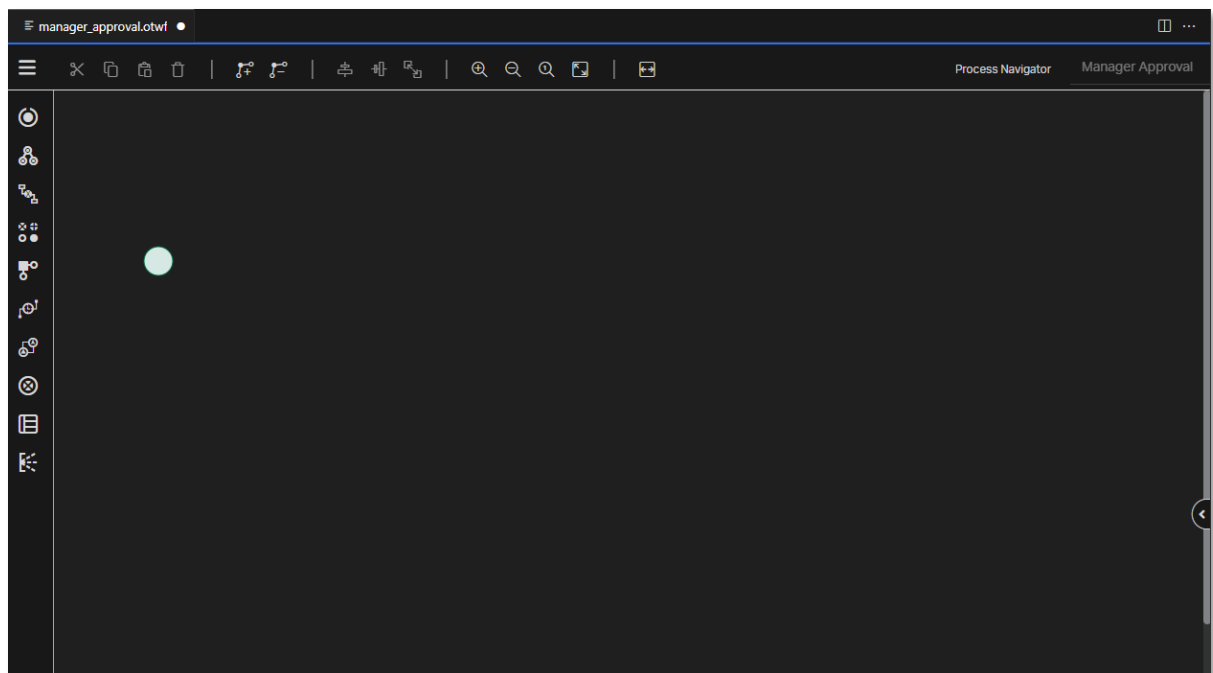
11.2 Create the Manager Approval workflow

For convenience, the creation of the Manager Approval workflow is split into sub chapters:

- Create the workflow model
- Set the workflow attributes
- Build the process definition

11.2.1 Create the workflow model

1. In VS Code, create a new workflow model using any of the three model creation methods explained in the following exercises:
 - [Create the Contract Approval namespace](#)
 - [Create the Approval trait](#)
 - [Create the Contract type](#)
2. Type `manager_approval` for the (file) name of the workflow.



11.2.2 Set the workflow attributes

1. Click the empty space in the **canvas** (that is, do not select the start event).
2. Expand the attribute bar and fill the workflow attributes using the following details:

Attribute	Value
Name	manager_approval
Display name	Manager Approval
Namespace	Select the contract_approval namespace.

Properties - Manager Approval

Basic Advanced

General details

Name
manager_approval

Display name
Manager Approval

*Namespace
contract_approval ▼

Start definitions

Signal definitions
+ Add

Message definitions
+ Add

Escalation definitions
+ Add

Context mapping

Data objects
+ Add

Execution details

Due date
Enter value

Execution listeners
+ Add

11.2.3 Build the process definition

1. Select the **Start event** that is already on the **canvas**.
2. Fill the start event attributes using the following details:

Attribute	Value
Display name	Start

Properties - start

General details


Id

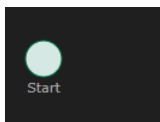
Display name

Initiator

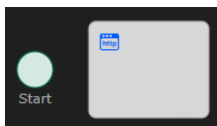
Execution details

Execution listeners

 Add



3. Drag and drop an **Http task** to the **canvas** next to the **Start event**.



4. Fill the Http task attributes using the following details:

Attribute	Value
Display name	Update Manager Approval trait
Authentication	Select Use current authentication token .
Request method	Select the PATCH request method.
Request URL	<code>\${base_url}/cms/instances/file/ca_contract/\${contract_id}</code>
Request headers	Content-Type: application/json
Request body	<p>This task sets the is_required property to true for the contract approval trait specified by \${approval_role} Approval (the concatenation of the approver_role process input parameter and “Approved”).</p> <p>Enter the following request body:</p> <pre>{ "traits": { "ca_approval": { "\${approver_role} Approval": { "is_required": true } } } }</pre>
Response variable name	contract
Save response as JSON	Choose to save the response as JSON.

Properties - Update Manager Approval trait

Basic Advanced

General details

Id
Enter value

Display name
Update Manager Approval trait

REST service details

Authentication
Authentication configured

*Request method
PATCH

*Request URL
\${base_url}/cms/instances/file/ca_...

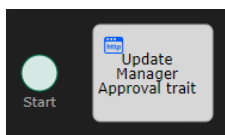
Request headers
Content-Type: application/json

Request body
{ "traits": { "ca_approval": { "\${ap...

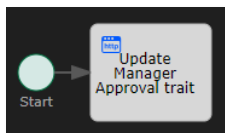
Response variable name
contract

Save response as JSON
☒

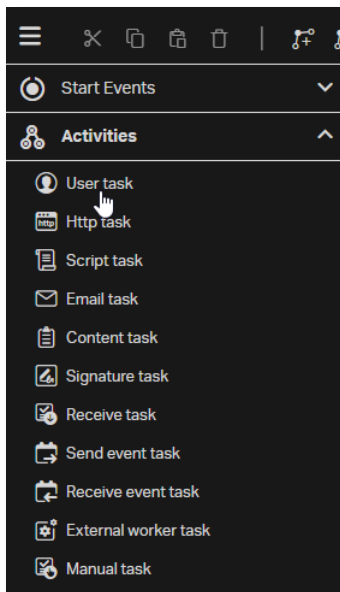
Save response as a transient variable
☐



5. Select the **Start event** and drag a **sequence flow** to the **Update Manager Approval trait http task**.



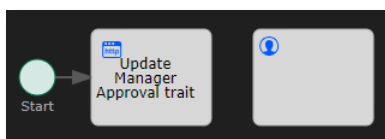
6. In the **palette**, find **Activities > User task**.



Note


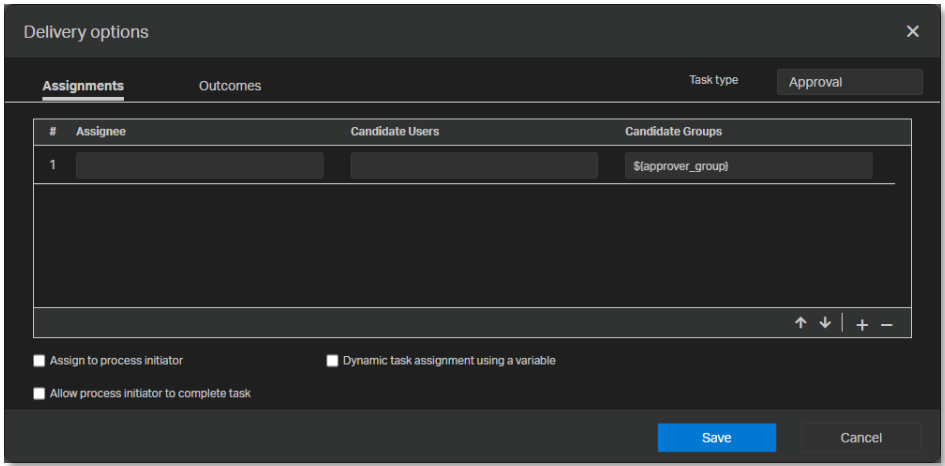
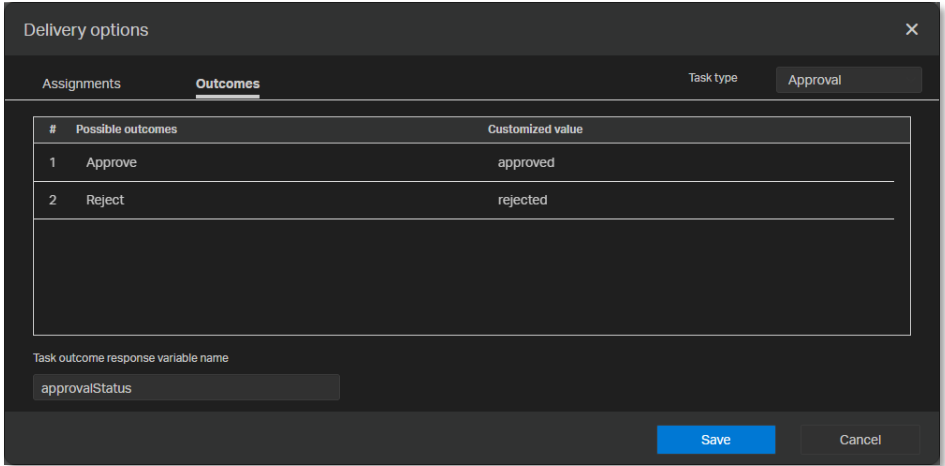
The user task is a typical workflow task where a user performs the task with the assistance of a software application. It is scheduled through a task list manager. In a workflow, user tasks are the primary way to interact with humans within a process. After the execution reaches such a task, a user is required to perform an action. As an outcome of the action, it is possible to create and update variables to use in other tasks or to control the flow of the process. Each task can be assigned to one or more users and shared with any number of groups. A task can optionally have a due date.

7. Drag and drop the **User task** to the **canvas** next to the **Update Manager Approval trait http task**.



8. Fill the User task attributes using the following details:

Attribute	Value
Display name	<p>In the manager_approval workflow, the display name of the manual approval task which signals the type of approval (that is, who is doing the approval, the line manager or the risk manager) is computed by concatenating the approver_role process input parameter and “Approved”.</p> <p>Enter the following display name:</p> <pre>\${approver_role} Approval</pre>

Attribute	Value
Delivery options	<p>Complete the user task behavior configuration using the following options:</p> <ul style="list-style-type: none"> Task type: The type of user task Assignments: How the user task gets assigned Outcomes: What to pass on (in a process variable) as result of the user action <p>This user task is an approval task to assign to one of the members of the approver group (that is, line managers or risk managers) which is passed as the approver_group process input parameter. The two possible user task outcomes are approval or rejection of the contract.</p> <p>Set the Task type to Approval.</p> <p>Under the Assignments tab, click the Add button  and enter the following for the Candidate Groups:</p> <pre>{approver_group}</pre>  <p>Select the Outcomes tab and change the Customized value entries to approved and rejected, set the Task outcome response variable name to approvalStatus, and click Save to confirm the delivery options.</p> 

Properties - \${approver_role} Approval

Basic Advanced

General details

Id

Display name

Task details

*Delivery options
 Configured

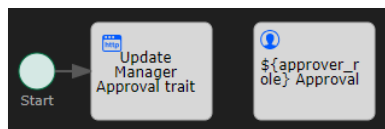
Priority

Execution details

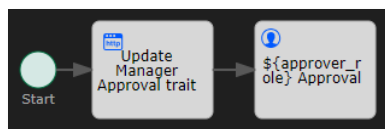
Due date

Task listeners

Execution listeners



9. Select the **Update Manager Approval trait http task** and drag a **sequence flow** to the **User task**.



10. Drag and drop another **Http task** from the **palette** to the **canvas** next to the **User task**.



11. Fill the Http task attributes using the following details:

Attribute	Value
Display name	Update Manager Approval trait
Authentication	Select Use current authentication token .

Attribute	Value
Request method	Select the PATCH request method.
Request URL	<code>\${base_url}/cms/instances/file/ca_contract/\${contract_id}</code>
Request headers	<code>Content-Type: application/json</code>
Request body	<p>This task sets the is_required, has_been_granted, approver, approver_role, and approval_date properties for the contract approval trait, specified by \${approver_role} Approver.</p> <p>Enter the following request body:</p> <pre> { "traits": { "ca_approval": { "\${approver_role} Approval": { "is_required": true, "has_been_granted": \${approvalStatus == "approved"}, "approver": "\${approver}", "approver_role": "\${approver_role}", "approval_date": "\${contract.update_time}" } } } } </pre>
Response variable name	<code>contract</code>
Save response as JSON	Choose to save the response as JSON.

Properties - Update Manager Approval trait

Basic Advanced

General details

Id
Enter value

Display name
Update Manager Approval trait

REST service details

Authentication
Authentication configured

*Request method
PATCH

*Request URL
\${base_url}/cms/instances/file/ca_...

Request headers
Content-Type: application/json

Request body
{ "traits": { "ca_approval": { "\${ap...

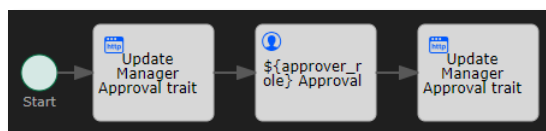
Response variable name
contract

Save response as JSON
☒

Save response as a transient variable
☐



12. Select the **User task** and drag a **sequence flow** to the second **Update Manager Approval trait http task**.



13. Select the second **Update Manager Approval trait http task** and drag and drop an **End event** next to it.



14. Fill the end event attributes using the following details:

Attribute	Value
Display name	End

Properties - End

General details

Id
Enter value

Display name
End

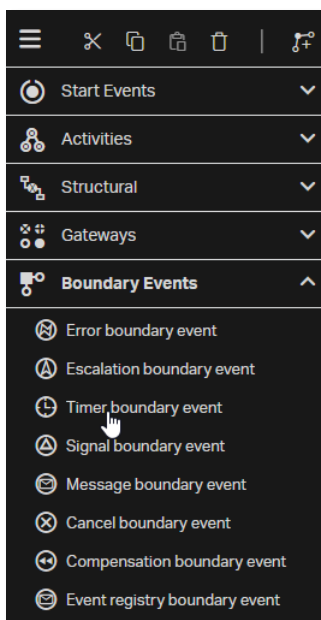
Execution details

Execution listeners

+ Add



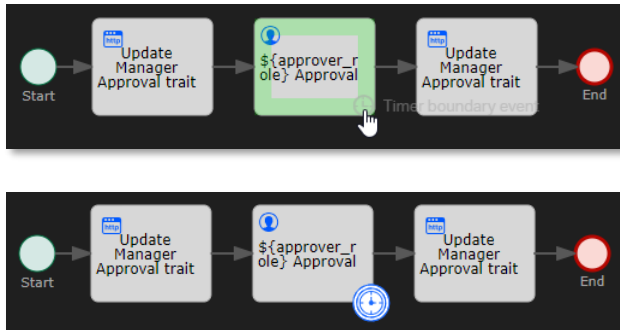
15. In the **palette**, find **Boundary Events > Timer boundary event**.




Note

The timer boundary event acts as a stopwatch and alarm clock. When an execution arrives at the activity where the boundary event is attached, a timer starts. When the timer fires (for example, after a specified interval), the activity is interrupted, and the outgoing sequence flow of the boundary event is followed.

16. Drag and drop the **Timer boundary event** on top of the **User task**'s bottom corner that is nearest to the **End event**. The **User task** changes to green to indicate when the timer boundary event can be released/dropped to be correctly linked.



17. Select the **Timer boundary event** and fill its attributes using the following details:

Attribute	Value
Time duration (e.g. PT5M)	<p>Specifies how long the timer must run before it is triggered. The ISO 8601 format is used as required by the BPMN 2.0 specification.</p> <div>  Note <p>There are three different ways to configure the timer:</p> <ul style="list-style-type: none"> Time cycle: Specifies (ISO 8601 format) a repeating interval for starting the process periodically or for sending multiple reminders. Time date: Specifies a fixed date (ISO 8601 format) when the trigger will fire. Time duration: Specifies (ISO 8601 format) how long the timer must run before it is fired. <p>For the timer boundary event, it is required to fill one of the three timer configuration properties.</p> <p>The manager approval task must expire (that is, the timer event must fire) after 5 minutes.</p> <p>Enter the following time duration:</p> <p>PT5M</p> </div>

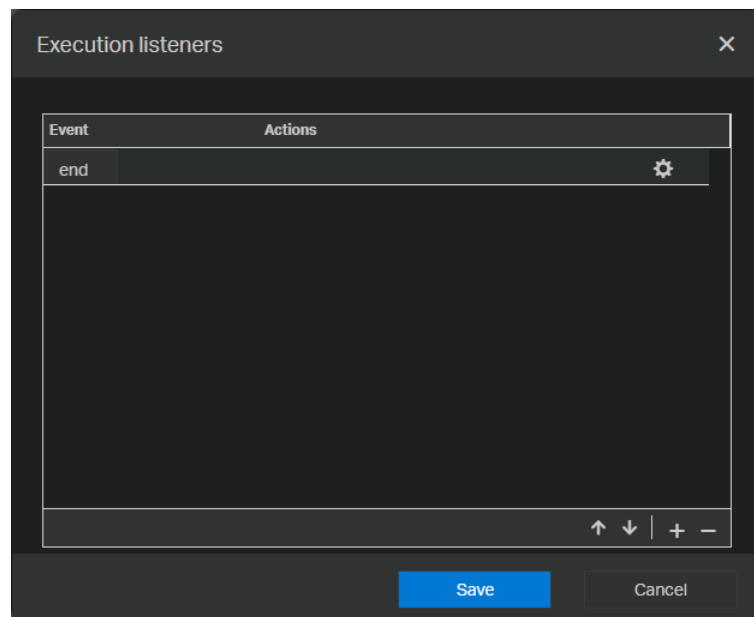
Execution listeners

Active execution listeners of the activity. Allows configuring an action when the **start**, **take**, or **end** events occur for the boundary time event.

For the manager approval task, the **approvalStatus** process variable must be set to **expired** when the timer fires.

Configure the Execution listeners as follows:

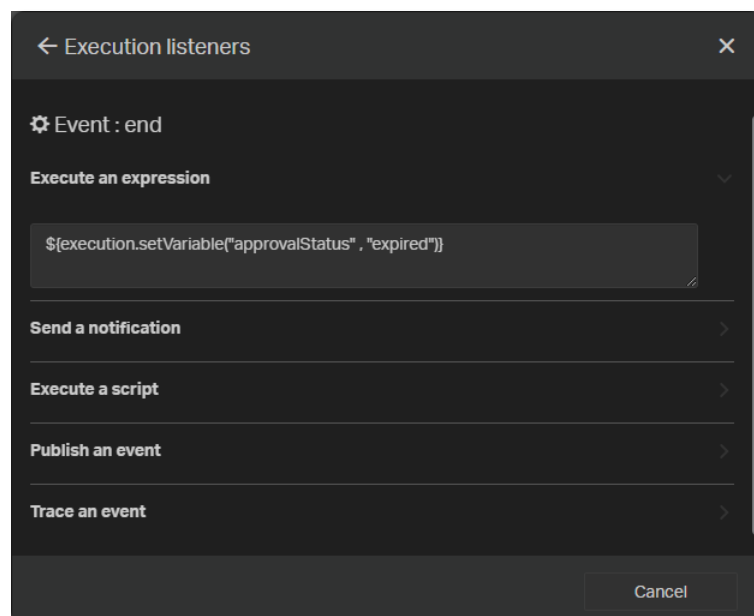
Click the **Add** button  and select the **end** event.




Click the **Configure listener definition** button  to configure the action and select **Execute an expression**.

In **Execute an expression** box, enter the following expression:

```
${execution.setVariable("approvalStatus" , "expired")}
```



Click the **Switch to previous view** button  to go back and click **Save** to confirm the execution listeners.

Properties

Basic Advanced

General details

Id
Enter value

Display name
Enter value

Timer details

* At least one input is required.

Date ⓘ
Enter value

End Date ⓘ
Enter value

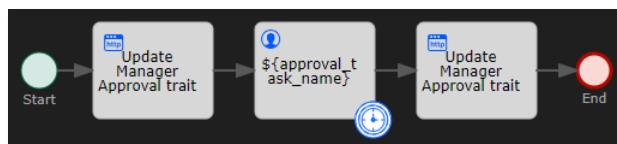
Time cycle ⓘ
Enter value

Time duration ⓘ
PT5M

Execution details


Execution listeners

1 execution listeners



18. Select the **Timer boundary event** and drag a **sequence flow** to the **End event**.

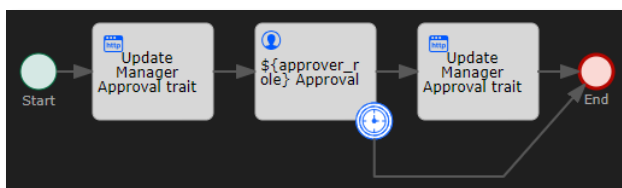
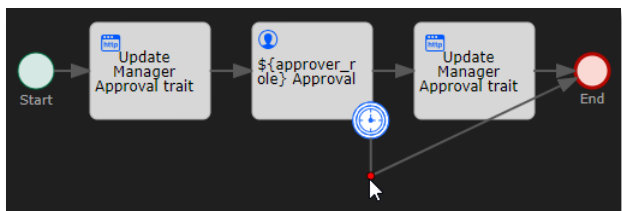
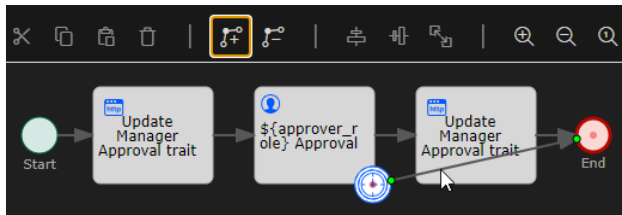
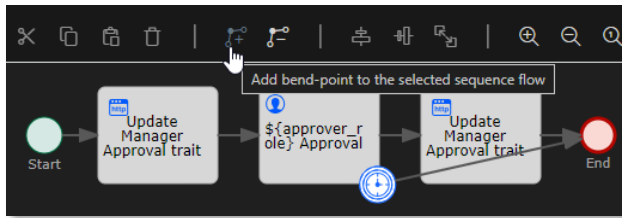


19. Click the **Sequence flow bend-point** button  to add two **bend-points** to the **sequence flow** between the **Timer boundary event** and the **End event**.

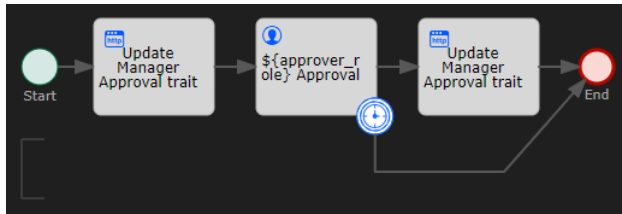


Note

Bend-points allow you to change the visual path of a sequence flow to display the BPMN process more clearly. Add bend-points where needed to make your BPMN process more readable.



20. Drag and drop a **Text annotation** from the **palette** to the **canvas** under the **start event**.



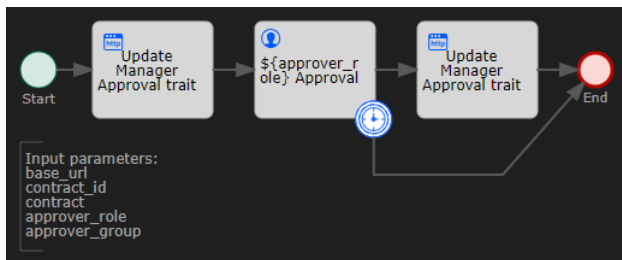
21. Double-click the **Text annotation** on the **canvas** to edit it inline.

22. Set the following text for the text annotation:

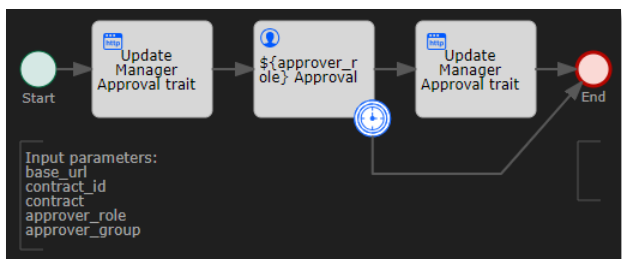
Text

```
Input parameters:
base_url
contract_id
contract
approver_role
approver_group
```

23. Resize the **text annotation** so that the text displays correctly and fits the square brackets.



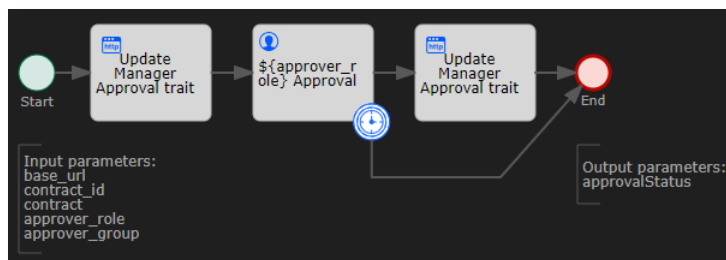
24. Drag and drop another **Text annotation** from the **palette** to the **canvas** under the **End event**.



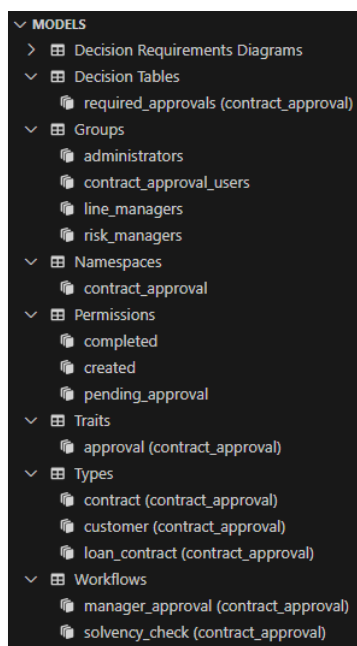
25. Double-click the **Text annotation** on the **canvas** to edit it inline and set the following text for the text annotation:

Text
Output parameters: approvalStatus

26. Resize the **Text annotation** so that the text displays correctly and fits the square brackets.



27. Save and close the Manager Approval workflow model. The model explorer displays the new **manager_approval (contract_approval)** workflow under **Workflows**.



Next step:

Create the Contract Approval workflow.

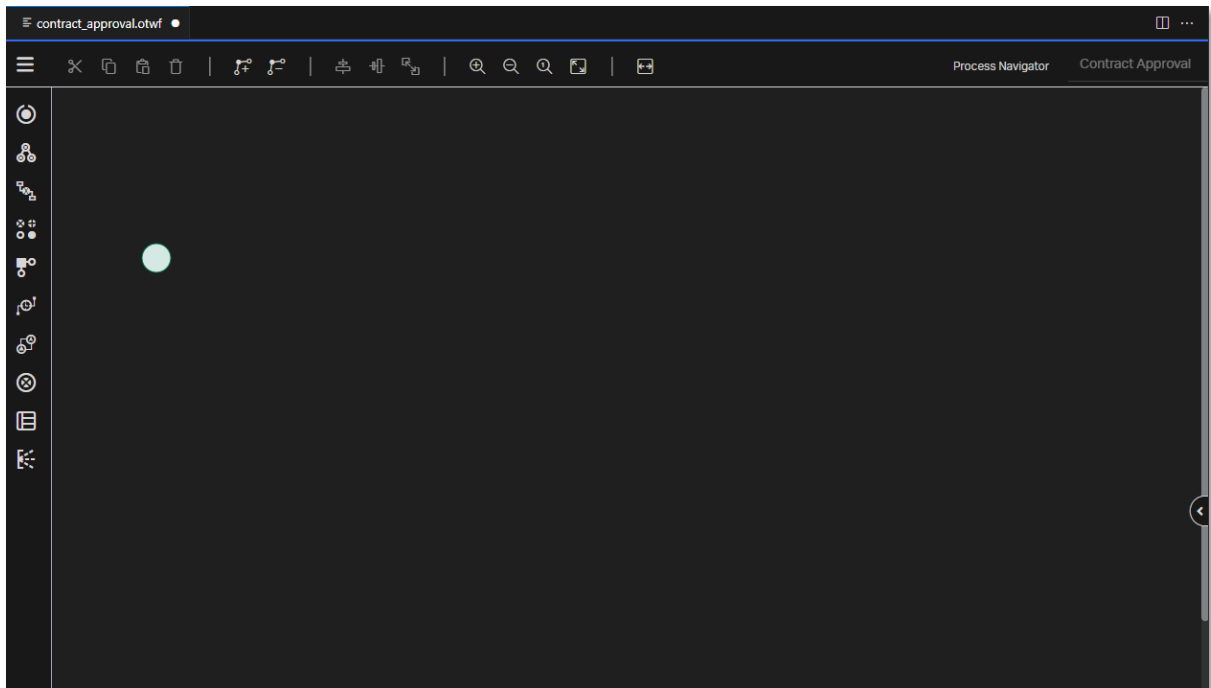
11.3 Create the Contract Approval workflow

For convenience, the creation of the Contract Approval workflow is split into several sub chapters:

- Create the workflow model
- Set the workflow attributes
- Build the process definition

11.3.1 Create the workflow model

1. In VS Code, create a new workflow model using any of the three model creation methods explained in the following exercises:
 - [Create the Contract Approval namespace](#)
 - [Create the Approval trait](#)
 - [Create the Contract type](#)
2. Type `contract_approval` for the (file) name of the workflow.



11.3.2 Set the workflow attributes

1. Click the empty space in the **canvas** (that is, do not select the start event).
2. Expand the attribute bar and fill the workflow attributes using the following details:

Attribute	Value
Name	contract_approval
Display name	Contract Approval
Namespace	Select the contract_approval namespace.

Properties - Contract Approval

Basic Advanced

General details

Name

Display name

*Namespace
 ▼

Start definitions

Signal definitions
+ Add

Message definitions
+ Add

Escalation definitions
+ Add

Context mapping

Data objects
+ Add

Execution details

Due date

Execution listeners
+ Add

11.3.3 Build the process definition

1. Select the **Start event** that is already on the **canvas**.
2. Fill the Start event attributes using the following details:

Attribute	Value
Display name	Start

Properties - start

General details

Id
Enter value

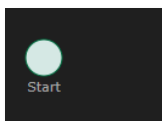
Display name
Start

Initiator
Enter value

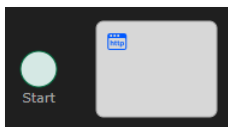
Execution details

Execution listeners

+ Add



3. Drag and drop an **Http task** to the **canvas** next to the **Start event**.



4. Fill the Http task attributes using the following details:

Attribute	Value
Display name	Get contract from CMS
Authentication	Select Use current authentication token .
Request method	This task fetches the contract metadata from CMS, so this implies using the GET method. Select the GET request method.
Request URL	<code>\${base_url}/cms/instances/file/ca_contract/\${contract_id}</code>
Request headers	Content-Type: application/json
Response variable name	contract

Attribute	Value
Save response as JSON	Choose to save the response as JSON.

Properties - Get contract from CMS

Basic Advanced

General details

Id
Enter value

Display name
Get contract from CMS

REST service details

Authentication
Authentication configured

*Request method
GET

*Request URL
\${base_url}/cms/instances/file/...

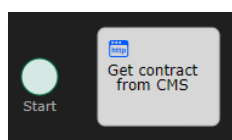
Request headers
Content-Type: application/json

Request body
Enter value

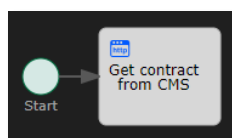
Response variable name
contract

Save response as JSON
☒

Save response as a transient variable
☐



5. Select the **Start event** and drag a **sequence flow** to the **Get contract from CMS http task**.



6. Drag and drop a second **Http task** from the **palette** to the **canvas** next to the **Get contract from CMS http task**.



7. Fill the Http task attributes using the following details:

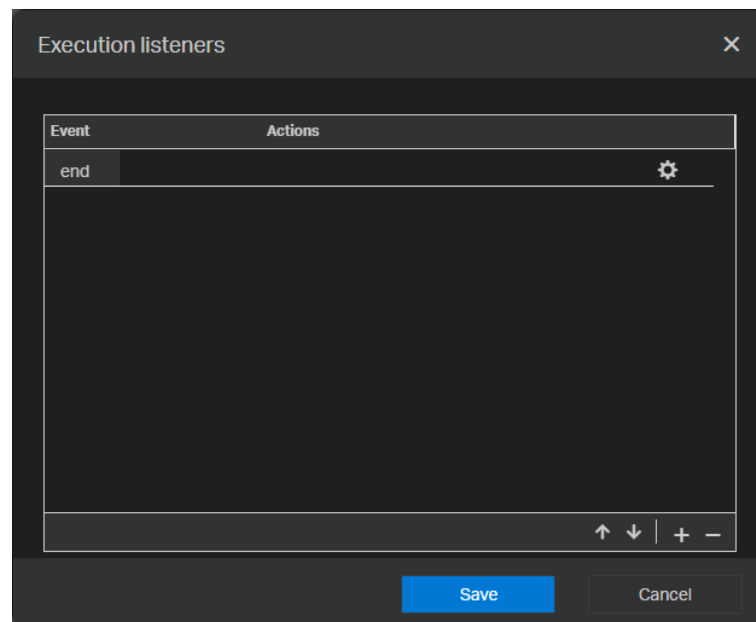
Attribute	Value
Display name	Get required approvals
Authentication	Select Use current authentication token .
Request method	This task calls the Decision Service to execute the decision table that calculates the required approvals. This implies using the POST method. Select the POST request method.
Request URL	Enter the following request URL to call the Decision Service : <code>\${base_url}/decision/v1/execute</code>
Request headers	Content-Type: application/json
Request body	This task executes the required_approvals decision table with the contract_type , contract_value , and risk_level contract property values as input variables. Enter the following request body: <pre>{ "decisionKey": "required_approvals", "inputVariables": [{ "name": "contract_type", "value": "\${contract.type}", "type": "string" }, { "name": "contract_value", "value": "\${contract.properties.value}", "type": "integer" }, { "name": "risk_level", "value": "\${contract.properties.risk_classification}", "type": "integer" }] }</pre>
Response variable name	required_approvals
Save response as JSON	Choose to save the response as JSON.

Execution listeners

Active execution listeners of the activity. Allows configuring an action when the **start**, **take**, or **end** events occur for the boundary time event.

For the Get required approvals http task the **solvency_check_required**, **line_manager_approval_required**, and **risk_manager_approval_required** process variables must be set according to the **required_approvals** result from the Decision Service call.

Click the **Add** button  and select the **End** event.



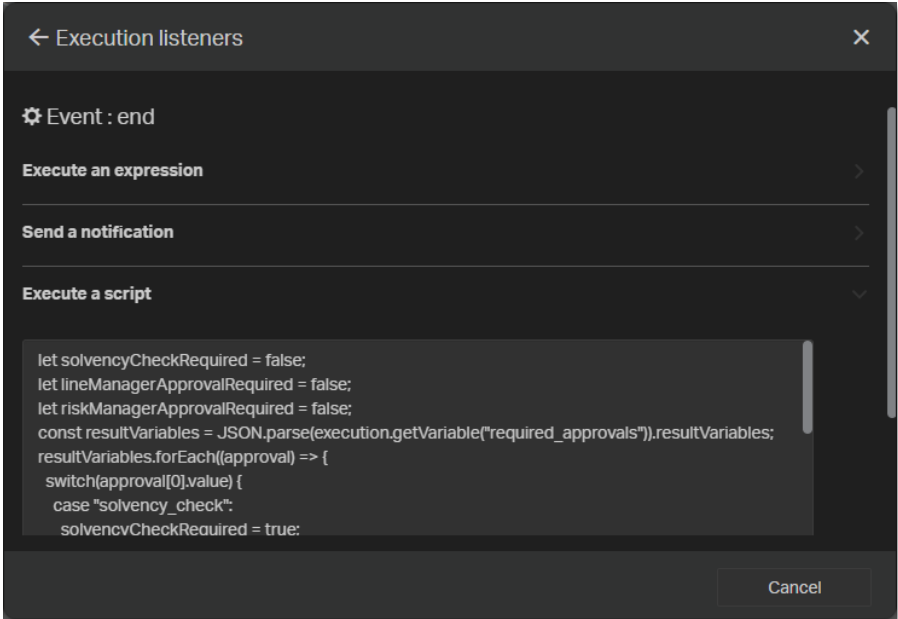
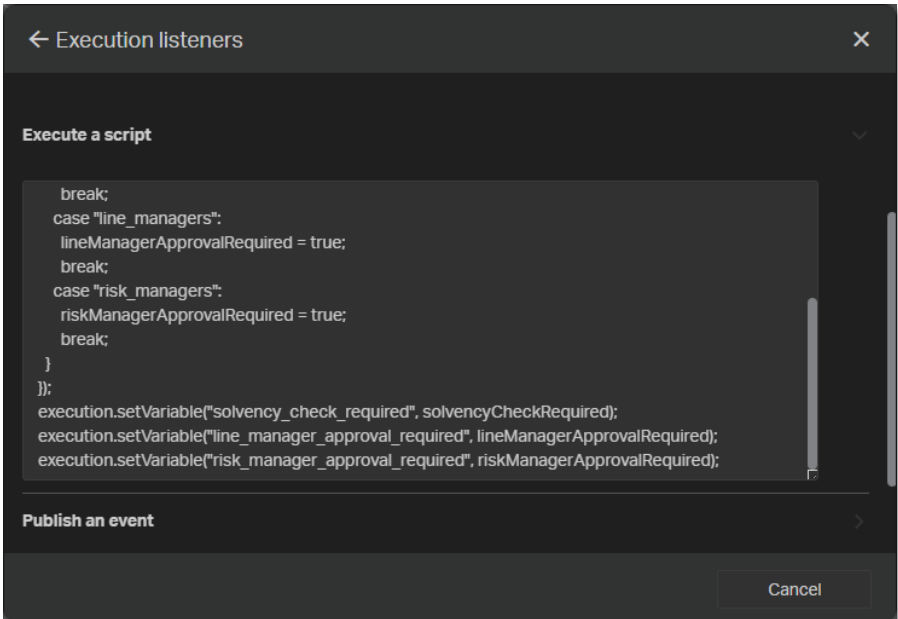

Click the **Configure listener definition** button  to configure the action and select **Execute a script**.

Expand the script input box (for readability) and enter the following script:

```
let solvencyCheckRequired = false;
let lineManagerApprovalRequired = false;
let riskManagerApprovalRequired = false;

const resultVariables = JSON.parse(
  execution.getVariable("required_approvals")
).resultVariables;
resultVariables.forEach((approval) => {
  switch(approval[0].value) {
    case "solvency_check":
      solvencyCheckRequired = true;
      break;
    case "line_managers":
      lineManagerApprovalRequired = true;
      break;
    case "risk_managers":
      riskManagerApprovalRequired = true;
      break;
  }
});

execution.setVariable("solvency_check_required",
  solvencyCheckRequired);
```

Attribute	Value
	<pre>execution.setVariable("line_manager_approval_required", lineManagerApprovalRequired); execution.setVariable("risk_manager_approval_required", riskManagerApprovalRequired);</pre>   <p>Click the Switch to previous view button  to go back and click Save to confirm the execution listeners.</p>

Properties - Get required approvals

Basic Advanced

General details

Id
Enter value

Display name
Get required approvals

REST service details

Authentication
Authentication configured

*Request method
POST

*Request URL
\${base_url}/decision/v1/ex...

Request headers
Content-Type: application/j...

Request body
{ "decisionKey": "required_...

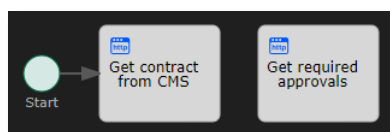
Response variable name
required_approvals

Save response as JSON
☒

Save response as a transient variable
☐

Execution details

Execution listeners
1 execution listeners



8. Select the Get contract from CMS http task and drag a sequence flow to the Get required approvals http task.



9. Drag and drop a third **Http task** from the **palette** to the **canvas** next to the **Get required approvals http task**.



10. Fill the Http task attributes using the following details:

Attribute	Value
Display name	Set contract status to PENDING APPROVAL
Authentication	Select Use current authentication token .
Request method	This task sets the value for the status attribute of the given contract, so this implies using the PATCH method. Select the PATCH request method.
Request URL	<code>\${base_url}/cms/instances/file/ca_contract/\${contract_id}</code>
Request headers	Content-Type: application/json
Request body	This task sets the contract's status property to PENDING APPROVAL . Enter the following request body: <pre>{ "properties": { "status": "PENDING APPROVAL" } }</pre>
Response variable name	contract
Save response as JSON	Choose to save the response as JSON.

Properties - Set contract status to PENDIN...

Basic Advanced

General details

Id
Enter value

Display name
Set contract status to PENDIN...

REST service details

Authentication
Authentication configured

*Request method
PATCH

*Request URL
\${base_url}/cms/instances/file/...

Request headers
Content-Type: application/json

Request body
{ "properties": { "status": "PEN...

Response variable name
contract

Save response as JSON
☒

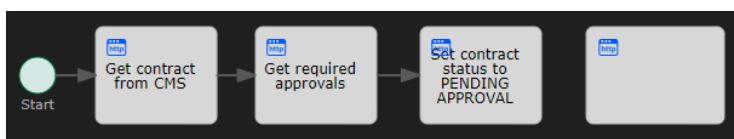
Save response as a transient variable
☐



11. Select the **Get required approvals http task** and drag a **sequence flow** to the **Set contract status to PENDING APPROVAL http task**.



12. Drag and drop a fourth **Http task** from the **palette** to the **canvas** next to the **Set contract status to PENDING APPROVAL** http task.



13. Fill the Http task attributes using the following details:

Attribute	Value
Display name	Set ACL to pending_approval
Authentication	Select Use current authentication token .
Request method	This task sets the ACL of the given contract. Setting the ACL for a file instance requires calling a separate (acl) file instance endpoint to replace the current ACL resource, which implies using the PUT method. Select the PUT request method.
Request URL	Enter the following request URL to call the acl endpoint for the given contract: <code>\${base_url}/cms/instances/file/ca_contract/\${contract_id}/acl</code>
Request headers	<code>Content-Type: application/json</code>
Request body	This task sets the ACL of the contract by using the pending_approval_acl_id process input parameter as the id of the ACL to apply. Enter the following request body: <pre>{ "id": "\${pending_approval_acl_id}" }</pre>
Save response as JSON	Choose to save the response as JSON.

Properties - Set ACL to pending_approval

Basic

Advanced

General details

Id

Enter value

Display name

Set ACL to pending_approval

REST service details

Authentication

Authentication configured

*Request method

PUT

*Request URL

{base_url}/cms/instances/file/...

Request headers

Content-Type: application/json

Request body

{ "id": "\${pending_approval_acl_...}

Response variable name

Enter value

Save response as JSON

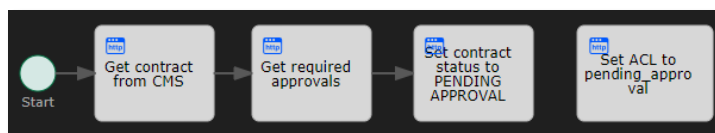
✓

●

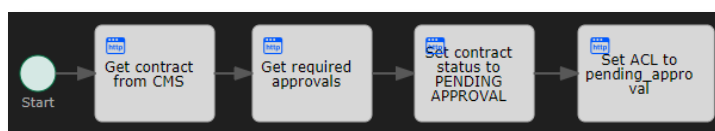
Save response as a transient variable

○

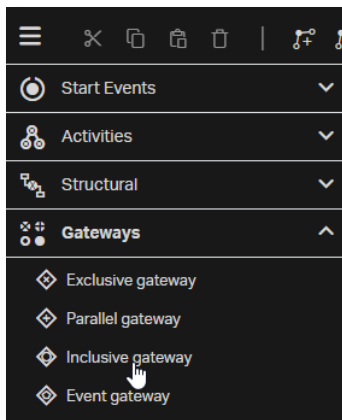
×



14. Select the **Set contract status to PENDING APPROVAL** http task and drag a **sequence flow** to the **Set ACL to pending_approval** http task.



15. In the **palette**, find **Gateways > Inclusive gateway**.



Note

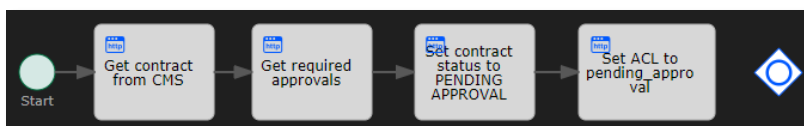
A gateway controls the flow of execution. A gateway is capable of consuming or generating tokens.

There are four types of gateways:

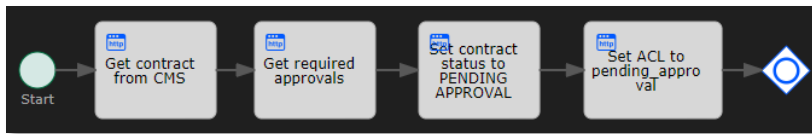
- **Exclusive gateway:** Is used to model a decision in the process. When the execution arrives at this gateway, all outgoing sequence flows are evaluated in the order in which they are defined. The first sequence flow whose condition evaluates to true is selected for continuing the process.
- **Parallel gateway:** Is the most straightforward gateway to introduce concurrency in a process model. It allows you to fork into multiple paths of execution or join multiple incoming paths of execution.
- **Inclusive gateway:** Can be seen as a combination of an exclusive and a parallel gateway. Like an exclusive gateway, you can define conditions on outgoing sequence flows and the inclusive gateway will evaluate them. However, the main difference is that like the parallel gateway, the inclusive gateway can take more than one sequence flow.
- **Event gateway:** Provides a way to take a decision based on events. Each outgoing sequence flow of the gateway must be connected to an intermediate catching event. When process execution reaches an event-based gateway, the gateway acts like a wait state: execution is suspended. In addition, for each outgoing sequence flow, an event subscription is created.

In this tutorial you will be using inclusive and exclusive gateways to control the contract approval flow execution.

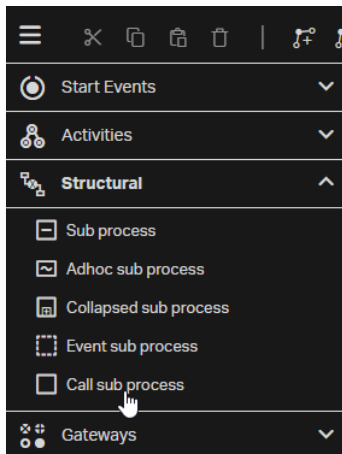
16. Drag and drop the **Inclusive gateway** to the **canvas** next to the **Set ACL to pending_approval http task**.



17. Select the **Set ACL to pending_approval http task** and drag a **sequence flow** to the **inclusive gateway**.



18. In the **palette**, find **Structural > Call sub process**.



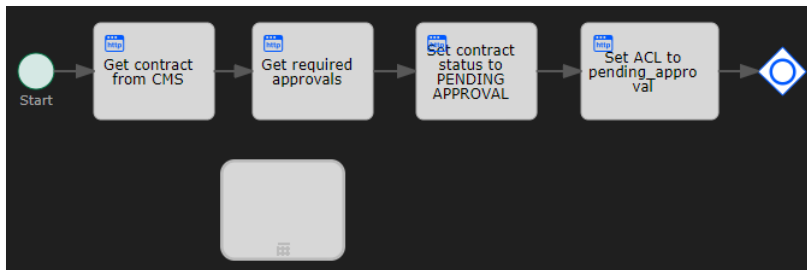
Note

BPMN 2.0 makes a distinction between a regular sub-process, often also called embedded sub-process, and a called sub process. Both call a sub-process when the process execution arrives at the activity. The difference is that call sub process references a process that is external to the process definition, whereas sub-process is embedded within the original process definition. The main use case for call sub process is to have a reusable process definition that can be called from multiple other process definitions.

When process execution arrives at call sub process, a new execution is created that is a sub-execution of the execution that arrived at call sub process. This sub-execution is then used to execute the sub-process, potentially creating parallel child executions within a regular process. The super-execution waits until the sub-process ends and continues with the original process afterward.


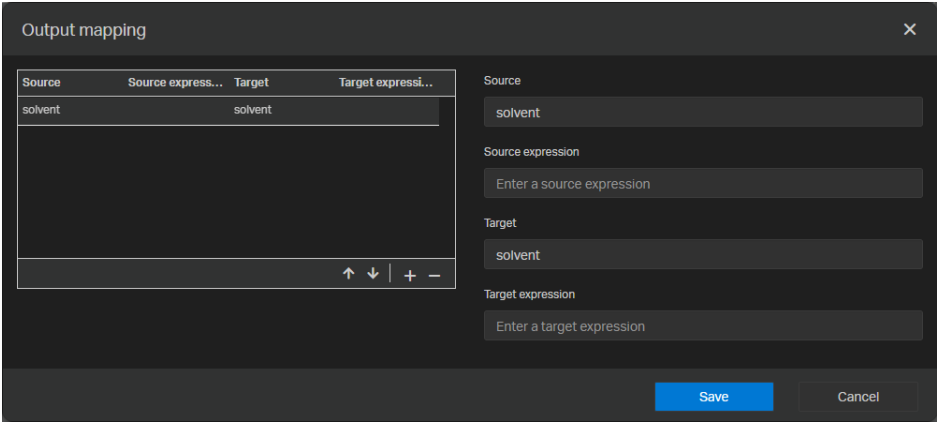
For the contract approval workflow, call sub process will be used to call both the previously created Solvency Check and Manager Approval workflows as sub-processes.

19. Drag and drop the **Call sub process** structural to the **canvas** below the previously created **Http tasks** and **Inclusive gateway**. Leave enough vertical space to draw sequence flow connectors from the inclusive gateway and leave enough horizontal space to add a text annotation under the **Start event** and **Get contract from CMS http task**.



