

OpenText IMaaS Developer Tutorial

Building a Contract Approval application

This tutorial has been created for software version OpenText™ IMaaS Tools for VS Code 22.1.4.

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

Open Text Corporation

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

Tel: +1-519-888-7111

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

Fax: +1-519-888-0677

Support: <https://support.opentext.com>

For more information, visit <http://www.opentext.com>

Copyright © 2022 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.

Last updated: 03/29/2022

Contents

1	Introduction	4
2	Prerequisites	5
2.1	[20'] Setting up an OpenText Developer Trial Account	5
3	Building the Contract Approval application	16
3.1	[25'] Setting up the IMaaS Developer IDE	16
3.2	[10'] Adding an organization and testing the connection	36
3.3	[15'] Creating an OpenText project	41
3.4	[10'] Creating a CMS namespace	49
3.5	[15'] Creating a CMS trait definition	52
3.6	[20'] Creating a CMS file type definition	57
3.7	[10'] Creating a CMS file type definition that is a subtype	64
3.8	[05'] Creating a CMS folder type definition	67
3.9	[120'] Creating a workflow model	69
3.10	[10'] Deploying the application to the IMaaS services	126
	<i>Resetting the Tenant password</i>	<i>131</i>
3.11	[25'] Working with the IMaaS APIs	134
3.12	[15'] Building the application backend	150
3.13	[15'] Building the application frontend	156
3.14	[50'] Testing your application	162
3.15	[00'] Bonus exercise: Using the OT2 Command Line Interface	190
	About OpenText	191

1 Introduction

This document is intended as a getting started guide for pro-code developers who want to learn how to build applications that utilize the OpenText IMaaS (Information Management as a Service) APIs, and how to do this in the most efficient way. More specifically, it shows how to develop a React based application in VS Code (Microsoft Visual Studio Code). The OpenText IMaaS developer functionality is added to the standard VS Code IDE (Integrated Development Environment) when you install the OpenText IMaaS Tools VS Code extension pack.

The application you will be building is a simple Contract Approval application which will allow to upload documents, store document related metadata for two types of contracts, do document analysis to detect PII (Personally Identifiable Information), and use a workflow to automate the different contract approval steps.

In this context, the following IMaaS services will be used:

- The Content Storage Service (CSS) for uploading and storing of documents
- The Content Metadata Service (CMS) for storing document metadata
- The OpenText Magellan Risk Guard Service for document analysis
- The Workflow Service for executing the contract approval process

In different step-by-step exercises throughout the document, we will cover the creation of models (IMaaS configuration artifacts) and how to deploy them, how to write code that talks to the IMaaS APIs, and how to run and test the finished Contract Approval application. Each exercise builds on the previous one, so you need to perform the steps exactly as written in the order in which they are presented, and without skipping any. It should take about 6 to 8 hours to complete the tutorial. You can complete all the exercises in a single session or break them up into multiple sessions.

This is what you will learn in the different exercises, and how much time it requires:

- [\[25'\] Setting up the IMaaS Developer IDE](#)
- [\[10'\] Adding an organization and testing the connection](#)
- [\[15'\] Creating an OpenText project](#)
- [\[10'\] Creating a CMS namespace](#)
- [\[15'\] Creating a CMS trait definition](#)
- [\[20'\] Creating a CMS file type definition](#)
- [\[10'\] Creating a CMS file type definition that is a subtype](#)
- [\[05'\] Creating a CMS folder type definition](#)
- [\[120'\] Creating a workflow model](#)
- [\[10'\] Deploying the application to the IMaaS services](#)
- [\[25'\] Working with the IMaaS APIs](#)
- [\[15'\] Building the application backend](#)
- [\[15'\] Building the application frontend](#)
- [\[50'\] Testing your application](#)
- [\[00'\] Bonus exercise: Using the OT2 Command Line Interface](#)

This tutorial is a complete “from scratch” application building guide. However, if you don't want to build the application but still want to familiarize yourself with the IMaaS API consuming example code, the different IMaaS models, and the Contract Approval sample application, you can directly open the finished application in VS Code without the need to completely go through all the exercises. The project sources for the completed Contract Approval application are available to be downloaded to your computer. To directly run the completed project from VS Code, you can skip ahead to the [Testing your application](#) section.

2 Prerequisites

As this is a complete end-to-end application building guide, we will be going through the downloading and installation of the required software and source code as part of the different tutorial steps. The only prerequisite to be able to start with the first exercise is that you have signed up for the OpenText Developer trial account, and that this trial has not yet expired. If you do not yet have your own trial account set up, please follow the instructions in this chapter.

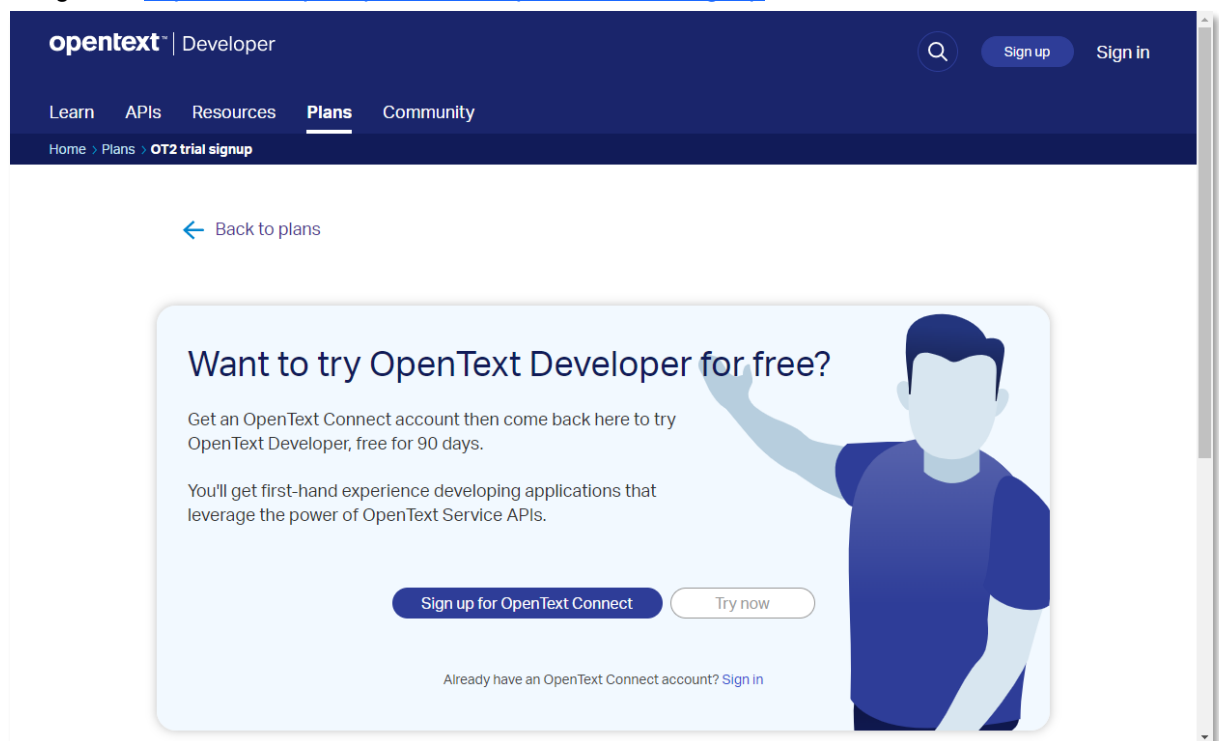
Also note that although you could theoretically use another OpenText Developer subscription than the aforementioned trial account, we recommend you still set up and use a trial account, as this tutorial has been explicitly constructed in that context. I.e.: differences between the trial account and another subscription might result in you getting stuck, or at least needing to deviate from the exact steps laid out in the tutorial.