20. Fill the call sub process attributes using the following details:

Attribute	Value
Display name	This call sub process calls the Solvency Check workflow to perform the solvency check. Enter the following display name: <code>Check Solvency</code>
Called process reference	Whether to use the key (that is, the name) or the ID of the deployed process definition to start the process referenced in the Called process property. Select the key value for the called process reference.
Called process	The name (also known as the key) or ID of the workflow to call. This call sub process calls the Solvency Check workflow by name to perform the solvency check. Enter the following for the called process: <code>solvency_check</code>

Attribute	Value
Output mapping	<p>Optional output parameter map. Allows the mapping of parameters and variables from the called sub-process to the calling process (that is, the workflow that contains the call sub process call).</p> <p>For the Check Solvency call sub process, the solvent process variable of the called (that is, source) solvency_check workflow must be mapped to a new solvent process variable of the calling (that is, target) workflow.</p> <p>Click the Add button  and enter <code>solvent</code> for both the Source and Target process variables.</p>  <p>Click Save to confirm the output mapping.</p>
Inherit variables in sub process	<p>Whether to inherit the parent process variables in the sub-process.</p> <p>Choose to inherit the input variables in the sub process.</p>

Properties - Check Solvency

Basic Advanced

General details

Id
Enter value

Display name
Check Solvency

Sub process details

Called process reference
key

*Called process
solvency_check

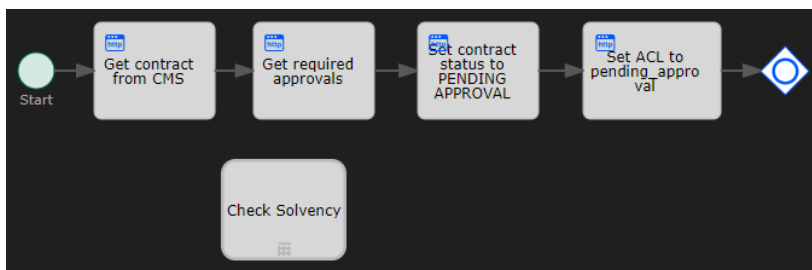
Input mapping
Add

Output mapping
1 out-parameters

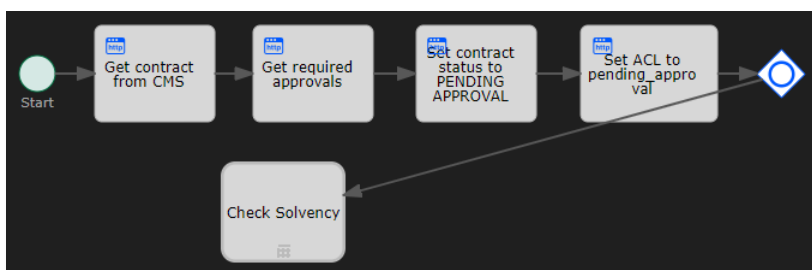
Inherit variables in sub process
☒

Execution details

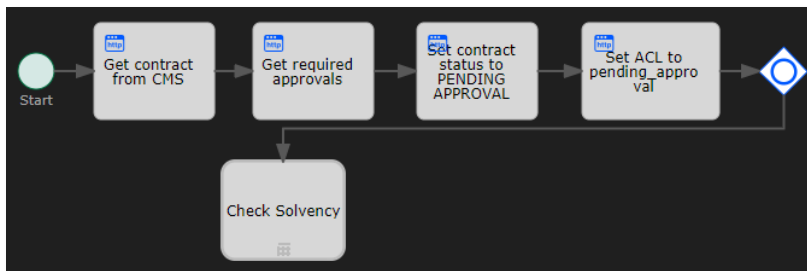
Execution listeners
Add



21. Select the **inclusive gateway** and drag a sequence flow to the Check Solvency call sub process.



22. Add two bend-points to the new sequence flow.



23. Select the new sequence flow and fill the sequence flow attributes using the following details:

Attribute	Value
Flow condition	<p>The condition that defines whether the sequence flow is selected from the connected gateway.</p> <p>This flow must be selected by the inclusive gateway if an automated solvency check is required. This information is available from the solvency_check_required process variable that was set after calling the Decision Service in the Get required approvals http task.</p> <p>Enter the following flow condition:</p> <pre>\${solvency_check_required}</pre>

Properties

General details

Display name

Connector details

* At least one input is required.

Flow condition

Default flow

☒

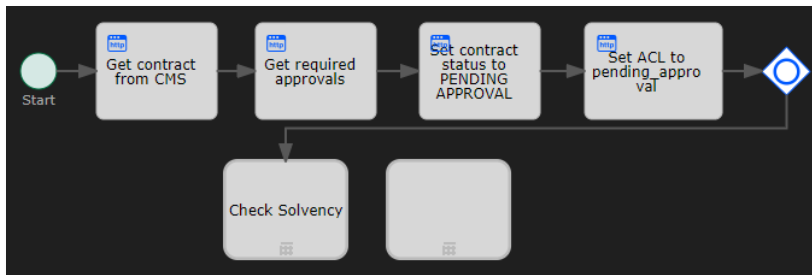
Skip condition

Execution details


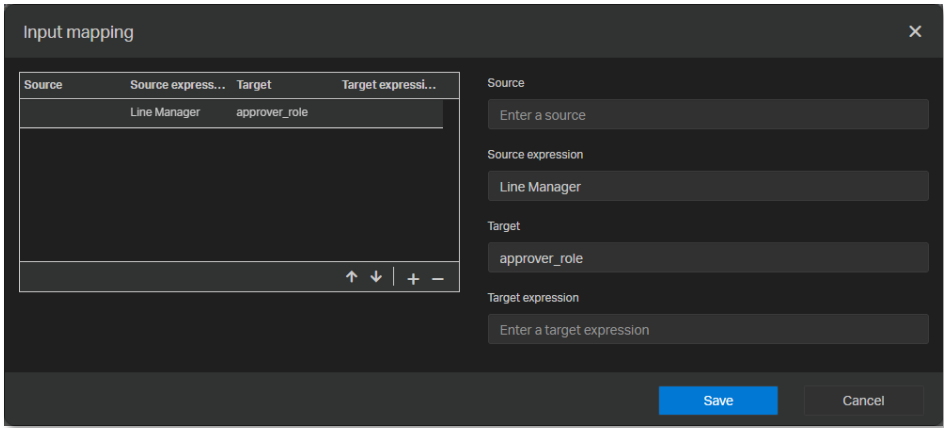
Execution listeners


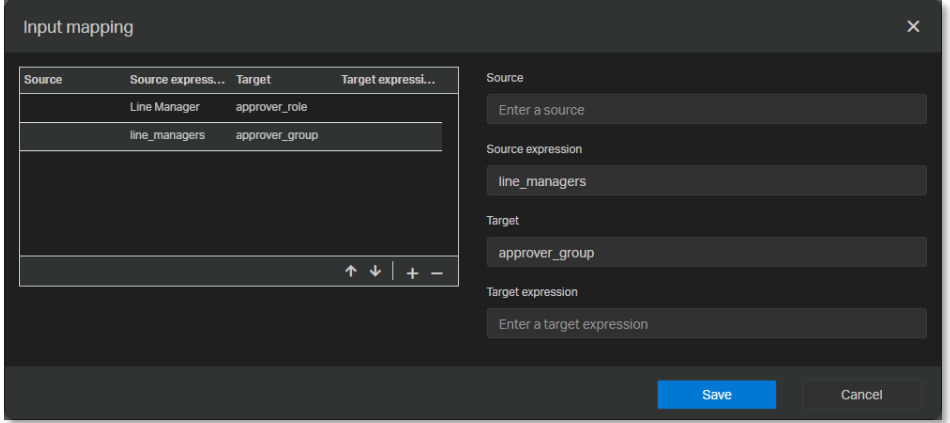

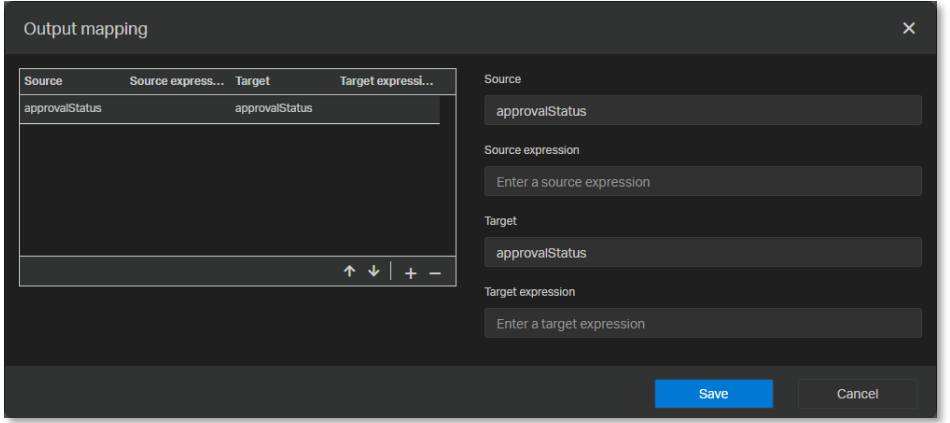
+ Add

24. Drag and drop a second **Call sub process** structural from the **palette** to the **canvas** next to the **Check Solvency call sub process**.



25. Fill the call sub process attributes using the following details:

Attribute	Value
Display name	This call sub process calls the Manager Approval workflow to perform the line manager approval. Enter the following display name: <code>Line Manager Approval</code>
Called process reference	Select the key value for the called process reference.
Called process	<code>manager_approval</code>
Input mapping	<p>Optional input parameter map. Allows the mapping of parameters and variables from the calling process (that is, the workflow that contains the call sub process call) to the called sub-process.</p> <p>For the Line Manager Approval call sub process, the values of the approver_role and approver_group process variables of the called manager_approval sub-process need to be set.</p> <p>Click the Add button  and enter <code>Line Manager</code> as the source expression (in this case a literal text value) and <code>approver_role</code> as the target process variable to map it to in the called sub-process.</p> 

Attribute	Value
	<p>Click the Add button  to add a second input parameter and enter <code>line_managers</code> (literal text value) as the source expression and <code>approver_group</code> as the target process variable to map it to in the called sub-process.</p>  <p>Click Save to confirm the input mapping.</p>
Output mapping	<p>For the Line Manager Approval call sub process, the approvalStatus process variable of the called (that is, source) manager_approval workflow must be mapped to a new approvalStatus process variable of the calling (that is, target) workflow.</p> <p>Click the Add button  and enter <code>approvalStatus</code> for both the Source and Target process variables.</p>  <p>Click Save to confirm the output mapping.</p>
Inherit variables in sub process	Choose to inherit the input variables in the sub process.

Properties - Line Manager Approval

Basic Advanced

General details

Id
Enter value

Display name
Line Manager Approval

Sub process details

Called process reference
key

*Called process
manager_approval

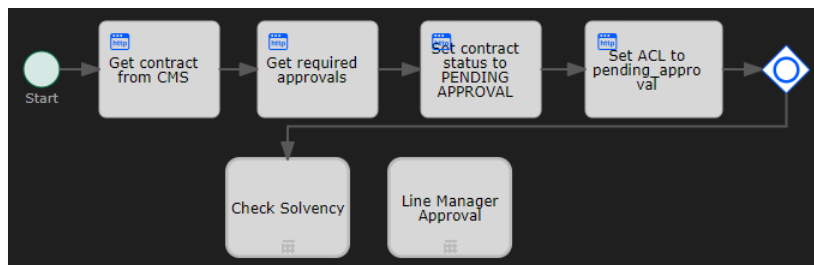
Input mapping
2 in-parameters

Output mapping
1 out-parameters

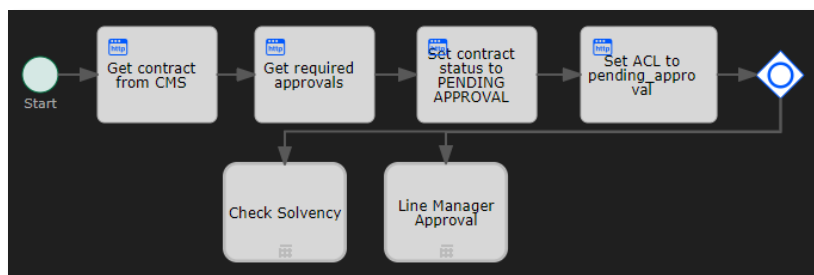
Inherit variables in sub process
☒

Execution details

Execution listeners
Add



26. Select the **inclusive gateway**, drag a **sequence flow** to the **Line Manager Approval** call sub process, and add two bend-points to the new sequence flow.



27. Select the new sequence flow and fill the sequence flow attributes using the following details:

Attribute	Value
Flow condition	<p>This flow must be selected by the inclusive gateway if a line manager approval is required. This information is available from the line_manager_approval_required process variable that was set after calling the Decision Service in the Get required approvals http task.</p> <p>Enter the following flow condition:</p> <pre>\${line_manager_approval_required}</pre>

Properties

General details

Display name

Connector details

* At least one input is required.

Flow condition

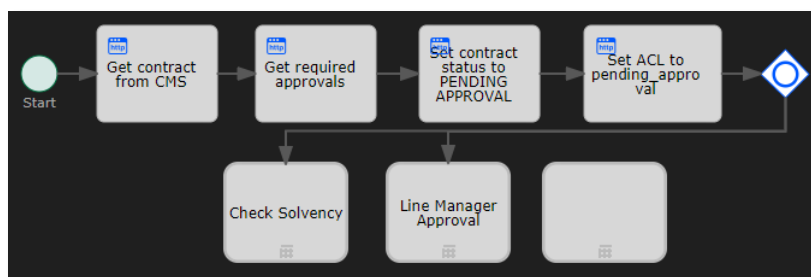
Default flow
☒

Skip condition

Execution details



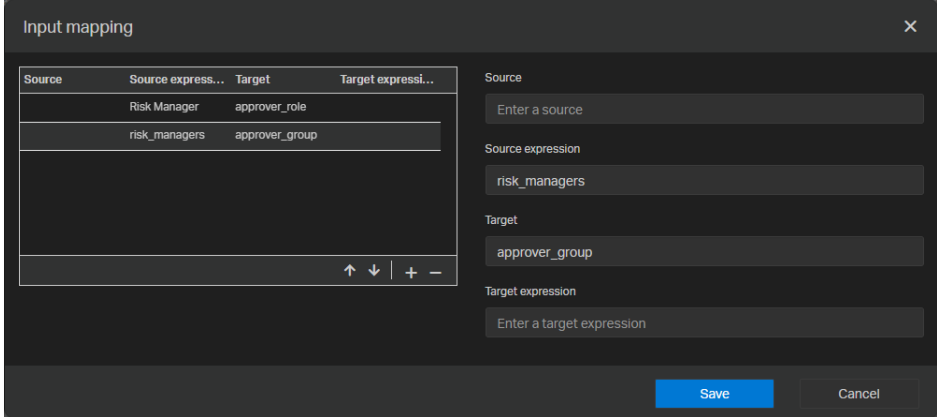
Execution listeners


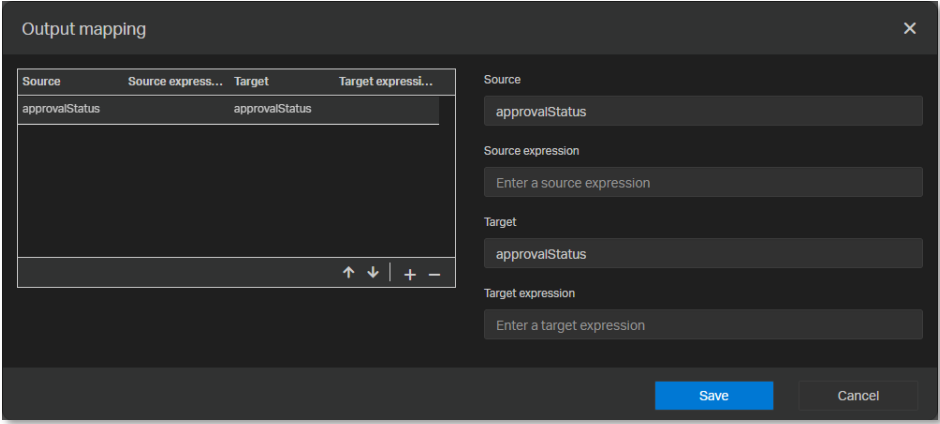
28. Drag and drop a third **Call sub process** structural from the **palette** to the **canvas** next to the **Line Manager Approval** call sub process.



29. Fill the call sub process attributes using the following details:

Attribute	Value
Display name	<p>This call sub process calls the Manager Approval workflow to perform the risk manager approval.</p> <p>Enter the following display name:</p> <pre>Risk Manager Approval</pre>

Attribute	Value
Called process reference	Select the key value for the called process reference.
Called process	manager_approval
Input mapping	<p>For the Risk Manager Approval call sub process, the values of the approver_role and approver_group process variables of the called manager_approval sub-process need to be set.</p> <p>Click the Add button  and enter Risk Manager (literal text value) as the Source expression and approver_role as the Target process variable to map it to in the called sub-process.</p> <p>Click the Add button  to add a second input parameter and enter risk_managers (literal text value) as the Source expression and approver_group as the Target process variable to map it to in the called sub-process.</p>  <p>Click Save to confirm the input mapping.</p>

Attribute	Value
Output mapping	<p>For the Risk Manager Approval call sub process, the approvalStatus process variable of the called (that is, the Source) manager_approval workflow must be mapped to a new approvalStatus process variable of the calling (that is the Target) workflow.</p> <p>Click the Add button  and enter <code>approvalStatus</code> for both the Source and Target process variables.</p>  <p>Click Save to confirm the output mapping.</p>
Inherit variables in sub process	Choose to inherit the input variables in the sub process.

Properties - Risk Manager Approval

Basic Advanced

General details

Id

Display name

Sub process details

Called process reference

*Called process

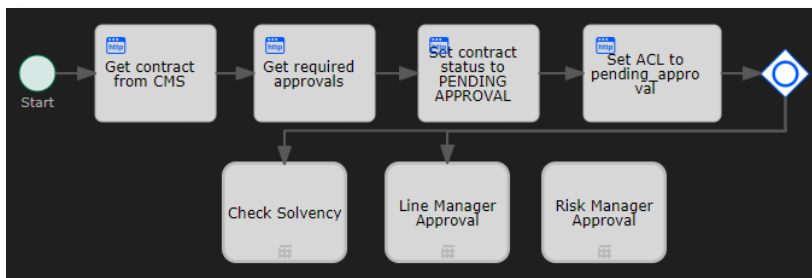
Input mapping
 2 in-parameters

Output mapping
 1 out-parameters

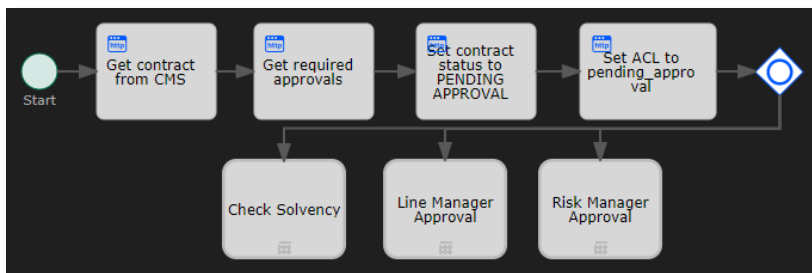
Inherit variables in sub process
☒

Execution details

Execution listeners



30. Select the **inclusive gateway**, drag a **sequence flow** to the **Risk Manager Approval** call sub process, and add two bend-points to the new sequence flow.



31. Select the new sequence flow and fill the sequence flow attributes using the following details:

Attribute	Value
Flow condition	<p>This flow must be selected by the inclusive gateway if a risk manager approval is required. This information is available from the risk_manager_approval_required process variable that was set after calling the Decision Service in the Get required approvals http task.</p> <p>Enter the following flow condition:</p> <pre>\${risk_manager_approval_required}</pre>

Properties

General details

Display name

Connector details

* At least one input is required.

Flow condition

`${risk_manager_approval`

Default flow

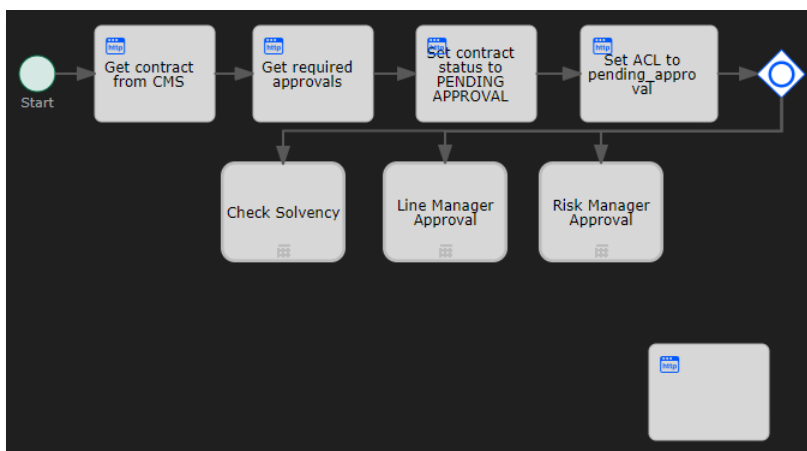
☒ ☐

Skip condition

Execution details

Execution listeners

32. Drag and drop a fifth **Http task** from the **palette** to the **canvas** under the bottom corner of the **Risk Manager Approval** call sub process.



33. Fill the Http task attributes using the following details:

Attribute	Value
Display name	Set contract status to APPROVED
Authentication	Select Use current authentication token .
Request method	Select the PATCH request method.
Request URL	<code>\${base_url}/cms/instances/file/ca_contract/\${contract_id}</code>
Request headers	Content-Type: application/json
Request body	<p>This task sets the status property of the contract to APPROVED. Enter the following request body:</p> <pre>{ "properties": { "status": "APPROVED" } }</pre>
Response variable name	contract
Save response as JSON	Choose to save the response as JSON.

Properties - Set contract status to APPROVED

Basic
Advanced

General details

Id

Display name

REST service details

Authentication

Authentication configured

*Request method

PATCH

*Request URL

Request headers

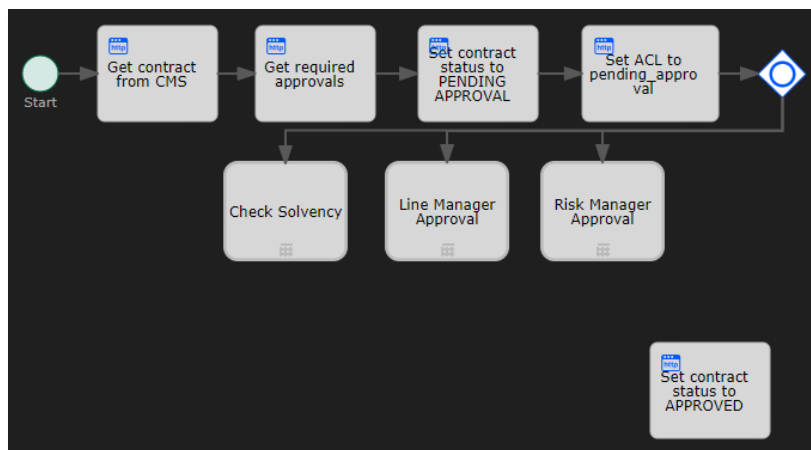
Request body

Response variable name

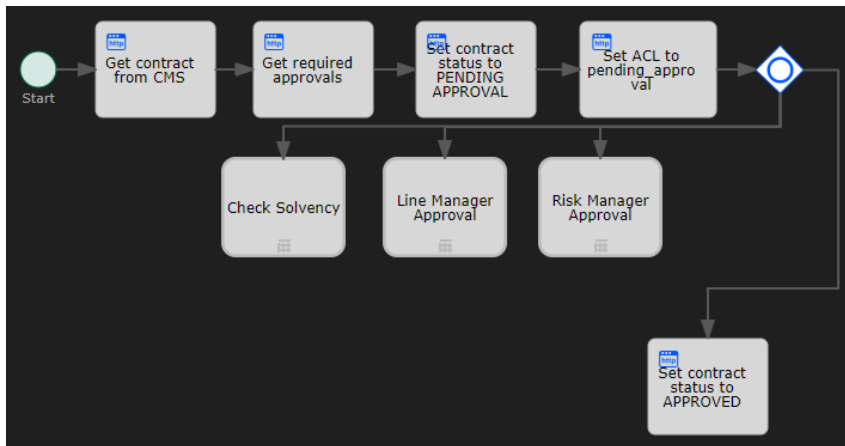
Save response as JSON

☒

Save response as a transient variable

☐


34. Select the **inclusive gateway**, drag a **sequence flow** to the **Set contract status to APPROVED** **http task**, and add three bend-points to the new sequence flow.



35. Select the new sequence flow and fill the sequence flow attributes using the following details:

Attribute	Value
Default flow	<p>Whether the sequence flow is the default flow to be selected from the connected gateway. The default flow gets selected when none of the other connected flows are (that is, none of the flow conditions of the other flows have been fulfilled).</p> <p>This flow is the default flow, and it results in setting the contract status directly to APPROVED when there are no required approvals.</p> <p>Choose to set this flow as the default flow.</p>

Properties

General details

Display name

Connector details

* At least one input is required.

Flow condition

+ Add

Default flow

☒

Skip condition

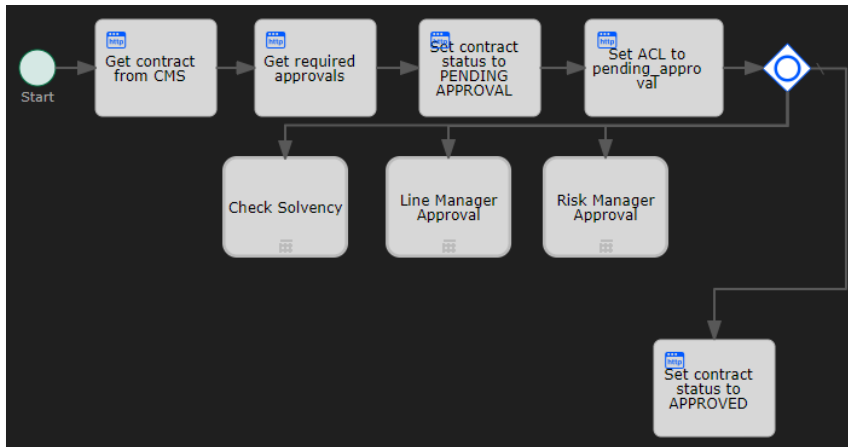
Execution details

Execution listeners

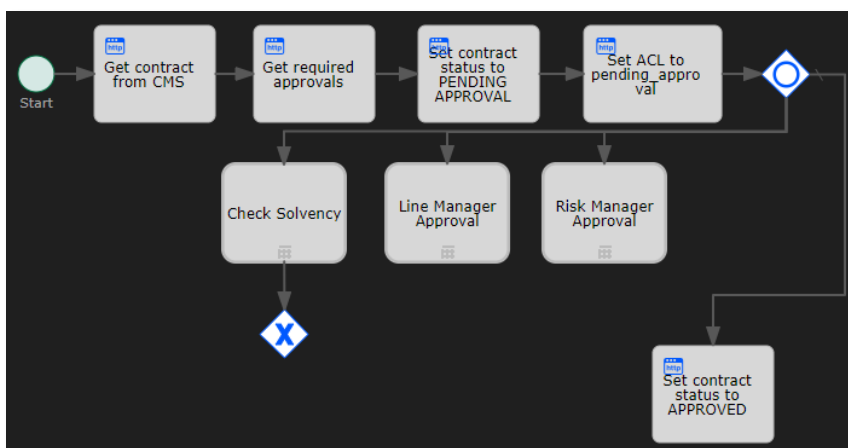
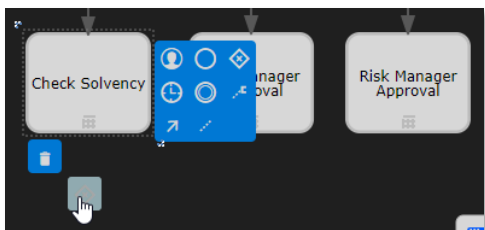
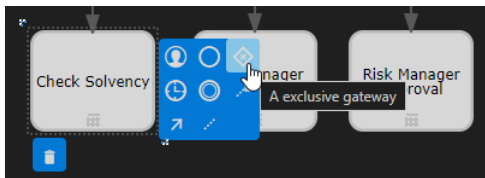
+ Add

**Note**

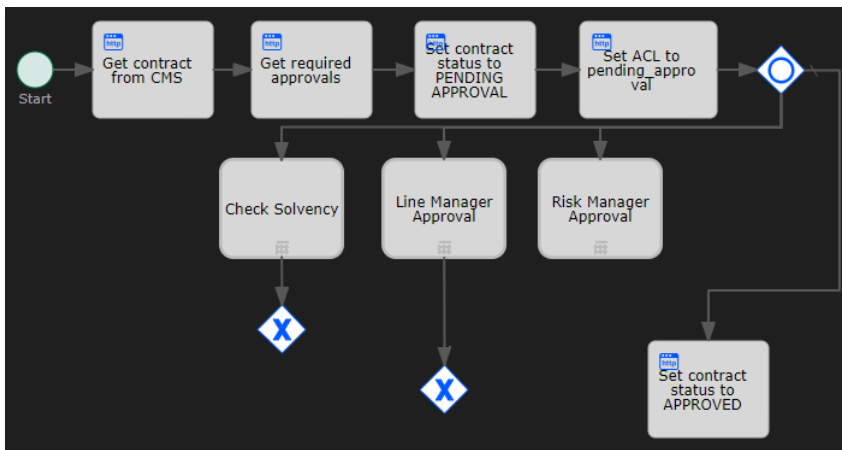
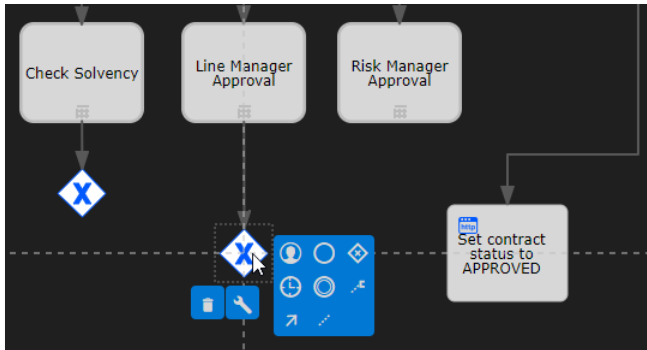
The default flow displays with an additional mark near the connected gateway:



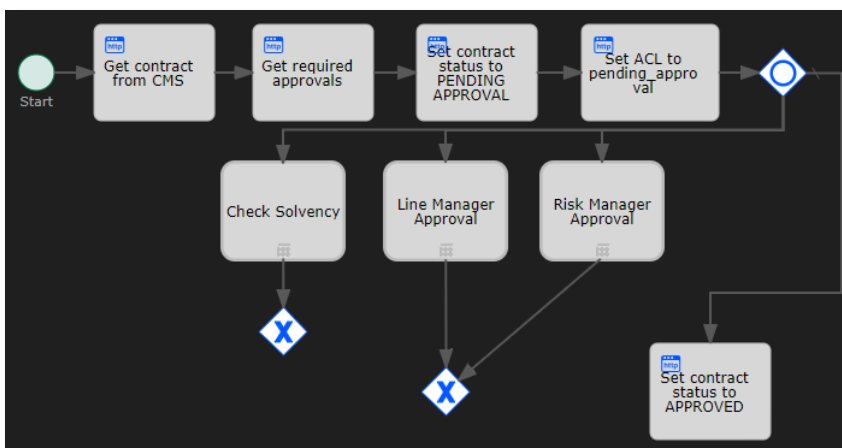
36. Select the **Check Solvency** call sub process and drag and drop an **Exclusive gateway** right below it.



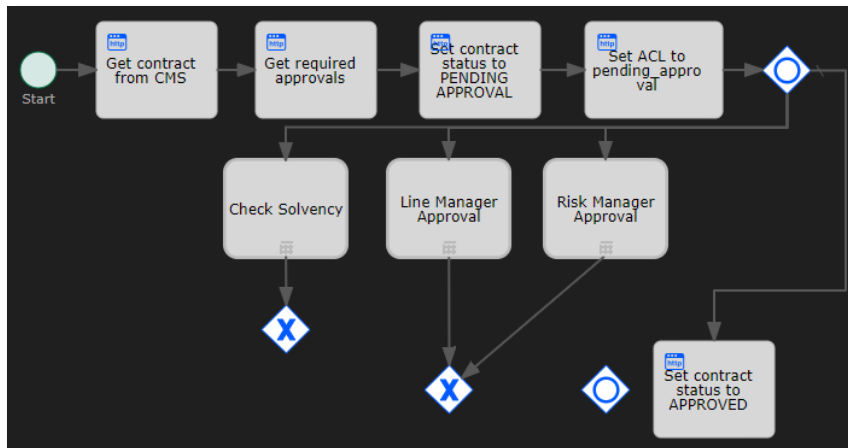
37. Select the **Line Manager Approval** call sub process and drag and drop an **Exclusive gateway** below it, making sure it is aligned with the **Set contract status to APPROVED** http task to ensure readability.



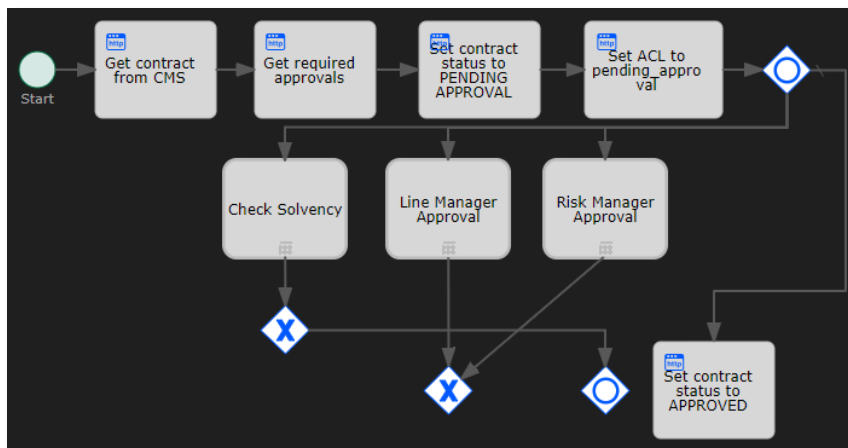
38. Select the **Risk Manager Approval** call sub process and drag a **sequence flow** to the second **exclusive gateway** (that is, the one below the Line Manager Approval call sub process).



39. Drag and drop a second **Inclusive gateway** to the **canvas** between the second **exclusive gateway** and the **Set contract status to APPROVED http task**. Make sure that the new inclusive gateway is close to the Set contract status to APPROVED http task, so that there is a maximum amount of space between the exclusive gateway and the inclusive gateway. This allows adding text to the connecting sequence flow.



40. Select the first **exclusive gateway** (that is, the one below the Check Solvency call task), drag a **sequence flow** to the **Inclusive gateway**, and add one bend-point to the new sequence flow.



41. Select the new sequence flow and fill the sequence flow attributes using the following details:

Attribute	Value
Display name	<p>The display name of a sequence flow displays on the canvas and can be filled to improve readability (to show what the sequence flow stands for).</p> <p>This sequence flow is selected when the outcome of the Check Solvency call sub process is that the contract approval requester is solvent, and more specifically that the Solvent process variable is equal to True.</p> <p>Enter the following display name:</p> <p>Solvent</p>
Default flow	<p>This flow is the default flow, and it moves the flow forward to the inclusive gateway that waits for all required approvals (that is, selected/active flows from the first inclusive gateway) to arrive.</p> <p>Choose to set this flow as the default flow.</p>

Properties - Solvent

General details

Display name
Solvent

Connector details

* At least one input is required.

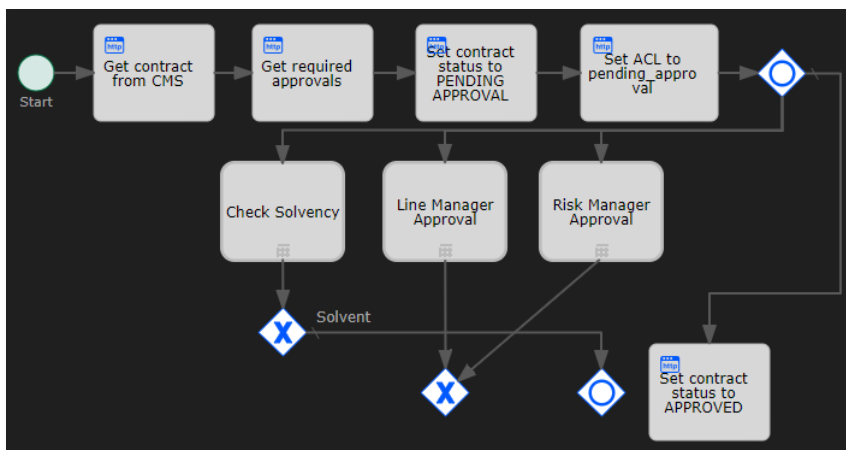
Flow condition
+ Add

Default flow
☒

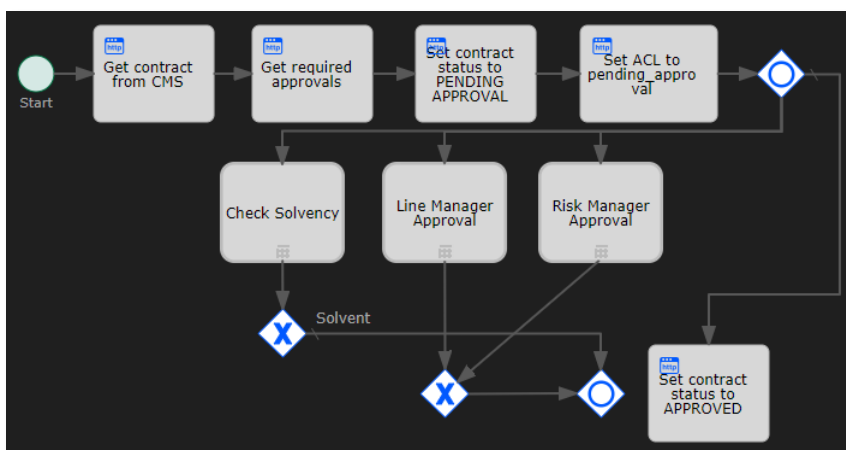
Skip condition
Enter value

Execution details

Execution listeners
+ Add



42. Select the second **exclusive gateway** (that is, the one below the Line Manager Approval call task) and drag a **sequence flow** to the **inclusive gateway**.



43. Select the new sequence flow and fill the sequence flow attributes using the following details:

Attribute	Value
Display name	This sequence flow is selected when the outcome of either the Line Manager Approval call sub process or the Risk Manager Approval call sub process is that the approver has approved the contract, and more specifically that the approvalStatus process variable is not equal to rejected or expired . Enter the following display name: Approved
Default flow	This flow is the default flow, and it moves the flow forward to the inclusive gateway that waits for all required approvals (that is, selected/active flows from the first inclusive gateway) to arrive. If both manager approver call activities are selected, the inclusive gateway requires this flow to be selected for each of them. Choose to set this flow as the default flow.

Properties - Approved

General details

Display name

Approved

Connector details

* At least one input is required.

Flow condition

+ Add

Default flow

☒

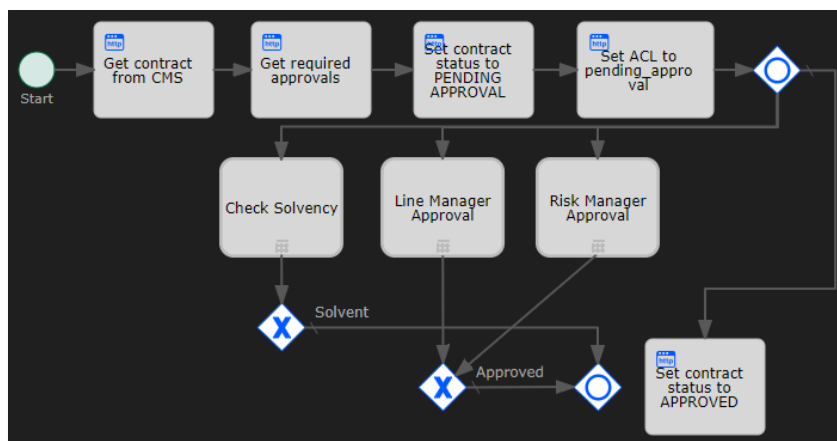
Skip condition

Enter value

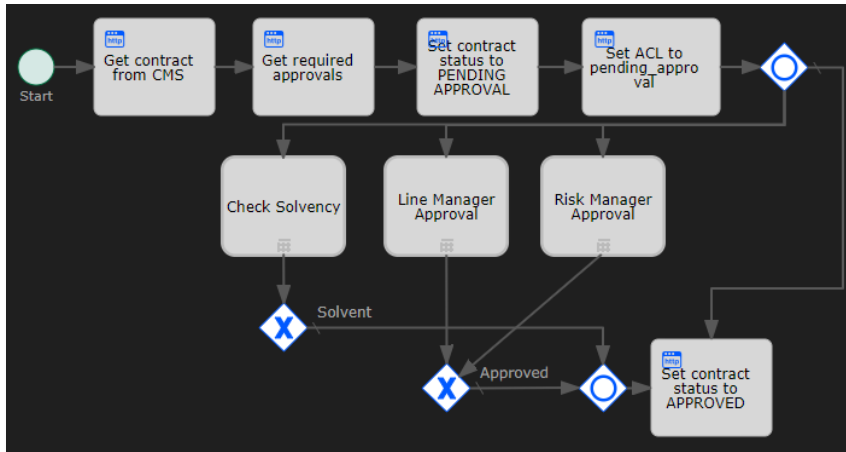
Execution details

Execution listeners

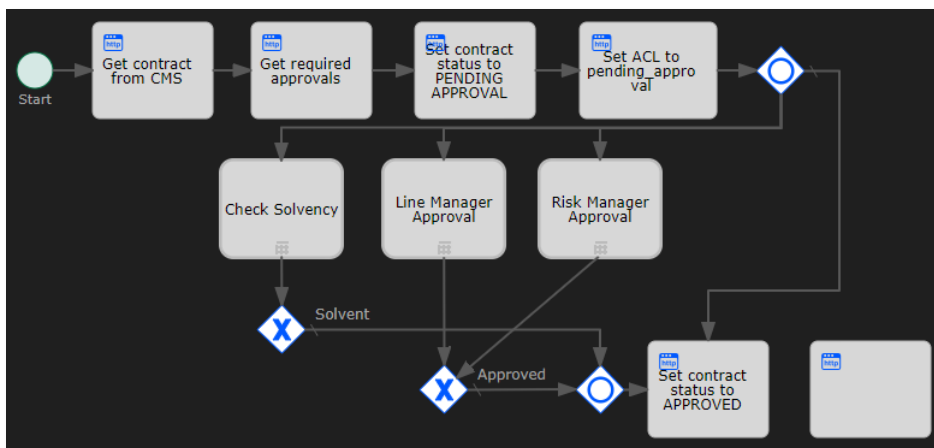
+ Add



44. Select the second **inclusive gateway** (that is, the one next to the Set contract status to APPROVED http task) and drag a **sequence flow** to the **Set contract status to APPROVED** http task.



45. Drag and drop a sixth **Http task** from the **palette** to the **canvas** next to the **Set contract status to APPROVED** http task.



46. Fill the Http task attributes using the following details:

Attribute	Value
Display name	Update Automatic Approval trait
Authentication	Select Use current authentication token .
Request method	Select the PATCH request method.
Request URL	<code>\${base_url}/cms/instances/file/ca_contract/\${contract_id}</code>
Request headers	Content-Type: application/json
Request body	<p>This task sets the has_been_granted, approver, approver_role, and approval_date properties for the Automatic Approval contract approval trait.</p> <p>Enter the following request body:</p> <pre>{ "traits": { "ca_approval": { "Automatic Approval": { "has_been_granted": true, "approver": "SYSTEM", "approver_role": "Automatic Approval", "approval_date": "\${contract.update_time}" } } } }</pre>
Response variable name	contract
Save response as JSON	Choose to save the response as JSON.

Properties - Update Automatic Approval trait

Basic Advanced

General details

Id
Enter value

Display name
Update Automatic Approval trait

REST service details

Authentication
Authentication configured

*Request method
PATCH

*Request URL
\${base_url}/cms/instances/file/ca_...

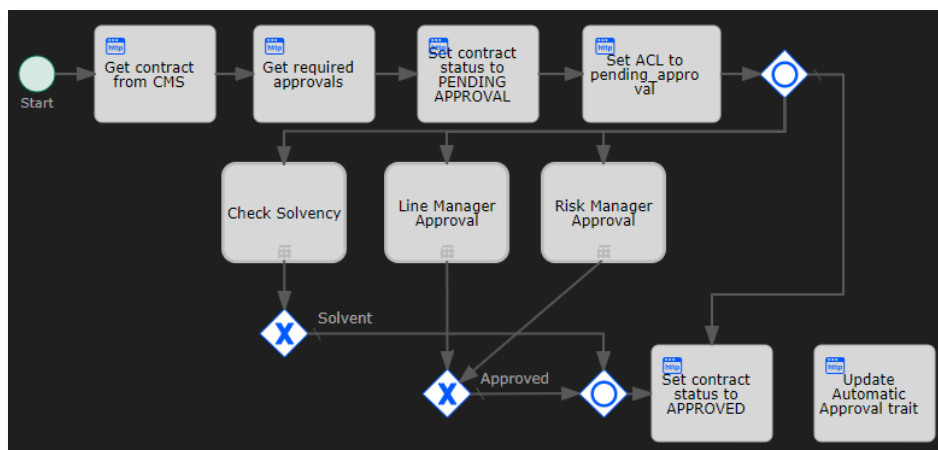
Request headers
Content-Type: application/json

Request body
{ "traits": { "ca_approval": { "Auto...

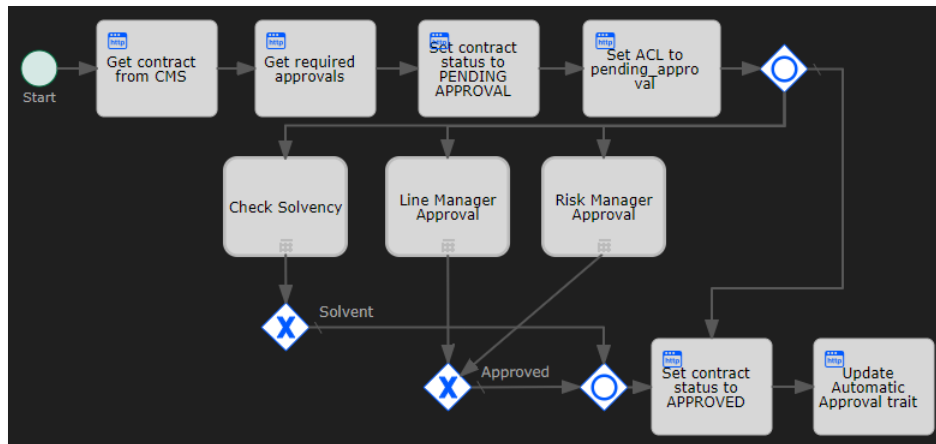
Response variable name
contract

Save response as JSON
☒

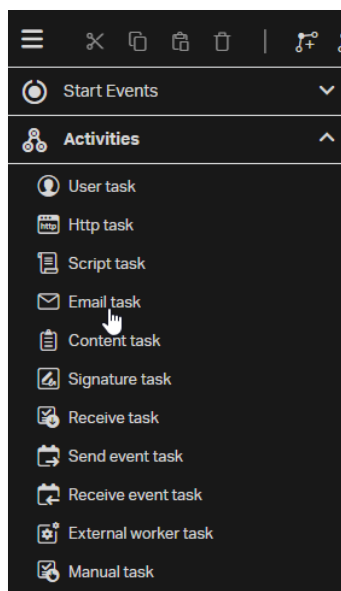
Save response as a transient variable
☐



47. Select the **Set contract status to APPROVED** http task and drag a **sequence flow** to the **Update Automatic Approval trait** http task.



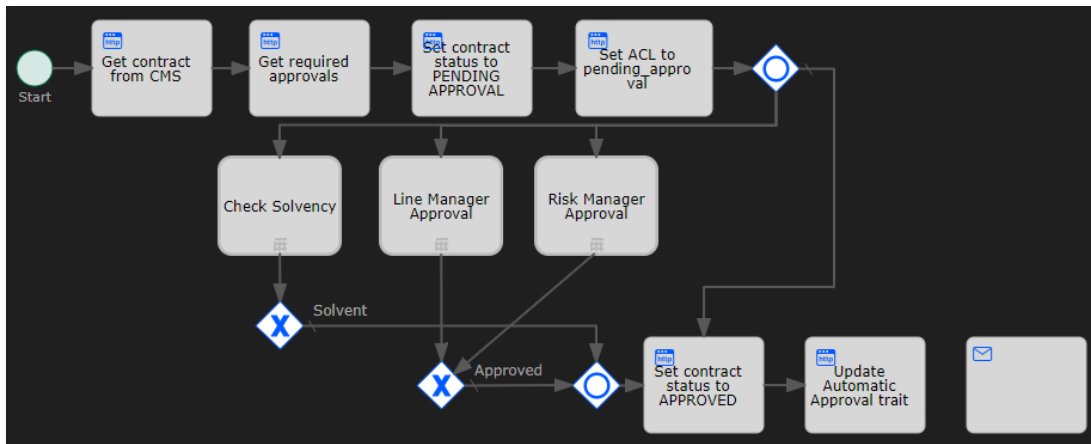
48. In the **palette**, find **Activities > Email task**.



Note

The email task sends emails to one or more recipients. It supports normal email features, such as cc lists, bcc lists, and HTML content.

49. Drag and drop the **Email task** to the **canvas** next to the **Update Automatic Approval trait http task**.



50. Fill the email task attributes using the following details:

Attribute	Value
Display name	Send Email on contract status
To	<p>The recipient of the email. Specify multiple recipients in a comma-separated list. This can be an expression.</p> <p>This task sends the email to the requester of the contract approval.</p> <p>Enter the following recipient email:</p> <pre>\${contract.properties.requester_email}</pre>
Subject	<p>The subject of the email. This can be an expression.</p> <p>Enter the following email subject:</p> <pre>Contract Approval Status</pre>
Text	<p>The text content of the email. This can be an expression.</p> <p>The email text details the name of the contract and its final status (that is, the result of the approvals).</p> <p>Enter the following email text:</p> <pre>Contract: \${contract.name} Status: \${contract.properties.status}</pre>

Properties - Send Email on contract status

Basic Advanced

General details

Id

Display name

Email details

To

Cc

Bcc

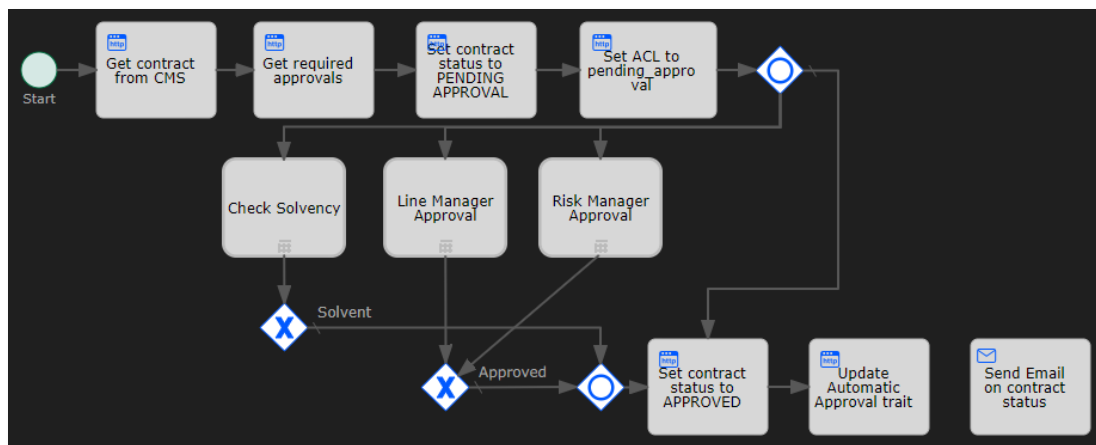
Subject

Text

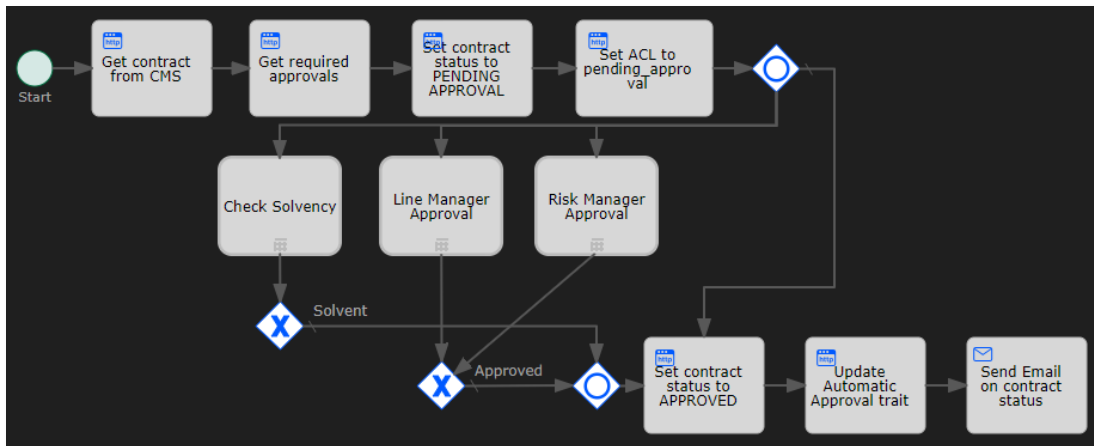
Execution details

Execution listeners

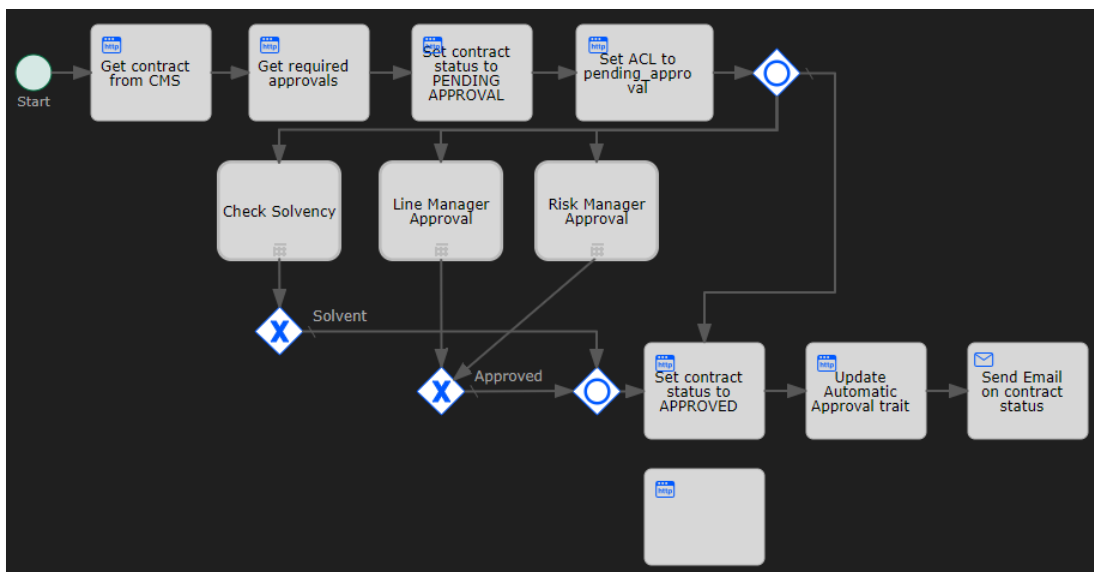
Add



51. Select the **Update Automatic Approval trait http task** and drag a **sequence flow** to the **Send Email on contract status email task**.



52. Drag and drop a seventh **Http task** from the **palette** to the **canvas** under the **Set contract status to APPROVED http task**.



53. Fill the Http task attributes using the following details:

Attribute	Value
Display name	Set contract status to EXPIRED
Authentication	Select Use current authentication token .
Request method	Select the PATCH request method.
Request URL	<code>\${base_url}/cms/instances/file/ca_contract/\${contract_id}</code>
Request headers	Content-Type: application/json

Attribute	Value
Request body	<p>This task sets the status property of the contract to EXPIRED.</p> <p>Enter the following request body:</p> <pre>{ "properties": { "status": "EXPIRED" } }</pre>
Response variable name	contract
Save response as JSON	Choose to save the response as JSON.

Properties - Set contract status to EXPIRED

Basic
Advanced

General details

Id

Display name

REST service details

Authentication
Authentication configured

*Request method
PATCH

*Request URL

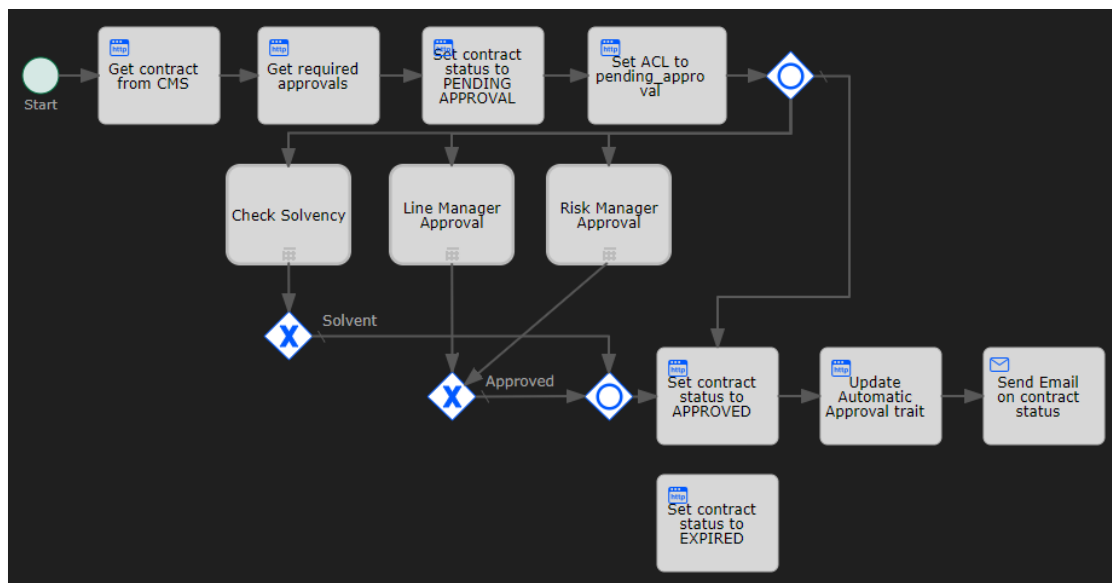
Request headers

Request body

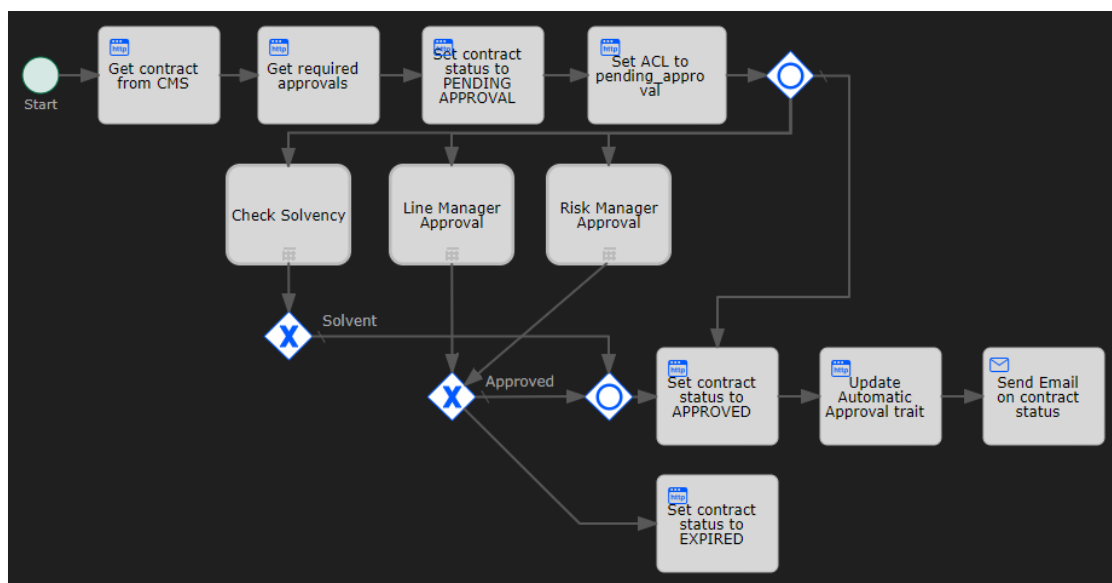
Response variable name

Save response as JSON
☒

Save response as a transient variable
☐



54. Select the second **exclusive gateway** (that is, the one below the Line Manager Approval call task), drag a **sequence flow** to the **Set contract status to EXPIRED** http task, and add 1 bend-point to the new sequence flow.



55. Select the new sequence flow and fill the sequence flow attributes using the following details:

Attribute	Value
Display name	Expired
Flow condition	<p>This sequence flow is selected when the outcome of either the Line Manager Approval call sub process or the Risk Manager Approval call sub process is that the approval task has expired, and more specifically that the approvalStatus process variable is equal to expired.</p> <p>Enter the following flow condition:</p> <pre>\${approvalStatus == "expired"}</pre>

Properties - Expired

General details

Display name
Expired

Connector details

* At least one input is required.

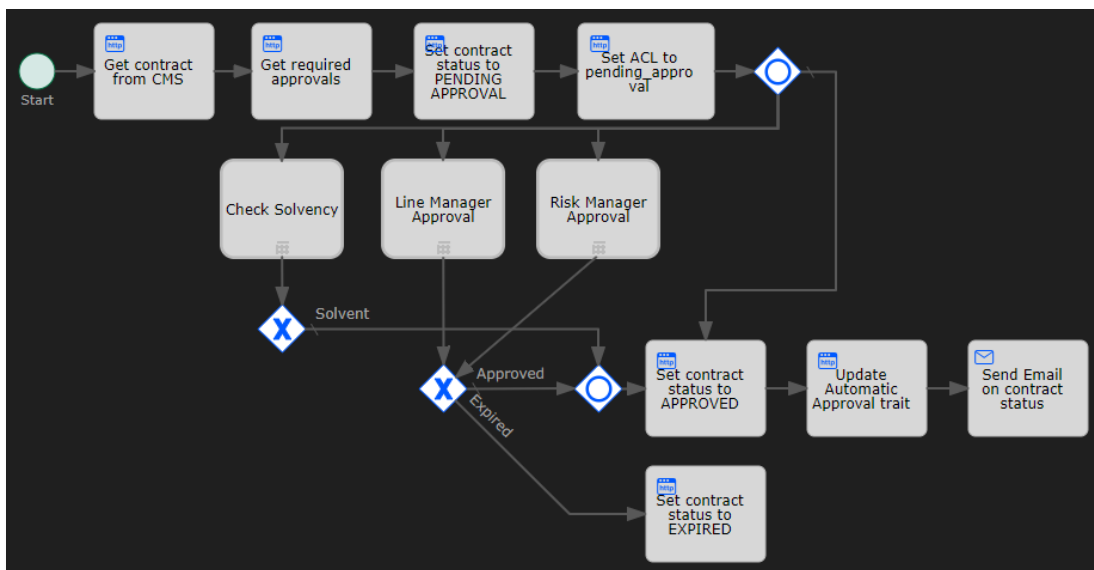
Flow condition
\${approvalStatus} ==

Default flow
☐ ☒

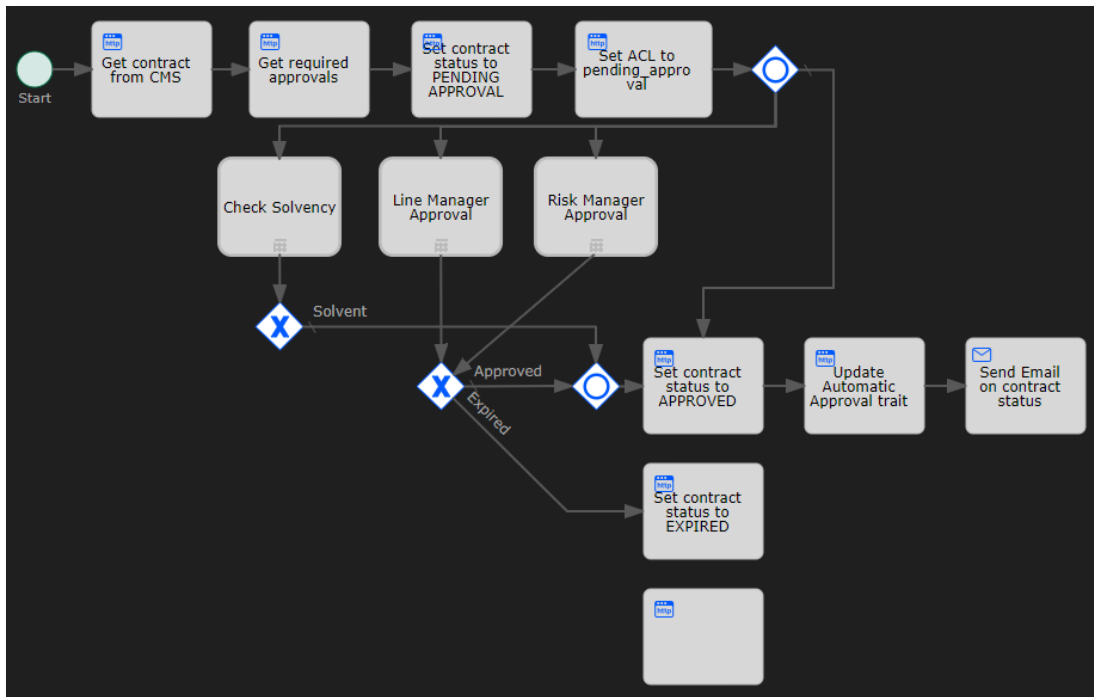
Skip condition
Enter value

Execution details

Execution listeners
[Add](#)



56. Drag and drop an eighth **Http task** from the **palette** to the **canvas** under the **Set contract status to EXPIRED** http task.



57. Fill the Http task attributes using the following details:

Attribute	Value
Display name	Set contract status to REJECTED
Authentication	Select Use current authentication token .
Request method	Select the PATCH request method.
Request URL	<code>\${base_url}/cms/instances/file/ca_contract/\${contract_id}</code>
Request headers	<code>Content-Type: application/json</code>
Request body	<p>This task sets the status property of the contract to REJECTED.</p> <p>Enter the following request body:</p> <pre>{ "properties": { "status": "REJECTED" } }</pre>
Response variable name	<code>contract</code>
Save response as JSON	Choose to save the response as JSON.

Properties - Set contract status to REJECTED

Basic Advanced

General details

Id

Display name

REST service details

Authentication
Authentication configured

*Request method
PATCH

*Request URL

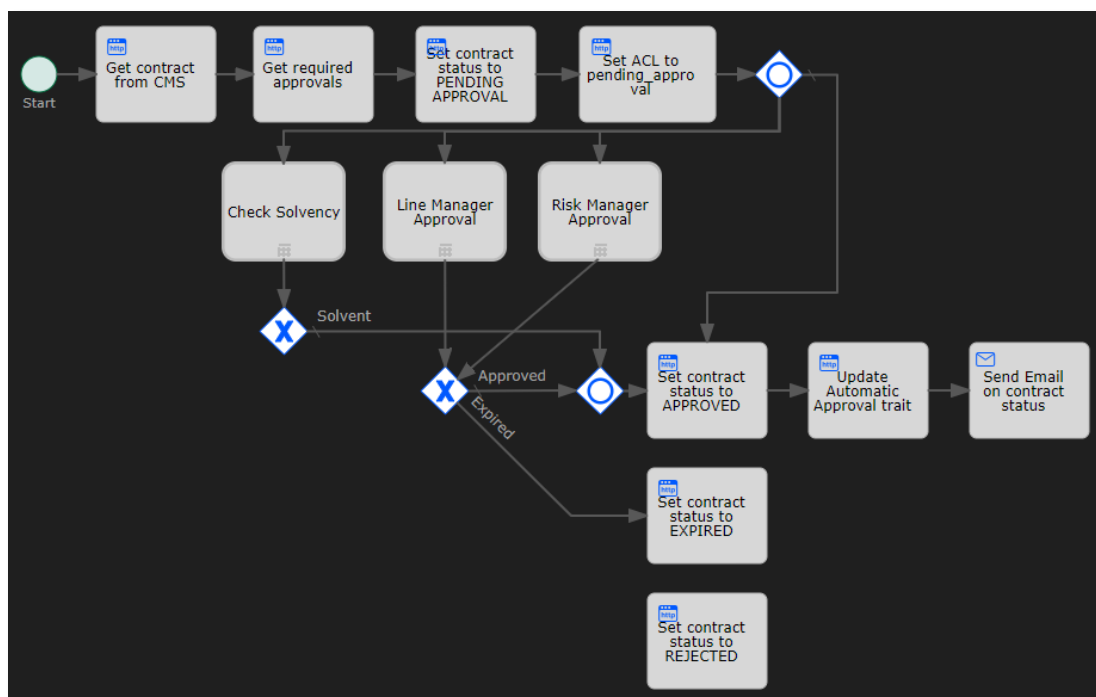
Request headers

Request body

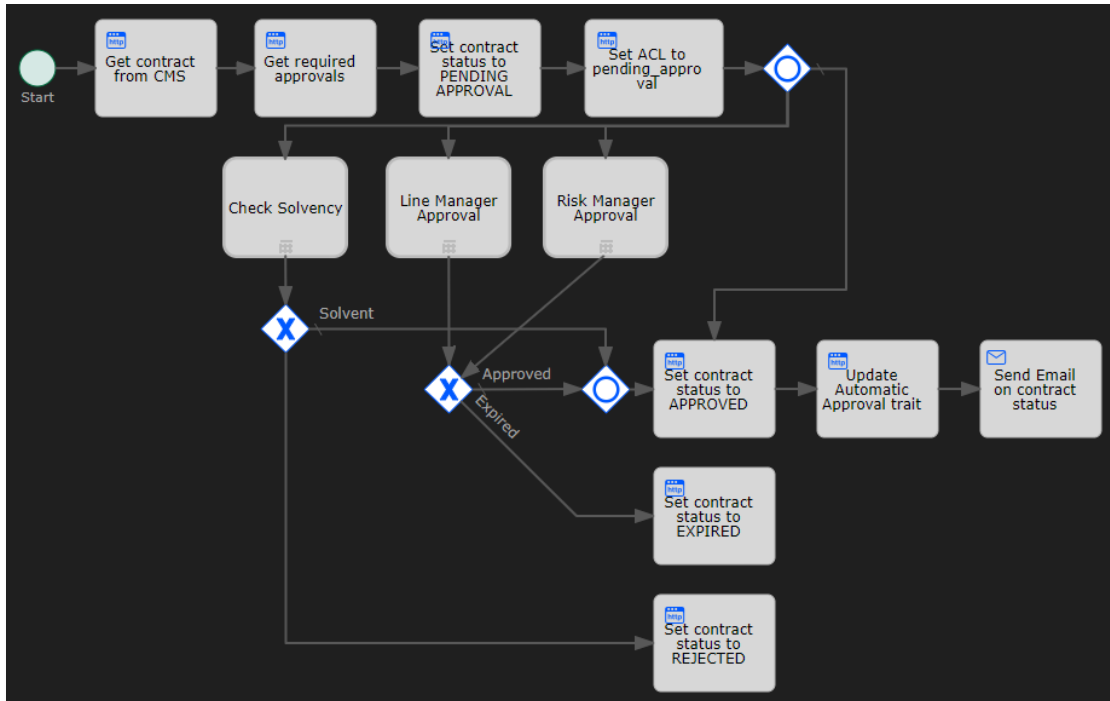
Response variable name

Save response as JSON
☒

Save response as a transient variable
☐



58. Select the first **exclusive gateway** (that is, the one below the Solvency Check call task), drag a **sequence flow** to the **Set contract status to REJECTED** http task, and add 1 bend-point to the new sequence flow.



59. Select the new sequence flow and fill the sequence flow attributes using the following details:

Attribute	Value
Display name	Not solvent
Flow condition	<p>This sequence flow is selected when the outcome of the Check Solvency call sub process is that the contract approval requester is not solvent, and more specifically that the solvent process variable is not equal to true.</p> <p>Enter the following flow condition:</p> <pre>\${!solvent}</pre>

Properties - Not solvent

General details

Display name

Not solvent

Connector details

* At least one input is required.

Flow condition

`${!solvent}`

Default flow

☒

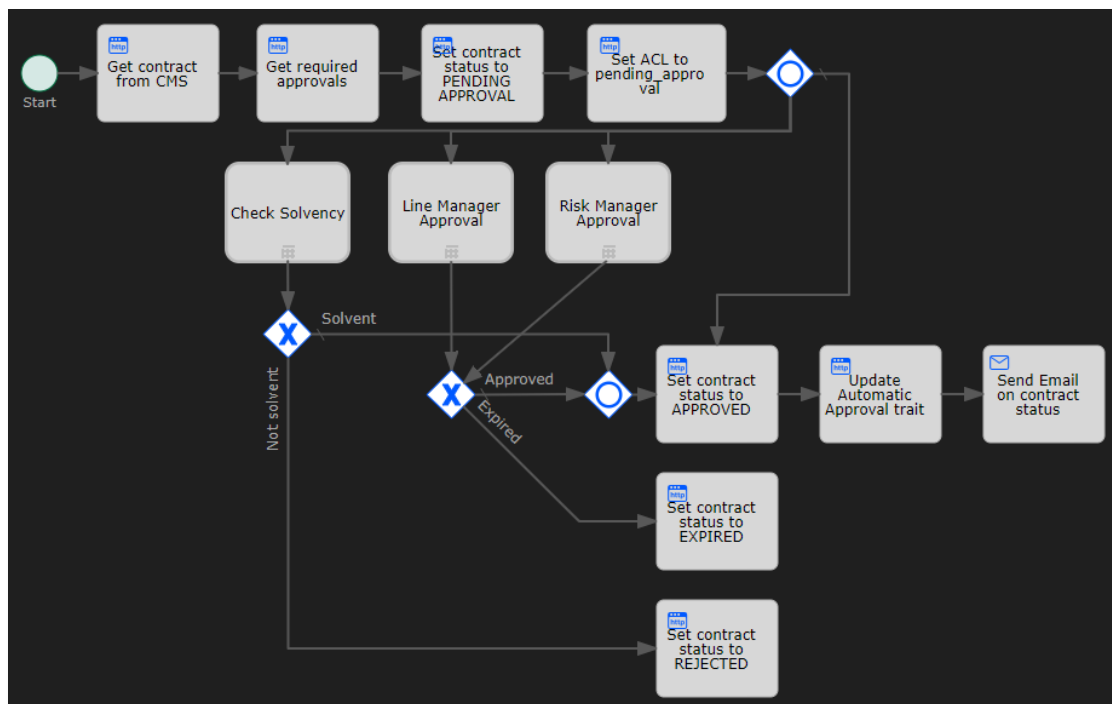
Skip condition

Enter value

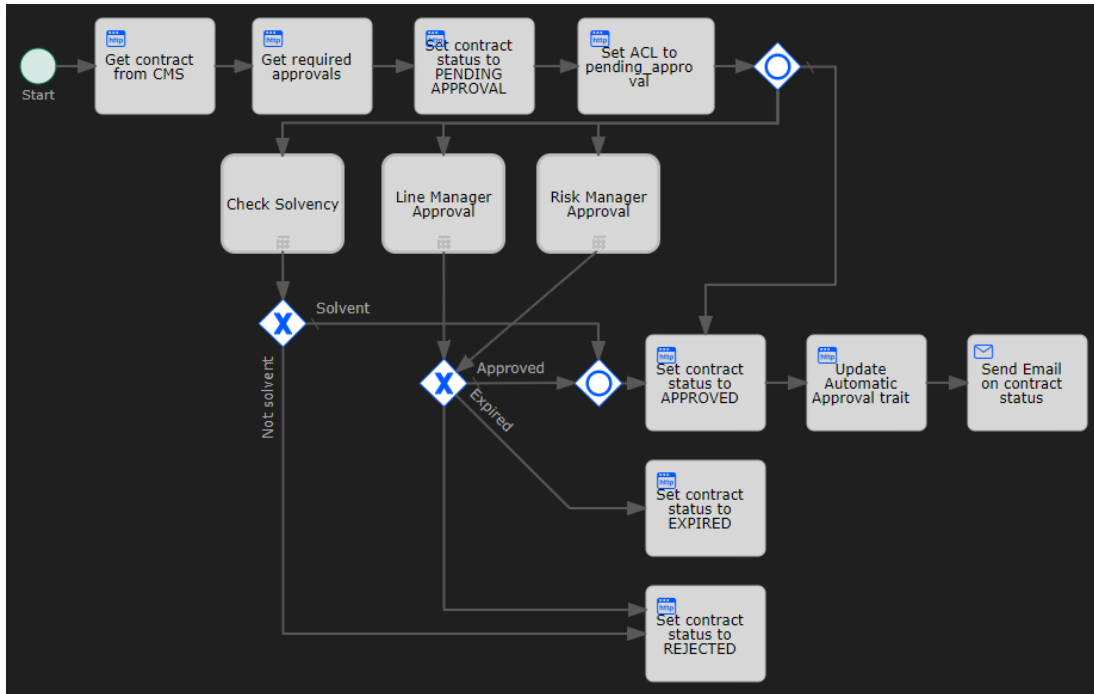
Execution details

Execution listeners

+ Add



60. Select the second **exclusive gateway** (that is, the one below the Line Manager Approval call task), drag a **sequence flow** to the **Set contract status to REJECTED http task**, and add 1 bend-point to the new sequence flow.



61. Select the new sequence flow and fill the sequence flow attributes using the following details:

Attribute	Value
Display name	Rejected
Flow condition	<p>This sequence flow is selected when the outcome of either the Line Manager Approval call sub process or the Risk Manager Approval call sub process is that the approval is rejected, and more specifically that the approvalStatus process variable is equal to rejected.</p> <p>Enter the following flow condition:</p> <pre>\${approvalStatus == "rejected"}</pre>

Properties - Rejected

General details

Display name

Rejected

Connector details

* At least one input is required.

Flow condition

\$[approvalStatus ==

Default flow

X

Skip condition

Enter value

Execution details

Execution listeners

+

Add

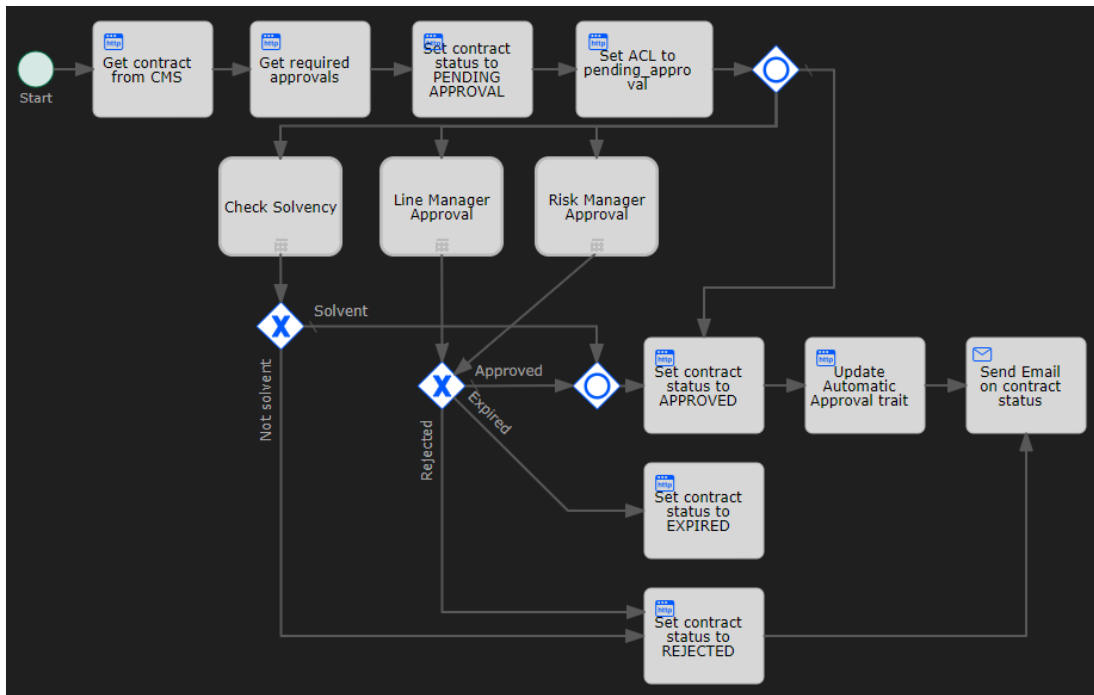
```

graph TD
    Start((Start)) --> GetContract[Get contract from CMS]
    GetContract --> GetApprovals[Get required approvals]
    GetApprovals --> SetPending[Set contract status to PENDING APPROVAL]
    SetPending --> SetACL{Set ACL to pending_approval}
    SetACL --> CheckSolvent{Check Solvency}
    SetACL --> LineManager[Line Manager Approval]
    SetACL --> RiskManager[Risk Manager Approval]
    CheckSolvent --> Solvent{Solvent}
    Solvent --> SetRejected1[Set contract status to REJECTED]
    LineManager --> Approved{Approved}
    RiskManager --> Approved
    Approved --> SetApproved[Set contract status to APPROVED]
    SetApproved --> UpdateTrait[Update Automatic Approval trait]
    UpdateTrait --> SendEmail[Send Email on contract status]
    Approved --> Expired{Expired}
    Expired --> SetExpired[Set contract status to EXPIRED]
    Rejected{Rejected} --> SetRejected2[Set contract status to REJECTED]
    
```

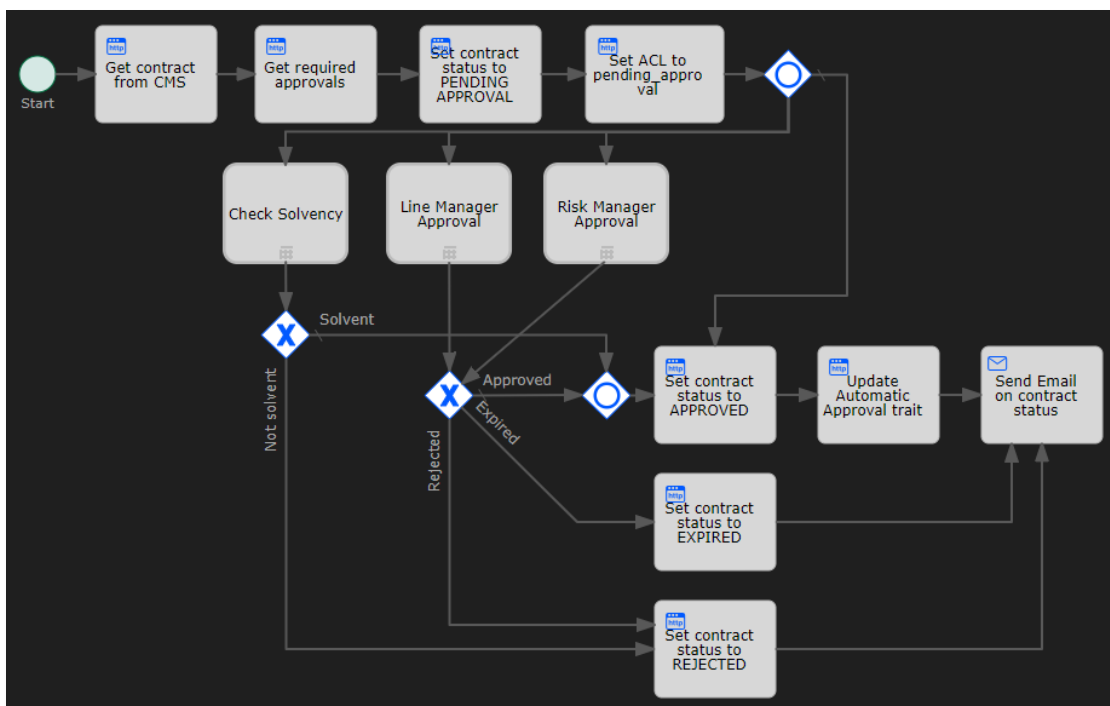
Copyright © 2025 Open Text. All rights reserved. Trademarks owned by Open Text.

156

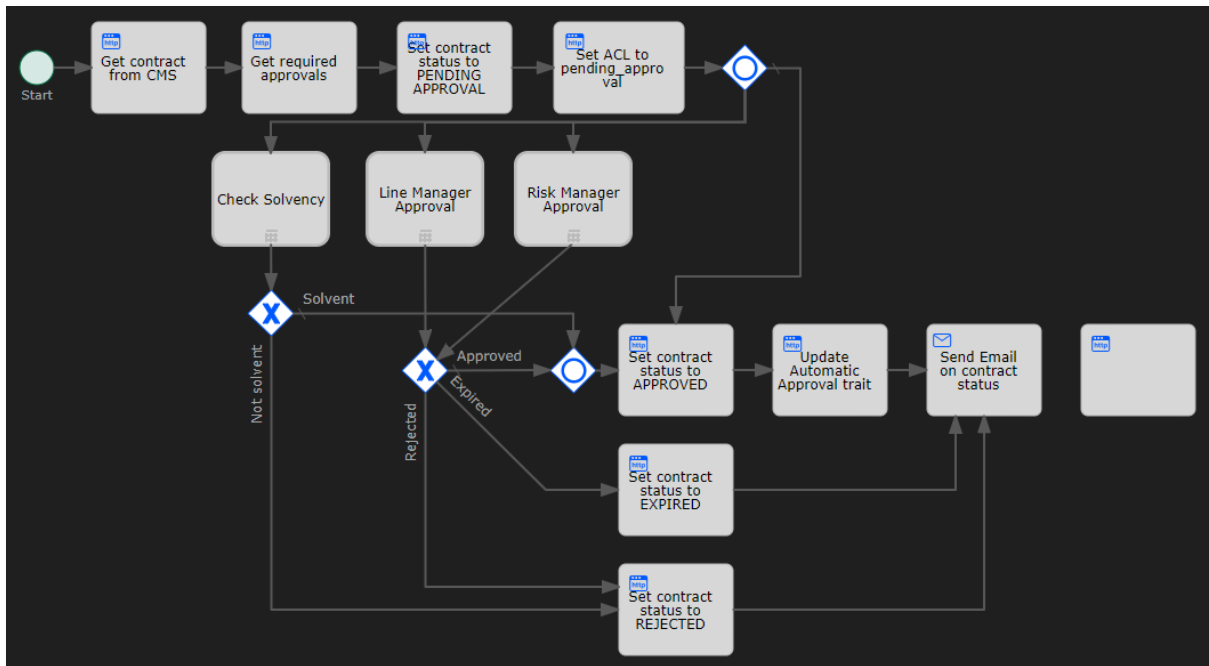
62. Select the **Set contract status to REJECTED** http task, drag a **sequence flow** to the **Send Email on contract status** email task, and add 1 bend-point to the new sequence flow.



63. Select the **Set contract status to EXPIRED** http task, drag a **sequence flow** to the **Send Email on contract status** email task, and add 1 bend-point to the new sequence flow.



64. Drag and drop a ninth **Http task** from the **palette** to the **canvas** next to the **Send Email on contract status email task**.



65. Fill the Http task attributes using the following details:

Attribute	Value
Display name	Set ACL to completed
Authentication	Select Use current authentication token .
Request method	Select the PUT request method.
Request URL	<code>\${base_url}/cms/instances/file/ca_contract/\${contract_id}/acl</code>
Request headers	<code>Content-Type: application/json</code>
Request body	<p>This task sets the ACL of the contract by using the completed_acl_id process input parameter as the id of the ACL to apply.</p> <p>Enter the following request body:</p> <pre>{ "id": "\${completed_acl_id}" }</pre>
Save response as JSON	Choose to save the response as JSON.

Properties - Set ACL to completed

Basic

Advanced

General details

Id

Enter value

Display name

Set ACL to completed

REST service details

Authentication

Authentication configured

*Request method

PUT

*Request URL

\$(base_url)/cms/instances/file/ca_...

Request headers

Content-Type: application/json

Request body

{ "id": "\${completed_acl_id}" }

Response variable name

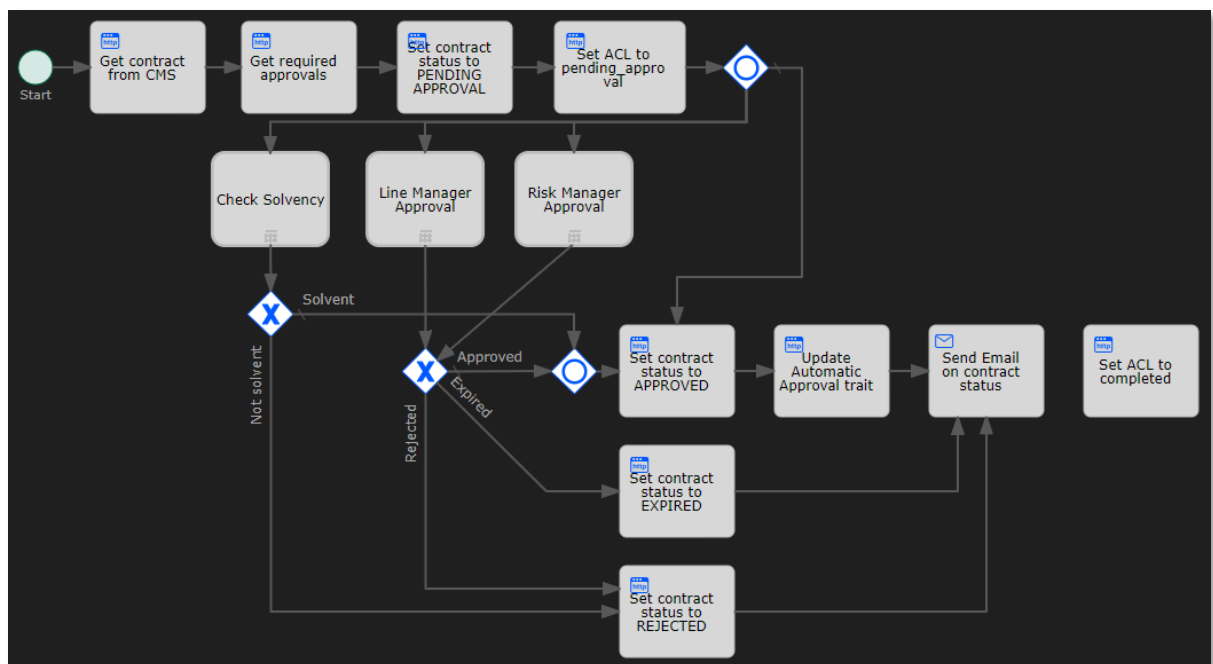
Enter value

Save response as JSON

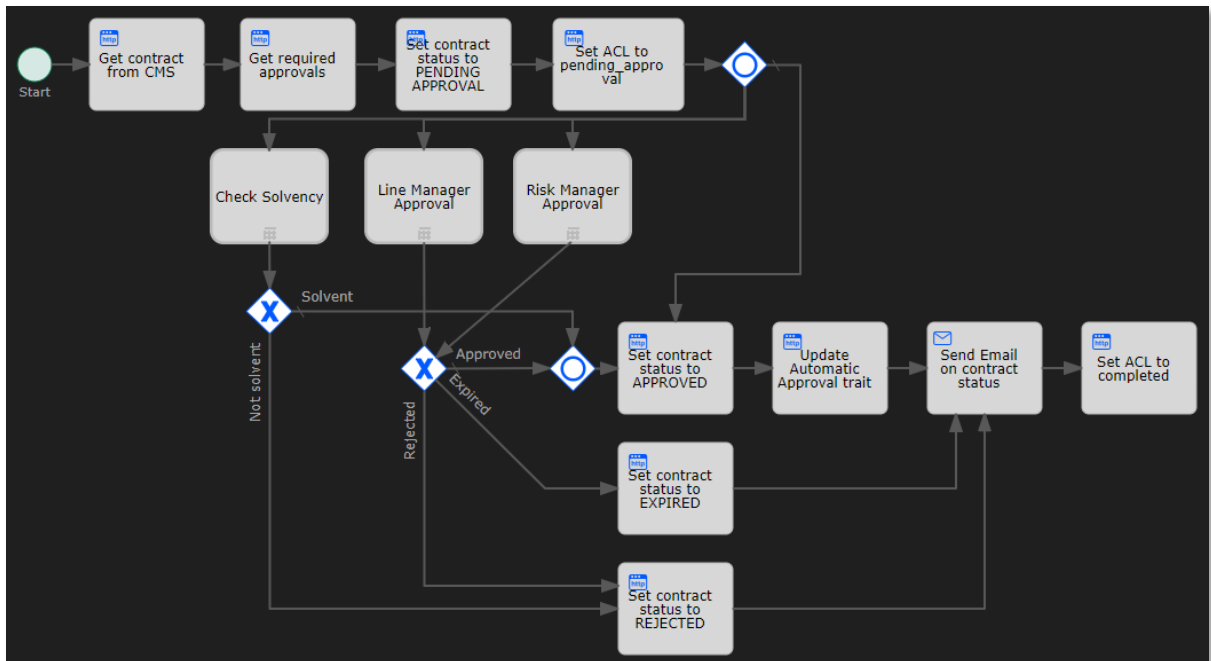
☒

Save response as a transient variable

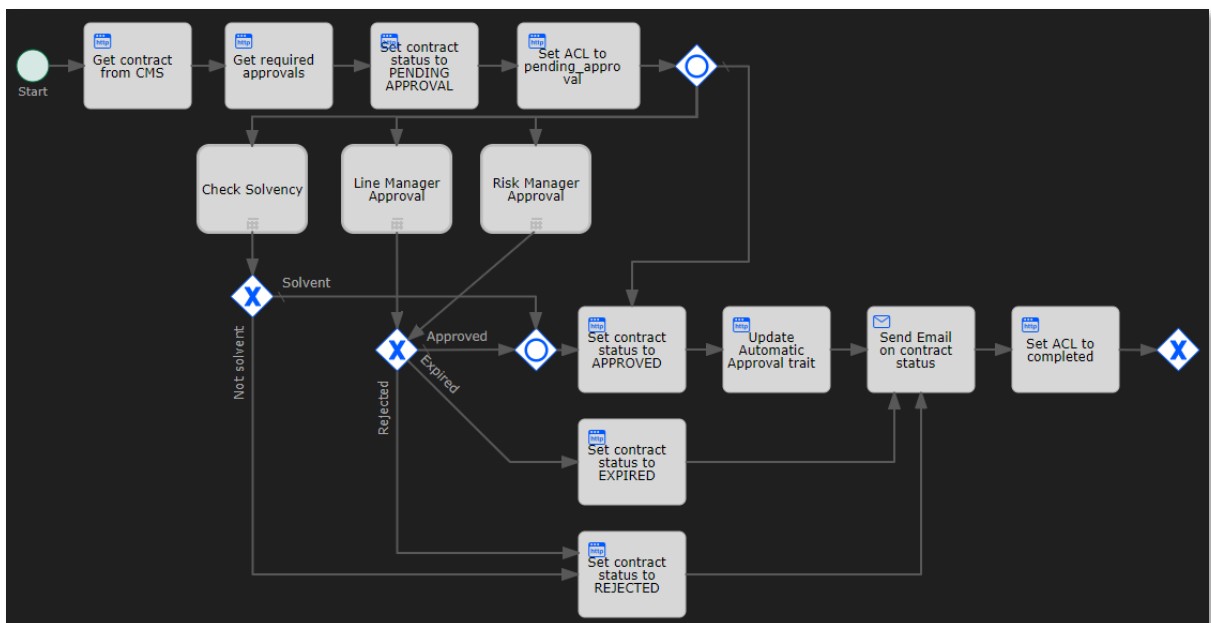
☐



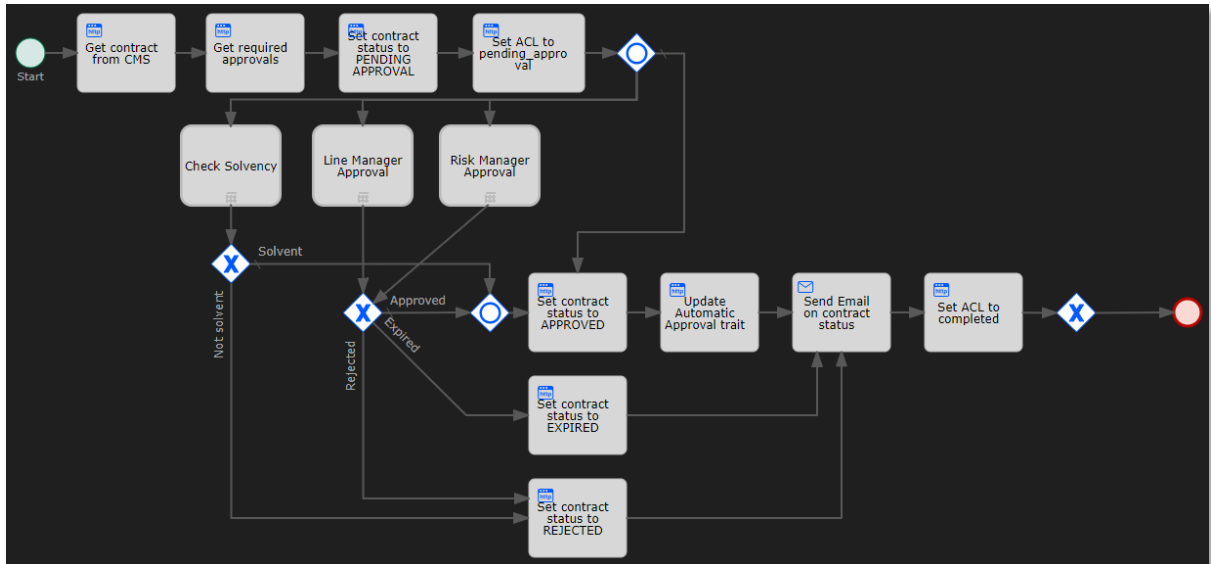
66. Select the **Send Email on contract status email task** and drag a **sequence flow** to the **Set ACL to completed http task**.



67. Select the **Set ACL to completed http task** and drag and drop an **Exclusive gateway** next to it.



68. Select the third (that is, newly created) **exclusive gateway** and drag and drop an **End event** next to it. Leave enough space between the exclusive gateway and the end event to allow adding text for the connecting sequence flow.



69. Fill the End event attributes using the following details:

Attribute	Value
Display name	End

Properties - End

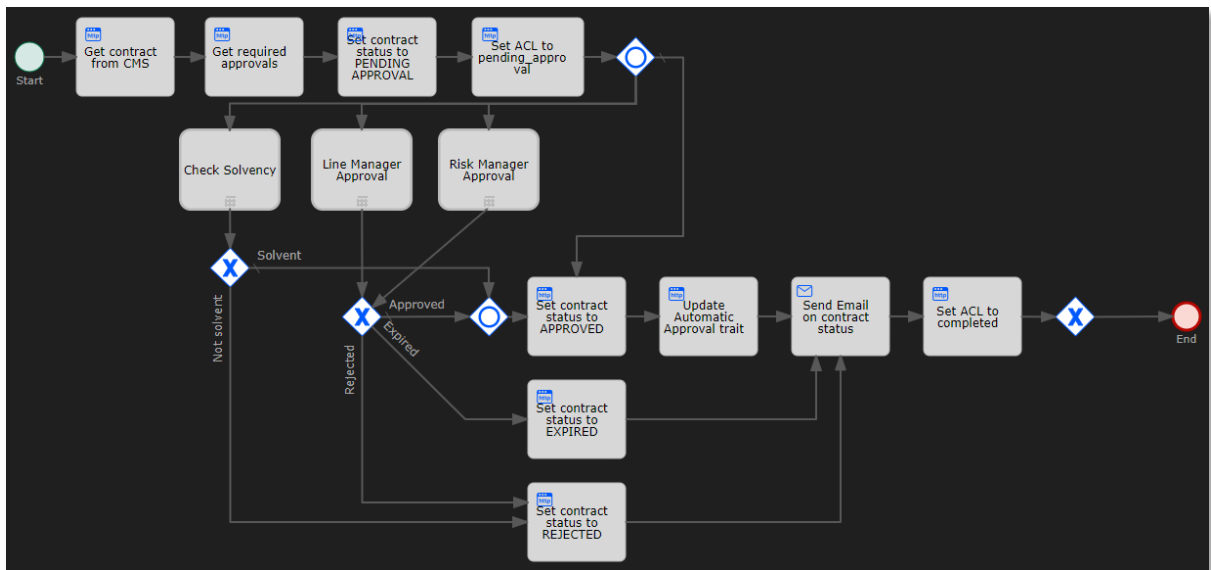
General details

Id

Display name

Execution details

Execution listeners



70. Select the new sequence flow between the third **exclusive gateway** and the **End event** and fill the sequence flow attributes using the following details:

Attribute	Value
Display name	This sequence flow is selected when the status property of the contract is equal to APPROVED . Enter the following display name: <input type="text" value="Approved"/>
Default flow	Choose to set this flow as the default flow.

Properties - Approved

General details

Display name

Connector details

* At least one input is required.

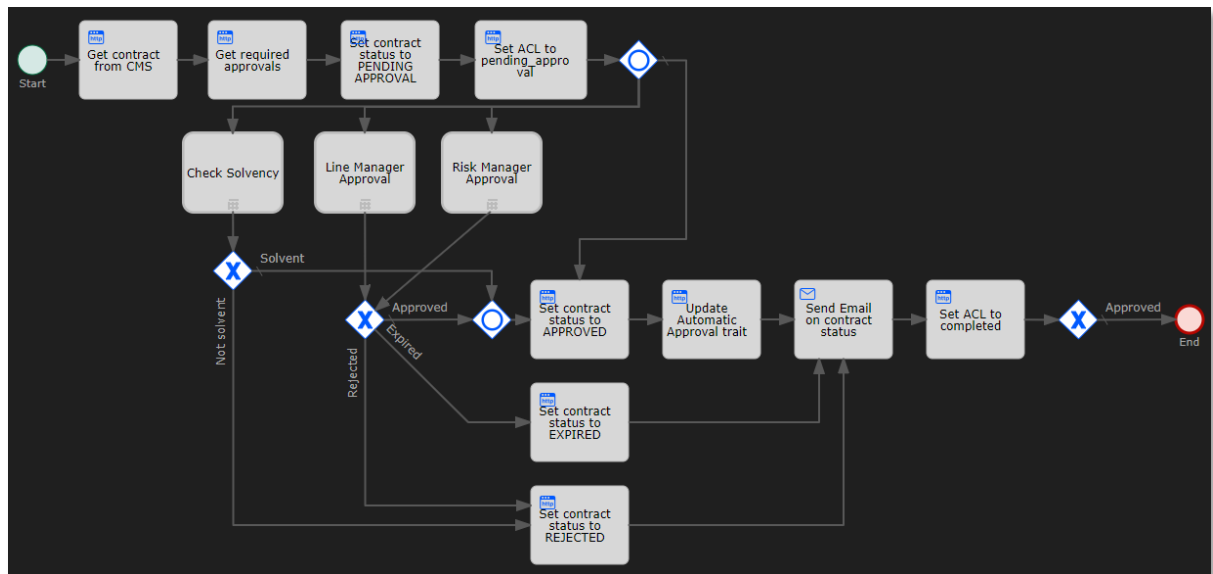
Flow condition
+ Add

Default flow
☒

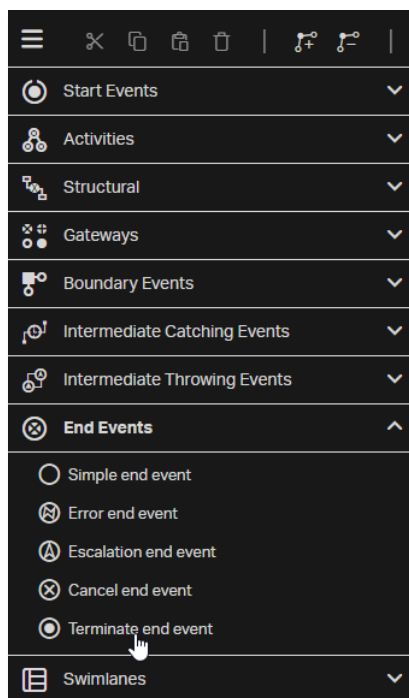
Skip condition

Execution details

Execution listeners
+ Add



71. In the **palette**, find **End events > Terminate end event**.

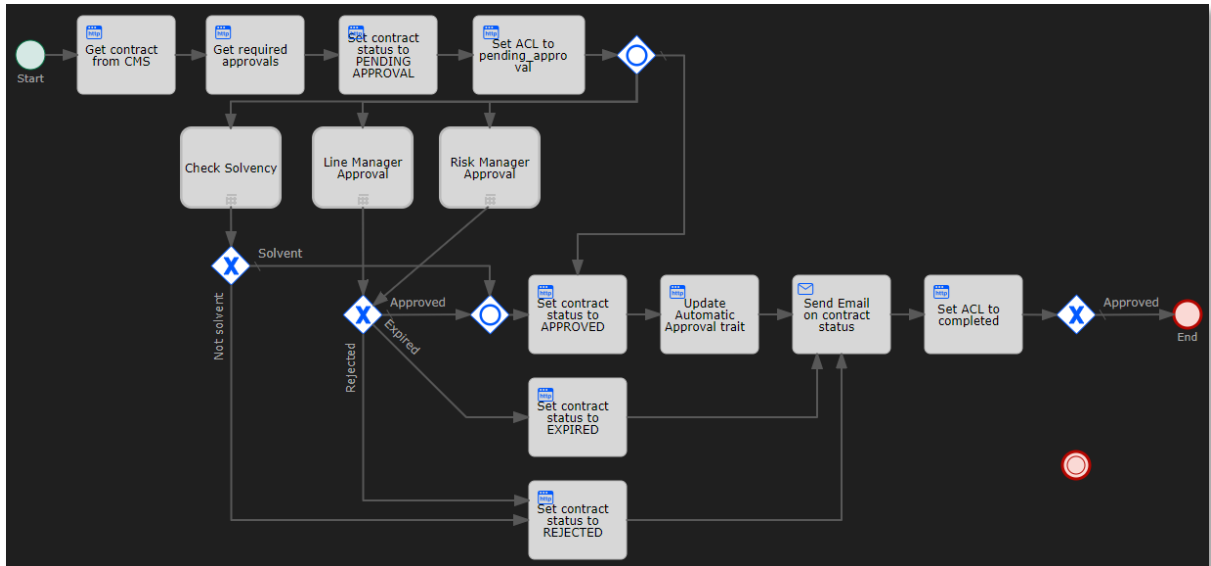


Note

The terminate end event is mostly used with parallel or inclusive gateways. While a normal (untyped) end event indicates that a single process sequence ends, the terminate end event ends the whole process and thereby, ends every activity that may be running at that time.

For example, in the contract approval workflow, if any of the manager approval tasks is still pending, the terminate end event results in terminating the sub processes and corresponding activities. This is required to ensure cleaning up in case of one of the selected required approvals marking the contract as EXPIRED or REJECTED.

72. Drag and drop the **Terminate end event** to the **canvas** under the third **exclusive gateway**. Leave enough space between the exclusive gateway and the terminate end event for adding text to the connecting sequence flow.



73. Fill the Terminate end event attributes using the following details:

Attribute	Value
Display name	End and cancel pending approvals

Properties - End and cancel pending approvals

General details

Id

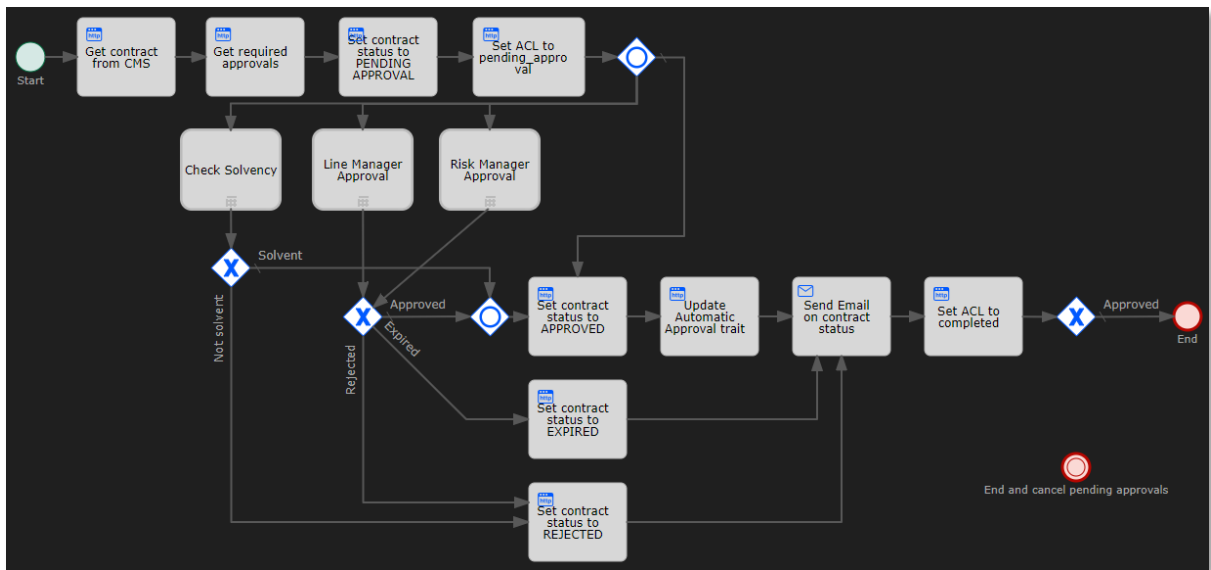
Display name

Termination details

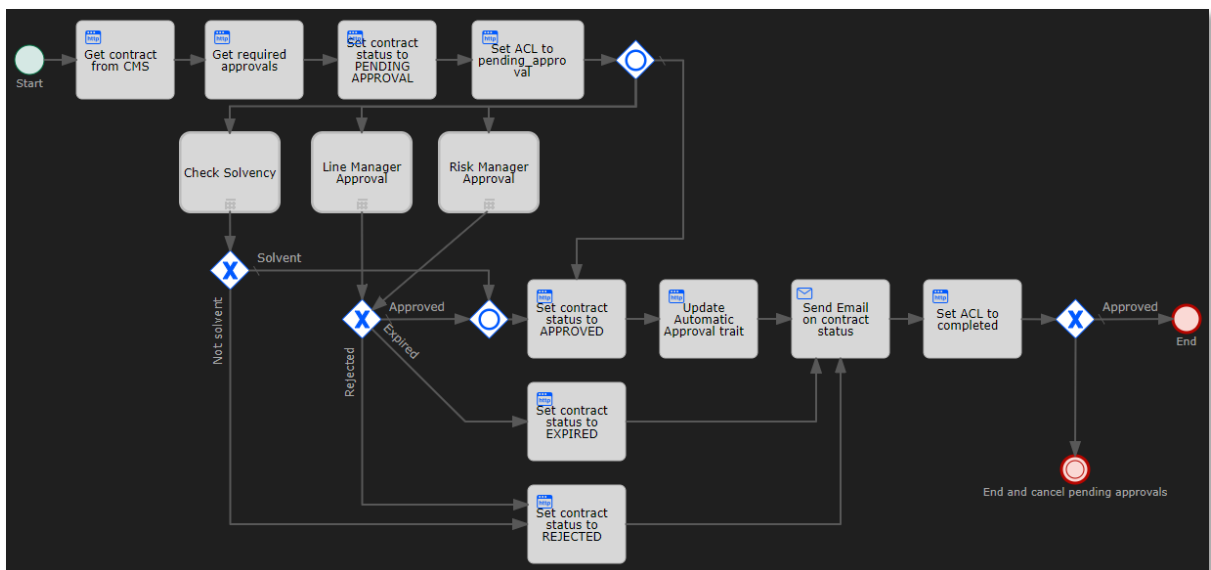
Terminate all
☒

Execution details

Execution listeners



74. Select the third **exclusive gateway** and drag a **sequence flow** to the **Terminate end event**.



75. Select the new sequence flow and fill the sequence flow attributes using the following details:

Attribute	Value
Display name	Not Approved
Flow condition	<p>This sequence flow is selected when the status property of the contract is not equal to APPROVED.</p> <p>Enter the following flow condition:</p> <pre>\${contract.properties.status != "APPROVED"}</pre>

Properties - Not Approved

General details

Display name

Connector details

* At least one input is required.

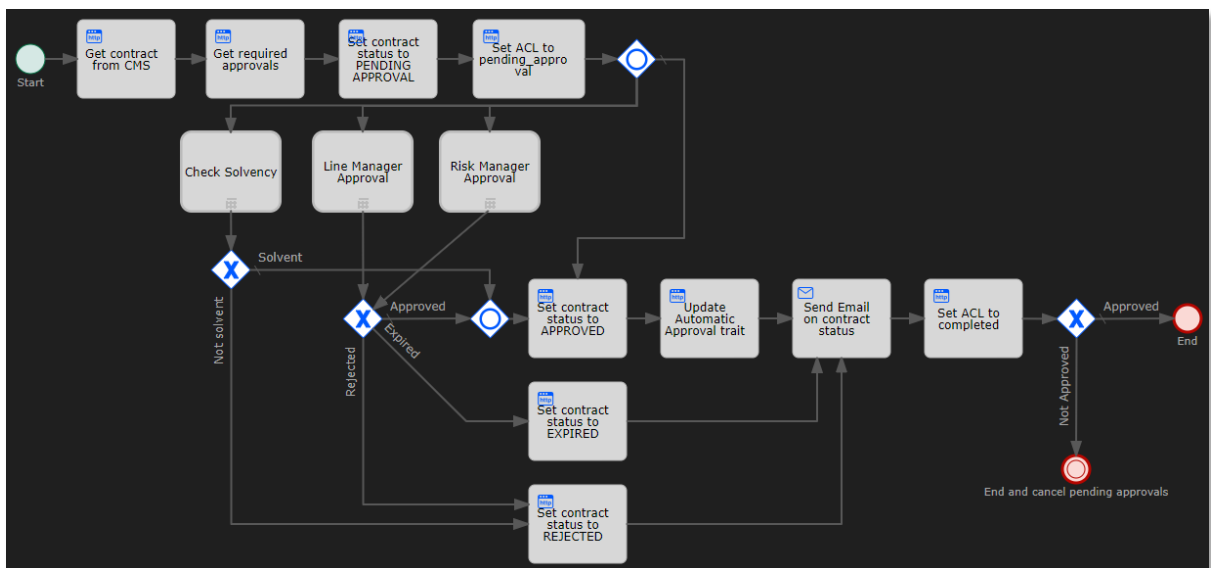
Flow condition
\${contract.propertie

Default flow
☒

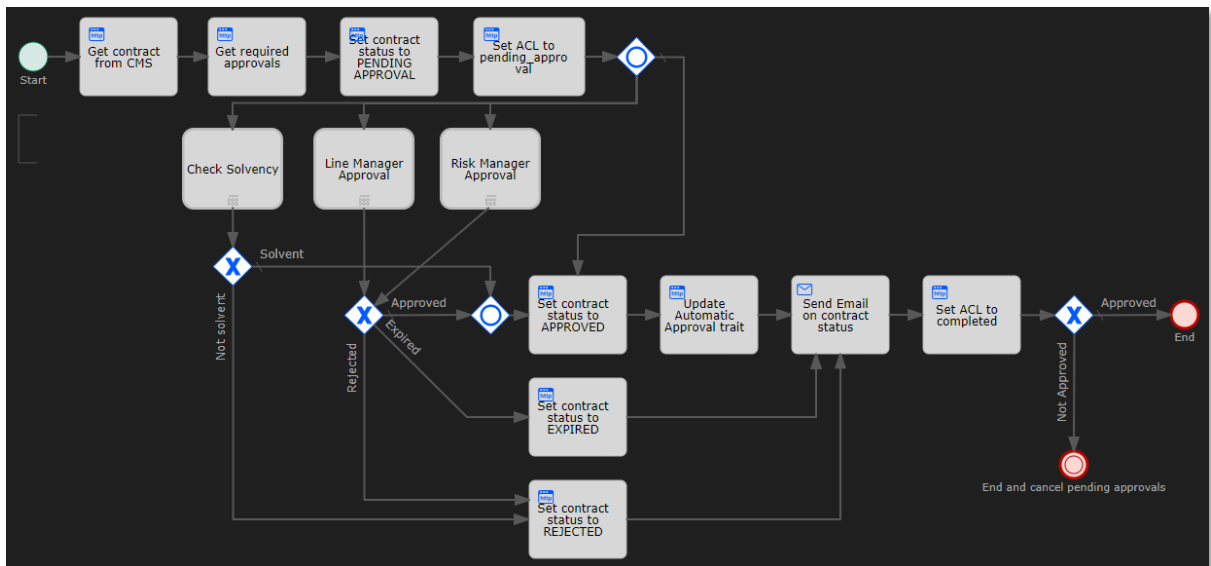
Skip condition

Execution details

Execution listeners



76. Drag and drop a **Text annotation** from the **palette** to the **canvas** under the **start event**.



77. Double-click the **Text annotation** on the **canvas** to edit it inline.

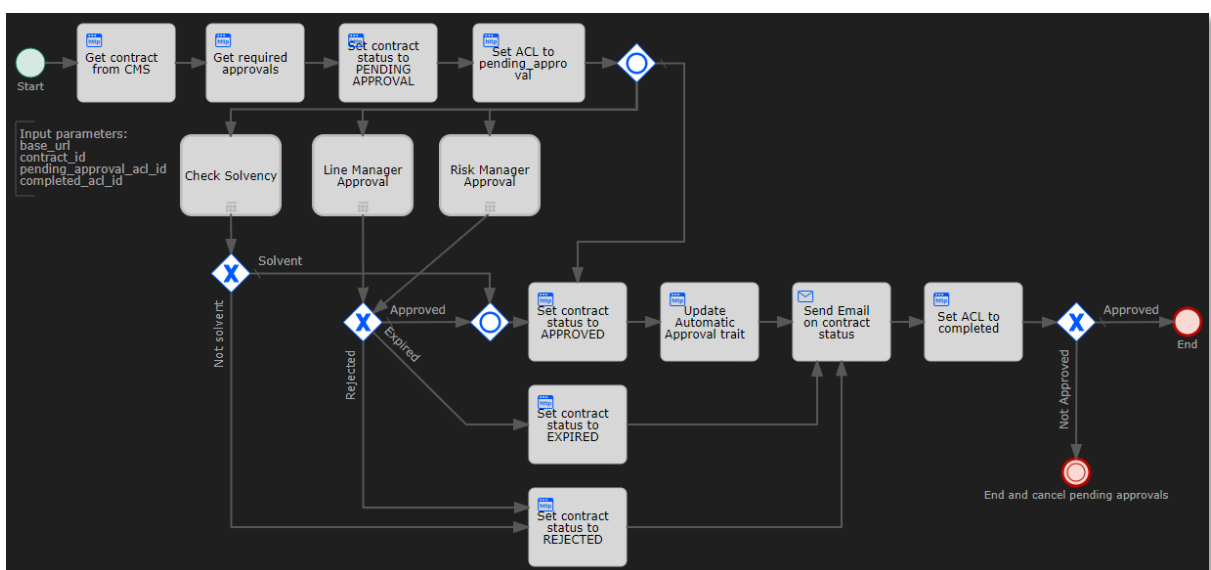
78. Set the following text for the text annotation:

Text

Input parameters:

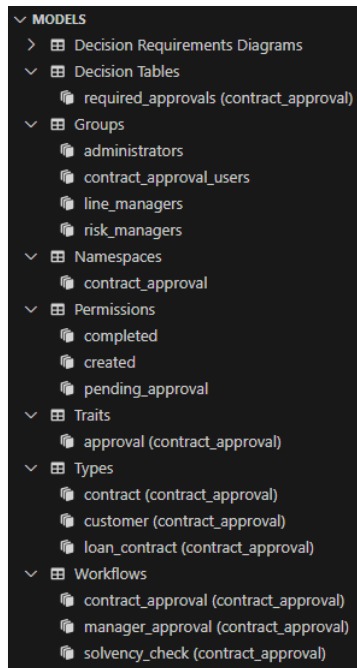
```
base_url
contract_id
pending_approval_acl_id
completed_acl_id
```

79. Resize the **Text annotation** so that the text displays correctly and fits the square brackets.



80. Save and close the Contract Approval workflow model.

81. The model explorer displays the new **contract_approval (contract_approval)** workflow under **Workflows**.



Next exercise module:


Deploy an application to the OpenText Thrust Services.

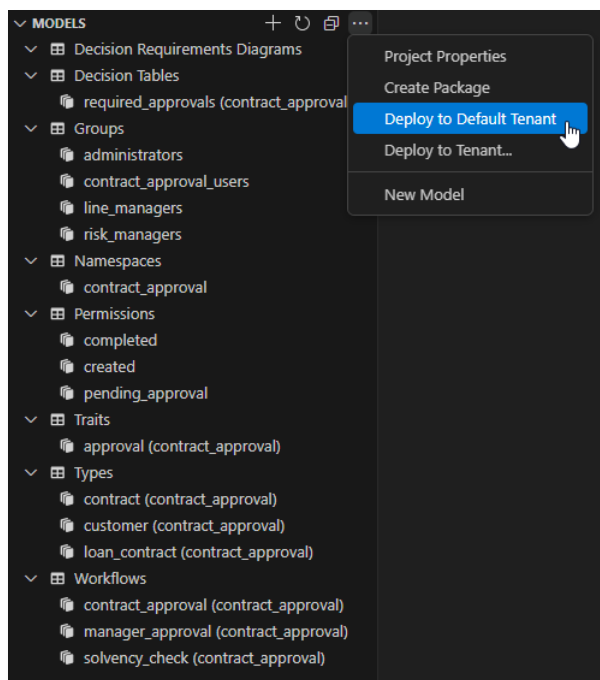
12 [25'] Deploy an application to the OpenText Thrust Services

Learn how to:

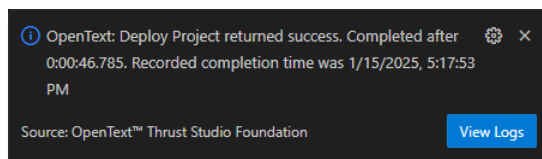
- Deploy an application project (with its models) to the OpenText Thrust Services and get the application credentials
- Verify that the application is deployed using Admin Center
- Add the redirect URLs for the deployed application using Admin Center
- Add the application users to the deployed groups using Admin Center

12.1 Deploy the Contract Approval application and get the application credentials

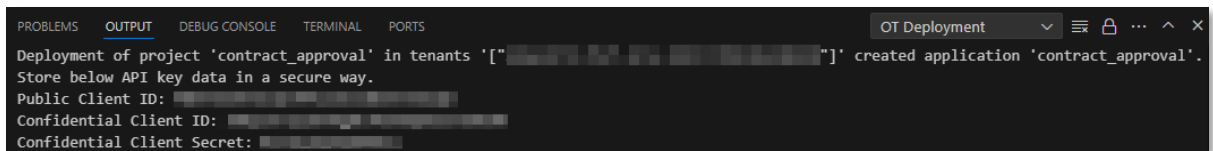
1. Open VS Code and, from the Activity Bar, select the **OpenText Thrust Studio** view.
2. In the **MODELS** section, click the **More Actions** button  and select **Deploy to Default Tenant** to deploy your application project.



After deployment, the success message is displayed.



The **OUTPUT** view automatically opens and displays the **OT Deployment** output.



```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
OT Deployment
Deployment of project 'contract_approval' in tenants '['...''] created application 'contract_approval'.
Store below API key data in a secure way.
Public Client ID: [...]
Confidential Client ID: [...]
Confidential Client Secret: [...]
  
```

3. Make note of the **Tenant ID** (value between '[' and ']') on the first line), the **Public Client ID**, the **Confidential Client ID**, and the **Confidential Client secret**. You require these values to test the deployment and run the application.

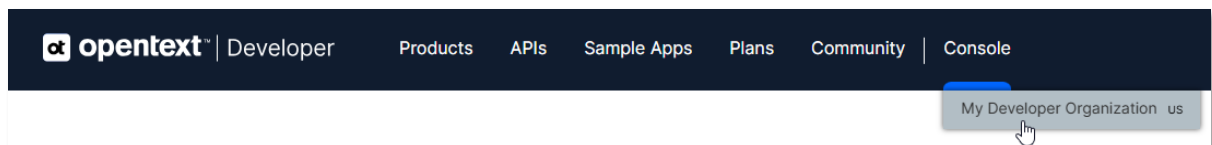


Note

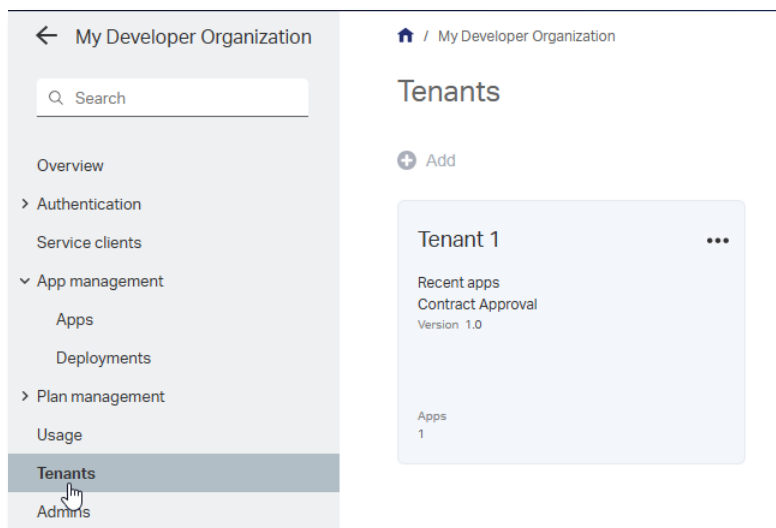
You can copy the above client credentials into a text file and save it. OpenText recommends that you store this type of sensitive information in a secure way.

12.2 Verify that the application is deployed

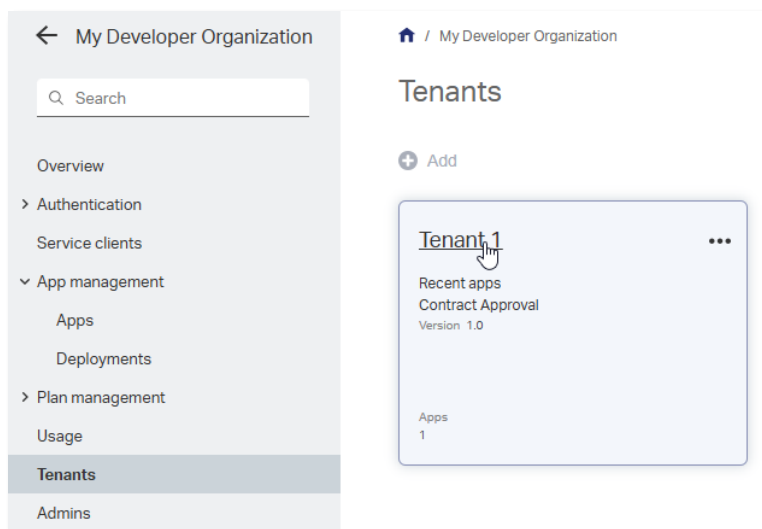
1. Sign in to the developer.opentext.com website.
2. Select the <organization name> <region> combination for your developer organization from the **Console** menu.



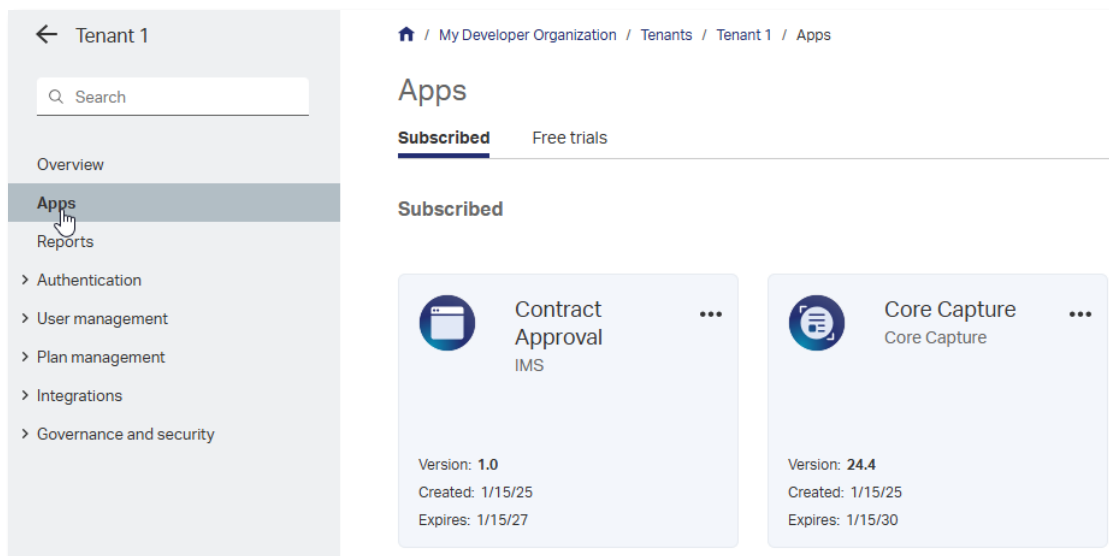
3. In the **Admin Center** navigation menu, click **Tenants**.



4. Select the tenant to which you deployed the application.



5. In the (tenant) navigation menu, click **Apps** and confirm the existence of the deployed **Contract Approval** app.



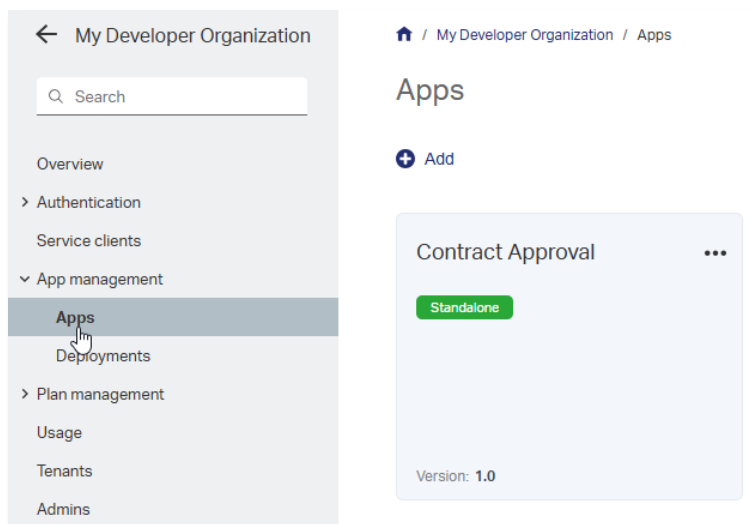
Next step:

Add the redirect URL for the application authentication flow.

12.3 Add the redirect URL for the application authentication flow

To authenticate, users of your Contract Approval application need to sign in using an external (to the application) login screen as part of the authentication (OAuth) flow. This means that the web browser in which you open the Contract Approval application will first be redirected to a login screen, and after it is authenticated, it will be redirected back to your Contract Approval application (which you will configure to run on <https://localhost:4000>). This redirect URL needs to be added at the organization level to the deployed application's public service client configuration (represented by the Public Client ID) for the authentication to work.

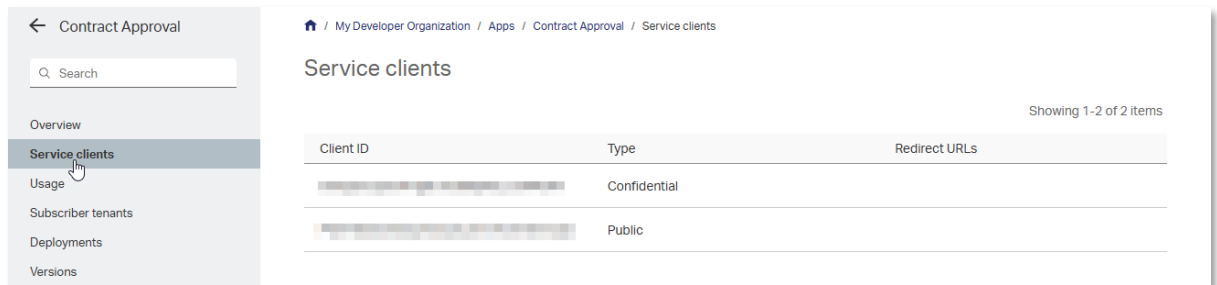
1. Sign in to developer.opentext.com.
2. Select the **<organization name> <region>** combination for your developer organization from the **Console** menu.
3. In the **Admin Center** navigation menu, click **Apps**.




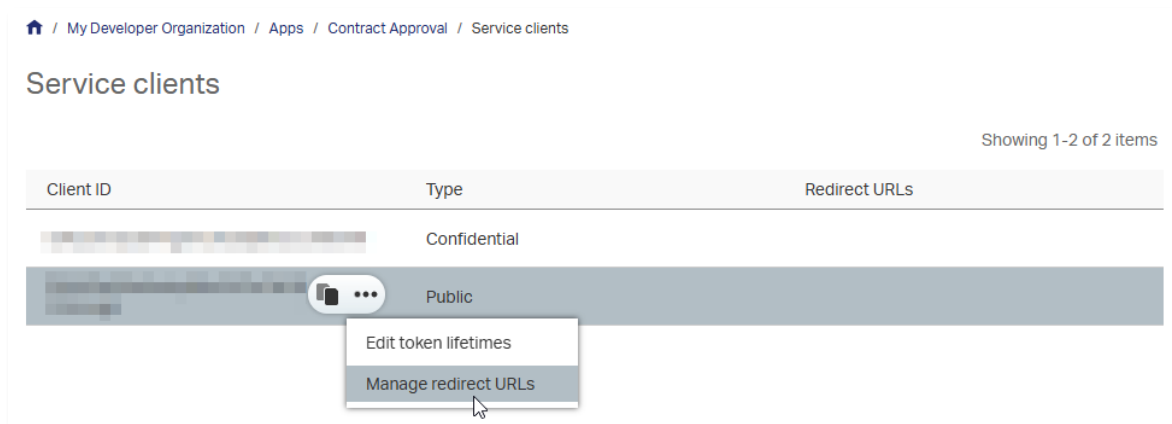
4. Click on the **Contract Approval** app tile.



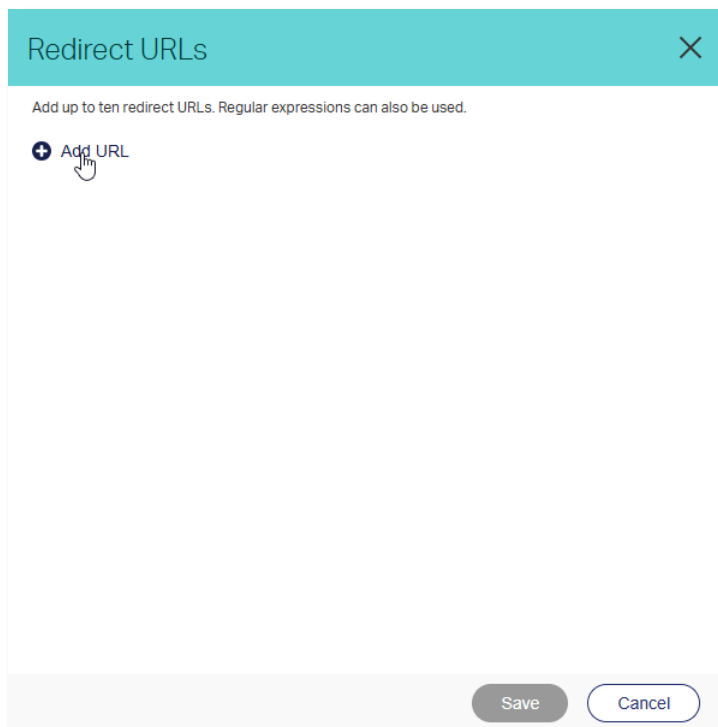
5. In the **Contract Approval** app navigation menu, click **Service clients**.



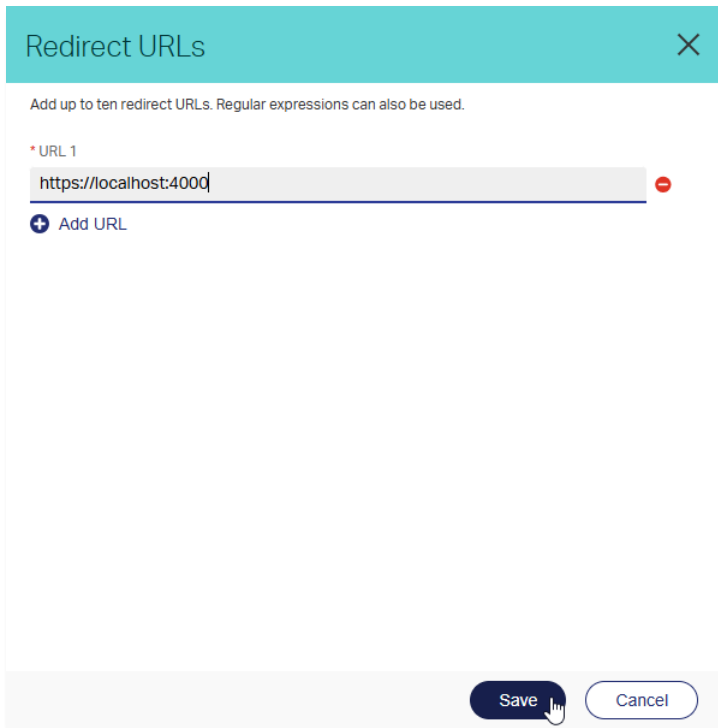
6. Hover over the **Public** client entry in the client list and click the **More** button  select **Manage redirect URLs**.



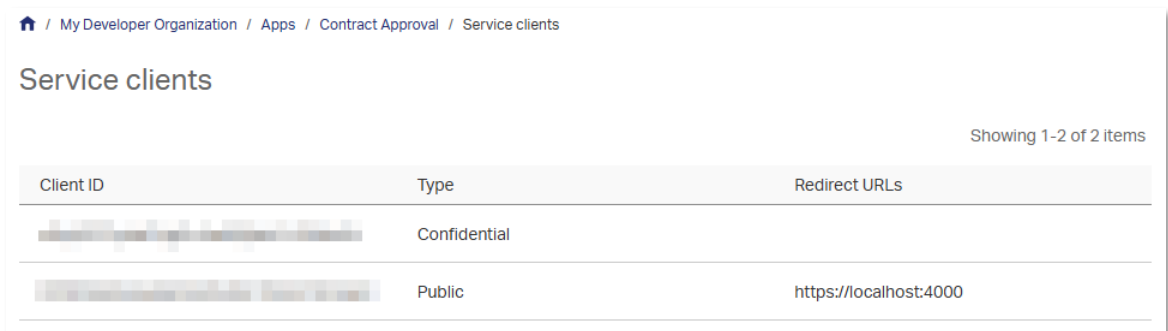
7. In the **Redirect URLs** screen, click **Add URL**.



8. In the URL 1 box, enter `https://localhost:4000` and click **Save**.



The dialog box is titled "Redirect URLs" with a close button (X) in the top right corner. Below the title bar, it says "Add up to ten redirect URLs. Regular expressions can also be used." There is a red asterisk followed by "URL 1" above a text input field. The input field contains the text "https://localhost:4000" and has a red minus button to its right. Below the input field is a blue plus button followed by the text "Add URL". At the bottom of the dialog are two buttons: "Save" and "Cancel".



The screenshot shows a breadcrumb trail: [Home](#) / [My Developer Organization](#) / [Apps](#) / [Contract Approval](#) / [Service clients](#). Below the breadcrumb is the heading "Service clients" and a status "Showing 1-2 of 2 items". A table with three columns: "Client ID", "Type", and "Redirect URLs" contains two rows. The first row has a blurred Client ID, "Confidential" Type, and an empty Redirect URLs field. The second row has a blurred Client ID, "Public" Type, and "https://localhost:4000" in the Redirect URLs field.

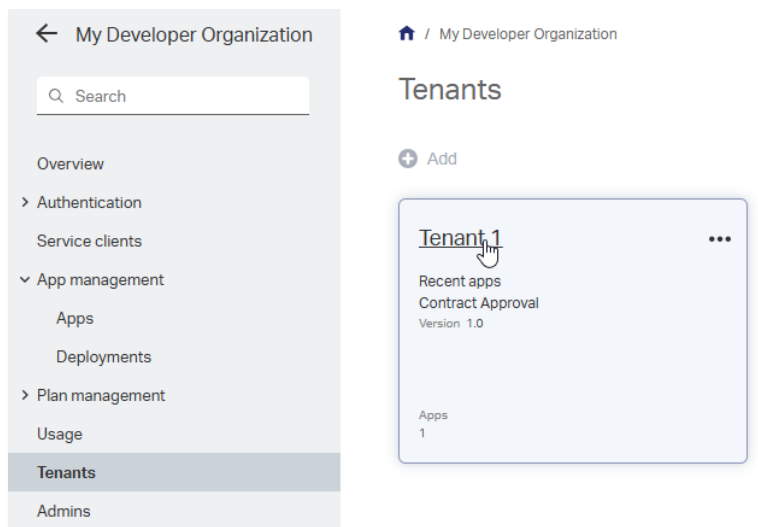
Client ID	Type	Redirect URLs
[blurred]	Confidential	
[blurred]	Public	https://localhost:4000

Next step:

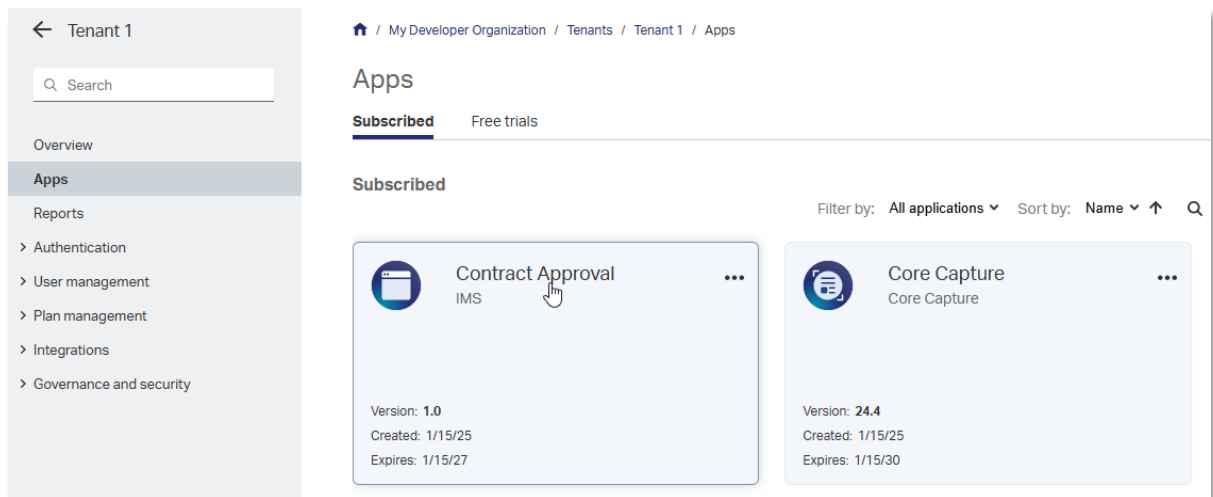
Add the application users to the application groups.

12.4 Add the application users to the application groups

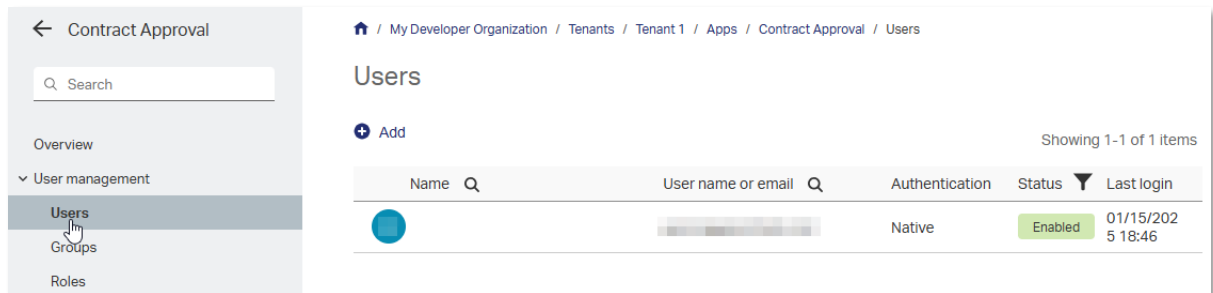
1. Sign in to developer.opentext.com.
2. Select the <organization name> <region> combination for your developer organization from the **Console** menu.
3. In the **Admin Center** navigation menu, navigate to **Tenants** and select the tenant to which you deployed the application.



4. In the (tenant) navigation menu, go to **Apps** and click on the **Contract Approval** app tile.



5. In the (tenant app) navigation menu, go to **User management > Users**.



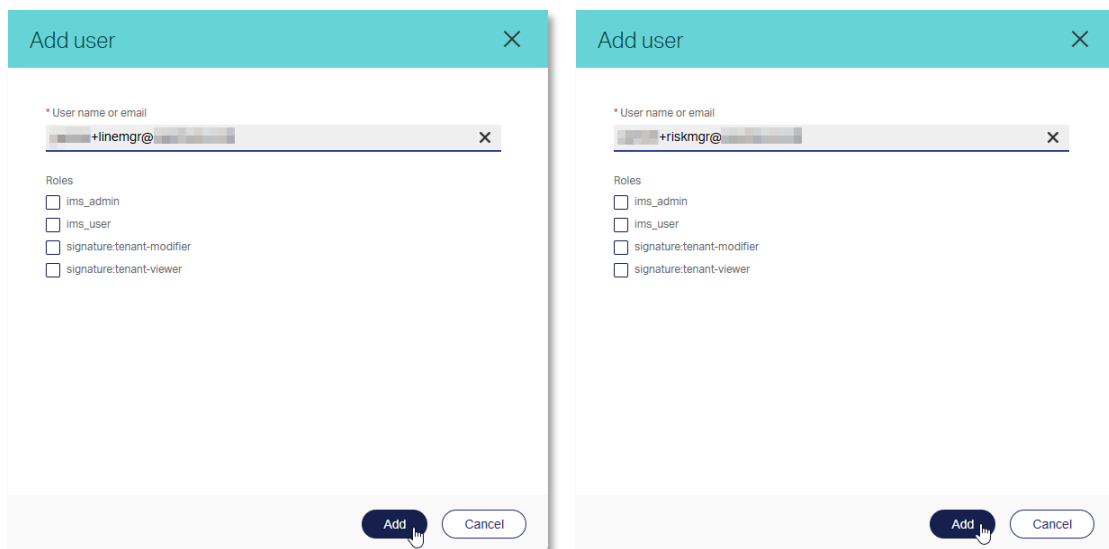
The user used for subscribing to the trial or developer plan is the only user listed in the **Users** list.

6. Click **Add** and select **Add user** to add two additional users (email accounts) to represent the line manager and risk manager as approvers.
7. From the Add user dialog box, in the **User name or email** box, to represent a role specific email address, enter the email address used for subscribing to the trial or developer plan, but add a “+” and the role (“linemgr” or “riskmgr”) right before the “@”.

Using a “+” in an email address is very useful as it allows differentiating recipients/email addresses while delivering them all to the same email address (the original address without the “+” and text before “@”).

So, for example, if the email address used to subscribe to the trial or developer plan is **myname@somedomain.com**, applying the approach of adding a “+” with the role results in the following two emails to represent the **line manager** and **risk manager**:

- Line manager: **myname+linemgr@somedomain.com**
- Risk manager: **myname+riskmgr@somedomain.com**



Note

Only two new users need to be added. For the **regular user** (that is, the approval requester), the email address used for subscribing to the trial or developer plan will be used.

Users

[+ Add](#)

Showing 1-3 of 3 items

Name	User name or email	Authentication	Status	Last login
	+linemgr@	Native	Invited	
	+riskmgr@	Native	Invited	
		Native	Enabled	01/15/2025 18:46

8. Go to your email client for the developer trial subscription email account and locate the two emails with the Subject: **Register your new OpenText IMS account**.

opentext™

Register to get started with IMS!

Please complete the registration process by entering your name and establishing a password using the link below.

[Complete Registration](#)

Need help?

Please contact [OpenText Support](#) or visit the [OpenText Community Forums](#).

This is a service communication sent from an unmonitored e-mail address.
Please do not reply to this message.
Copyright © 2024 OpenText. All rights reserved.
Trademarks owned by OpenText.

9. For each email, click **Complete Registration** and fill the **OpenText Cloud Platform** registration form using the following details:

Field	Value
First name	Thrust Services Tutorial
Last name	Use the application role as value for the last name field. Depending on the email address in the Email field of the form, fill the correct application role, that is: Line Manager or Risk Manager
Email	The email address for the specific approver role and it appears automatically.
Password	Specify the password.
Confirm password	Confirm the password.




10. Click **I accept** to accept the terms and proceed with the registration of the two new users.

11. Go back to **Admin Center** and refresh the **User management > Users** page to confirm that the status of the two newly added users is **Enabled**.

🏠 / My Developer Organization / Tenants / Tenant 1 / Apps / Contract Approval / Users

Users

+ Add Showing 1-3 of 3 items

	Name 🔍	User name or email 🔍	Authentication	Status ▼	Last login
	Thrust Services Tutorial Line Manager	+linemgr@	Native	Enabled	01/15/2025 19:01
	Thrust Services Tutorial Risk Manager	+riskmgr@	Native	Enabled	01/15/2025 19:04
			Native	Enabled	01/15/2025 18:46

12. In the (tenant app) navigation menu, select **User management > Groups**.

← Contract Approval

🔍 Search

- Overview
- ▼ User management
 - Users
 - Groups**
 - Roles
 - Admins
- > Metrics
- > Plan management
- > Integrations

🏠 / My Developer Organization / Tenants / Tenant 1 / Apps / Contract Approval / Groups

Groups

+ Add Showing 1-4 of 4 items

Group name 🔍	Description
administrators	Group containing the administrators for the Contract Approval application
contract_approval_users	Group containing the users that are allowed to use the Contract Approval application
line_managers	Group containing the users that have the Line Manager role
risk_managers	Group containing the users that have the Risk Manager role

13. From Groups, select **line_managers**.

Groups

+ Add Showing 1-4 of 4 items

Group name 🔍	Description
administrators	Group containing the administrators for the Contract Approval application
contract_approval_users	Group containing the users that are allowed to use the Contract Approval application
line_managers	Group containing the users that have the Line Manager role
risk_managers	Group containing the users that have the Risk Manager role

14. In the **Members** section, click **Add > Add user** to add the line manager approver as a group member.

← line_managers

General Members

General

Group name
line_managers

Description
Group containing the users that have the Line Manager role

+ Add

Add user
Add group
Add service client

<input type="checkbox"/>	Name	Username/Client ID	Type
--------------------------	------	--------------------	------

15. In the **User name or mail** box, enter the **line manager approver email** and click **Add**.

Add user

* User name or email

+linemgr@


Add another Add Cancel

Members


+ Add





Showing 1-1 of 1 items

<input type="checkbox"/>	Name	Username/Client ID	Type
<input type="checkbox"/>	Thrust Services Tutorial Line Manager	+linemgr@	User


16. Click  (back arrow icon) to go back to the **User management > Groups** list.
17. Click the **risk_managers** group.

Groups

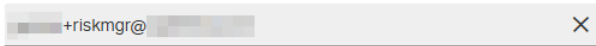

 Add Showing 1-4 of 4 items



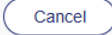
Group name 	Description
administrators	Group containing the administrators for the Contract Approval application
contract_approval_users	Group containing the users that are allowed to use the Contract Approval application
line_managers	Group containing the users that have the Line Manager role
risk_managers 	Group containing the users that have the Risk Manager role  

18. In the **Members** section, click **Add > Add user** to add the risk manager approver as a group member.
19. Enter the **risk manager approver email** and click **Add**.


Add user 



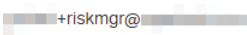
* User name or email


 +riskmgr@ 

Members

 Add Showing 1-1 of 1 items

<input type="checkbox"/>	Name	Username/Client ID 	Type
<input type="checkbox"/>	 Thrust Services Tutorial Risk Manager	 +riskmgr@	User

20. Click  (back arrow icon) to go back to the **User management > Groups** list.

21. Click the **contract_approval_users** group.

Groups

+ Add Showing 1-4 of 4 items

Group name	Description
administrators	Group containing the administrators for the Contract Approval application
contract_approval_users	Group containing the users that are allowed to use the Contract Approval application
line_managers	Group containing the users that have the Line Manager role
risk_managers	Group containing the users that have the Risk Manager role

22. In the **Members** section, click **Add > Add user** to add the regular user (that is, the user requesting the contract approval and not having line manager or risk manager approval privileges).

← contract_approval_users

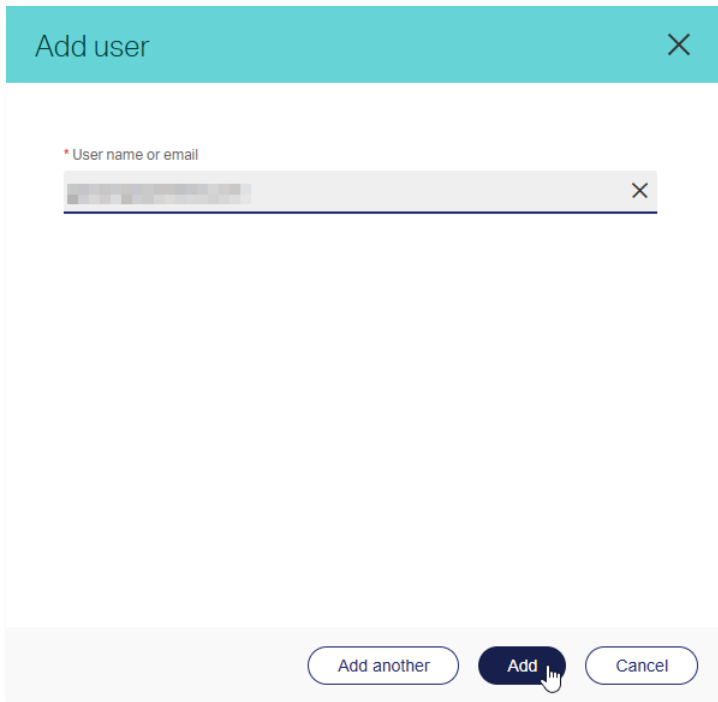
General **Members**

Members

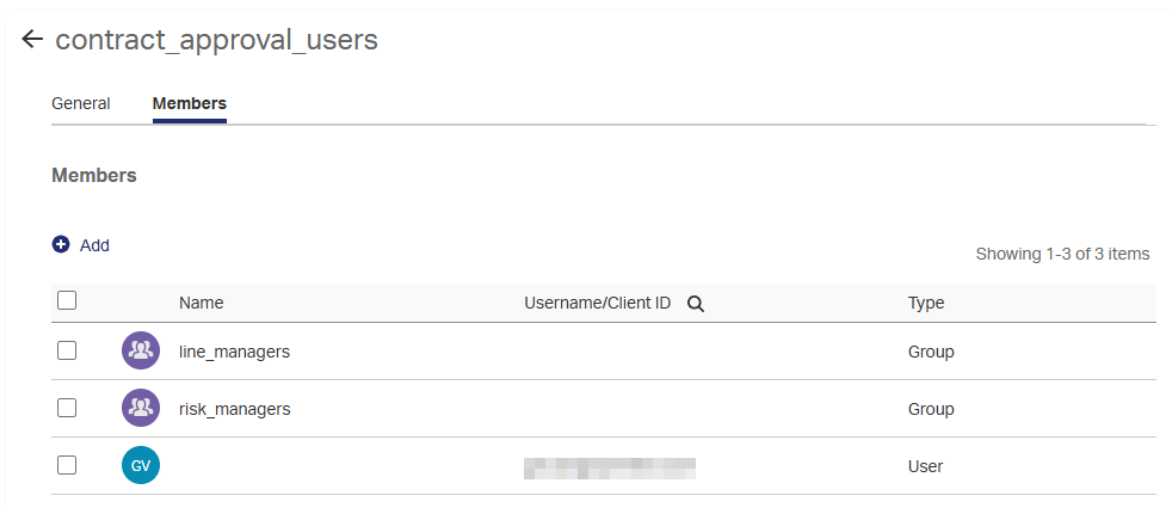
+ Add Showing 1-2 of 2 items

Username/Client ID	Type
	Group
<input type="checkbox"/> risk_managers	Group





23. Enter the **regular user email** (the email address used for subscribing to the trial or developer plan) and click **Add**.



The 'Add user' dialog box features a teal header with the title 'Add user' and a close button (X). Below the header is a text input field with a red asterisk and the label 'User name or email'. The input field contains a blurred email address and a clear button (X). At the bottom of the dialog, there are three buttons: 'Add another', 'Add' (highlighted with a mouse cursor), and 'Cancel'.



The 'contract_approval_users' page shows the 'Members' tab. It includes a '+ Add' button and a 'Showing 1-3 of 3 items' indicator. The table lists three members: 'line_managers' (Group), 'risk_managers' (Group), and 'GV' (User). The 'GV' user's email address is blurred.

<input type="checkbox"/>	Name	Username/Client ID	Type
<input type="checkbox"/>	 line_managers		Group
<input type="checkbox"/>	 risk_managers		Group
<input type="checkbox"/>	 GV		User

Next exercise module:

Work with the OpenText Thrust Services APIs.

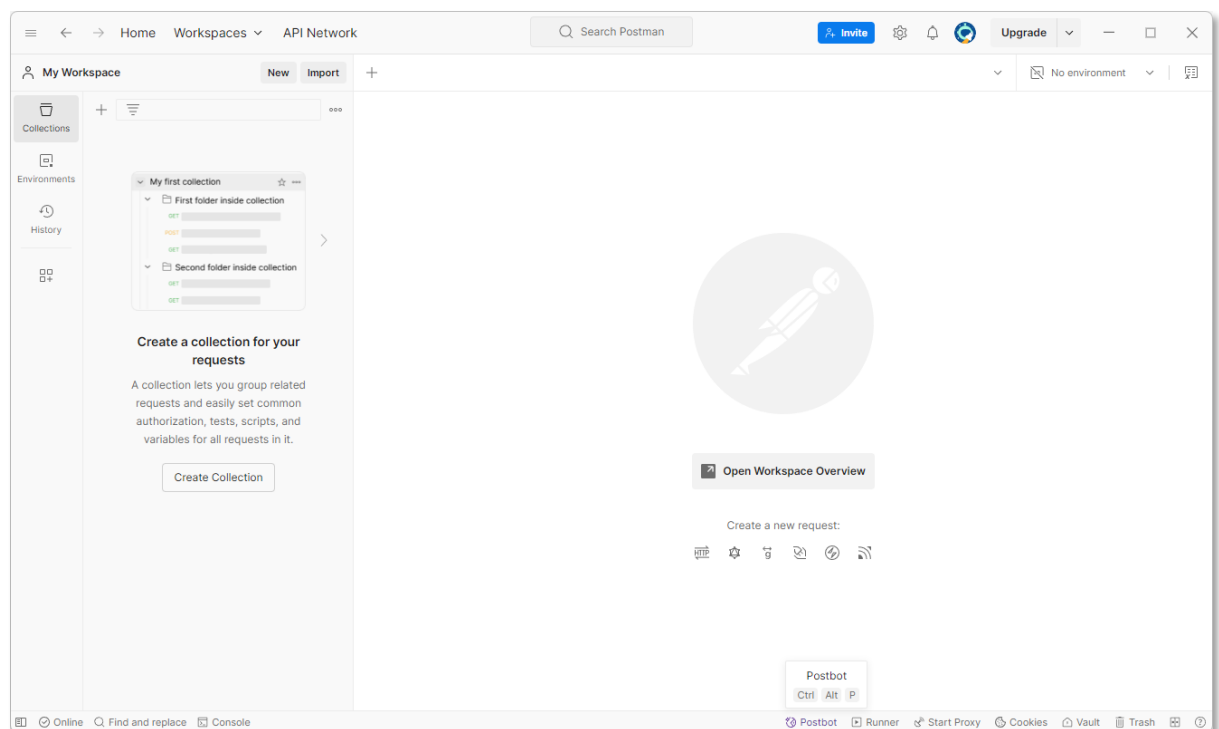
13 [25'] Work with the OpenText Thrust Services APIs

Learn how to:

- Download and install Postman
- Download the finished Contract Approval App from GitHub
- Import the Postman collection and environment
- Verify the deployment of the different models using the OpenText Thrust Services APIs through Postman

13.1 Download and install Postman

1. Download and install postman from <https://www.postman.com/downloads>.
2. Sign in or create an account to allow using collections.



Next step:

Download the Contract Approval application.

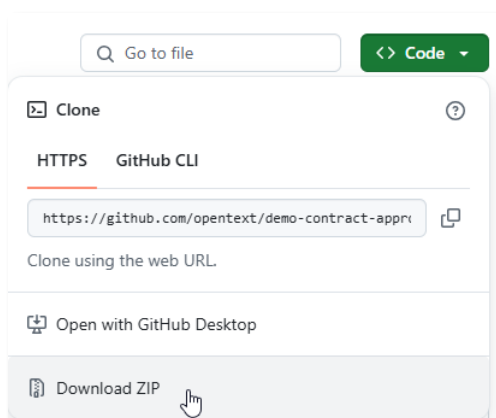
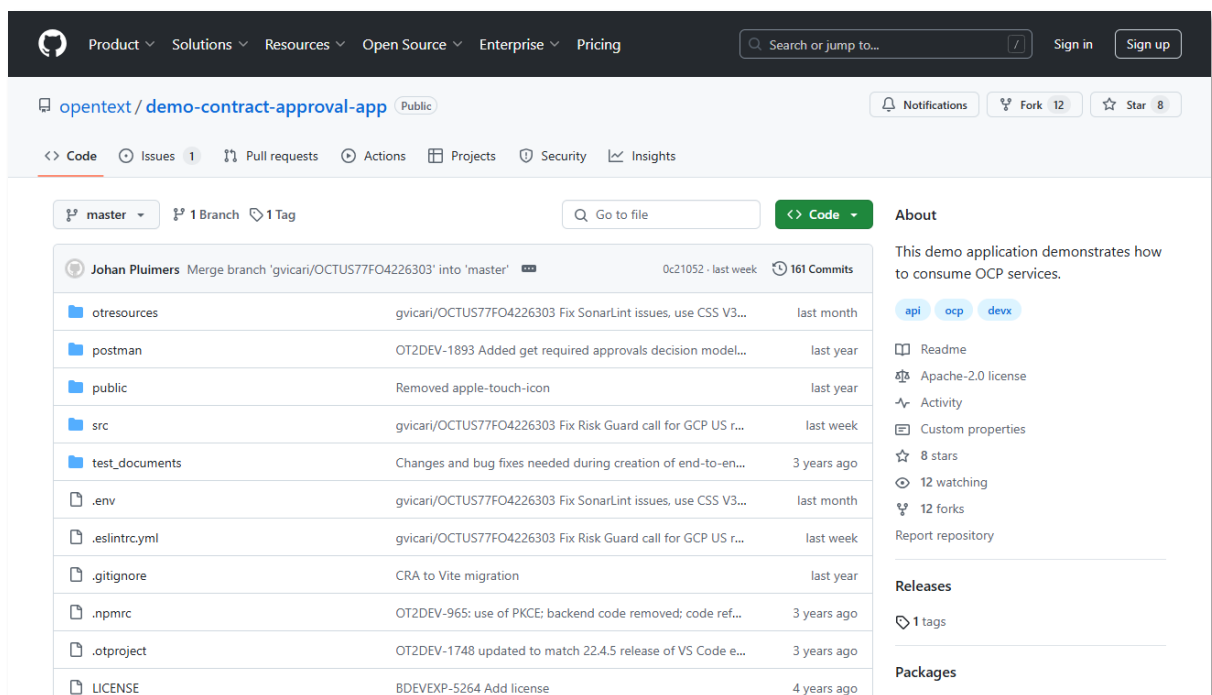
13.2 Download the Contract Approval application

1. Go to web address <https://github.com/opentext/demo-contract-approval-app> and download the Contract Approval App (demo-contract-approval-app) from GitHub.

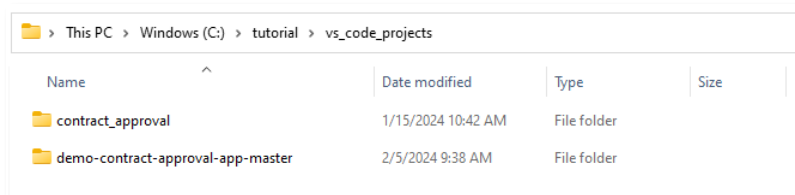


Note

The latest version of the Contract Approval App is available from GitHub. For this tutorial, you will use “Download ZIP” to download the application. To ensure having the latest version of the Contract Approval App, you can download the ZIP at any time, but you can also choose to clone the repository and regularly perform a git pull.



2. Extract the **demo-contract-approval-app-master** folder from the downloaded ZIP file to your file system (for example, in the same folder as the tutorial project).

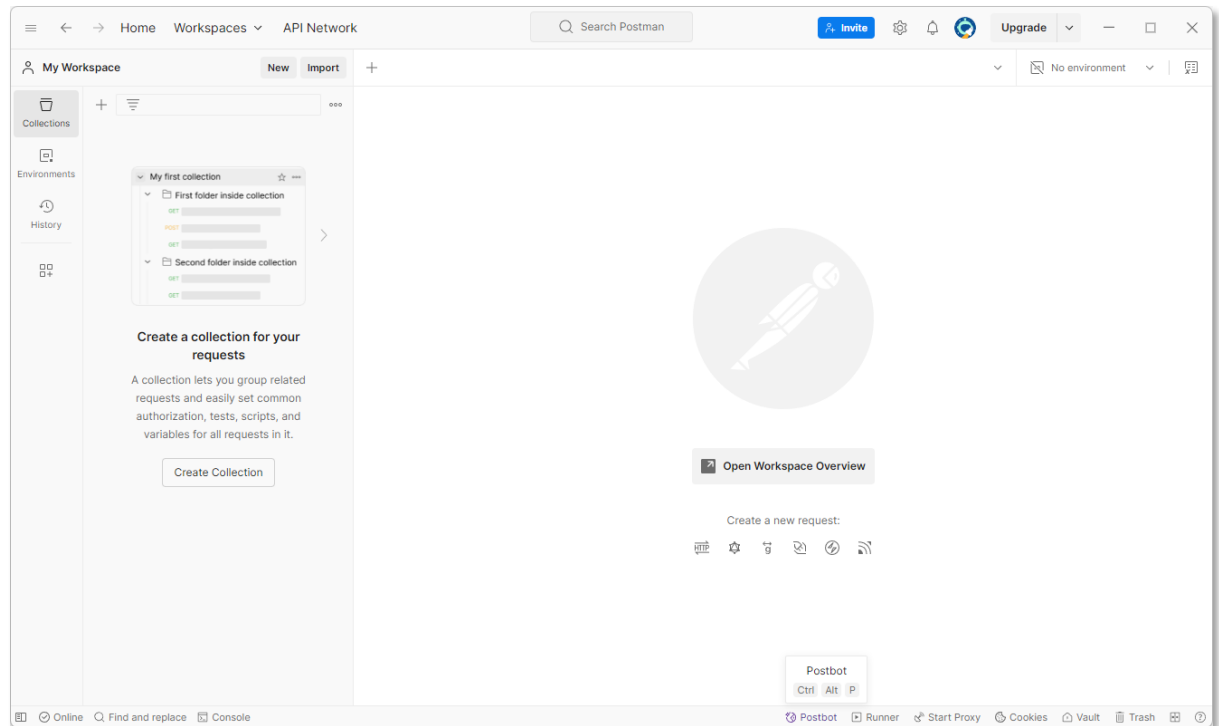


Next step:

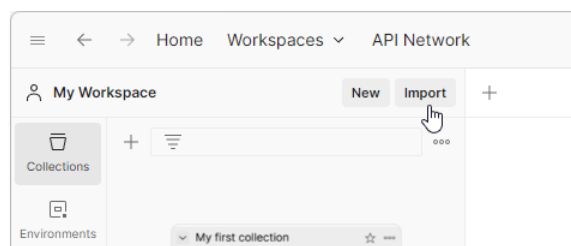
Import the Postman collection and environment into Postman.

13.3 Import the Postman collection and environment into Postman

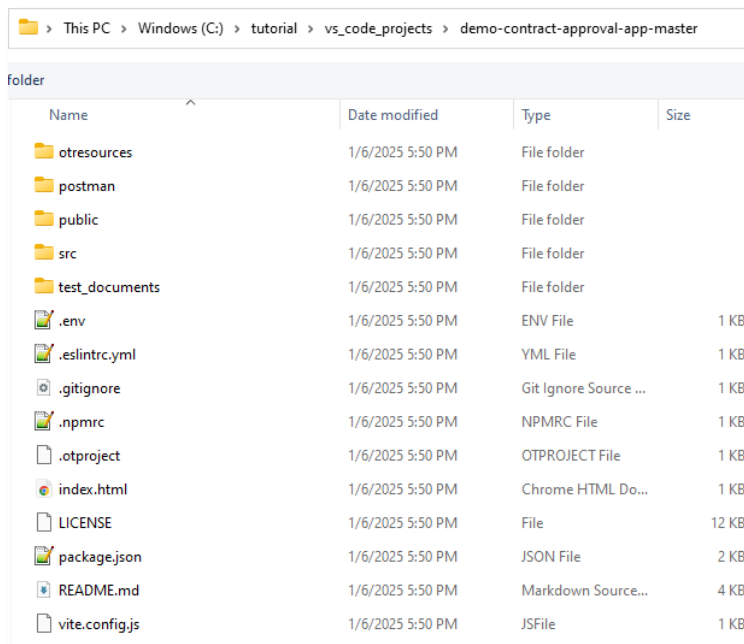
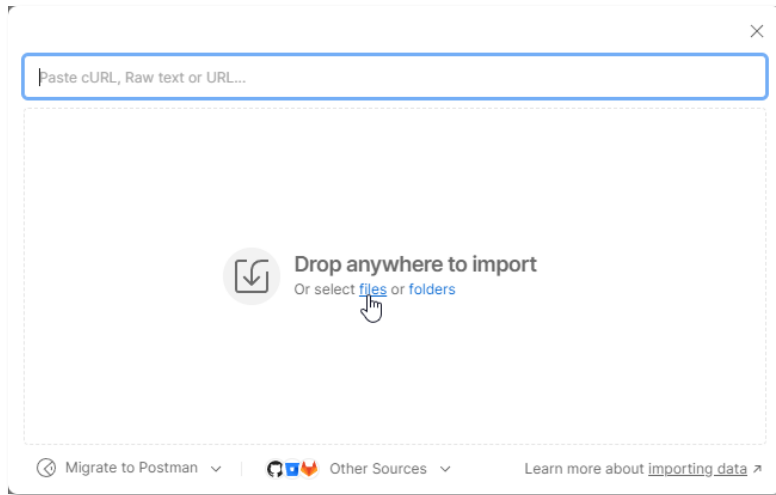
1. Open **Postman**.



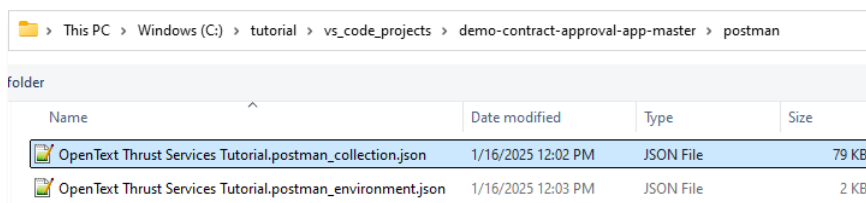
2. Select **Collections** from the sidebar and click **Import**.



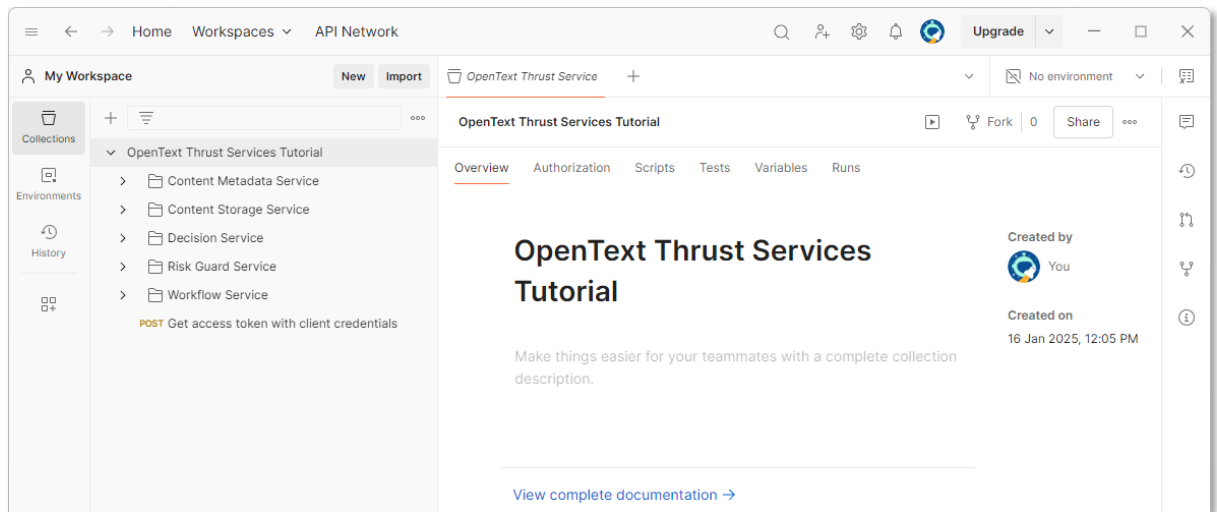
- From the import dialog box, click to select **files** and navigate to the previously extracted **demo-contract-approval-app-master** folder.



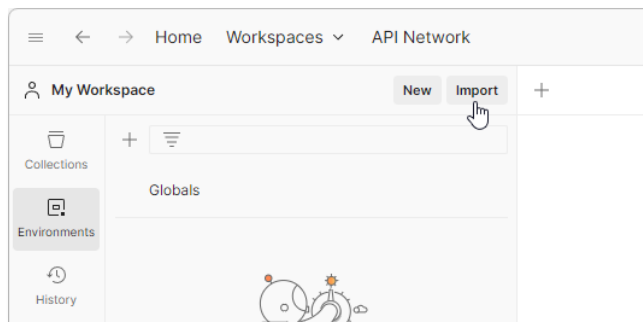
- In the **demo-contract-approval-app-master** folder, navigate to the **/postman** folder and select the **OpenText Thrust Services Tutorial.postman_collection.json** collection file.



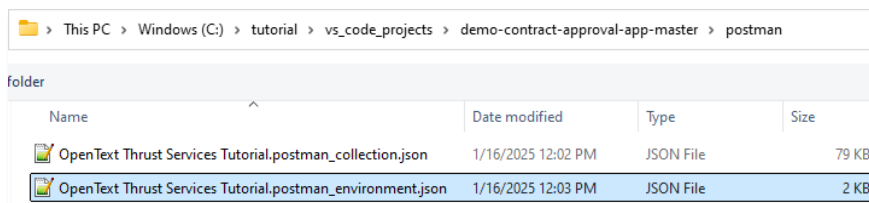
- Click **Open** to import the **OpenText Thrust Services Tutorial** collection into Postman.



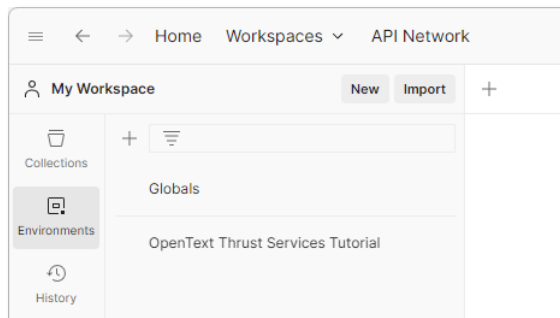
- Select **Environments** from the Postman sidebar and click **Import**.



- From the import dialog box, click to select **files** and navigate to the previously extracted **demo-contract-approval-app-master** folder.
- In the **demo-contract-approval-app-master** folder, navigate to the **/postman** folder and select the **OpenText Thrust Services Tutorial.postman_environment.json** environment file.



9. Click **Open** to import the **OpenText Thrust Services Tutorial** environment into Postman.

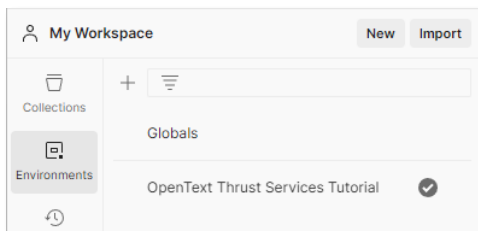
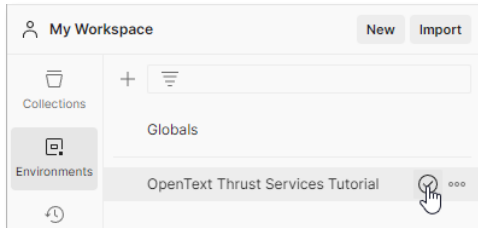


Next step:

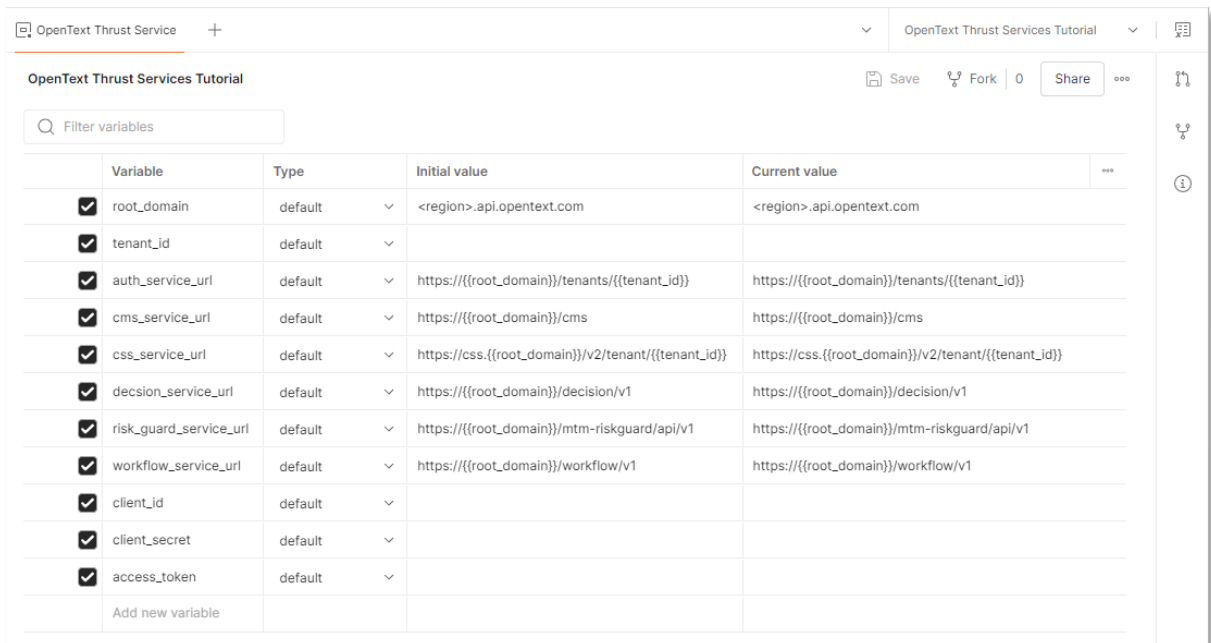
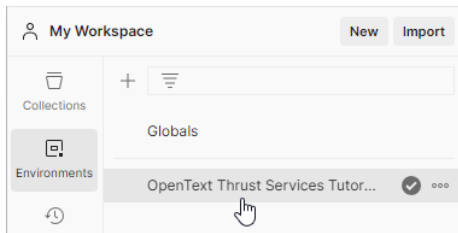
Verify the deployment of the application models using the OpenText Thrust Services APIs.

13.4 Verify the deployment of the application models using the OpenText Thrust Services APIs

1. Select **Environments** from the Postman sidebar and In Postman and set the **OpenText Thrust Services Tutorial** environment as the active environment.



2. Select the **OpenText Thrust Services Tutorial** to fill the environment variables.



3. Use the following details to fill the OpenText Thrust Services Tutorial environment's environment variables with the values that correspond with the deployed application:

Field	Value
root_domain	Replace the <region> placeholder in <region>.api.opentext.com with the region of your developer subscription. For this tutorial we use us as the region, but if your developer subscription (trial or other) is in a different region, you must use that region.
tenant_id	Use the tenant id you saved after deploying the application project (that is, the text value between "[" and "] " on the first line).
client_id	Use the Confidential Client ID you saved after deploying the application project.
client_secret	Use the Confidential Client Secret you saved after deploying the application project.



Note

Make sure to fill both the **Initial value** and **Current value** columns for each of the environment variables (the ones that are not mentioned, you must leave as is). The initial value can be used if you want to reset the environment variables to their initial value, and the current value is the environment variable value that will be used when performing API calls.

OpenText Thrust Services Tutorial					Save	Fork	0	Share	...
Q Filter variables									
	Variable	Type		Initial value	Current value				
<input checked="" type="checkbox"/>	root_domain	default	▼	us.api.opentext.com	us.api.opentext.com				
<input checked="" type="checkbox"/>	tenant_id	default	▼	[REDACTED]	[REDACTED]				
<input checked="" type="checkbox"/>	auth_service_url	default	▼	https://{{root_domain}}/tenants/{{tenant_id}}	https://{{root_domain}}/tenants/{{tenant_id}}				
<input checked="" type="checkbox"/>	cms_service_url	default	▼	https://{{root_domain}}/cms	https://{{root_domain}}/cms				
<input checked="" type="checkbox"/>	css_service_url	default	▼	https://css.{{root_domain}}/v2/tenant/{{tenant_id}}	https://css.{{root_domain}}/v2/tenant/{{tenant_id}}				
<input checked="" type="checkbox"/>	decision_service_url	default	▼	https://{{root_domain}}/decision/v1	https://{{root_domain}}/decision/v1				
<input checked="" type="checkbox"/>	risk_guard_service_url	default	▼	https://{{root_domain}}/mtm-riskguard/api/v1	https://{{root_domain}}/mtm-riskguard/api/v1				
<input checked="" type="checkbox"/>	workflow_service_url	default	▼	https://{{root_domain}}/workflow/v1	https://{{root_domain}}/workflow/v1				
<input checked="" type="checkbox"/>	client_id	default	▼	[REDACTED]	[REDACTED]				
<input checked="" type="checkbox"/>	client_secret	default	▼	[REDACTED]	[REDACTED]				
<input checked="" type="checkbox"/>	access_token	default	▼						
	Add new variable								

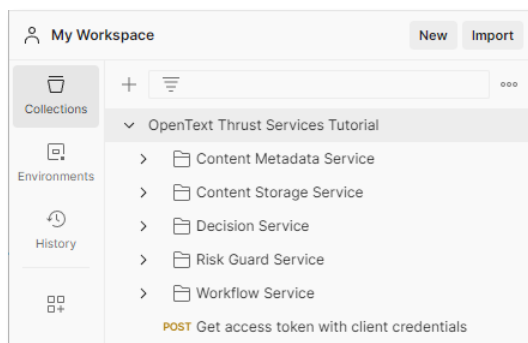
4. Click **Save** and close the **OpenText Thrust Services Tutorial** environment configuration screen.

**Note**

In this context you are using the confidential client ID and confidential client secret. This is typically the type of credentials you would use to authenticate from a back-end service (that is, running on a server, and not an end user device).

Later in the tutorial (when looking at the application code), you will be using a different authentication mechanism that uses the public client ID to authenticate through a trusted (OpenText Cloud Platform) login web page so that the application, which is front-end code (that is, running on the end user device, in the browser), can authenticate against the OpenText Thrust Services APIs in a secure manner.

5. Select **Collections** from the Postman sidebar and expand the **OpenText Thrust Services Tutorial** Postman collection.



You can see five folders representing the different OpenText Thrust Services APIs (**Content Metadata Service**, **Content Storage Service**, **Decision Service**, **Risk Guard Service**, and **Workflow Service**) for which the collection has example requests.

There is also a **Get access token with client credentials** POST request in the root of the collection, as this is the single request you will be using to get an access token to use for all other requests. After you run this request successfully, the **access_token** environment variable gets populated and can be used in every other request.

**Note**

If the **access_token** expires, the API requests will start failing with the token not valid error. To resolve this error, re-run the **Get access token with client credentials** request.

6. Run the **Get access token with client credentials** POST request to get the token.

Make sure you have selected the (previously configured) **OpenText Thrust Services Tutorial** environment, open the **Get access token with client credentials** request and click **Send**.

The screenshot displays the OpenText Thrust Services Tutorial interface. At the top, a browser window shows the 'OpenText Thrust Services Tutorial' title. Below it, the 'My Workspace' sidebar lists various services under the 'OpenText Thrust Services Tutorial' environment. The 'POST Get access token with client credentials' request is highlighted. The main panel shows the details of this request, including the URL, headers, and body. The 'Send' button is visible, and the response is shown in the 'Body' tab.

Request Details:

- Method:** POST
- URL:** `{{auth_service_url}}/oauth2/token`
- Body (JSON):**

```

1 {
2   "client_id": "{{client_id}}",
3   "client_secret": "{{client_secret}}",
4   "grant_type": "client_credentials"
5 }
```

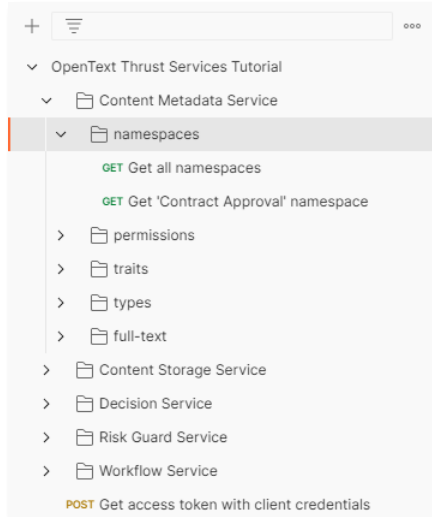
Response Details:

- Status:** 200 OK
- Time:** 721 ms
- Size:** 3.37 KB
- Body (JSON):**

```

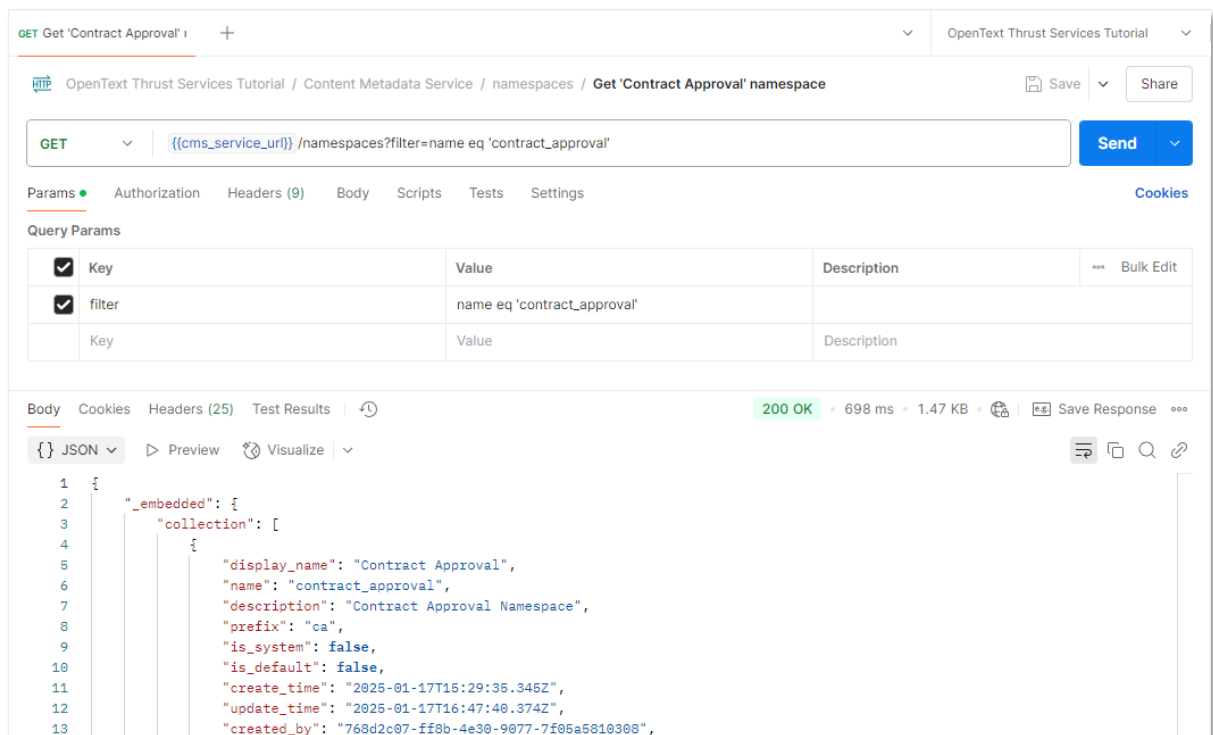
1 {
2   "refresh_token_expires_in": "",
3   "api_product_list": "[devx-us-prod1-publication, devx-us-prod1-capture, devx-us-prod1-dictionary,
4     devx-us-prod1-workflow, devx-us-prod1-highlight, devx-us-prod1-mtm-riskguard, devx-us-prod1-retention,
5     devx-us-prod1-ui, devx-us-prod1-css, devx-us-prod1-workflow-history, devx-us-prod1-cms, devx-us-prod1-oauth2,
6     devx-us-prod1-viewer, devx-us-prod1-decision, devx-us-prod1-markup, devx-us-prod1-retentionondisposition,
7     devx-us-prod1-retentionqualification, devx-us-prod1-admin]",
8   "api_product_list_json": [
9     "devx-us-prod1-publication",
10    "devx-us-prod1-capture",
11    "devx-us-prod1-dictionary",
12    "devx-us-prod1-workflow",
13    "devx-us-prod1-highlight",
14    "devx-us-prod1-mtm-riskguard",
15    "devx-us-prod1-retention",
16    "devx-us-prod1-retentionondisposition",
17    "devx-us-prod1-retentionqualification",
18    "devx-us-prod1-admin"]
19 }
```

7. To verify that the **contract_approval** namespace is correctly deployed or created in CMS, select **OpenText Thrust Services Tutorial > Content Metadata Service > namespaces**. The examples related to the namespace requests are listed.



You will have a look at the one that allows to retrieve the namespace you created.

8. Open the **Get 'Contract Approval' namespace** request and click **Send**.



As you can see, the Contract Approval namespace is displayed, which means the collection is deployed. The **display_name**, **name**, **description**, and **prefix** attributes are shown.

Note that the request that is used, **{{base_url}}/cms/namespaces?filter=name eq 'contract_approval'** has a filter to retrieve the namespaces with a name equal to 'contract_approval'.

All the requests are based on the API reference documentation. Go to developer.opentext.com, select the appropriate request explanation. This applies to all requests in the collection. For related documentation, see:

- [Content Metadata Service API reference](#)
- [Content Storage Service API reference](#)
- [Decision Service API reference](#)
- [Risk Guard Service API reference](#)
- [Workflow Service API reference](#)

In the case of this specific request, you will find the explanation under **[GET] Get list of Namespaces** for the **Namespace** resource.

Thrust Services > Services > Content Metadata > Content Metadata Service

GET /namespaces Get list of namespaces

Returns a list of namespaces

Parameters Try it out

Name	Description
page integer(\$int32) (query)	The page number required <input type="text" value="page"/>
items-per-page integer(\$int32) (query)	Number of items per page <input type="text" value="items-per-page"/>
filter string (query)	Filter the list of namespaces. For e.g. ?filter = name like '%de%' <input type="text" value="filter"/>
orderby string (query)	Sort by field/attribute names. Default sort order is descending and multiple sort condition should be separated by comma. Attribute name(refer response schema) containing underscore should be converted to camel case. Pass 'asc/desc' text after the value to change the sort order. e.g., to sort by 'display_name' and 'active' attribute in ascending order use ?orderby=displayName asc, active asc Note: Not all attributes support sort. <input type="text" value="orderby"/>

Search

- > Full-Text Query
- > Zip and Download
- > History
- > Relation
- > Attribute
- > IndexDefinition
- > RequiredTrait
- > **Namespace**
 - GET** Get list of namespaces
 - POST** Create a new...
 - GET** Get namespace details
 - PUT** Update a namespace
 - DELETE** Delete a namespace
 - PATCH** Update a namespace b...

In theory, the request to use to get a specific namespace is the **Get Namespace Details** GET request, but this requires the unique ID (in UUID string format) of the namespace to be passed.

That's why the filtering mechanism on the **Get list of Namespaces** request is used (so that it always works, no matter the namespace's ID value).

9. To verify the **approval** trait, select **OpenText Thrust Services Tutorial > Content Metadata Service > traits** and perform the **Get 'Approval' trait** GET request.

The screenshot shows the OpenText Thrust Services Tutorial interface for the 'Get 'Approval' trait' request. The URL is `{{cms_service_url}}/trait-definitions/ca_approval`. The response is a 200 OK status with a JSON body:

```

1 {
2   "name": "approval",
3   "system_name": "ca_approval",
4   "display_name": "Approval",
5   "description": "Approval Trait",
6   "namespace": "contract_approval",
7   "namespace_prefix": "ca",
8   "priority": 0,
9   "created_by": {
10    "id": "768d2c07-ff8b-4e30-9077-7f05a5810300",
11    "service_account": true,
12    "oauth2_client_id": "alm",

```

10. To verify the deployment of the **contract**, **loan_contract** and **customer** types, perform the following requests:

- **/Content Metadata Service/types/file/Contract/types/Get 'Contract' type**

The screenshot shows the OpenText Thrust Services Tutorial interface for the 'Get 'Contract' type' request. The URL is `{{cms_service_url}}/type-definitions/ca_contract`. The response is a 200 OK status with a JSON body:

```

1 {
2   "display_name": "Contract",
3   "description": "Contract Type",
4   "name": "contract",
5   "namespace": "contract_approval",
6   "namespace_prefix": "ca",
7   "parent": "cms_file",
8   "parent_display_name": "File",
9   "system_name": "ca_contract",
10  "category": "file",
11  "version": 0,
12  "rel_integrity_type": 0,
13  "create_time": "2025-01-17T15:30:09.714Z",
14  "update_time": "2025-01-17T16:47:49.583Z",
15  "created_by": {

```

- **/Content Metadata Service/types/file/Loan Contract/types/Get 'Loan Contract' type**

The screenshot shows a REST client interface with the following details:

- Request Method:** GET
- URL:** `{{cms_service_url}}/type-definitions/ca_loan_contract`
- Status:** 200 OK
- Response Body (JSON):**

```

1 {
2   "display_name": "Loan Contract",
3   "description": "Loan Contract Type",
4   "name": "loan_contract",
5   "namespace": "contract_approval",
6   "namespace_prefix": "ca",
7   "parent": "ca_contract",
8   "parent_display_name": "Contract",
9   "system_name": "ca_loan_contract",
10  "category": "file",
11  "version": 0,
12  "rel_integrity_type": 0,
13  "create_time": "2025-01-17T15:30:26.487Z",
14  "update_time": "2025-01-17T16:48:17.459Z",

```

- **/Content Metadata Service/types/folder/Customer/types/Get 'Customer' type**

The screenshot shows a REST client interface with the following details:

- Request Method:** GET
- URL:** `{{cms_service_url}}/type-definitions/ca_customer`
- Status:** 200 OK
- Response Body (JSON):**

```

1 {
2   "display_name": "Customer",
3   "description": "Customer Type",
4   "name": "customer",
5   "namespace": "contract_approval",
6   "namespace_prefix": "ca",
7   "parent": "cms_folder",
8   "parent_display_name": "Folder",
9   "system_name": "ca_customer",
10  "category": "folder",
11  "version": 0,
12  "rel_integrity_type": 0,
13  "create_time": "2025-01-17T15:30:22.744Z",
14  "update_time": "2025-01-17T16:48:10.058Z",

```

For types, the collection structure has additional levels to distinguish between the **file** and **folder** type category, and the individual types of the Contract Approval application. Requests to retrieve and manipulate **type instances** (that are created when using the application) are provided as well.

11. To verify the **required_approvals** decision model, execute the **/Decision Service/ Get 'Required Approvals' decision model** request.

GET Get 'Required Approvals' decision model

OpenText Thrust Services Tutorial / Decision Service / Get 'Required Approvals' decision model

GET `{{decision_service_url}}/runtime/decision-models?namespace=contract_approval&key=required_approvals` Send

Params Authorization Headers (9) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
namespace	contract_approval		
key	required_approvals		

Body Cookies Headers (21) Test Results 200 OK 861 ms 2.72 KB Save Response

```

1 {
2   "_embedded": {
3     "decision-models": [
4       {
5         "id": "fd43fe2b-d4e7-11ef-9b3a-6aac50f2406d",
6         "subscriptionId": "38be3761-ea3e-484e-bce4-cb3b84d5e41b",
7         "name": "Required Approvals",
8         "key": "required_approvals",
9         "version": 1,
10        "tenantId": "592b3f87-9bb4-463d-a541-8858934bd7d1",
11        "decisionDefinitionId": "fd460f9d-d4e7-11ef-9b3a-6aac50f2406d",
12        "category": null,
13        "namespace": "contract_approval",
14        "deploymentSpace": "TENANT",
15        "decisionType": "DECISION_TABLE"
16      }
17    ]
18  }
19 }

```

12. The last deployed model to verify is the **contract_approval** workflow model. To do this, perform the **/Workflow Service/Get 'Contract Approval' process model** request.

GET Get 'Contract Approval' process model

OpenText Thrust Services Tutorial / Workflow Service / Get 'Contract Approval' process model

GET `{{workflow_service_url}}/runtime/models?modelType=json&key=contract_approval` Send

Params Authorization Headers (9) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
modelType	json		
key	contract_approval		

Body Cookies Headers (21) Test Results 200 OK 1.05 s 2.57 KB Save Response

```

1 {
2   "_embedded": {
3     "models": [
4       {
5         "id": "f3300937-d4e7-11ef-ae5b-7eb03f3f778e",
6         "name": "Contract Approval",
7         "key": "contract_approval",
8         "tenantId": "592b3f87-9bb4-463d-a541-8858934bd7d1",
9         "version": 1,
10        "processDefinitionId": "contract_approval:1:f3300937-d4e7-11ef-ae5b-7eb03f3f778e",
11        "subscriptionId": "38be3761-ea3e-484e-bce4-cb3b84d5e41b",
12        "deploymentSpace": "TENANT",
13        "namespace": "contract_approval"
14      }
15    ]
16  }
17 }

```

Next exercise module:

Build the Contract Approval application.

14 [30'] Build the Contract Approval application

Learn how to:

- Import the finished Contract Approval App code into your project
- Understand the main logic of the Contract Approval application from the App.jsx React code
- Authenticate and get authorized with the OpenText Thrust Services APIs
- Set the environment variables for the Contract Approval application to run from your computer
- Use the content (CSS and CMS) APIs
- Use the Workflow Service API
- Use the Viewer Service API
- Use the Risk Guard Service API

During this exercise module you will be going through the code of the Contract Approval application. Although you will import and not actually write the (JavaScript and React) code, except for setting the environment variables in the `.env` file, you will go over its structure, logic and how it calls the different OpenText Thrust services APIs (CSS, CMS, Decision Service, Workflow Service, Viewer Service and Risk Guard Service).

It is not the intent to go over every single detail of how the code was written but you will touch upon the key aspects of how the Contract Approval application is developed, so that you have a good starting knowledge to build any application of your own.

For more information on the CSS, CMS, Decision Service, Workflow Service, Viewer Service and Risk Guard Service APIs, you can refer to their API reference documentation, respectively [CSS API Reference](#), [CMS API reference](#), [Decision Service API reference](#), [Workflow Service API reference](#), [Viewer Service API reference](#) and [Risk Guard Service API reference](#). Each service also has a more functional product documentation (that is, how to do things): [Content Storage product documentation](#), [Content Metadata product documentation](#), [Decision product documentation](#), [Workflow product documentation](#), [Viewing & Transformation product documentation](#), and [Risk Guard product documentation](#).

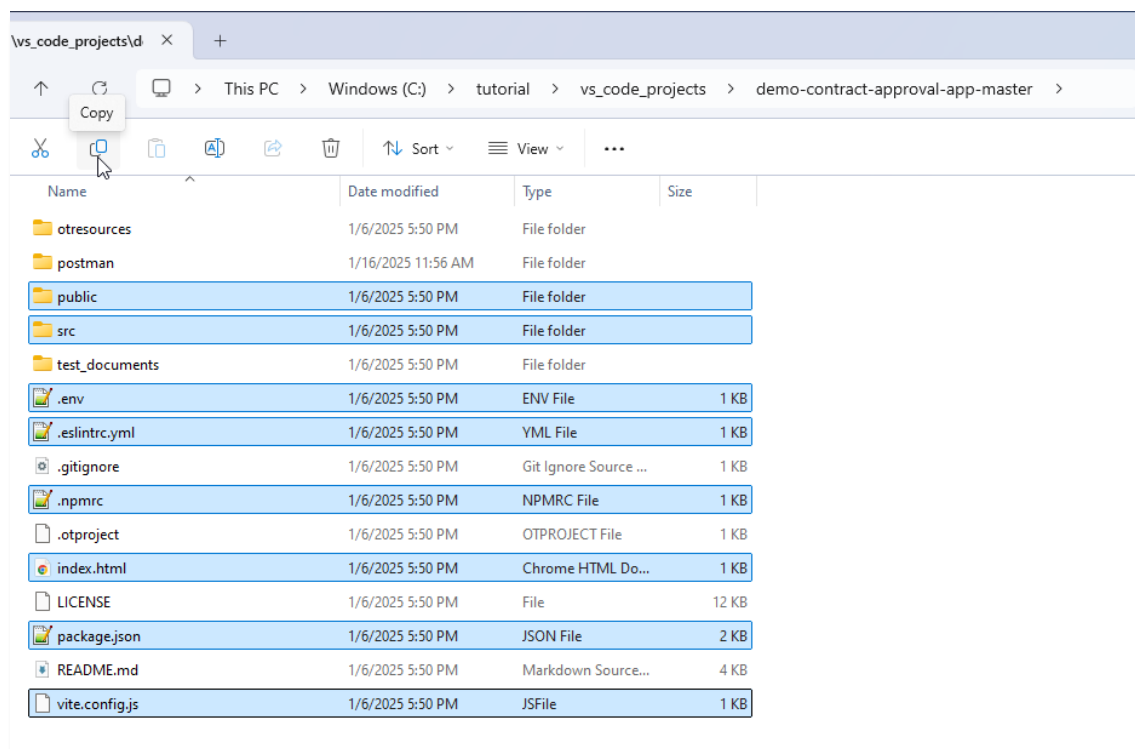
After completing this session, you will understand how the code of the Contract Approval application is written, how it consumes the deployed models, and how it calls the different OpenText Thrust Services APIs.

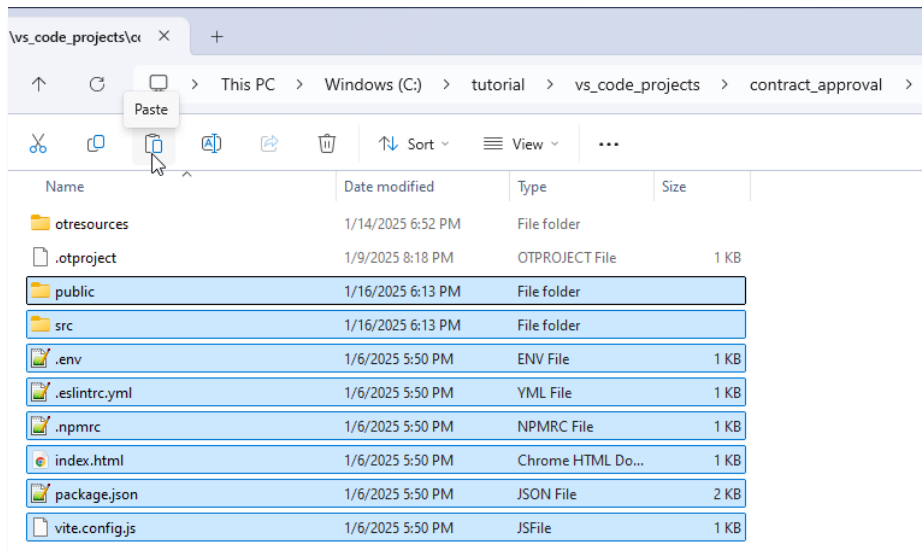
14.1 Import the Contract Approval App code into your project

The first step in this exercise is to import the code into your VS Code project. Make sure you have saved a copy of the finished Contract Approval application on your file system as described in [Download the Contract Approval application](#).

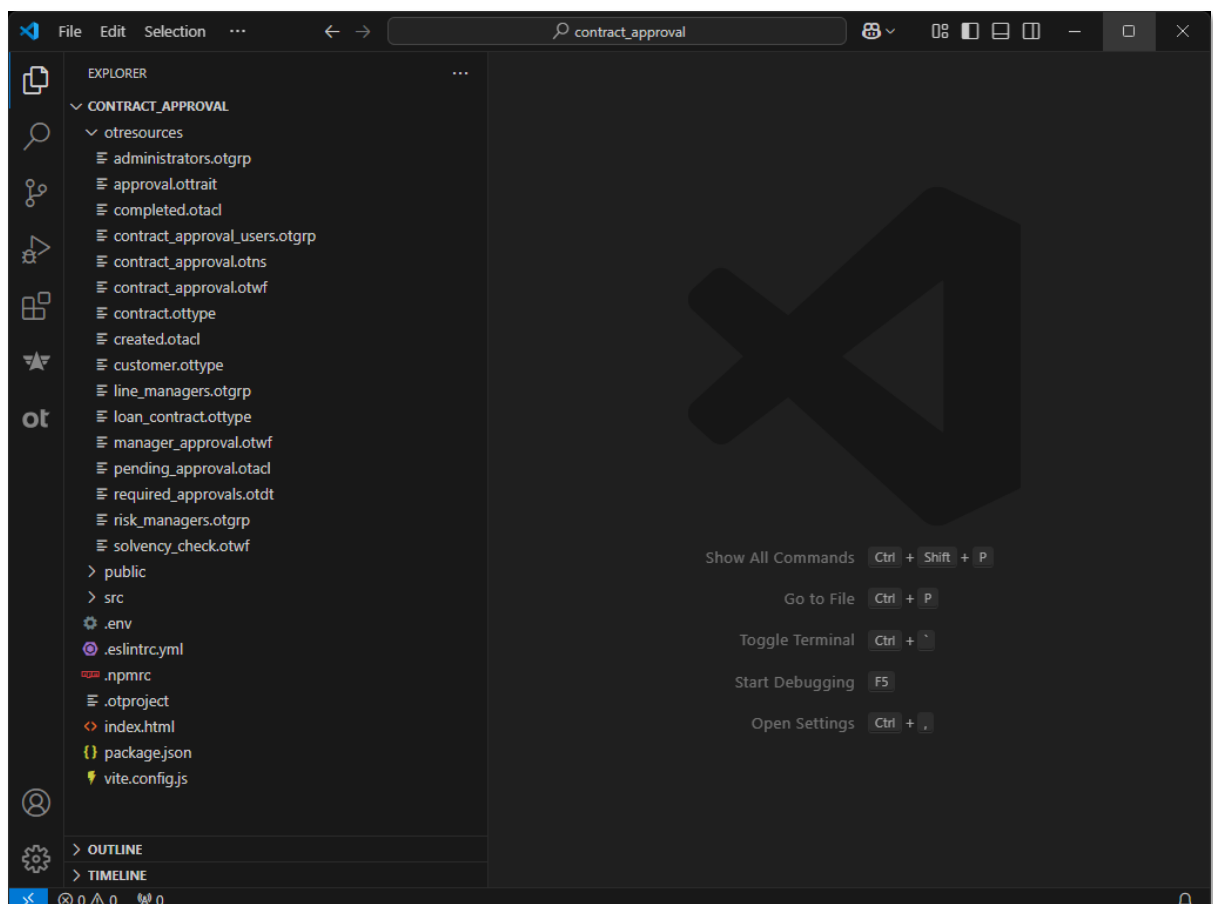
1. Navigate into the project folder of the finished Contract Approval application (that is, **demo-contract-approval-app-master** if you did not modify the project folder name) and copy the following folders and files into the root of your Contract Approval project:

- **public** folder
- **src** folder
- **.env** file
- **.eslintrc.yml** file
- **.npmrc** file
- **index.html** file
- **package.json** file
- **vite.config.js** file





2. Open your project in VS Code. The copied folders and files are visible in the **Explorer** view.



Next step:

Understand the main logic of the Contract Approval application.

14.2 Understand the main logic of the Contract Approval application

The **App.jsx** file contains the main logic for the application (React application). This file is available from the **/src** folder.

1. Open the **App.jsx** file and scroll to the return statement of the **App()** functional component that returns the JSX (XML-like syntax that allows to write HTML in React) that represents the main application user interface (UI).

```

78   return (
79     <div className="App">
80       <Header />
81       {
82         groups.includes('contract_approval_users')
83         ? (
84           <div className="page-content">
85             <Tabs
86               orientation="horizontal"
87               value={tabValue}
88               onChange={handleTabChange}
89             >
90               <Tab className="tab-caption" label="Created Contracts" />
91               {
92                 groups.includes('line_managers')
93                 && <Tab className="tab-caption" label="Line Manager Tasks" />
94               }
95               {
96                 groups.includes('risk_managers')
97                 && <Tab className="tab-caption" label="Risk Manager Tasks" />
98               }
99               <Tab className="tab-caption" label="All Contracts" />
100             </Tabs>
101             <ApplicationProvider>
102               <TabPanel value={tabValue} index={tabIndex}>
103                 <CreatedContractList />
104               </TabPanel>
105               {
106                 groups.includes('line_managers')
107                 && (
108                   // eslint-disable-next-line no-plusplus
109                   <TabPanel value={tabValue} index={++tabIndex}>
110                     <TasksList taskName="Line Manager Approval" />
111                   </TabPanel>
112                 )
113               }
114               {
115                 groups.includes('risk_managers')
116                 && (
117                   // eslint-disable-next-line no-plusplus
118                   <TabPanel value={tabValue} index={++tabIndex}>
119                     <TasksList taskName="Risk Manager Approval" />
120                   </TabPanel>
121                 )
122               }
123               <TabPanel value={tabValue} index={tabIndex + 1}>
124                 <ContractList />
125               </TabPanel>
126             </ApplicationProvider>
127           </div>
128         )
129         : (
130           <div className="page-content">
131             <p>You are not authorized to use this application</p>
132             <button type="button" style={{ margin: '0.50rem' }} onClick={signoutRedirect}>
133               Logout
134             </button>
135           </div>
136         )
137       }
138     </div>
139   );
140 }

```

- The **Header** (cf. **Header.jsx** for more details) component is the header bar of the Contract Approval application. It displays the OpenText logo, the application name, and the username of the connected user. The username is also a menu that allows the connected user to log out from the application.
- The **groups.includes('contract_approval_users')** statement is used to verify that the user is part of the **contract_approval_users** group (or one of its sub groups) to determine whether they are allowed to use the application.
- If the user is authorized to use the application, a four (horizontal) tabs UI layout is displayed, with the **Tabs** component and its **Tab** child components representing the tabs, and the **ApplicationProvider** component with its **TabPanel** child components representing the views to show when clicking the tabs.

The four horizontally stacked tabs generated by this part of the code are:

- The **Created Contracts** tab with the **CreatedContractList** component providing the “created contracts list” view to show all newly created contracts, that is, where status = ‘CREATED’.
- The **Line Manager Tasks** tab with the **TasksList** component providing the “Line Manager Approval” tasks view to show all approval tasks for the Line Manager. This tab is only shown if the list of groups, the authenticated user is a member of, includes the **line_managers** group.
- The **Risk Manager Tasks** tab with the **TasksList** component providing the “Risk Manager Approval” tasks view to show all approval tasks for the Risk Manager. This tab is only shown if the list of groups, the authenticated user is a member of, includes the **risk_managers** group.
- The **All Contracts** tab with the **ContractList** component providing the “all contracts” view to show all contracts in the application, independently of their status.

If the user is not authorized to use the application, they are presented with a “not authorized” message and a button to log out.

Next step:

Understand the main logic of the Contract Approval application.

14.3 Authenticate and get authorized with the OpenText Thrust Services APIs

Beyond the main application UI, the **App.jsx** file also contains code that manages the authentication and authorization logic in combination with the code in the following files:

- **WrappedSecuredApp.jsx** (under **/src**)
 - **OidcConfig.js** (under **/src/authorization**)
1. Open the **WrappedSecuredApp.jsx** file. This file is the component that “wraps” the application with security.

```

1  import { AuthProvider } from 'react-oidc-context';
2  import App from './App';
3  import OidcConfig from './authorization/OidcConfig';
4
5  function WrappedSecuredApp() {
6    return (
7      /* eslint-disable-next-line react/jsx-props-no-spreading */
8      <AuthProvider {...OidcConfig}>
9        <App />
10     </AuthProvider>
11   );
12 }

```



Note

To authenticate with the OpenText Thrust Services APIs, the Contract Approval App uses **OpenID Connect (OIDC)**, an authentication protocol that sits on top of OAuth 2.0. More specifically, using the **Public Client ID** for the deployed application, an authentication flow that uses an external (to the Contract Approval application) login screen is used to authenticate with the OpenText Cloud Platform, allowing to subsequently call the different Thrust Services APIs.

- The **AuthProvider** component wraps the security around the **App** component (from the **App.jsx** file). It is imported from the **react-oidc-context** library to deliver the OIDC authentication mechanism. The authentication and authorization parameters for the security to apply to the contained **App** are defined through the **OidcConfig** configuration object, which the **AuthProvider** component takes as a parameter.
- The **OidcConfig** configuration object is imported from **OidcConfig.js**



Note

The **react-oidc-context** library is available to be imported/referenced, because it is defined as a dependency in the **package.json** file. To check the different project dependencies, you can open the **package.json** file.

The **package.json** file is the main configuration file for a Node.js project, and the Contract Approval application is a React application set up to run in Node.js.

2. Open the **OidcConfig.js** file to examine the different OIDC (that is, authentication and authorization) configuration parameters.

```

1  const OidcConfig = {
2    authority: `${process.env.REACT_APP_BASE_SERVICE_URL}/tenants/${process.env.REACT_APP_TENANT_ID}`,
3    client_id: process.env.REACT_APP_CLIENT_ID,
4    redirect_uri: process.env.REACT_APP_REDIRECT_URI,
5    response_type: 'code',
6    scope: 'openid otds:groups',
7    post_logout_redirect_uri: process.env.REACT_APP_REDIRECT_URI,
8    onSignInCallback: () => {
9      window.history.replaceState({}, document.title, window.location.pathname);
10   },
11 };

```

- The **authority** is the URL of the authority (or identity provider) and it is used as the main URL for the authentication flow.
- The **client_id** is the **Public Client ID** for the deployed application.
- The **redirect_uri** is passed as part of the authentication flow to allow redirecting back into the (Contract Approval) application once authenticated.
- The **response_type** tells the authorization server which grant to execute.
- The **scope** is a space-delimited list of permissions that the application requires.
- The **post_logout_redirect_uri** defines the application page to redirect to after the logout completes.
- The **onSignInCallback** is the callback function that defines the logic to execute when the sign in occurs. Here the authorization code is removed from the url for security reasons.

As you can see, to construct the parameter values, **process.env.<ENVIRONMENT_VARIABLE>** is used, which corresponds with the environment variables filled in the **.env** file (available in the project root).



Note

The **.env** file needs to be correctly filled for the application to work and how to do this is described in the [Set the environment variables](#) section.

- Go back to the **App.jsx** file to examine the use of the security mechanism and configuration provided by the wrapping **WrappedSecuredApp** component and configured by the **OidcConfig** configuration object.

```

17  const {
18    activeNavigator,
19    error,
20    isAuthenticated,
21    isLoading,
22    user,
23    signinRedirect,
24    signoutRedirect,
25  } = useAuth();
26  const [isAppLoaded, setIsAppLoaded] = useState(false);
27  const [tabValue, setTabValue] = useState(0);
28  const [groups, setGroups] = useState([]);
29
30  const handleTabChange = (event, newTabValue) => {
31    setTabValue(newTabValue);
32  };
33
34  useEffect(() => {
35    if (!isLoading) {
36      if (!isAuthenticated) {
37        signinRedirect();
38      } else if (!isAppLoaded && isAuthenticated) {
39        setGroups(
40          jwtDecode(user.id_token)
41            .grp
42            .map((group) => group.substring(0, group.indexOf('@'))),
43        );
44        setIsAppLoaded(true);
45      }
46    }
47  }, [isAuthenticated, isLoading, isAppLoaded]);
48
49  if (activeNavigator === 'signoutRedirect') {
50    return (
51      <div className="loading">
52        <CircularProgress color="inherit" />
53      </div>
54    );
55  }
56
57  if (error) {
58    return (
59      <div className="error">
60        <Alert severity="error">
61          Authentication error:
62          {''}
63          {error.message}
64        </Alert>
65      </div>
66    );
67  }
68
69  if (!isAppLoaded) {
70    return (
71      <div className="loading">
72        <CircularProgress color="inherit" />
73      </div>
74    );
75  }

```

- The **useAuth** method is imported from the **react-oidc-context** library to provide the **activeNavigator**, **error**, **isAuthenticated**, **isLoading**, **user**, **signinRedirect**, and **signoutRedirect**. These are used for signing in (if not authenticated), signing out, returning user information (including user groups), handling errors, and managing whether the application is still loading and authenticating.
- The **jwtDecode** method is imported from the **jwt-decode** library to allow decoding the JWT id token available from the (authenticated) user object. This is important because it allows retrieving the groups the user belongs to in order to display the correct application tabs.

Next step:

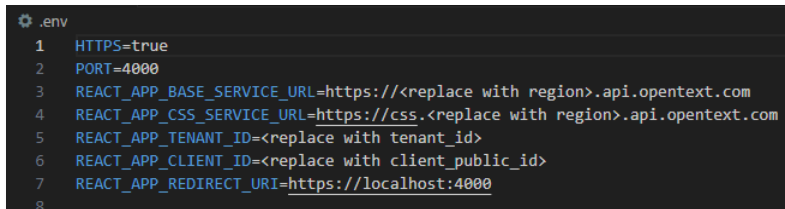
Set the environment variables.

14.4 Set the environment variables

In the previous section on authentication and authorization, it is explained that to access the different OIDC configuration parameters, `process.env.<ENVIRONMENT_VARIABLE>` is being used.

The corresponding environment variables are configured in the `.env` file.

1. Open the `.env` file (available from the project root folder) to configure the environment variables to be able to run the Contract Approval application.



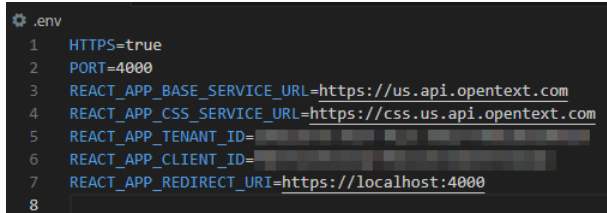
```

1  HTTPS=true
2  PORT=4000
3  REACT_APP_BASE_SERVICE_URL=https://<replace with region>.api.opentext.com
4  REACT_APP_CSS_SERVICE_URL=https://css.<replace with region>.api.opentext.com
5  REACT_APP_TENANT_ID=<replace with tenant_id>
6  REACT_APP_CLIENT_ID=<replace with client_public_id>
7  REACT_APP_REDIRECT_URI=https://localhost:4000
8

```

2. Update the `.env` file as follows:

- Replace the `<replace with region>` with the region of your developer subscription (for this tutorial we use `us`, but if your developer subscription is in a different region, you must use that region).
- Replace `<replace with tenant_id>` with the corresponding value you saved after deploying the application project. More specifically, use the tenant id (from the text in **tenants** `["<tenant id>"]`).
- Replace `<replace with client_public_id>` with the corresponding value you saved after deploying the application project. More specifically, use **Public Client ID**.



```

1  HTTPS=true
2  PORT=4000
3  REACT_APP_BASE_SERVICE_URL=https://us.api.opentext.com
4  REACT_APP_CSS_SERVICE_URL=https://css.us.api.opentext.com
5  REACT_APP_TENANT_ID=
6  REACT_APP_CLIENT_ID=
7  REACT_APP_REDIRECT_URI=https://localhost:4000
8

```

3. **Save** and close the `.env` file.

Next step:

Use the content (CSS and CMS) APIs.

14.5 Use the content (CSS and CMS) APIs

To store and manage content in the OpenText Cloud Platform, two Thrust Services APIs need to be used. The Content Storage Service (CSS) API allows to store and retrieve the actual (binary) content files (that is, the different file renditions), whereas the Content Metadata Service (CMS) API can be used to manage the associated file object metadata. It is through combining these two APIs that you can manage file objects.

Beyond file objects (that is, objects with content), the CMS API also provides the capability to manage folders (containers for other folders, files, and objects), objects (contentless objects) and relations.

To understand how to use the CSS and CMS APIs, let's have a closer look at the first of the four application views (that is, the view for the first tab). The “created contracts list” view provided through the **CreatedContractList** component does not only show the newly created contracts (status = 'CREATED'), but it also provides the button to add new contracts to the system.

1. Open the **CreatedContractList.jsx** file from the **/src/components** folder.

```

34  /**
35   * This view displays the list of created contracts.
36   * From here the user can request approval for any of them.
37   */
38  function CreatedContractList() {
39    const { user } = useAuth();
40    const [state, setState] = useState(
41      {
42        contracts: [],
43        pendingApprovalAclId: '',
44        completedAclId: '',
45        openContractDetails: false,
46        selectedContract: { properties: {} },
47        openAddContract: false,
48        addNumberOfContracts: 0,
49        pageNumber: 0,
50        count: -1,
51        openDocumentDialogView: false,
52        fileId: '',
53        fileName: '',
54        showBackdrop: false,
55        showSnackBar: false,
56        snackBarMessage: '',
57        snackBarSeverity: 'success',
58      },
59    );
60    const didMountRef = useRef(false);
61    const addNumberOfContractsRef = useRef(state.addNumberOfContracts);
62    const pageNumberRef = useRef(state.pageNumber);
63
64    const raiseError = useCallback((errorMessage) => {
65      setState((prevState) => ({
66        ...prevState,
67        snackBarSeverity: 'error',
68        snackBarMessage: errorMessage,
69        showSnackBar: true,
70      }));
71    }, []);
72
73    const getAcls = useCallback(() => {
74      if (
75        state.pendingApprovalAclId.length === 0
76        || state.completedAclId.length === 0) {
77        let pendingApprovalAclFound = false;
78        let completedAclFound = false;
79        axios({
80          method: 'get',
81          url: `${baseUrl}/cms/permissions?filter=name eq "pending_approval" or name eq "completed"`,
82          headers: {
83            Authorization: `Bearer ${user.access_token}`,
84          },
85        }).then((res) => {
86          const acls = res.data._embedded.collection;
87          acls.forEach((acl) => {

```

2. Scroll to the return statement of the **CreatedContractList()** functional component as it represents the “created contract list” UI.

```

283 return (
284   <div>
285     <Button variant="contained" color="primary" disabled={!user.profile.preferred_username} startIcon={<Add />} onClick={() => openContractDetails}>Add Contract</Button>
286     <div className="content-header">All created contracts</div>
287     <TableContainer component={Paper}>
288       <Table size="small" aria-label="a dense table">
289         <TableHead>
290           <TableRow>
291             <TableCell>Contract name</TableCell>
292             <TableCell align="left">Creation date</TableCell>
293             <TableCell align="left">Value</TableCell>
294             <TableCell align="left">Risk classification</TableCell>
295             <TableCell align="left">View document</TableCell>
296             <TableCell align="left">Action</TableCell>
297           </TableRow>
298         </TableHead>
299         <TableBody>
300           {state.contracts.map((row) => (
301             <TableRow key={row.id}>
302               <TableCell component="th" scope="row">
303                 {row.name}
304               </TableCell>
305               <TableCell align="left">{getDateValue(row.create_time)}</TableCell>
306               <TableCell align="left">{row.properties.value}</TableCell>
307               <TableCell align="left"><RiskClassification row={row} /></TableCell>
308               <TableCell align="left">
309                 <Button size="small" variant="outlined" color="primary" onClick={() => openDocumentDialogView(row.id, row.name)}>Original</Button>
310               </TableCell>
311               <TableCell align="left">
312                 <Button size="small" variant="outlined" color="primary" onClick={() => startContractForApproval(row.id)}>Request approval</Button>
313               </TableCell>
314               <TableCell align="left">
315                 <IconButton size="small" variant="outlined" color="primary" title="Show details" onClick={() => showDetails(row)}>
316                   <ArrowForwardIos />
317                 </IconButton>
318               </TableCell>
319             </TableRow>
320           ))}
321         </TableBody>
322       </Table>
323     </TableContainer>
324     <Pagination
325       pageNumber={state.pageNumber}
326       count={state.count}
327       handlePageNumber={handlePageNumber}
328     />
329     <ContractDetails
330       open={state.openContractDetails}
331       selectedContract={state.selectedContract}
332       parentRaiseError={raiseError}
333       onClose={handleCloseContractDetails}
334     />
335     <AddContract
336       open={state.openAddContract}
337       onAddContract={handleContractAdded}
338       onClose={handleCloseAddContract}
339     />
340     <DocumentDialogView
341       open={state.openDocumentDialogView}
342       fileId={state.fileId}
343       onClose={handleCloseDocumentDialogView}
344     />
345     <Backdrop style={{ zIndex: 9999 }} open={state.showBackdrop}>
346       <CircularProgress color="inherit" />
347     </Backdrop>
348     <Snackbar
349       anchorOrigin={{
350         vertical: 'bottom',
351         horizontal: 'center',
352       }}
353       open={state.showSnackBar}
354       autoHideDuration={5000}
355       onClose={handleSnackBarClose}
356       action={
357         (
358           <IconButton size="small" aria-label="close" color="inherit" onClick={handleSnackBarClose}>
359             <Close fontSize="small" />
360           </IconButton>
361         )
362       }
363     />
364   </div>

```

- The “Add” **Button** at the top of created contract list allows adding new contracts via the **AddContract** component.
- The **TableContainer** component provides the table layout to display the list of the different contracts that have the ‘CREATED’ status.
- The **TableHead** and **TableBody** components represent the column headers and the rows with the values, each table row in the table body will show the following information for the displayed contract:
 - **Contract name**
 - **Creation date**
 - **Value**
 - **Risk classification**
 - **Original** (this is not a property value but a button to open the actual document in the viewer from the Viewer Service through the **DocumentDialogView** component)
 - **Request Approval** (this is not a property value but a button to start the contract approval workflow)
 - An **arrow icon button** allowing to open the contract details (the contract attributes screen) through the **ContractDetails** component

The table rows are generated by iterating over the **state.contracts** array, and in its turn **state.contracts** is populated by the **getContracts** method (called when the created contracts list needs updating).

3. If you look at the **getContracts** method, you easily recognize the (axios) GET request to the **/cms/instances** endpoint of the **CMS** API with the **user.access_token** (bearer) token passed as the **Authorization** header.

```

117 const getContracts = useCallback(() => {
118   if (user.profile.preferred_username) {
119     setState((prevState) => ({ ...prevState, showBackdrop: true }));
120     axios({
121       method: 'get',
122       url: `${baseUrl}/cms/instances/file/ca_contract/?include-total=true&sortby=create_time desc&filter=status eq "CREATED"&page=
123       headers: {
124         Authorization: `Bearer ${user.access_token}`,
125       },
126     }).then((res) => {
127       setState((prevState) => ({
128         ...prevState,
129         contracts: res.data?._embedded?.collection ?? [],
130         count: res.data.total,
131       }));
132     }).catch((error) => {
133       let errorMessage = 'Could not get contracts: ';
134       if (error.response?.data) {
135         errorMessage += error.response.data.exception;
136       } else {
137         errorMessage += error.message;
138       }
139       raiseError(this, errorMessage);
140     }).finally(() => {
141       setState((prevState) => ({ ...prevState, showBackdrop: false }));
142     });
143   } else {
144     setState((prevState) => ({ ...prevState, contracts: [], count: -1 }));
145   }
146 }, [
147   raiseError,
148   user.access_token,
149   user.profile.preferred_username,
150   state.pageNumber,
151 ];

```


4. To understand how to use the CSS and CMS APIs to create a new contract, open the **AddContract.jsx** file (again from the **/src** folder) and first look for the **// Adding contract content file** comment in the code.

```

259 // Adding contract content file
260 const formData = new FormData();
261 formData.append(
262   'file',
263   state.selectedFile,
264   state.selectedFile.name,
265 );
266
267 let createFileResponse;
268 try {
269   createFileResponse = await axios.post(
270     `${process.env.REACT_APP_CSS_SERVICE_URL}/v3/files/fromStream`,
271     formData,
272     {
273       headers: {
274         'Content-Type': 'multipart/form-data',
275         Authorization: `Bearer ${user.access_token}`,
276       },
277     },
278   );
279 } catch (error) {
280   exitAddContract(`Error saving contract content file: ${JSON.stringify(error.response.data, null, 2)}`);
281 }

```

- The (axios) POST call uploads the selected (contract) file to CSS.

5. After uploading the file to CSS, the second step in the contract creation code is to create the metadata object. Look for **// Setting contract metadata** comment.

```

283 // Setting contract metadata
284 const { originalFileName } = createFileResponse.data;
285 let cmsType;
286 if (isLoanContract()) {
287   cmsType = 'ca_loan_contract';
288 } else {
289   cmsType = 'ca_contract';
290 }
291
292 let createMetadataResponse;
293 try {
294   createMetadataResponse = await axios.post(
295     `${baseUrl}/cms/instances/file/${cmsType}`,
296     {
297       name: state.newContractName,
298       parent_folder_id: parentFolderId,
299       acl_id: aclId,
300       renditions: [
301         {
302           name: originalFileName,
303           rendition_type: 'primary',
304           source: 'FILE_ID',
305           blob_id: createFileResponse.data.id,
306         },
307       ],
308       properties: {
309         value: parseInt(state.newContractValue, 10),
310         status: 'CREATED',
311         requester_email: customerEmail,
312         risk_classification: risk.data.riskClassification,
313         extracted_terms: risk.data.extractedTerms,
314         ...isLoanContract() && {
315           monthly_installments: parseInt(state.newContractMonthlyInstallments, 10),
316           yearly_income: parseInt(state.newContractYearlyIncome, 10),
317         },
318       },
319       traits: {
320         ca_approval: {
321           'Automatic Approval': {
322             is_required: true,
323             has_been_granted: false,
324             approver: '',
325             approver_role: '',
326           },
327           'Line Manager Approval': {
328             is_required: false,
329             has_been_granted: false,
330             approver: '',
331             approver_role: '',
332           },
333           'Risk Manager Approval': {
334             is_required: false,
335             has_been_granted: false,
336             approver: '',
337             approver_role: '',
338           },
339           ...isLoanContract() && {
340             'Solvency Check': {
341               is_required: false,
342               has_been_granted: false,
343               approver: '',
344               approver_role: '',
345             },

```

- The type of contract to create is determined using the system names, **ca_loan_contract** or **ca_contract**, where the ca prefix represents the namespace.
- Another (axios) POST call creates the file metadata object in CMS. It passes the contract's properties, rendition (linked to the previously uploaded CSS file via the **blob_id**), and traits as payload.

Next step:

Use the Workflow Service API.

14.6 Use the Workflow Service API

To illustrate the use of the Workflow Service API, you will look at the piece of code where the approval workflow is launched.

1. Open the **CreatedContractList.jsx** file again and locate the **startContractForApproval** method.

```
200 const startContractForApproval = (contractId) => {
201   setState((prevState) => ({ ...prevState, showBackdrop: true }));
202
203   const data = {
```

2. First, the **data** object to pass as payload to the Workflow Service API gets constructed. It contains the **processDefinitionKey** matching the name of the workflow model and allowing to identify the workflow to create an workflow instance for, a user-friendly **name**, and the process **variables** to pass to the new workflow instance (cf. [Create the Contract Approval workflow](#)).

```
203   const data = {
204     processDefinitionKey: 'contract_approval',
205     name: 'Approve contract',
206     outcome: 'none',
207     variables: [
208       {
209         name: 'base_url',
210         value: baseUrl,
211       },
212       {
213         name: 'contract_id',
214         value: contractId,
215       },
216       {
217         name: 'pending_approval_acl_id',
218         value: state.pendingApprovalAclId,
219       },
220       {
221         name: 'completed_acl_id',
222         value: state.completedAclId,
223       },
224     ],
225     returnVariables: true,
226   };

```

3. Secondly, the Workflow Service API that launches the workflow instance (with the **data** payload) gets called via a (axios) POST call.

```
228   axios({
229     method: 'post',
230     url: `${baseUrl}/workflow/v1/process-instances`,
231     headers: {
232       Authorization: `Bearer ${user.access_token}`,
233     },
234     data,
235   }).then(() => {
236     setState((prevState) => ({ ...prevState, snackBarMessage: 'Approval requested successfully.' }));
237     setState((prevState) => ({ ...prevState, showSnackBar: true }));
238     getContracts();
239   }).catch((error) => {
240     const statusCode = error.response.status;
241     let errorMessage = 'Error requesting approval: ';
242     if (statusCode === 400) {
243       errorMessage += error.response.data.exception;
244     } else {
245       errorMessage += error.message;
246     }
247     raiseError(errorMessage);
248   }).finally(() => {
249     setState((prevState) => ({ ...prevState, showBackdrop: false }));
250   });
251 };

```

4. To have an example of how to call the task related service endpoints, feel free to explore the **Tasks.js** file under **/src/services/workflow**.

The Workflow Service tasks API is used in the Contract Approval application to interact with the contract approval workflow's manual user tasks (that is, the Line Manager approval tasks and the Risk Manager approval tasks). More specifically, it provides the different REST API endpoints that are called from the **getTasks()**, **claimTask()**, and **completeTask()** methods from the **Tasks** utility class.

Next step:

Use the Viewer Service API.

14.7 Use the Viewer Service API

The Viewer Service exposes a powerful file viewer that can be configured in a very granular way.

To understand how to use and configure the viewer from the Viewer Service, it is recommended that you have an in-depth look at the **FileViewer.jsx** file available from the **/src** folder.

For this tutorial, you will focus on how to interact with the viewer API (**bravaApi**).

1. Open the **FileViewer.jsx** file and explore the **toolbarWithMarkupStuff**, **tabContainerWithMarkups** and **markupTools** constants. They represent the detailed configuration of the different viewer components/features.

```

20 const toolbarWithMarkupStuff = {
21   left: [
22     { component: 'ToggleSidebarButton', side: 'tabContainerWithMarkups' },
23     { component: 'DownloadButton' },
24     { component: 'PanButton' },
25     { component: 'ZoomToRectangleButton' },
26     { component: 'SaveButton' },
27     { component: 'ZoomInButton' },
28     { component: 'ZoomOutButton' },
29     { component: 'ZoomExtentsButton' },
30     { component: 'ZoomWidthButton' },
31     { component: 'RotateButton' },
32   ],
33   center: [{ component: 'TitleText' }],
34   right: [
35     { component: 'PageSelector', style: { marginLeft: '0.5em' } },
36     { component: 'CloseButton', size: 18 },
37   ],
38 };
39
40 const tabContainerWithMarkups = {
41   sidebarName: 'tabContainerWithMarkups',
42   primary: 'primary',
43   tabs: [
44     { component: 'ThumbnailPane', title: 'tab.thumbnails' },
45     { component: 'MarkupPane', title: 'tab.markups', layoutKey: 'markupTools' },
46   ],
47 };
48
49 const markupTools = [
50   {
51     title: 'Tools',
52     tools: [
53       {
54         label: 'select markup',
55         tool: 'select',
56         icon: 'Select',
57       },
58     ],
59   },
60   {
61     title: 'toolPalette.annotations',
62     tools: [
63       {
64         label: 'text', tool: 'text', icon: 'Text', props: {},
65       },
66       {
67         label: 'arrow', tool: 'arrow', icon: 'Arrow', props: {},
68       },
69       {
70         label: 'ellipse', tool: 'ellipse', icon: 'Ellipse', props: {},
71       },
72       {
73         label: 'arc', tool: 'arc', icon: 'Arc', props: {},
74       },
75       {
76         label: 'scratchout', tool: 'scratchout', icon: 'Scratchout', props: {},
77       },

```

2. Explore the **useEffect()** hook method to understand how the configuration gets applied to the **bravaApi** and ultimately returned to render the viewer.

```

189   useEffect(() => {
190     if (didMountRef.current) {
191       if (bravaApi) {
192         bravaApi.setHttpHeaders({
193           Authorization: `Bearer ${user.access_token}`,
194         });
195         bravaApi.setScreenBanner('Viewer Service by OpenText | Document Viewed at %Time');
196         bravaApi.enableMarkup(true);
197         bravaApi.editableMarkupPredicate = () => true;
198         bravaApi.commentableMarkupPredicate = () => true;
199         bravaApi.visibleToolPropertyPredicate = () => true;
200         bravaApi.deletableMarkupPredicate = () => true;
201         bravaApi.deletableStampPredicate = () => true;
202         bravaApi.editableStampPredicate = () => true;
203         bravaApi.addableStampPredicate = () => true;
204         bravaApi.setMarkupHost(window.ViewerAuthority);
205         bravaApi.setUserName(user.profile.preferred_username);
206         bravaApi.setScreenWatermark('ORIGINAL');
207         bravaApi.setLayout({
208           topToolbar: 'toolbarWithMarkupStuff',
209           toolbarWithMarkupStuff,
210           mainContainer: [
211             { component: 'TabContainer', layoutKey: 'tabContainerWithMarkups' },
212             { component: 'PageContainer' },
213           ],
214           tabContainerWithMarkups,
215           markupTools,
216         });
217         bravaApi.addPublication(publicationData, true);
218         bravaApi.render(VIEWER_ID);
219       }
220     } else {
221       window.addEventListener('bravaReady', bravaReadyEventListener);
222       loadBravaViewer();
223       didMountRef.current = true;
224     }
225   }, [
226     bravaApi,
227     publicationData,
228     user.access_token,
229     user.profile.preferred_username,
230     loadBravaViewer,
231   ]);

```

3. The absolute minimum for the viewer to render correctly is the following code snippet (showing a drastically reduced **useEffect()** method):

```

54   useEffect(() => {
55     if (didMountRef.current) {
56       if (bravaApi) {
57         bravaApi.setHttpHeaders({
58           Authorization: `Bearer ${user.access_token}`,
59         });
60         bravaApi.addPublication(publicationData, true);
61         bravaApi.render(VIEWER_ID);
62       }
63     } else {
64       window.addEventListener('bravaReady', bravaReadyEventListener);
65       loadBravaViewer();
66       didMountRef.current = true;
67     }
68   }, [
69     bravaApi,
70     publicationData,
71     user.access_token,
72     loadBravaViewer,
73   ]);

```

It is recommended you try out different configurations by modifying the **useEffect()** method's content.

4. To help with understanding the Viewer Service configuration, two additional files in the **/src/components** folder are included: **FileViewer.jsx-minimal** and **FileViewer.jsx-full**. They respectively provide an example of the minimal set of configuration settings (as shown in the above code snippet) and an example of the viewer with all configuration settings enabled and configured. Feel free to try adding or removing configuration settings using these two example files as reference.

The **FileViewer.jsx** file that is used for the Contract Approval application shows you an example of how the configuration can be applied to display/overlay a screen banner (footer) and a watermark on the viewed document pages, and to provide markup tools from the left panel under a markups tab.

Next step:

Use the Risk Guard Service API.

14.8 Use the Risk Guard Service API

The last API to look into is the Risk Guard Service API. Risk Guard offers the ability to submit the content of a file/document to analyze it and identify potentially risky or sensitive content.

The Contract Approval application calls the Risk Guard Service API to determine the risk level of a newly created contract (that is, uploaded file).

1. First, (re)open the **AddContract.jsx** file as it holds the code that creates a new contract. More specifically, the Risk Guard Service API gets called at the beginning of the **submitContract** method.

The **processDoc()** method of the **riskGuardService** (instance of **RiskGuard** class) is called to return the **riskClassification** and **extractedTerms** for the contract that is being submitted.

```
139 const submitContract = async () => {
140   setState((prevState) => ({ ...prevState, showBackdrop: true }));
141   const risk = await riskGuardService.processDoc(
142     state.selectedFile,
143     state.selectedFile.name,
144   );
145   setState((prevState) => ({
146     ...prevState,
147     contractRisk: risk.data.riskClassification,
148     extractedTerms: risk.data.extractedTerms,
149   }));

```

2. The code that interacts with the Risk Guard Service API sits within the **RiskGuard** utility class, so the second bit of code to look at is in the **RiskGuard.js** file, located under **/src/services/riskguard**.

The **processDoc()** method calls the Risk Guard Service API through a (axios) POST request, passing the given (contract) file as the **FormData** data payload of the call (which requires to set the **Content-Type** header to **multipart/form-data**).

```
106 async processDoc(fileData, fileName) {
107   const form = new FormData();
108   form.append('File', fileData, fileName);
109   const postRequest = {
110     method: 'post',
111     url: this.url,
112     headers: {
113       Authorization: `Bearer ${this.user.access_token}`,
114       'Content-Type': 'multipart/form-data',
115     },
116     data: form,
117   };
118
119   return new Promise((resolve, reject) => {
120     axios(postRequest).then((postResponse) => {
121       resolve({ data: RiskGuard.calculateRisk(postResponse) });
122     }).catch((err) => {
123       // eslint-disable-next-line no-console
124       console.error(err);
125       reject(new Error(err.response));
126     });
127   });
128 }
129 }

```

3. Feel free to also have a look at how the risk classification is computed in the **calculateRisk()** method.

CONGRATULATIONS!

You have now finished building the Contract Approval application. In the next exercise module, you will test it and run through the different contract approval scenarios.

Next exercise module:

Test the Contract Approval application.

15 [60'] Test the Contract Approval application

Learn how to:

- Run a React application that consumes the OpenText Thrust Services APIs
- Sign in to the application using an external login screen and redirect into the application
- Explore the Contract Approval application UI and features
- Test uploading a (contract) file and filling its metadata (consuming the Content Storage Service and Content Metadata Service APIs)
- Test the applying of ACLs (consuming Content Metadata Service API)
- Test the updating of document metadata and traits (consuming the Content Metadata Service API)
- Test the detection of sensitive content (consuming the Risk Guard Service API)
- Test the Viewer (consuming the Viewer Service API)
- Test the different contract approval flows, including rejection and expiry of manual approval tasks (consuming the Workflow Service and Decision Service APIs)

During this exercise module you will be testing the Contract Approval application. You will run through different scenarios to demonstrate the different behaviors that depend on the automated and manual choices that can be made within the application.



Note

If you have not gone through all the exercise modules to build the Contract Approval application yourself, you must first set up your testing environment according to the following steps:

- Be certain that you have checked and fulfilled the [Prerequisites](#).
- Set up your IDE as described in [Set up the OpenText Thrust Studio IDE](#).
- Connect to your developer organization as described in [Set up a connection to the developer organization](#).
- Download the finished version of the Contract Approval application as described in [Download the Contract Approval application](#).
- Once downloaded and extracted, make sure to open the **demo-contract-approval-app-master** folder in VS Code, as this is the root of your project.
- Deploy the application into your developer organization as described in [Deploy an application to the OpenText Thrust Services](#).
- Update the **.env** file as described in [Set the environment variables](#).

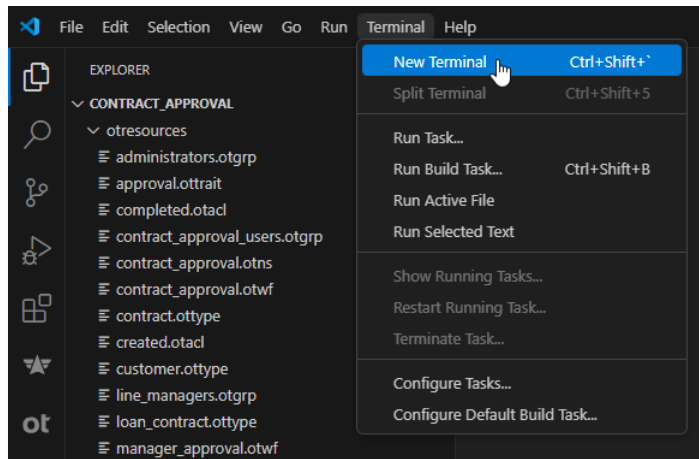
To help with understanding the application that you are going to test, this is a very short explanation of its main project folders:

- **src**: contains the JavaScript and React code that communicates with the OpenText Thrust Services APIs and provides the User Interface (UI) of the application
- **otresources**: contains the different models (built with OpenText Thrust Studio) that you deployed to the OpenText Thrust Services

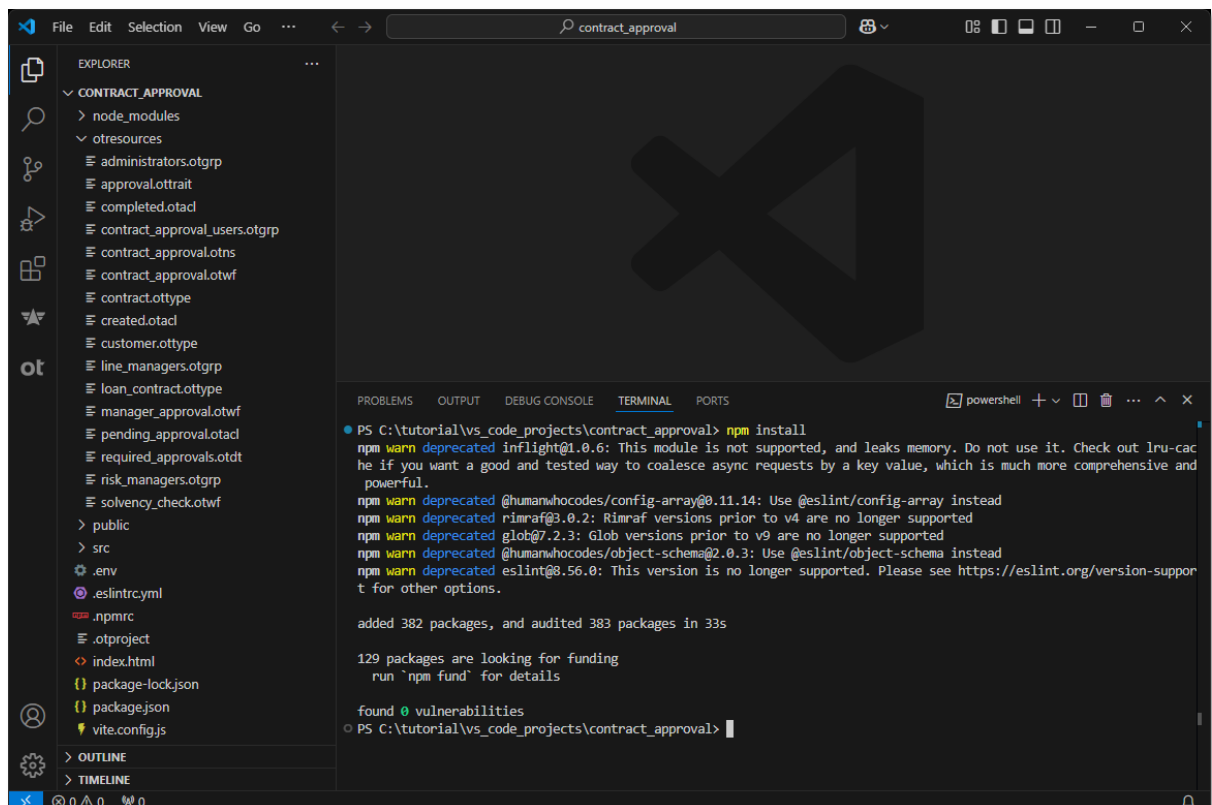
You are now ready to proceed with this exercise and test the Contract Approval application.

15.1 Start the Contract Approval application and sign in

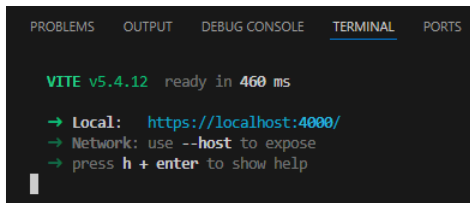
1. Open the Contract Approval application project in VS Code and open a new **Terminal** window.



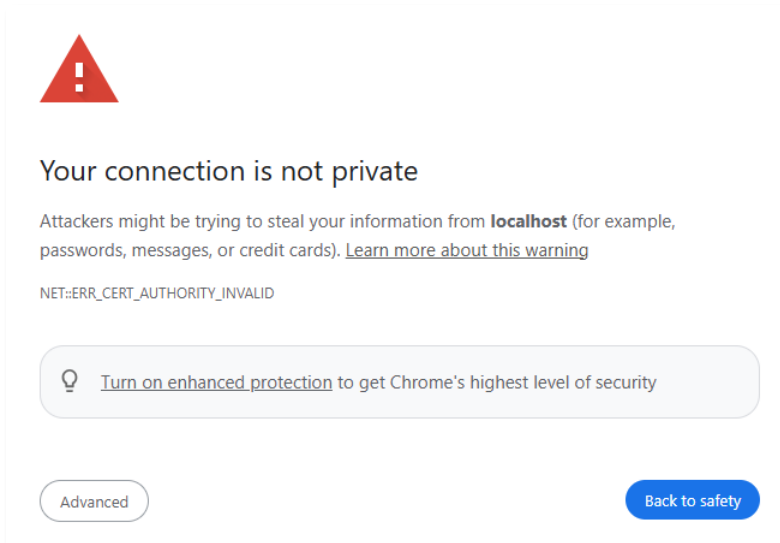
2. In the **Terminal** window run the `npm install` command. This will install all dependencies needed by your application to run.



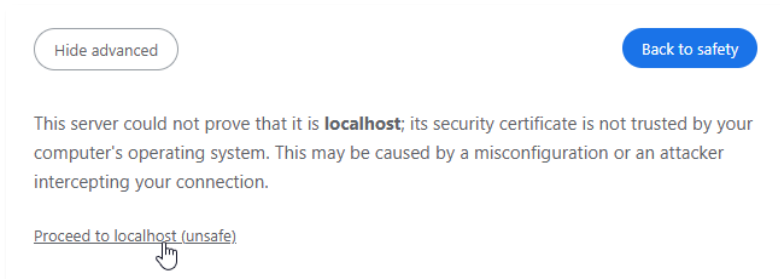
- After the npm install process has completed, launch the application using `npm start`. This will result in a new browser window opening on **https://localhost:4000**.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
  
VITE v5.4.12 ready in 460 ms  
→ Local: https://localhost:4000/  
→ Network: use --host to expose  
→ press h + enter to show help
```



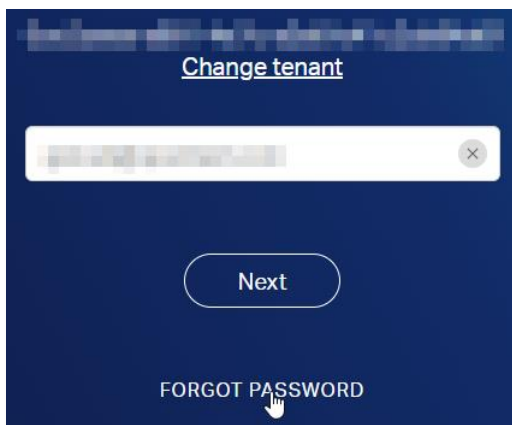
Click the **Advanced** button and select **Proceed to localhost (unsafe)**. Note that the screen shots are of Google Chrome and that the equivalent action on your own web browser can be different.



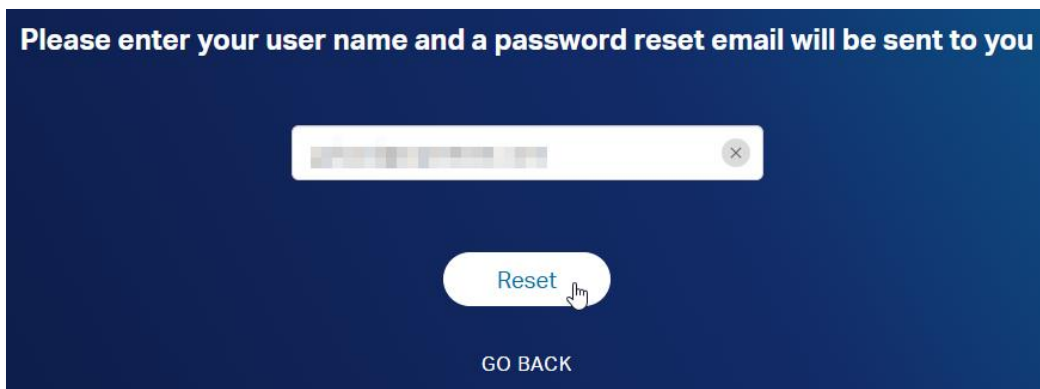


The screenshot shows the OpenText Cloud Platform login interface. At the top, the logo 'opentext™' is followed by 'Cloud Platform'. Below this is a blurred header area. A link labeled 'Change tenant' is centered. Underneath is a text input field with the placeholder text 'User name or email'. Below the input field is a rounded button labeled 'Next'. At the bottom, the text 'FORGOT PASSWORD' is displayed.

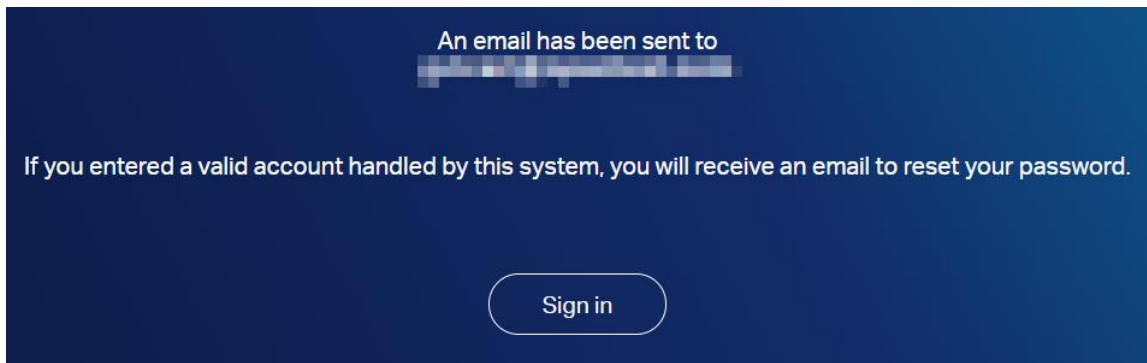
4. In case you have a new subscription and/or have not yet set your tenant admin password, you need to (re)set it from this screen before logging in as you are using the tenant admin email for the **regular user** (the **line manager** and **risk manager** passwords have been set earlier in this tutorial). To reset the tenant admin password, fill the **regular user email address** and select **FORGOT PASSWORD**.



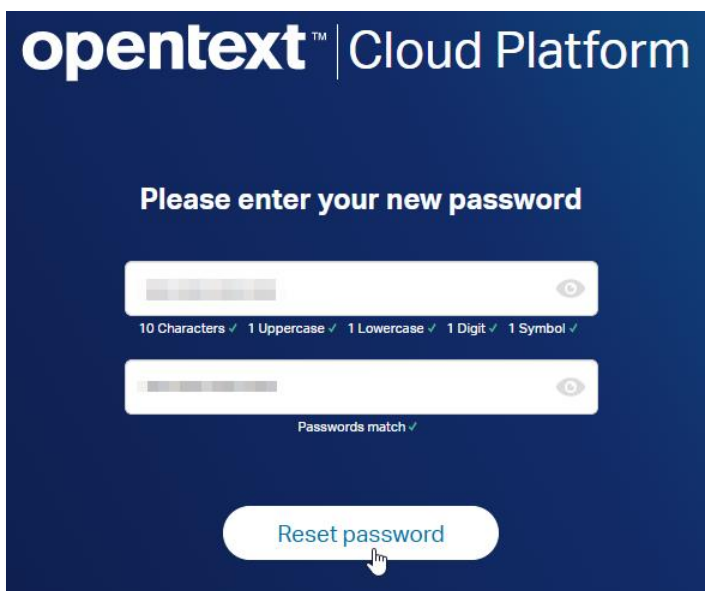
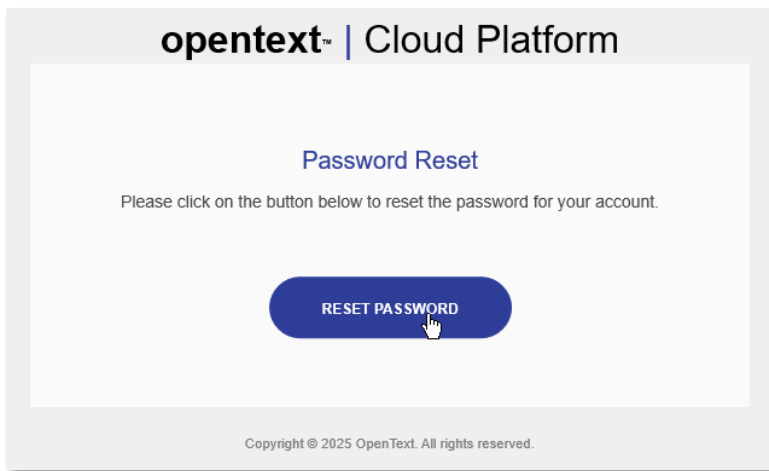
This screenshot shows the same login screen as before, but the text input field now contains a blurred email address. The 'Next' button and 'FORGOT PASSWORD' link remain visible.



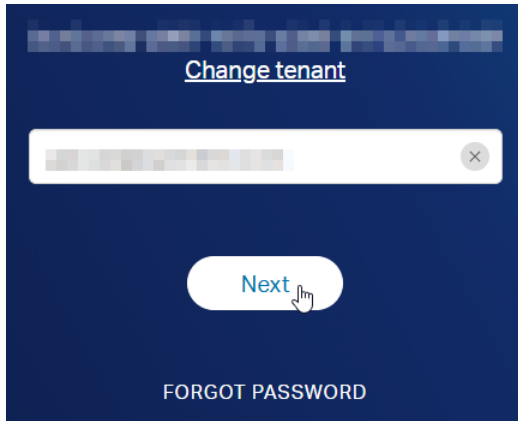
The screenshot shows a new screen titled 'Please enter your user name and a password reset email will be sent to you'. It features a text input field with a blurred email address. Below the input field is a rounded button labeled 'Reset' in blue text. At the bottom, the text 'GO BACK' is displayed.



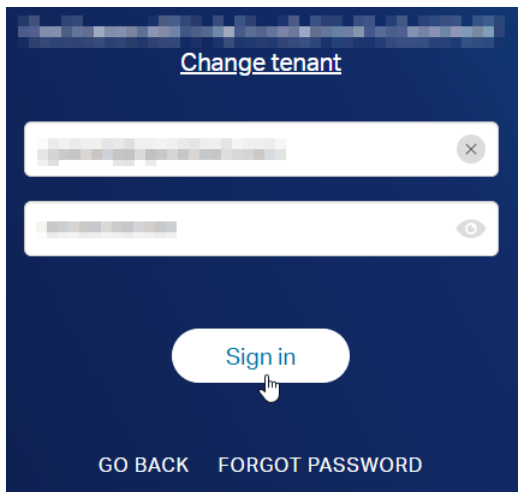
5. Go to your email inbox and from the new **OpenText Password Reset** email, click the **RESET PASSWORD** button, and reset the password.



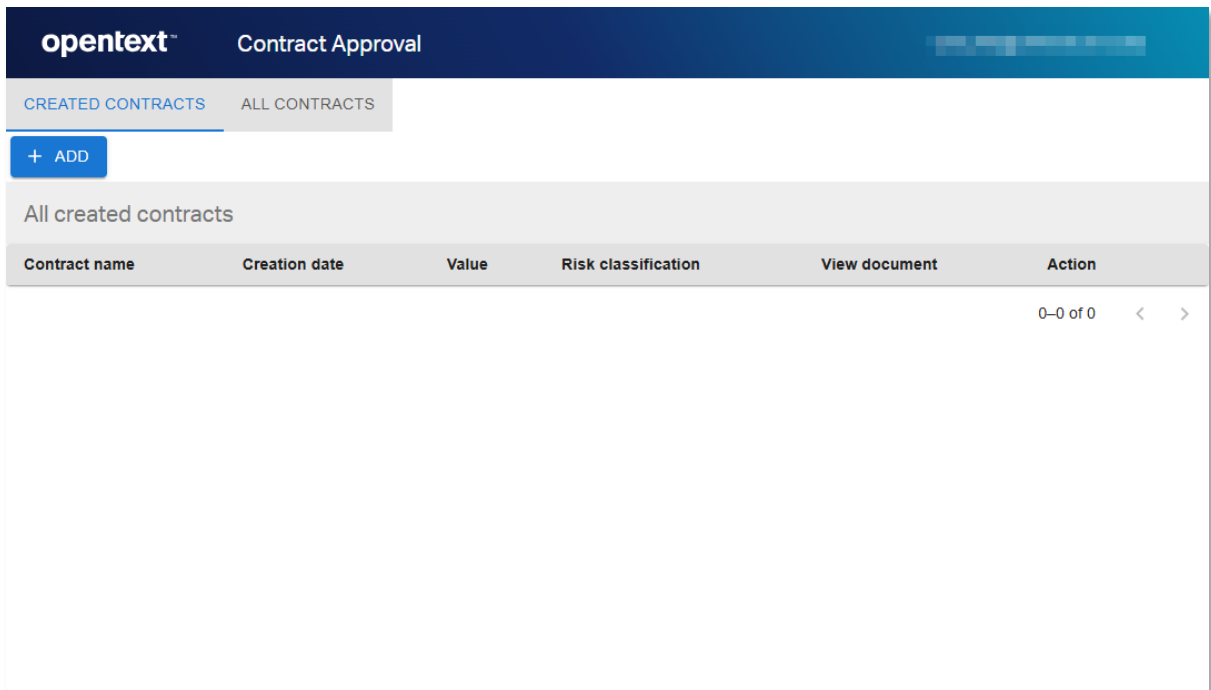
6. Close and reopen the browser, navigate to **localhost:4000** and sign in.



Change tenant

[Next](#)
[FORGOT PASSWORD](#)

Change tenant

[Sign in](#)
[GO BACK](#) [FORGOT PASSWORD](#)

opentext™ Contract Approval

CREATED CONTRACTS ALL CONTRACTS

+ ADD

All created contracts

Contract name	Creation date	Value	Risk classification	View document	Action
0-0 of 0 < >					

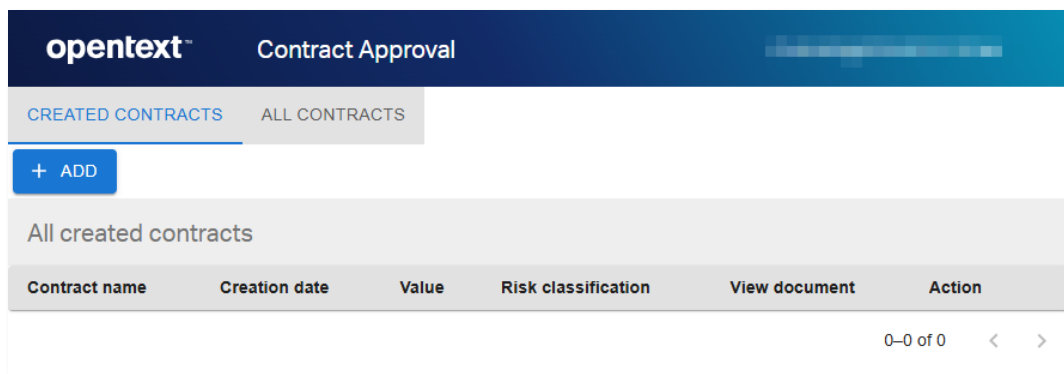
You are now logged in to the Contract Approval application.

Next step:

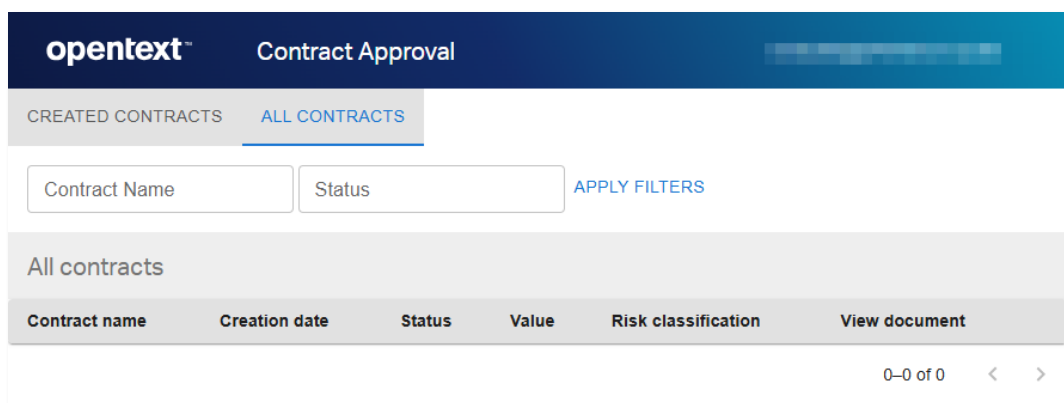
Approve a standard contract that only requires automatic approval (no additional required approvals).

15.2 Approve a standard contract that only requires automatic approval (no additional required approvals)

1. Have a look at the different tabs. As previously described when discussing the application code, as a regular user (that is, not being a member of the **line_managers** or **risk_managers** groups), the currently connected user can only see two tabs from the four tabs in total:
 - **CREATED CONTRACTS**: this tab shows all newly created contracts that have not yet been submitted for approval (that is, contracts with the status attribute equal to 'CREATED')

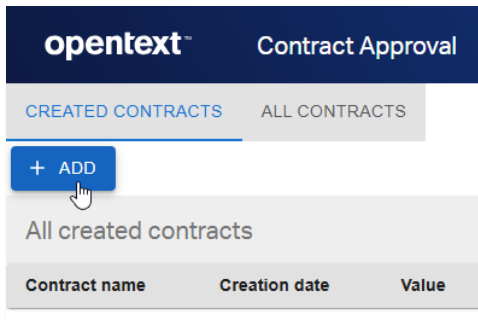


- **ALL CONTRACTS**: this tab shows all contracts, independently of their status (that is, created, pending approval, approved, rejected, expired) with the ability to filter on contract name and status (not case sensitive).



2. After checking the different tabs that the regular user can see, create a first standard contract. To trigger the simplest approval process, you will create a contract with the following characteristics:
 - **Type: standard contract** (doesn't require solvency check)
 - **Value: below 1000** (doesn't require Line Manager approval)
 - **Risk classification: below 4, that is, NONE, LOW or MEDIUM** (doesn't require Risk Manager approval)

3. Select the **CREATED CONTRACTS** tab and click the **+ ADD** button to open the contract creation form.

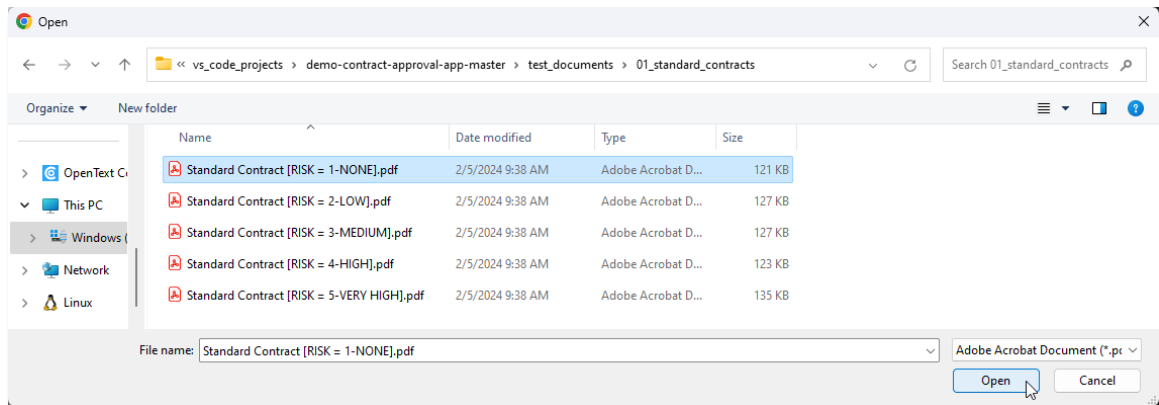


4. From the **Add Contract** screen, click **SELECT DOCUMENT** to add the contract content file.

A screenshot of the 'Add Contract' form in the OpenText Contract Approval application. The form has a white background and a thin grey border. At the top, the title 'Add Contract' is displayed. Below the title, there is a blue button with a white cloud icon and the text 'SELECT DOCUMENT'. A mouse cursor is pointing at this button. Below the button, there are two radio buttons: 'Standard Contract' (which is selected) and 'Loan Contract'. Below the radio buttons, there are three input fields: 'Document name', 'Contract value', and 'Contract requester email'. At the bottom right of the form, there are two buttons: 'ADD' (disabled, grey) and 'CANCEL' (active, blue).

5. To help with selecting a file that matches the intended contract properties (certainly the risk classification, as this gets determined by what the Risk Guard Service discovers in the document), a **test_documents** folder is provided under the finished version of the Contract Approval application project.

From this **test_documents** folder, open the **01_standard_contracts** subfolder and select the **Standard Contract [RISK = 1-NONE].pdf** file.




6. Make sure the **Standard Contract** option is selected and fill the contract properties as follows:

Property	Value
Document name	First standard contract
Contract value	500
Contract requester email	<your email>

7. Click **Add** to create the contract.

Add Contract

 **SELECT DOCUMENT** Standard Contract [RISK = 1-NONE].pdf

☒ Standard Contract ☐ Loan Contract

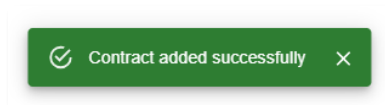
Document name

Contract value


Contract requester email

ADD **CANCEL**


At the bottom of the screen, the creation of the new contract is confirmed by a “Contract added successfully” message.



The first standard contract is now created.

CREATED CONTRACTS					
ALL CONTRACTS					
+ ADD					
All created contracts					
Contract name	Creation date	Value	Risk classification	View document	Action
First standard contract	2/13/2024, 2:17:53 PM	500	NONE 	ORIGINAL	REQUEST APPROVAL >
1-1 of 1 < >					

8. Explore the contract list capabilities from the **CREATED CONTRACTS** view.

First, note that the **Risk classification** property indeed shows **NONE** as risk level. Click on the  (information icon) next to the **NONE** risk classification value to see which terms the call to the Risk Guard Service API has identified and extracted.

CREATED CONTRACTS

ALL CONTRACTS

+ ADD

All created contracts

Contract name	Creation date	Value	Risk classification	View document	Action
First standard contract	2/13/2024, 2:17:53 PM	500	NONE	ORIGINAL	REQUEST APPROVAL

Show extracted personal data

1-1 of 1

Extracted Terms	
Social Security Numbers:	
Credit Card Numbers:	
Bank Accounts:	
Person Names:	
- Peter M. Townsend	
- Tomas U. Britton	
Phone Numbers:	
Addresses:	
Geographic Locations:	
- Helena, Georgia	
- State of Georgia	
Organization Names:	
- Innovate	
CLOSE	

A few names, addresses, geographic locations, and organization names were found, but nothing that warrants increasing the risk level (hence risk classification = NONE).


- Click **CLOSE** to close the **Extracted Terms** information screen.
- Click on the **ORIGINAL** button (in the **View document** column) to view the uploaded document content.

CREATED CONTRACTS

ALL CONTRACTS

+ ADD

All created contracts

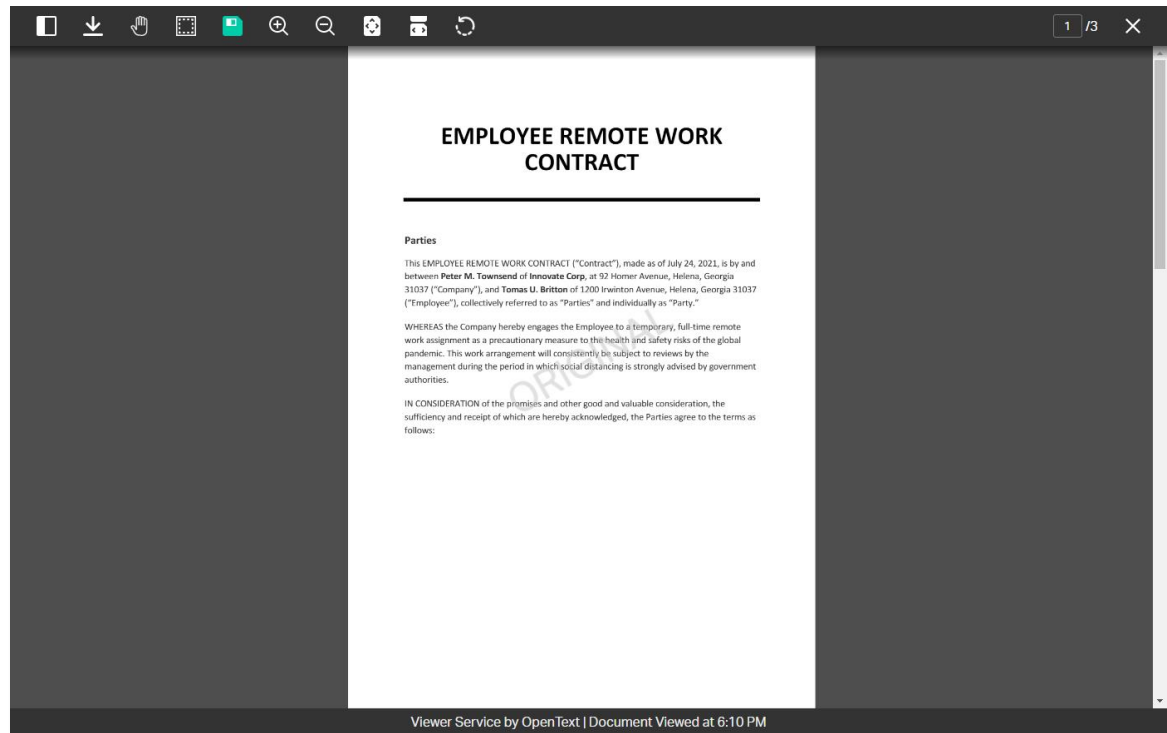
Contract name	Creation date	Value	Risk classification	View document	Action
First standard contract	2/13/2024, 2:17:53 PM	500	NONE 	<div>ORIGINAL</div>	<div>REQUEST APPROVAL</div> <div>></div>

1-1 of 1












<

>

The file content of the contract displays to be viewed (in the viewer from the Viewer Service).

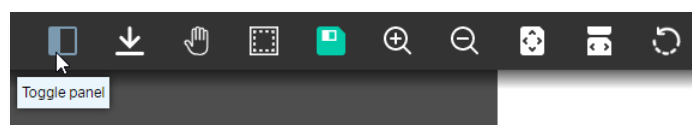


The toolbar at the top of the viewer allows multiple actions (as defined in the code of the FileViewer.jsx file):

-  Toggle panel
-  Download document
-  Toggle pan mode (allowing to use the cursor to navigate the document)
-  Zoom window (allowing to draw a window on the document to zoom into)
-  Save markups
-  Zoom in
-  Zoom out
-  Fit document to screen
-  Fit width to screen
-  Rotate left
-  Page selector

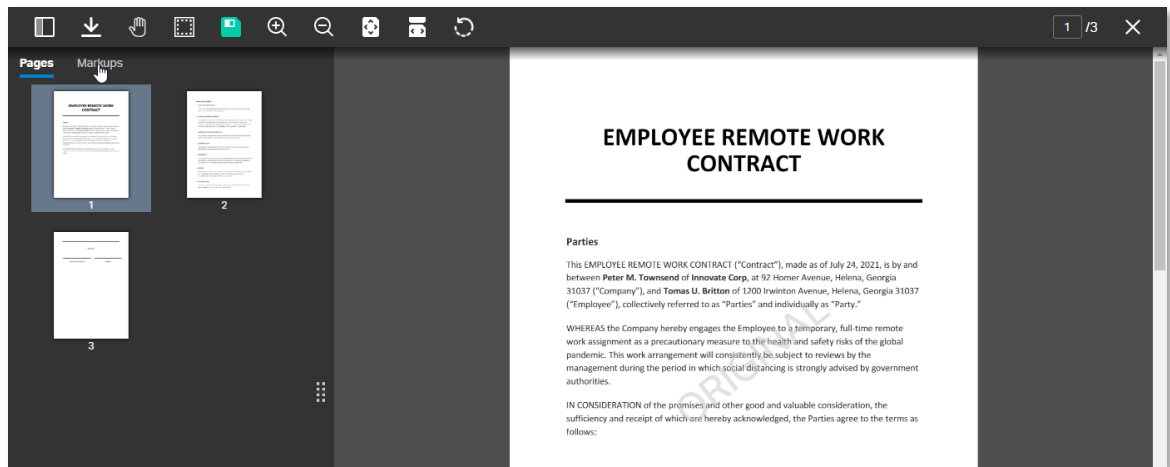
11. Although most of the actions in the toolbar are self-explanatory, have a look at the panel toggle.

Click the **Toggle panel** button to open the (left) side panel.

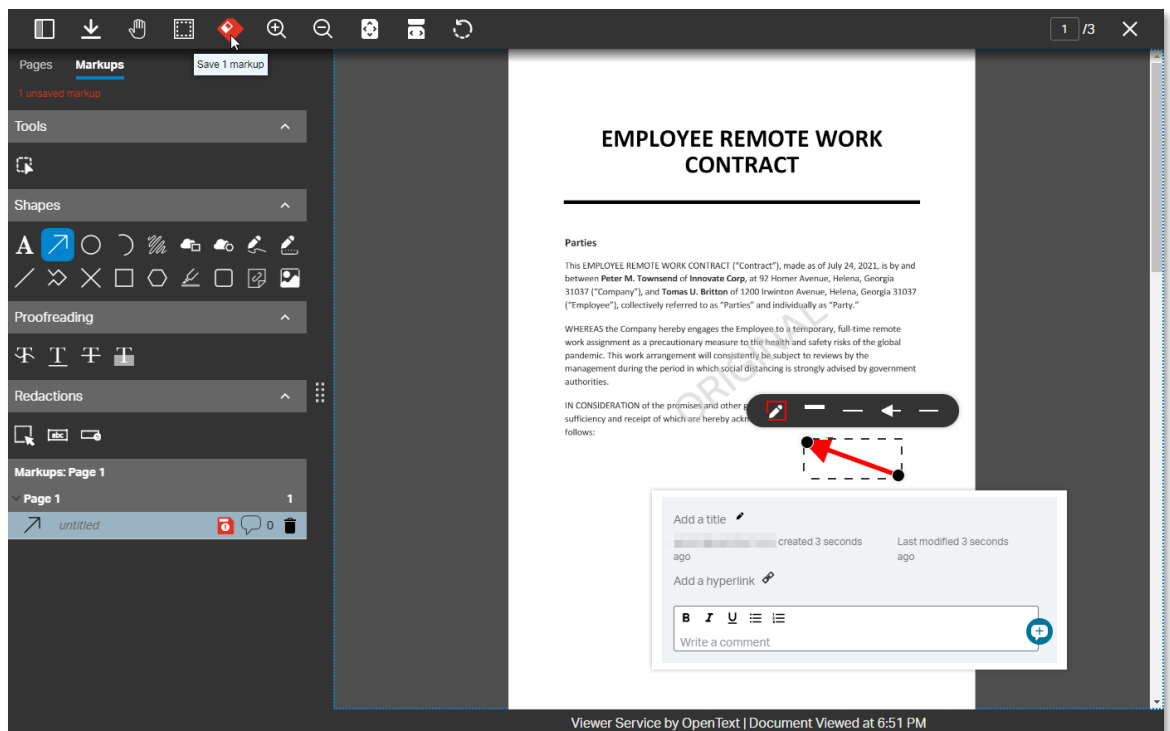


The side panel shows two tabs (again as per the code in FileViewer.jsx). The **Pages** tab shows the page thumbnails, and the **Markups** tab provides the markup tools.

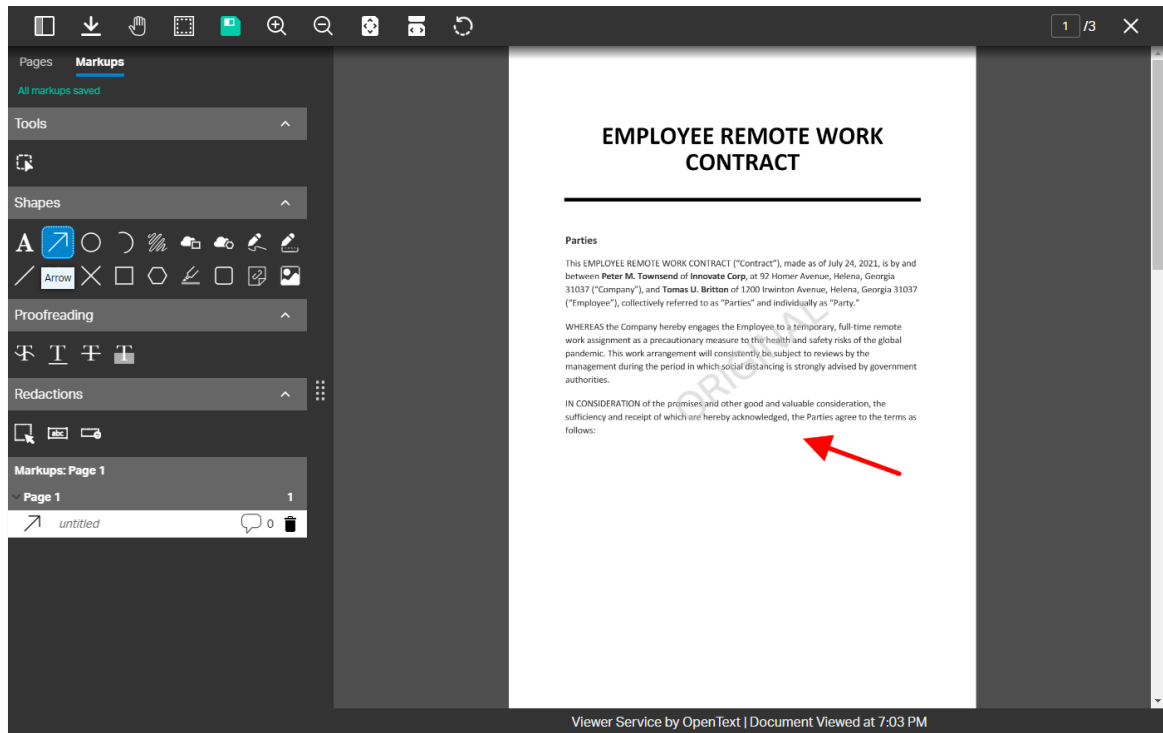
12. Click the **Markups** tab to switch to the markup tools view.




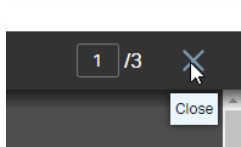
13. Select a markup shape, draw it on the document (for example, an **arrow**), and click the **Save markups** button to save it.



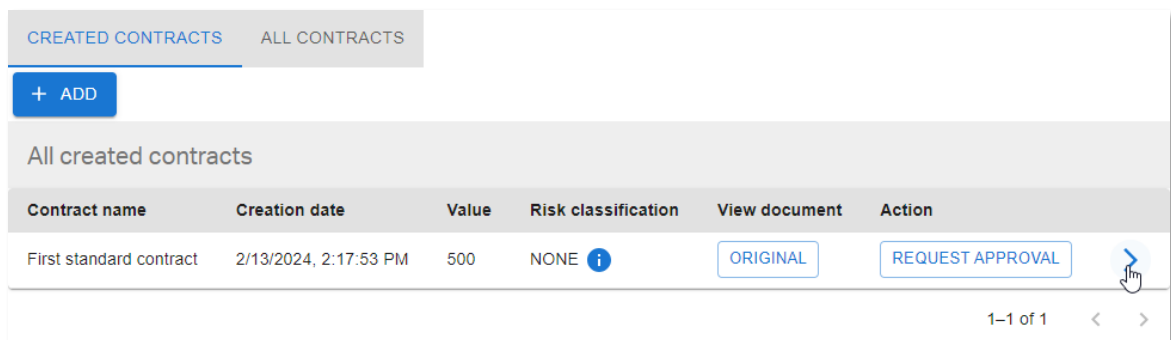
14. You can explore creating/modifying/deleting/commenting other markups, but for the purpose of this tutorial just press the **esc** key to unselect the markup and proceed with further examining the viewer.



15. The two last things defined in the FileViewer.jsx code to draw your attention to are the **ORIGINAL** watermark (across the document pages) and the **Viewer Service by OpenText | Document Viewed at <computed viewing time>** footer.
16. To go back to the main application screen, close the viewer with the  on the top right.



17. Click on  (caret arrow icon) to view the contract details.



18. In the **Contract details** screen, there are three tabs. The first one shows the contract properties.

Note that the **PROPERTIES** tab indeed displays a status of 'CREATED' and that the risk classification is 1 (which is the corresponding integer value for the NONE risk level).

19. Click the **PERMISSIONS** tab to have a look at the different permissions for this contract.

The screenshot shows a web application window titled "Contract details". At the top, there are three tabs: "PROPERTIES" (which is selected and underlined in blue), "PERMISSIONS", and "APPROVALS". A mouse cursor is pointing at the "PERMISSIONS" tab. Below the tabs, there are six input fields, each with a label and a value:

- Name: First standard contract
- Status: CREATED
- Value: 500
- Risk classification: 1
- Creation date: 2/13/2024, 2:17:53 PM
- Contract requester email: [redacted]

In the bottom right corner of the window, there is a blue button labeled "CLOSE".

20. The **PERMISSIONS** tab displays the permissions for the contract (full control access for the **administrators**, read access for the **contract_approval_users**, and full control for the **owner**).

Contract details

PROPERTIES **PERMISSIONS** APPROVALS

Identity	Permissions
administrators	<ul style="list-style-type: none"> • browse • change_folder_links • change_location • change_owner • change_permission • create_link • delete • delete_link • lock • read_content • relate • version • write • write_content
contract_approval_users	<ul style="list-style-type: none"> • browse • read_content
owner	<ul style="list-style-type: none"> • browse • change_folder_links • change_location • change_owner • change_permission • create_link • delete • delete_link • lock • read_content • relate • version • write • write_content

CLOSE

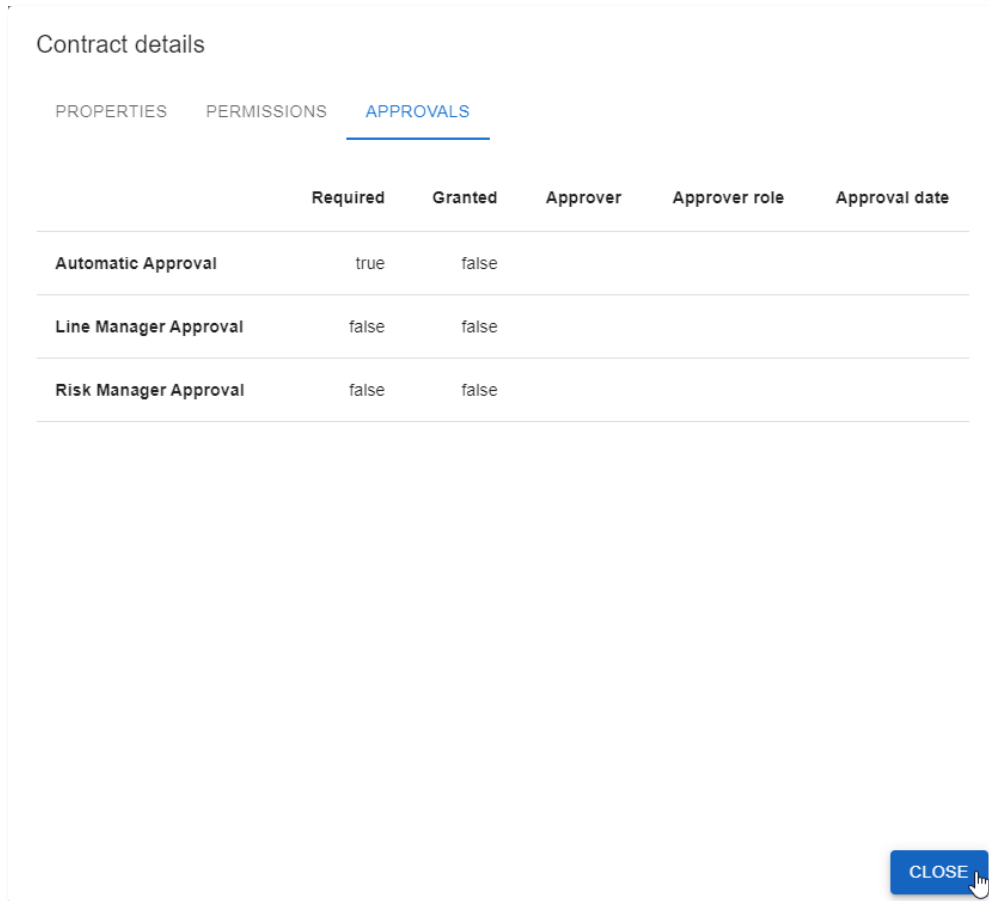
21. Click the **APPROVALS** tab to view the different approvals for this contract.

Contract details

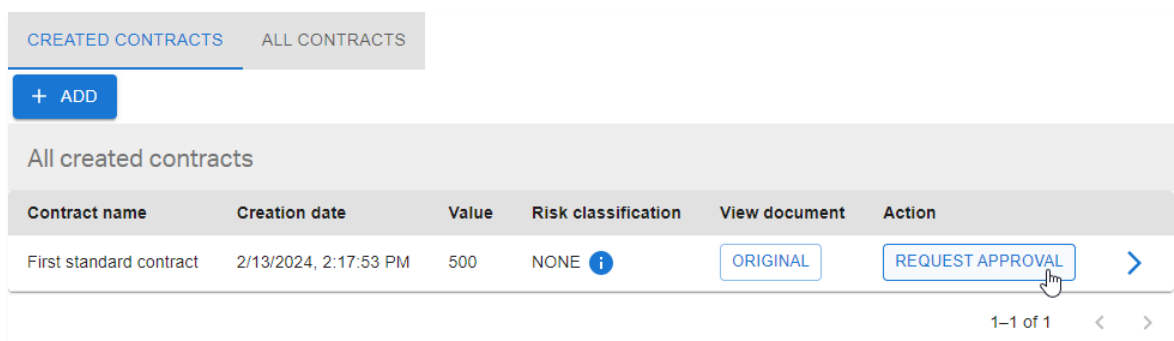
PROPERTIES **PERMISSIONS** APPROVALS

Identity	Permissions
administrators	<ul style="list-style-type: none"> • browse • change_folder_links

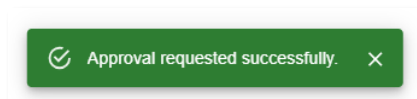
22. The **APPROVALS** tab displays the different approval steps (traits) for the standard contract type (**Automatic Approval**, **Line Manager Approval** and **Risk Manager Approval**). As you can see, the only approval step marked as required is the **Automatic Approval**.
23. Click **CLOSE** to close the **Contract details** screen.



24. Back in the **CREATED CONTRACTS** view, you can now launch the approval workflow. Click the **REQUEST APPROVAL** button in the **Action** column to do that.



At the bottom of the screen the “Approval requested successfully” message confirms that the approval process has started.





25. The new contract has now disappeared from the **CREATED CONTRACTS** view, as it is no longer in 'CREATED' status.

Contract name	Creation date	Value	Risk classification	View document	Action
0-0 of 0					

26. Since the value of the contract is below 1000, no line manager approval is required. There is also no need for the risk manager to approve, as the risk classification is NONE (1) which is below HIGH (4).

Open the **ALL CONTRACTS** view to confirm the contract has been automatically approved (**Status** column shows **APPROVED** status), without needing any manual approval by the line manager or risk manager.

27. Open the contract details as well, and more specifically the approval steps/traits by clicking  (caret arrow icon) and selecting the **APPROVALS** tab.

Contract name	Creation date	Status	Value	Risk classification	View document
First standard contract	1/20/2025, 3:46:52 PM	APPROVED	500	NONE 	ORIGINAL 

28. The **APPROVALS** tab still shows that only the **Automatic Approval** was required, but with the difference that it has now been granted by **Approver SYSTEM** with the **Approver role** of **Automatic Approval** at a specific **Approval date** and time.

Click **CLOSE** to return to the **ALL CONTRACTS** view.

Contract details

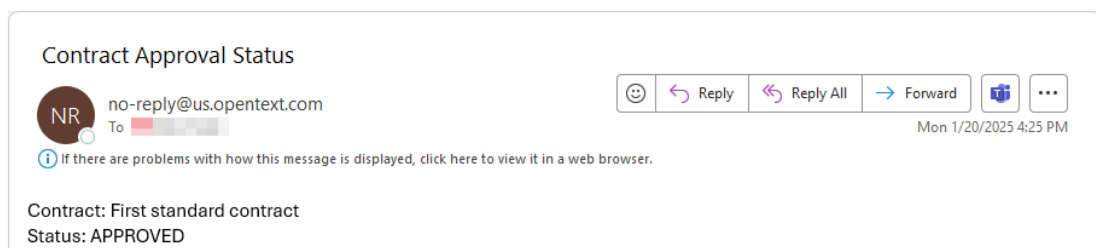
PROPERTIES PERMISSIONS **APPROVALS**

	Required	Granted	Approver	Approver role	Approval date
Automatic Approval	true	true	SYSTEM	Automatic Approval	2025-01-20
Line Manager Approval	false	false			
Risk Manager Approval	false	false			

CLOSE

29. The first standard contract is now approved.

Go to your email client and check the inbox and confirm receiving (due to the email task in the workflow) a **Contract Approval Status** email from **no-reply@<region>.opentext.com**.

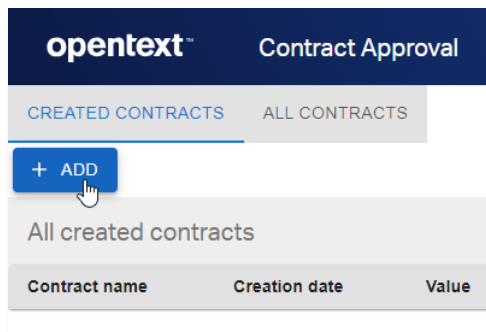


Next step:

Approve a loan contract that requires all additional approvals.

15.3 Approve a loan contract that requires all additional approvals

- The second contract to create will follow the most extensive process flow (requiring all automated and manual approvals). To that end, you will create a contract with the following characteristics:
 - Type: loan contract** (requires solvency check)
 - Monthly loan cost is below or equal to 25% of monthly income** (requester is solvent, so automatic solvency check will not reject the contract approval request)
 - Value: above 5000** (requires Line Manager approval)
 - Risk classification: above 3, that is, HIGH or VERIFY HIGH** (requires Risk Manager approval)
- Select the **CREATED CONTRACTS** tab and click the **+ ADD** button to open the contract creation form.



- From the **Add Contract** screen, click **SELECT DOCUMENT** to add the contract content file.

Add Contract

SELECT DOCUMENT

☒ Standard Contract ☐ Loan Contract

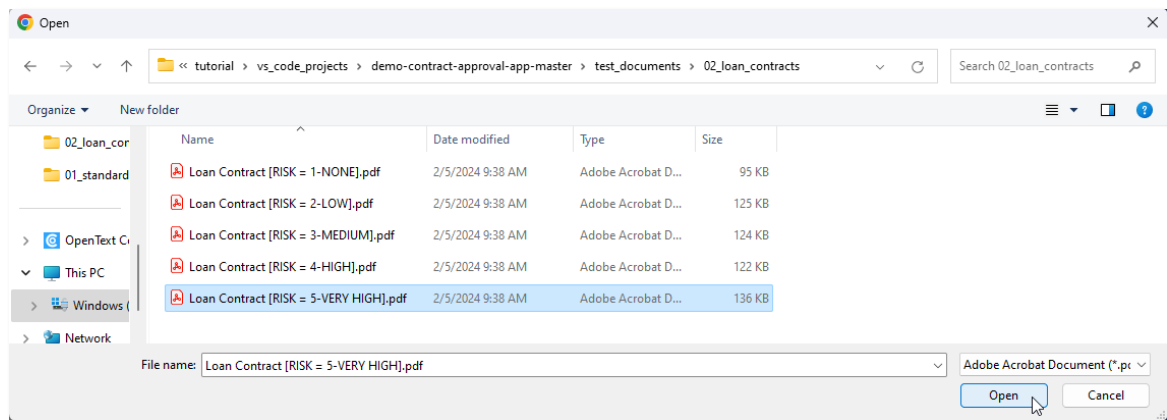
Document name

Contract value

Contract requester email

ADD CANCEL

4. From the **test_documents** folder, open the **02_loan_contracts** subfolder and select the **Loan Contract [RISK = 5-VERY HIGH].pdf** file.



5. Select the **Loan Contract** option and fill the contract properties as follows:

Property	Value
Document name	First loan contract
Contract value	12000
Monthly installments	12
Yearly income	100000
Contract requester email	<your email>

6. Click **Add** to create the contract.

Add Contract

SELECT DOCUMENT
Loan Contract [RISK = 5-VERY HIGH].pdf

☐ Standard Contract
☒ Loan Contract

Document name
First loan contract

Contract value
12000

Monthly installments
12

Yearly income
100000

Contract requester email



ADD CANCEL

CREATED CONTRACTS



ALL CONTRACTS

+ ADD

All created contracts

Contract name	Creation date	Value	Risk classification	View document	Action
First loan contract	2/13/2024, 6:33:19 PM	12000	VERY HIGH 	ORIGINAL	REQUEST APPROVAL 

1-1 of 1



7. Click on  (information icon) next to the **VERY HIGH** risk classification value to see which terms the call to the Risk Guard Service API has identified and extracted.

CREATED CONTRACTS

ALL CONTRACTS

+ ADD

All created contracts

Contract name	Creation date	Value	Risk classification	View document	Action
First loan contract	2/13/2024, 6:33:19 PM	12000	VERY HIGH	<div>ORIGINAL</div>	<div>REQUEST APPROVAL</div>

1-1 of 1

< >

Contrary to the previous contract you created, this contract contains high risk personal information, such as a social security number (considered very high risk), a credit card number, a bank account, and many person names (hence risk classification = VERY HIGH). Some addresses, geographic locations, and organization names were also found.

8. Click **CLOSE** to close the **Extracted Terms** information screen.

Extracted Terms

Social Security Numbers:

- 778-62-8144

Credit Card Numbers:

- 5105 1051 0510 5100

Bank Accounts:

- BE71 0961 2345 6769

Person Names:

- Anna C. Quebral
- James B. Davis
- John W. Snow
- Janette Jamison-Johnson
- Sandra Kauffmann
- Anthony Hopkins
- James Bond
- Brenda Ann Oliver-Johnson
- Frank Joe Parker
- Michael Colbert-Johnson
- Andy McIntosh
- Emily Jackson
- Pete Peterson

Phone Numbers:

Addresses:

Geographic Locations:



- Indianapolis, Indiana
- Elk City Road
- Bloomington, Indiana

Organization Names:

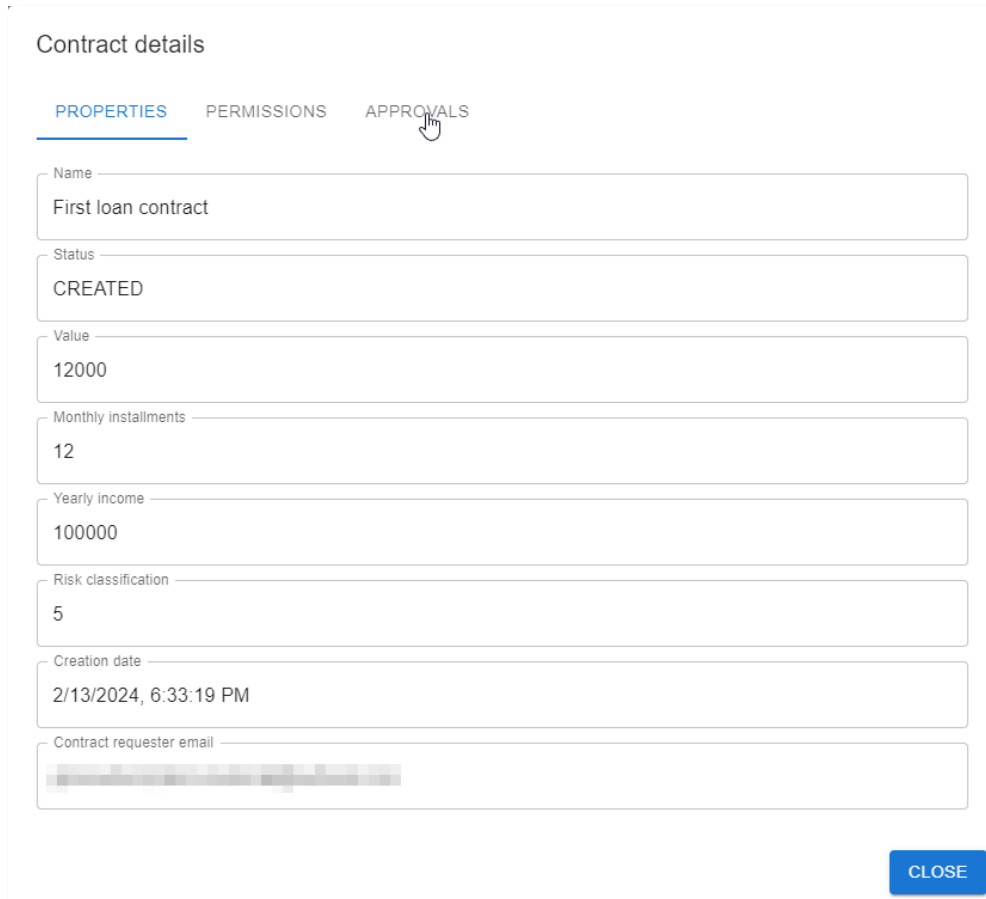
- Innovate Bank

CLOSE

9. Click on > (caret arrow icon) to view the contract details.

CREATED CONTRACTS ALL CONTRACTS						
+ ADD						
All created contracts						
Contract name	Creation date	Value	Risk classification	View document	Action	
First loan contract	2/13/2024, 6:33:19 PM	12000	VERY HIGH 	ORIGINAL	REQUEST APPROVAL	
1-1 of 1 < >						

10. In the **PROPERTIES** tab of the **Contract details** screen note that, since this is a loan contract, the monthly installments and yearly income are also displayed. The risk classification is now equal to 5 (the corresponding integer value for the VERY HIGH risk level).
11. Click the **APPROVALS** tab to have a look at the different approvals for this contract.



Contract details

PROPERTIES PERMISSIONS APPROVALS

Name
First loan contract

Status
CREATED

Value
12000

Monthly installments
12

Yearly income
100000

Risk classification
5

Creation date
2/13/2024, 6:33:19 PM

Contract requester email
[Redacted]

CLOSE

12. The **APPROVALS** tab displays the different approval steps (traits) for the loan contract type (**Automatic Approval**, **Line Manager Approval**, **Risk Manager Approval**, and the additional **Solvency Check**).

13. Click **CLOSE** to close the **Contract details** screen.

Contract details

PROPERTIES PERMISSIONS APPROVALS

	Required	Granted	Approver	Approver role	Approval date
Automatic Approval	true	false			
Line Manager Approval	false	false			
Risk Manager Approval	false	false			
Solvency Check	false	false			

CLOSE

14. Back in the **CREATED CONTRACTS** view, you can now launch the approval workflow. Click the **REQUEST APPROVAL** button in the **Action** column to do that.

CREATED CONTRACTS ALL CONTRACTS

+ ADD


All created contracts

Contract name	Creation date	Value	Risk classification	View document	Action
First loan contract	2/13/2024, 6:33:19 PM	12000	VERY HIGH <i>i</i>	ORIGINAL	REQUEST APPROVAL <i>h</i>

1-1 of 1 < >

15. The new contract has now disappeared from the **CREATED CONTRACTS** view, as it is no longer in 'CREATED' status. Open the **ALL CONTRACTS** tab to see its current status.

The status is now 'PENDING APPROVAL'.

16. Click  (caret arrow icon) to view the contract details again.

CREATED CONTRACTS





ALL CONTRACTS

Contract Name

Status

APPLY FILTERS

All contracts

Contract name	Creation date	Status	Value	Risk classification	View document
First loan contract	1/20/2025, 4:37:50 PM	PENDING APPROVAL	12000	VERY HIGH 	ORIGINAL 
First standard contract	1/20/2025, 3:46:52 PM	APPROVED	500	NONE 	ORIGINAL 

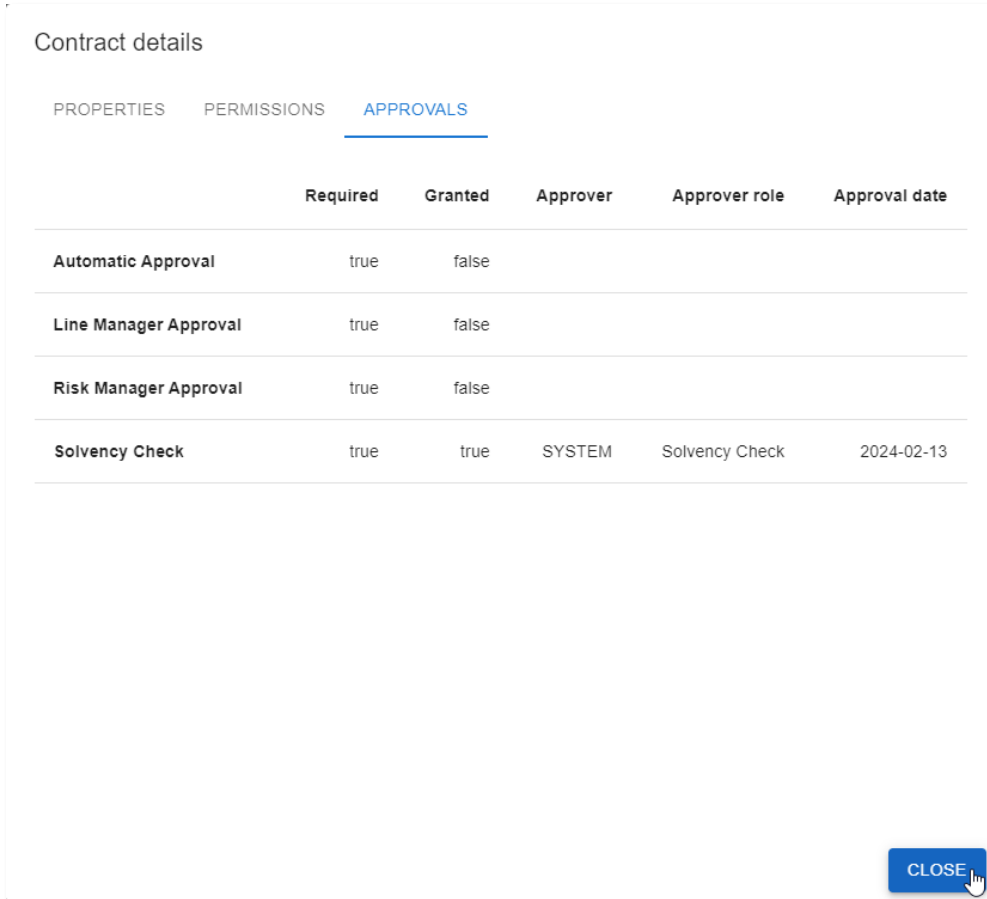
1-2 of 2

17. From the **PERMISSIONS** tab you can see that the ACL is changed to grant custom permissions to allow the approval by the line manager and the risk manager.

Contract details	
	<ul style="list-style-type: none"> lock read_content relate version write write_content
contract_approval_users	<ul style="list-style-type: none"> browse read_content
line_managers	<ul style="list-style-type: none"> browse change_permission read_content write write_content
owner	<ul style="list-style-type: none"> browse change_permission read_content write write_content
risk_managers	<ul style="list-style-type: none"> browse change_permission read_content write write_content
CLOSE	

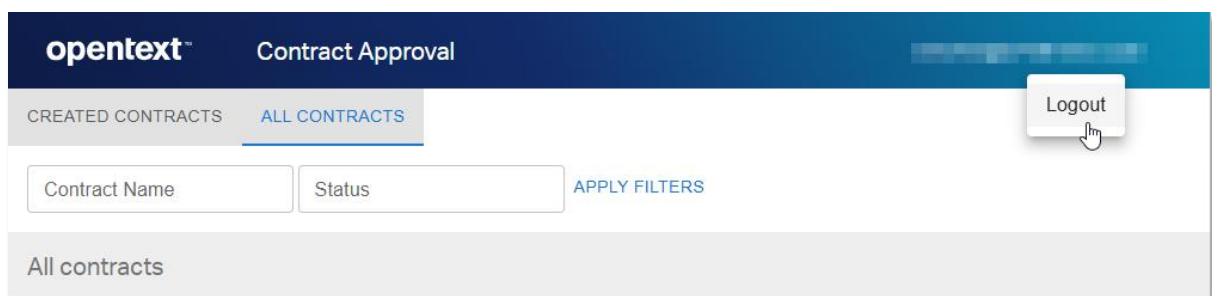
18. Additionally, from the **APPROVALS** tab, you can see that the **Solvency Check** approval is granted (because the requester is solvent) and that the **Line Manager Approval** and **Risk Manager Approval** are required.

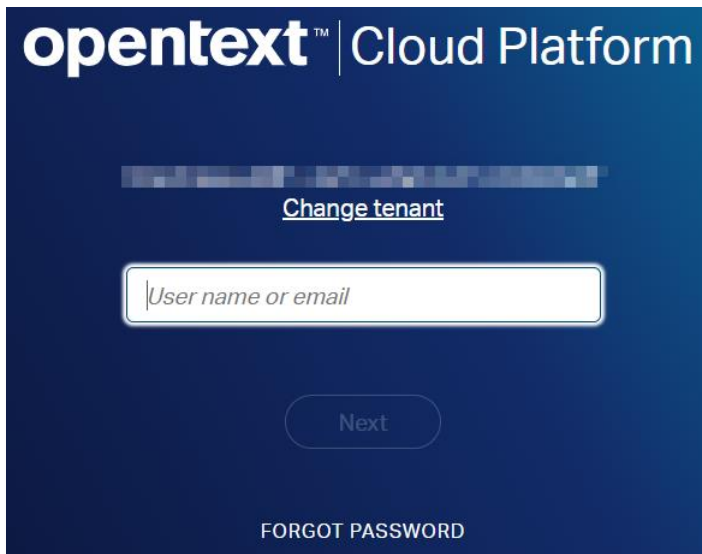
Click **CLOSE** to close the **Contract details** screen.



	Required	Granted	Approver	Approver role	Approval date
Automatic Approval	true	false			
Line Manager Approval	true	false			
Risk Manager Approval	true	false			
Solvency Check	true	true	SYSTEM	Solvency Check	2024-02-13

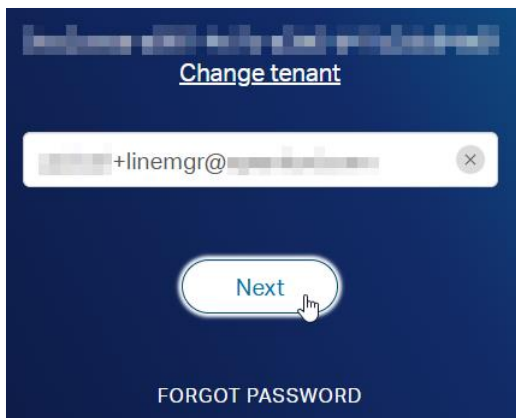
19. To perform the manual approval by the line manager, you must first sign in as the line manager. To do this, click the user name (email) at the top of the application screen and choose **Logout**.



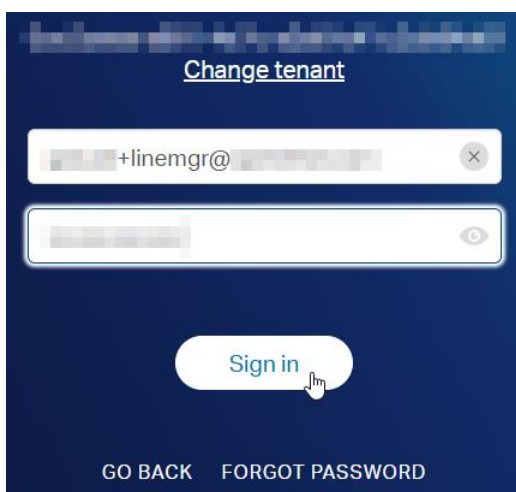


The screenshot shows the OpenText Cloud Platform login interface. At the top, the logo "opentext™" is followed by "Cloud Platform". Below this is a link "Change tenant". A text input field contains the placeholder text "User name or email". Below the input field is a "Next" button. At the bottom, there is a link "FORGOT PASSWORD".

20. Sign in with the **line manager** email (the one with **+linemgr** before the **@**).



This screenshot shows the login screen after the email has been entered. The text input field now contains "+linemgr@". A cursor is visible at the end of the text. The "Next" button is highlighted with a mouse cursor. The "Change tenant" link and "FORGOT PASSWORD" link are still present.



This screenshot shows the login screen after the password has been entered. The text input field now contains a password. A "Sign in" button is highlighted with a mouse cursor. The "Change tenant" link, the email input field, and the "FORGOT PASSWORD" link are still present. A "GO BACK" link has appeared at the bottom left.

21. Since you are now logged in as line manager, in addition to the regular user's **CREATED CONTRACTS** and **ALL CONTRACTS** tabs, you can also see the **LINE MANAGER TASKS** tab.

CREATED CONTRACTS LINE MANAGER TASKS ALL CONTRACTS					
+ ADD					
All created contracts					
Contract name	Creation date	Value	Risk classification	View document	Action
0-0 of 0 < >					

22. Click the **LINE MANAGER TASKS** tab. An approval task is waiting for the Line Manager to approve.
23. First a line manager has to claim the approval task. So, click **CLAIM** to assign the approval to your user.

CREATED CONTRACTS LINE MANAGER TASKS ALL CONTRACTS					
All Tasks					
Contract name	Creation date	Value	Risk classification	Assignee	View document Action
First loan contract	2/13/2024, 7:36:07 PM	12000	VERY HIGH ⓘ		ORIGINAL CLAIM >
1-1 of 1 < >					

24. Now you can click **APPROVE** to approve as the Line Manager.

CREATED CONTRACTS LINE MANAGER TASKS ALL CONTRACTS					
All Tasks					
Contract name	Creation date	Value	Risk classification	Assignee	View document Action
First loan contract	2/13/2024, 7:36:07 PM	12000	VERY HIGH ⓘ	+linemgr@	ORIGINAL APPROVE REJECT >
1-1 of 1 < >					

25. The contract has now disappeared from the **LINE MANAGER TASKS** view. Open the **ALL CONTRACTS** tab to see its current status.
- The status is still 'PENDING APPROVAL' because the risk manager approval is also pending.

26. Click on > (caret arrow icon) to view the contract details again.

CREATED CONTRACTS

LINE MANAGER TASKS

ALL CONTRACTS

Contract Name

Status

APPLY FILTERS

All contracts

Contract name	Creation date	Status	Value	Risk classification	View document
First loan contract	1/20/2025, 4:37:50 PM	PENDING APPROVAL	12000	VERY HIGH	<div>ORIGINAL</div> <div>></div>
First standard contract	1/20/2025, 3:46:52 PM	APPROVED	500	NONE	<div>ORIGINAL</div> <div>></div>

1-2 of 2 < >

27. The **APPROVALS** tab shows that the **Line Manager Approval** is granted.

Click **CLOSE** to return to the **ALL CONTRACTS** view.

Contract details					
PROPERTIES PERMISSIONS <u>APPROVALS</u>					
	Required	Granted	Approver	Approver role	Approval date
Automatic Approval	true	false			
Line Manager Approval	true	true	+linemgr@	Line Manager	2025-01-20
Risk Manager Approval	true	false			
Solvency Check	true	true	SYSTEM	Solvency Check	2025-01-20
CLOSE					

28. In the same way as with the line manager, to perform the manual approval by the risk manager, choose to **Logout** and sign in as the risk manager.

opentext™

Contract Approval

CREATED CONTRACTS

LINE MANAGER TASKS

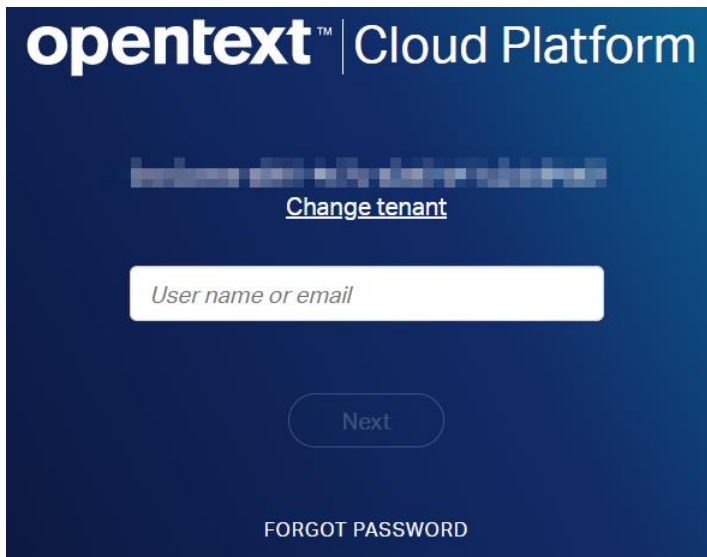
ALL CONTRACTS

Contract Name

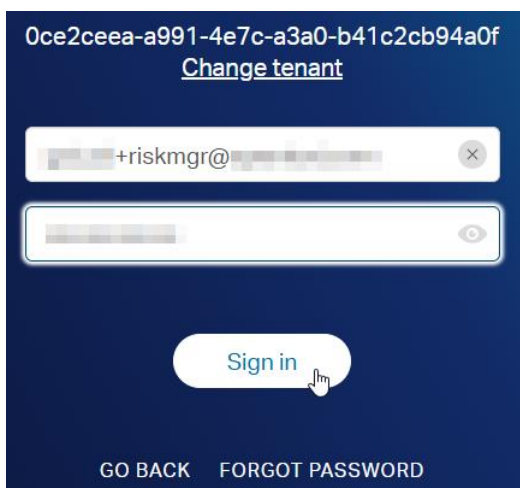
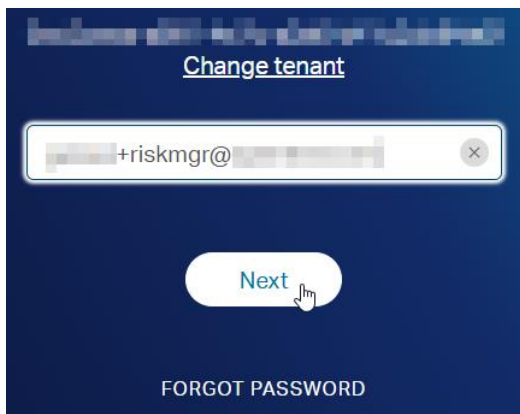
Status

APPLY FILTERS

Logout



29. Sign in with the **risk manager** email (the one with **+riskmgr** before the **@**).



30. Since you are now logged in as risk manager, you can see the **RISK MANAGER TASKS** tab (and no **LINE MANAGER TASKS** tab).

CREATED CONTRACTS

RISK MANAGER TASKS

ALL CONTRACTS

+ ADD

All created contracts

Contract name	Creation date	Value	Risk classification	View document	Action
0-0 of 0					

31. Click the **RISK MANAGER TASKS** tab. An approval task is now waiting for the Risk Manager to approve.
32. First a risk manager has to claim the approval task. So, click **CLAIM** to assign the approval to your user.

CREATED CONTRACTS

RISK MANAGER TASKS

ALL CONTRACTS

All Tasks

Contract name	Creation date	Value	Risk classification	Assignee	View document	Action
First loan contract	2/13/2024, 7:45:50 PM	12000	VERY HIGH 		ORIGINAL	CLAIM 

1-1 of 1  





33. Now you can click **APPROVE** to approve as the Risk Manager.

CREATED CONTRACTS



RISK MANAGER TASKS

ALL CONTRACTS

All Tasks

Contract name	Creation date	Value	Risk classification	Assignee	View document	Action
First loan contract	2/13/2024, 7:45:50 PM	12000	VERY HIGH 	 +riskmgr@ 	<div>ORIGINAL</div>	<div>APPROVE</div> <div>REJECT</div> <div></div>

1-1 of 1



34. The contract has now disappeared from the **RISK MANAGER TASKS** view. Open the **ALL CONTRACTS** view to confirm that the contract has been automatically approved (**Status** column shows **APPROVED** status).

35. Click on > (caret arrow icon) to view the contract details.

CREATED CONTRACTS

RISK MANAGER TASKS





ALL CONTRACTS

Contract Name

Status

APPLY FILTERS

All contracts

Contract name	Creation date	Status	Value	Risk classification	View document
First loan contract	1/20/2025, 4:37:50 PM	APPROVED	12000	VERY HIGH 	ORIGINAL 
First standard contract	1/20/2025, 3:46:52 PM	APPROVED	500	NONE 	ORIGINAL 

1-2 of 2

36. From the **PERMISSIONS** tab you can see that the ACL is changed so that everyone except the administrators has read permissions.

Contract details	
PROPERTIES	PERMISSIONS
APPROVALS	
Identity	Permissions
administrators	<ul style="list-style-type: none"> • browse • change_folder_links • change_location • change_owner • change_permission • create_link • delete • delete_link • lock • read_content • relate • version • write • write_content
contract_approval_users	<ul style="list-style-type: none"> • browse • read_content
owner	<ul style="list-style-type: none"> • browse • read_content
CLOSE	

37. The **APPROVALS** tab now shows that both the **Risk Manager Approval** and the **Automatic Approval** are granted. That is, all four approvals were required for this loan contract, and all four approvals are granted.

Click **CLOSE** to return to the **ALL CONTRACTS** view.

Contract details

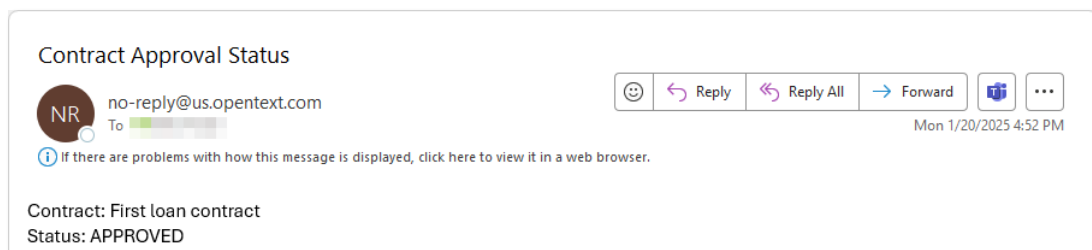
PROPERTIES PERMISSIONS APPROVALS

	Required	Granted	Approver	Approver role	Approval date
Automatic Approval	true	true	SYSTEM	Automatic Approval	2025-01-20
Line Manager Approval	true	true	+linemgr@	Line Manager	2025-01-20
Risk Manager Approval	true	true	+riskmgr@	Risk Manager	2025-01-20
Solvency Check	true	true	SYSTEM	Solvency Check	2025-01-20

CLOSE

38. The first loan contract has now been approved.

Go to your email client and check the inbox to confirm receiving the corresponding **Contract Approval Status** email from **no-reply@<region>.opentext.com**.



Next step:

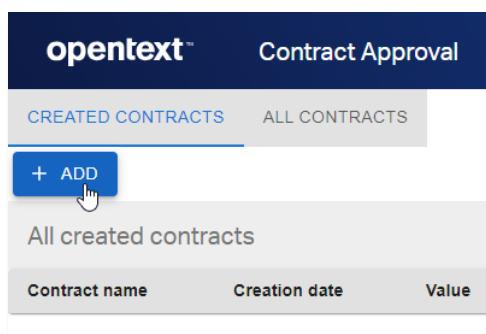
Reject a manual contract approval task.

15.4 Reject a manual contract approval task

You have now successfully approved two contracts with two completely different approval flows. In this section, you will test the scenario where an approver does not approve (that is, rejects) the contract.

You will create a contract with the following characteristics:

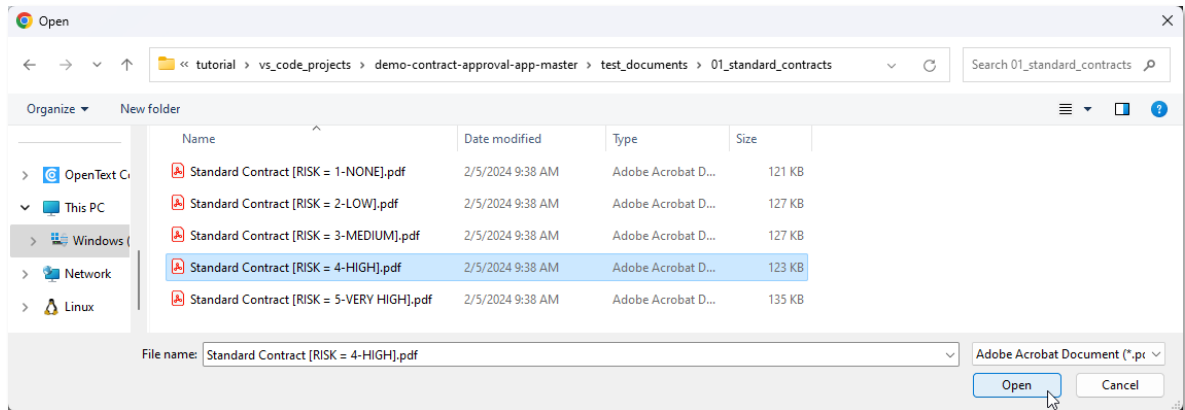
- **Type: standard contract**
 - **Value: above 1000** (requires Line Manager approval)
 - **Risk classification: above 3, that is, HIGH or VERIFY HIGH** (requires Risk Manager approval)
1. Log back in as the regular user, select the **CREATED CONTRACTS** tab and click the **+ ADD** button to open the contract creation form.



2. From the **Add Contract** screen, click **SELECT DOCUMENT** to add the contract content file.

 The screenshot shows the 'Add Contract' form. At the top, the title 'Add Contract' is displayed. Below the title is a blue button with a white cloud icon and the text 'SELECT DOCUMENT'. A mouse cursor is pointing at this button. Below the button, there are two radio buttons: 'Standard Contract' (which is selected) and 'Loan Contract'. Below the radio buttons are three input fields: 'Document name', 'Contract value', and 'Contract requester email'. At the bottom right of the form, there are two buttons: 'ADD' (disabled, grey) and 'CANCEL' (active, blue).

3. From the **test_documents** folder, open the **01_standard_contracts** subfolder and select the **Standard Contract [RISK = 4-HIGH].pdf** file.



4. Make sure the **Standard Contract** option is selected and fill the contract properties as follows:

Property	Value
Document name	Second standard contract
Contract value	5000
Contract requester email	<your email>

5. Click **Add** to create the contract.

Add Contract

SELECT DOCUMENT
Standard Contract [RISK = 4-HIGH].pdf

☒ Standard Contract
☐ Loan Contract

Document name
Second standard contract

Contract value
5000

Contract requester email

ADD CANCEL



6. Click the **REQUEST APPROVAL** button in the **Action** column to launch the approval workflow.

CREATED CONTRACTS

ALL CONTRACTS

+ ADD

All created contracts

Contract name	Creation date	Value	Risk classification	View document	Action
Second standard contract	2/14/2024, 9:15:05 AM	5000	HIGH 	<div>ORIGINAL</div>	<div>REQUEST APPROVAL</div> <div></div>

1-1 of 1

<

>

7. The new contract has now disappeared from the **CREATED CONTRACTS** view, as it is no longer in 'CREATED' status.
8. Sign in as the line manager and go to the **LINE MANAGER TASKS** tab. You can see the new Line Manager approval task.
9. Click **CLAIM** to assign the approval to your user.

CREATED CONTRACTS

LINE MANAGER TASKS

ALL CONTRACTS

All Tasks

Contract name	Creation date	Value	Risk classification	Assignee	View document	Action
Second standard contract	2/14/2024, 9:29:39 AM	5000	HIGH 		<div>ORIGINAL</div>	<div>CLAIM</div> 

1-1 of 1



10. Click **REJECT** to reject the contract approval.

CREATED CONTRACTS

LINE MANAGER TASKS

ALL CONTRACTS

All Tasks

Contract name	Creation date	Value	Risk classification	Assignee	View document	Action
Second standard contract	2/14/2024, 9:29:39 AM	5000	HIGH	+linemgr@	ORIGINAL	APPROVE REJECT

1-1 of 1

11. The contract has now disappeared from the **LINE MANAGER TASKS** view. Open the **ALL CONTRACTS** view to see that the contract has indeed been rejected (**Status** column shows **REJECTED** status).

12. Click > (caret arrow icon) to view the contract details and check the different approvals (traits).

CREATED CONTRACTS

LINE MANAGER TASKS

ALL CONTRACTS

Contract Name

Status

APPLY FILTERS

All contracts

Contract name	Creation date	Status	Value	Risk classification	View document
Second standard contract	1/20/2025, 6:48:31 PM	REJECTED	5000	HIGH	<div>ORIGINAL</div> <div></div>
First loan contract	1/20/2025, 4:37:50 PM	APPROVED	12000	VERY HIGH	<div>ORIGINAL</div> <div></div>
First standard contract	1/20/2025, 3:46:52 PM	APPROVED	500	NONE	<div>ORIGINAL</div> <div></div>

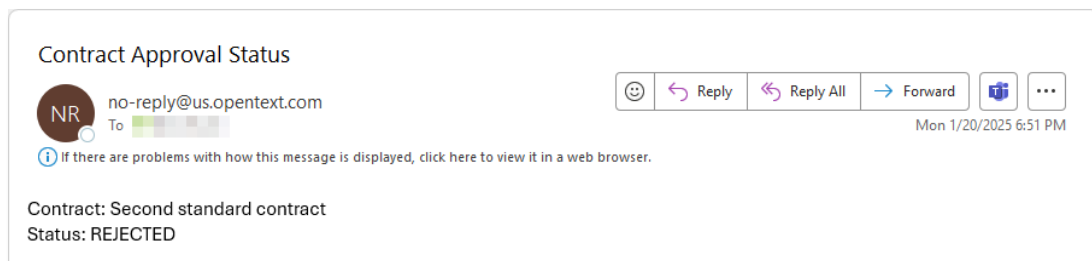
1-3 of 3

13. From the **Line Manager Approval** entry, you can see that the contract has been rejected because, although the **Approval date** has been set, **Granted** is false.

Contract details						
PROPERTIES PERMISSIONS APPROVALS						
	Required	Granted	Approver	Approver role	Approval date	
Automatic Approval	true	false				
Line Manager Approval	true	false	+linemgr@	Line Manager	2025-01-20	
Risk Manager Approval	true	false				
						CLOSE

14. The second standard contract has not been approved (that is, it has been rejected).

Go to your email inbox and confirm receiving the corresponding **Contract Approval Status** email from **no-reply@<region>.opentext.com**.



Next step:

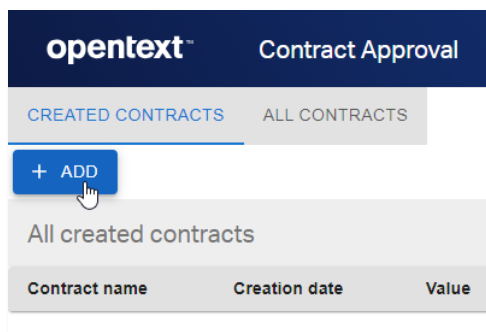
Expire a manual contract approval task.

15.5 Expire a manual contract approval task

1. The last scenario to run through is to let an approval task expire (happens after 5 minutes).

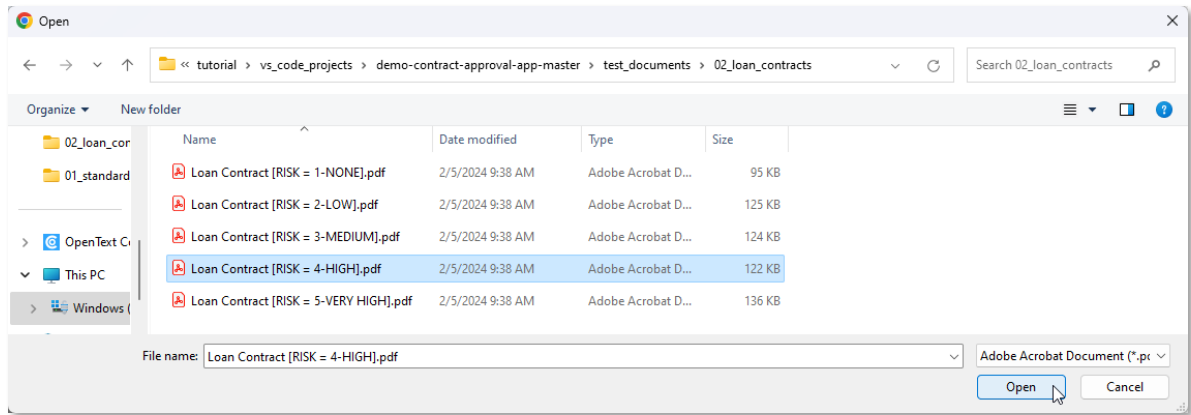
You will create a contract with the following characteristics:

- **Type: loan contract**
 - **Value: below 5000** (doesn't require Line Manager approval)
 - **Risk classification: above 3, that is, HIGH or VERIFY HIGH** (requires Risk Manager approval)
2. One more time, login with the regular user, select the **CREATED CONTRACTS** tab and click the **+ ADD** button to open the contract creation form.



3. From the **Add Contract** screen, click **SELECT DOCUMENT** to add the contract content file.

4. From the **test_documents** folder, open the **02_loan_contracts** subfolder and select the **Loan Contract [RISK = 4-HIGH].pdf** file.



5. Select the **Loan Contract** option and fill the contract properties as follows:

Property	Value
Document name	Second loan contract
Contract value	3500
Monthly installments	12
Yearly income	50000
Contract requester email	<your email>

6. Click **Add** to create the contract.

Add Contract

SELECT DOCUMENT
Loan Contract [RISK = 4-HIGH].pdf

☐ Standard Contract
☒ Loan Contract

Document name
Second loan contract

Contract value
3500

Monthly installments
12

Yearly income
50000

Contract requester email

ADD
CANCEL



7. Click the **REQUEST APPROVAL** button in the **Action** column to launch the approval workflow.

CREATED CONTRACTS



ALL CONTRACTS

+ ADD

All created contracts

Contract name	Creation date	Value	Risk classification	View document	Action
Second loan contract	2/14/2024, 9:53:49 AM	3500	HIGH 	<div>ORIGINAL</div>	<div>REQUEST APPROVAL</div> <div></div>

1-1 of 1


8. The new contract has now disappeared from the **CREATED CONTRACTS** view, as it is no longer in 'CREATED' status.
9. Sign in as the risk manager and go to the **RISK MANAGER TASKS** tab. You can see the new Risk Manager approval task.

CREATED CONTRACTS

RISK MANAGER TASKS

ALL CONTRACTS

All Tasks

Contract name	Creation date	Value	Risk classification	Assignee	View document	Action
Second loan contract	2/14/2024, 10:09:16 AM	3500	HIGH 		<div>ORIGINAL</div>	<div>CLAIM</div> <div>></div>

1-1 of 1

<>

10. To test whether or not the approval task will expire, wait for more than 5 minutes and refresh the application page from the web browser.
11. After refreshing the application screen, go back to the **RISK MANAGER TASKS** view.

The contract has now disappeared from the **RISK MANAGER TASKS** view.

CREATED CONTRACTS

RISK MANAGER TASKS

ALL CONTRACTS

All Tasks

Contract name	Creation date	Value	Risk classification	Assignee	View document	Action
						0-0 of 0 < >

12. Open the **ALL CONTRACTS** view to see that the contract approval has indeed expired (**Status** column shows **EXPIRED** status).

13. Click > (caret arrow icon) to view the contract details and check the different approvals (traits).

CREATED CONTRACTS

RISK MANAGER TASKS









ALL CONTRACTS

Contract Name



Status

APPLY FILTERS

All contracts

Contract name	Creation date	Status	Value	Risk classification	View document	
Second loan contract	1/20/2025, 6:58:07 PM	EXPIRED	3500	HIGH 	ORIGINAL	
Second standard contract	1/20/2025, 6:48:31 PM	REJECTED	5000	HIGH 	ORIGINAL	
First loan contract	1/20/2025, 4:37:50 PM	APPROVED	12000	VERY HIGH 	ORIGINAL	
First standard contract	1/20/2025, 3:46:52 PM	APPROVED	500	NONE 	ORIGINAL	

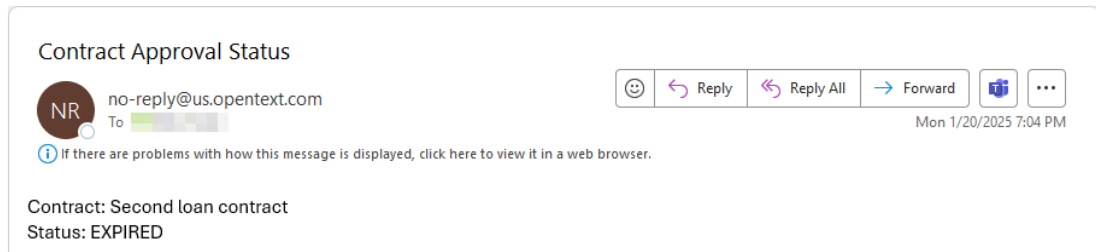
1-4 of 4

14. You can see that the contract is not approved by the Risk Manager since the **Risk Manager Approval** has NOT been granted. However, you can also see that there has not been a rejection action as the approval date is not filled. The approval activity simply timed out.

Contract details						
PROPERTIES PERMISSIONS APPROVALS						
	Required	Granted	Approver	Approver role	Approval date	
Automatic Approval	true	false				
Line Manager Approval	false	false				
Risk Manager Approval	true	false				
Solvency Check	true	true	SYSTEM	Solvency Check	2025-01-20	
						CLOSE

15. The second loan contract has not been approved since the Risk Manager Approval step expired. Go to your email inbox and confirm receiving the corresponding **Contract Approval Status** email from **no-reply@<region>.opentext.com**.



CONGRATULATIONS!

You have now finished building and testing the Contract Approval application. You are at the end of the main part of the tutorial.

There is one more bonus exercise where you will learn about the ocp cli. If you are interested in build automation and CI/CD for your applications, it is recommended that you do that exercise module as well.

Next exercise module:

Bonus exercise: Use the ocp command line interface.

16 [15'] Bonus exercise: Use the ocp command line interface

Learn how to:

- Install the stand-alone ocp command line interface (cli)
- Use organization profiles with the ocp cli
- Deploy projects with the ocp cli

In this exercise module you will use the OpenText Cloud Platform command line interface, or in short **ocp cli**. This cli allows you to manage organization profiles, deploy projects, and to create local packages.



Note

One of the main purposes of this cli is to be used in a CI/CD scenario. It allows to automate the deployment of the application models needed to run automated tests. Please refer to the [OpenText Thrust Studio User Guide](#) for more details on CI/CD and how to use the ocp cli in that context.

16.1 Install the ocp cli

The ocp cli is available from npm, which is a public software registry. Installing the ocp cli is done with the npm command, which is already available as part of the Node.js installation.

1. Open a command prompt or terminal and execute the command:

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

This installs the ocp cli on your system and makes it available globally.

```
C:\>npm install -g @opentext/ocp
added 61 packages in 1s
C:\>
```

2. In the command prompt or terminal execute the following command to see the information on how to use the ocp cli:

```
ocp --help
```

This shows the command groups and actions that can be used with the cli.

```
C:\>ocp --help
Usage of ocp CLI (version '24.3.2'):
  ocp <command-group> <command> <sub-command> [options] [arguments]
Available command groups for the OCP CLI:
Command Group: alm:
  Commands:
    app credentials          - generate new client credentials for the application
    deployment monitor      - monitor the state of an ongoing deployment
Command Group: local:
  Commands:
    profile list (or ls)    - displays available profiles
                           --name <profile-name> --region <region> --org <org-id> --org-name <org-name> [--pu
    profile add             --name <profile-name> [--region <region>] [--org <org-id>] [--org-name <org-name>]
    profile update          --name <profile-name>
    profile delete          --name <profile-name>
    profile authenticate    - tests the authentication of the profile credentials
    profile property global-update - add, delete, update and list global profile properties
    profile tenant list (or ls) - displays tenants available in local profile
    profile tenant add      --tenant-id <tenant_id> --tenant-name <tenant_name> [--default]
    profile tenant update   --tenant-id <tenant_id> [--tenant-name <tenant_name>] [--default]
    profile tenant delete   --tenant-id <tenant_id>
    profile tenant default  --tenant-id <tenant_id>
    project deploy          - upload and deploy the current contents of the project
    package create          - create a package from a project
    package upload          - upload a package file
    package deploy          - create, upload and deploy a package
To access top level help (this page):
  ocp --help
To access help for a command group:
  ocp <command-group> --help
To access help for a command within a command group:
  ocp <command-group> <command> --help
C:\>
```

3. To see the detailed information on how to use a specific command use:

```
ocp <command group> --help
```

Next step:

Use the developer profile.

16.2 Use the developer profile

To access the OpenText Thrust Services APIs you already set up a developer profile in VS Code (see [Add an organization profile](#)). This profile's data is shared between VS Code and the ocp cli. This means that the cli can be used to see and manage the already added profile.

1. In the command prompt or terminal execute the following command to display the already configured profile:

```
ocp local profile list
```

This shows the available profile as added earlier:

```
C:\>ocp local profile list
Profiles available in configuration file: C:\Users\...\ot2\profiles.json
Name      Project      Org Name      Org ID      Default
Dev        My Developer  My Developer  Org         YES
C:\>
```

If you have set up your organization profile and tenant according to the tutorial and have not added any other organization, there is only one organization profile being shown and it is also the Default organization profile. Otherwise, multiple organization profiles will be listed with one being the Default organization profile.

2. Use the **authenticate** action to authenticate using the default profile. In the command prompt or terminal execute the command:

```
ocp local profile authenticate
```

This starts the authentication flow and after successful authentication a message is displayed that an authentication code was received:

```
C:\>ocp local profile authenticate
Using public credentials for authentication.
Server is running on http://localhost:22682
Authentication flow continues via the browser. We are awaiting a response callback to url: "http://localhost:22682/login/oauth2/code".
closed authentication server
Authentication code received. Performing token exchange.
Updating existing profile Dev.
Updated OCP profile C:\Users\...\ot2\profiles.json.
Authentication successful: {"profile_name":"Dev","default":true,"config_file":"C:\\Users\\...\\ot2\\profiles.json","updated_auth":tr
C:\>
```

Next step:

Deploy the application project from command line.

16.3 Deploy the application project from command line

One of the main purposes of the ocp cli is to deploy application projects. When deploying an application project, first a temporary package is created, after which it gets uploaded to the ALM deployment service. The package is validated and its models are deployed. When an application project is deployed for the first time the API key data is returned and displayed.

Deploy an application project with the ocp cli using the **deploy** action.

1. In the command prompt or terminal, change the directory to be the folder that contains the **.otproject** file, as this is the main project folder of the application project.

```
C:\tutorial\vs_code_projects\contract_approval>dir
Volume in drive C is Windows
Volume Serial Number is 1111-1111

Directory of C:\tutorial\vs_code_projects\contract_approval

21/01/2025 12:35 <DIR>      .
21/01/2025 12:35 <DIR>      ..
17/01/2025 15:48          290 .env
16/01/2025 20:13          399 .eslintrc.yml
16/01/2025 20:13          221 .gitignore
16/01/2025 20:13           15 .npmrc
16/01/2025 20:13          378 .otproject
16/01/2025 20:13          598 index.html
16/01/2025 20:13       11.358 LICENSE
21/01/2025 12:35 <DIR>      node_modules
21/01/2025 12:35 <DIR>      oresources
20/01/2025 14:40       213.091 package-lock.json
20/01/2025 14:40        1.200 package.json
21/01/2025 12:35 <DIR>      postman
21/01/2025 12:35 <DIR>      public
16/01/2025 20:13        3.275 README.md
21/01/2025 12:35 <DIR>      src
21/01/2025 12:35 <DIR>      test_documents
16/01/2025 20:13          705 vite.config.js
                        15 File(s)      234.044 bytes
                        8 Dir(s)    188.569.329.664 bytes free

C:\tutorial\vs_code_projects\contract_approval>
```

2. In the command prompt or terminal execute the command to deploy the application project:

```
ocp local project deploy
```

```
C:\tutorial\vs_code_projects\contract_approval>ocp local project deploy
The project directory is [C:\tutorial\vs_code_projects\contract_approval]
Using public credentials for authentication.
Using existing access token from profile.

Status: success
Name: oresources/contract_approval_users.otgrp
ID: 30ec7468-2ffe-48fc-b44c-96637121521b. Model ID: 6d5b69fc-6f20-465c-bb59-31402a3eaa0b
Status: success
Name: oresources/manager_approval.otwf
ID: 9ec7e09d-49ae-4988-bdb5-cd8333a4c8fc. Model ID: 69cb85de-cbaf-4d5c-af5e-a31e75ea3203
Status: success
Name: oresources/risk_managers.otgrp
ID: 90780011-001a-46a3-ac0b-114c9a944cae. Model ID: b60cbccf-8d01-4c7f-b11b-730a4077ceb2
Status: success
Name: oresources/contract.ottype
ID: 875d9217-06d2-4df8-816a-331df07afc32. Model ID: cf1359bf-c5aa-462d-97e6-6bc98352742b
Status: success
Name: oresources/pending_approval.otacl
ID: 56487a01-d3d8-48ce-842f-e69cd58ff7fd. Model ID: 902bf59d-9a00-47ff-b4c7-739ddc7aff4
Status: success
Name: oresources/line_managers.otgrp
ID: e1410134-67b3-4e75-9be3-479e66ab4f20. Model ID: 09f74bb9-0542-4b45-93d2-4243516158ec
Status: success

-----
In tenants [""] application "contract_approval" was updated.
C:\tutorial\vs_code_projects\contract_approval>
```

- Normally the temporary application package is deleted after it is uploaded to the ALM deployment service. By using the package create action the package can be saved in a local file. This allows the file to be stored for deploying at a later moment or to a different environment.

In the command prompt or terminal execute the following command to save the package to a local file:

```
ocp local package create
```

```
C:\tutorial\vs_code_projects\contract_approval>ocp local package create
The project directory is [C:\tutorial\vs_code_projects\contract_approval]
Adding https://www.opentext.com/ocp/devx/security/1.0.0/Group artifact: C:\tutorial\vs_code_projects\contract_approval\otresources\administrators.otgrp
Adding https://www.opentext.com/ocp/devx/metadata/1.0.0/Trait artifact: C:\tutorial\vs_code_projects\contract_approval\otresources\approval.ottrait
Adding https://www.opentext.com/ocp/devx/security/1.0.0/ACL artifact: C:\tutorial\vs_code_projects\contract_approval\otresources\completed.otacl
Adding https://www.opentext.com/ocp/devx/metadata/1.0.2/Type artifact: C:\tutorial\vs_code_projects\contract_approval\otresources\contract.ottype
Adding https://www.opentext.com/ocp/devx/common/1.0.0/Namespce artifact: C:\tutorial\vs_code_projects\contract_approval\otresources\contract_approval.otns
Adding https://www.opentext.com/ocp/devx/workflow/1.0.0/Flow artifact: C:\tutorial\vs_code_projects\contract_approval\otresources\contract_approval.otwf
Adding https://www.opentext.com/ocp/devx/security/1.0.0/Group artifact: C:\tutorial\vs_code_projects\contract_approval\otresources\contract_approval_users.otgrp
Adding https://www.opentext.com/ocp/devx/security/1.0.0/ACL artifact: C:\tutorial\vs_code_projects\contract_approval\otresources\created.otacl
Adding https://www.opentext.com/ocp/devx/metadata/1.0.2/Type artifact: C:\tutorial\vs_code_projects\contract_approval\otresources\customer.ottype
Adding https://www.opentext.com/ocp/devx/security/1.0.0/Group artifact: C:\tutorial\vs_code_projects\contract_approval\otresources\line_managers.otgrp
Adding https://www.opentext.com/ocp/devx/metadata/1.0.2/Type artifact: C:\tutorial\vs_code_projects\contract_approval\otresources\loan_contract.ottype
Adding https://www.opentext.com/ocp/devx/workflow/1.0.0/Flow artifact: C:\tutorial\vs_code_projects\contract_approval\otresources\manager_approval.otwf
Adding https://www.opentext.com/ocp/devx/security/1.0.0/ACL artifact: C:\tutorial\vs_code_projects\contract_approval\otresources\pending_approval.otacl
Adding https://www.opentext.com/ocp/devx/decision/1.0.0/DecisionTable artifact: C:\tutorial\vs_code_projects\contract_approval\otresources\required_approvals.otdt
Adding https://www.opentext.com/ocp/devx/security/1.0.0/Group artifact: C:\tutorial\vs_code_projects\contract_approval\otresources\risk_managers.otgrp
Adding https://www.opentext.com/ocp/devx/workflow/1.0.0/Flow artifact: C:\tutorial\vs_code_projects\contract_approval\otresources\solvency_check.otwf
Outputting deployable package archive to local file: "contract_approval_1.0.otpp".
Progression: 0.00%.
Current file = "manifest".
Progression: 5.56%.
Current file = "otresources/".
Progression: 11.11%.
Current file = "otresources/administrators.otgrp".
Progression: 16.67%.
Current file = "otresources/approval.ottrait".
Progression: 22.22%.
Current file = "otresources/manager_approval.otwf".
Progression: 27.78%.
Current file = "otresources/pending_approval.otacl".
Progression: 33.33%.
Current file = "otresources/required_approvals.otdt".
Progression: 38.89%.
Current file = "otresources/risk_managers.otgrp".
Progression: 44.44%.
Current file = "otresources/solvency_check.otwf".
Progression: 50.00%.
Completed outputting deployable package archive to local file: "contract_approval_1.0.otpp".
C:\tutorial\vs_code_projects\contract_approval>
```

This saves the application package to the file called **contract_approval_1.0.otpp** in the current folder.

- The created package can be deployed in a different tenant in the organization, or in another tenant in another organization. The package doesn't contain any tenant or organization specific data.

In the command prompt or terminal execute the following command to deploy the saved package:

```
ocp local package deploy --source contract_approval_1.0.otpp
```

This deploys the application package in the same way as a deploy of the application project for which the package was created.

```
C:\tutorial\vs_code_projects\contract_approval>ocp local package deploy --source contract_approval_1.0.otpp
Using public credentials for authentication.
Using existing access token from profile.
Uploading existing deployable package archive contract_approval_1.0.otpp to ALM.
Retrieved service info from ALM.
Queried ALM GW for version information: {"supportedApiVersions":[{"version":"v1","stopWorkingDate":"2022-08-31T00:00:00Z"}],{
Auto Selection ALM API Version V2
Progression: 0.00%.
Current file = "manifest".
Progression: 5.56%.
Current file = "otresources/".
Progression: 11.11%.
Current file = "otresources/administrators.otgrp".
Progression: 16.67%.
Current file = "otresources/approval.ottrait".
Progression: 22.22%.
```

```

      Status: success
Name: oresources/contract_approval.otwf
ID: 7e2d3305-73c3-48c8-bdf2-ac04cb580b3d. Model ID: 68f09739-aace-4c37-a07c-fbeee217c7b2
      Status: success
Name: oresources/administrators.otgrp
ID: faf7a913-bd53-4e3e-9118-7a11f09db6f8. Model ID: 9b7a150c-072d-4140-b02c-58850dce2c42
      Status: success
Name: oresources/solvency_check.otwf
ID: d15a06e9-ec2e-4c5a-b31d-4de1ab6a3b41. Model ID: 51bd0f86-f261-48a1-a19a-3b28146c4c0d
      Status: success
Name: oresources/approval.ottrait
ID: 27b0a058-1872-4b6d-8d65-d46b9de285a7. Model ID: 1b42a482-41de-4c26-a7a0-2ceff39e0f29
      Status: success
Name: oresources/line_managers.otgrp
ID: e1410134-67b3-4e75-9be3-479e66ab4f20. Model ID: 09f74bb9-0542-4b45-93d2-4243516158ec
      Status: success
=====
In tenants [" "] application "contract_approval" was updated.
C:\tutorial\vs_code_projects\contract_approval>

```

You have now completed this bonus exercise about using the stand-alone ocp command line interface.

About OpenText

OpenText enables the digital world, creating a better way for organizations to work with information, on-premises or in the cloud. For more information about OpenText (NASDAQ/TSX: OTEX), visit opentext.com.

Connect with us:

[OpenText CEO Mark Barrenechea's blog](#)

[Twitter](#) | [LinkedIn](#)