2.1 [20'] Setting up an OpenText Developer Trial Account

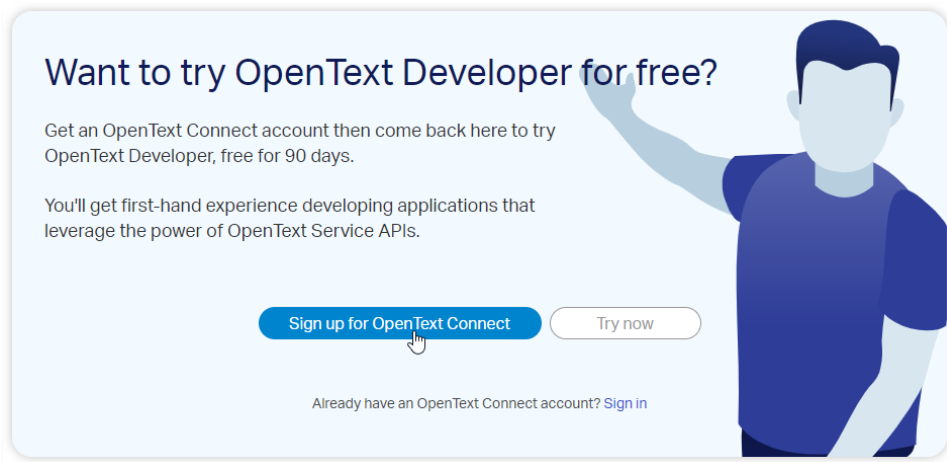
This tutorial uses the beta release of the OpenText IMaaS Tools and as previously stated, this requires using a trial account to avoid running into issues related to differences between subscriptions.

To set up an OpenText Developer free trial Account, proceed as follows:

- Navigate to <https://developer.opentext.com/plans/ot2-trial-signup>.



- If you haven't already signed up for OpenText Connect, click the **Sign up for OpenText Connect** button, otherwise you can skip to the [next step](#).



Fill your email address, choose a password, and confirm you are “not a robot”. Click **Create account** to continue with the OpenText Connect account creation.

Register for an Account

Welcome to OpenText Connect, the entry point for access to My Support, Communities and exclusive thought-leadership content.

Already have an account? [Sign in](#)

Email

Password

Confirm password

I'm not a robot

reCAPTCHA
[Privacy](#) - [Terms](#)

[Need help?](#)

For additional information on how your personal data is processed view our [privacy notice](#).

Create account

Fill your personal information, do not request full access to My Support and agree to the processing of your personal data for account creation purposes. Click **Create your account** to continue.

Registration – Step 2 of 2

Please provide your name and contact information for your opentext.com account.

First name

Last name

Position

Department

Company

Country

Phone

Your industry


Software maintenance customers

If your company has an active Customer Support contract, you are eligible for access to **My Support**. If your company has an active Customer Support maintenance contract, you can request access to **My Support** where you will find software, documentation, a complete Knowledge Base, and many other services included in your contract. For ticket creation, your company administrator manages the contact roles; if you would like to have this option, please contact your company administrator.

<p>Not sure if your company has a customer support contract?</p> <p>If you are not sure your company has an OpenText Customer Support contract, you can proceed with this request access form, and a Customer Service Representative will contact you in the event you do not have an active Customer Support contract.</p>	<p>Don't have a support contract?</p> <p>If your company does not have an OpenText Customer Support contract and you are interested in discussing your options please contact Sales:</p> <p>North America: +1-800-499-6544</p> <p>International: +800-4996-5440</p> <p>Email: Please use our contact form</p>
--	---

☐ Request full access to My Support.

☒ I agree to let OpenText use my personal data for creating an account, allow me access to OpenText web offerings and for OpenText to process my data in connection to these services. For additional information, refer to our [privacy notice](#).

Create your account 

Thank you for registering with OpenText

A link to complete registration has been sent to the email address you provided.

When you receive this email from OpenText Connect, please click on the link to confirm your registration.

When you complete your registration, you will be able to

Discover

- Access exclusive whitepapers, webinars and presentations.
- Register for OpenText events and seminars online or in your area.

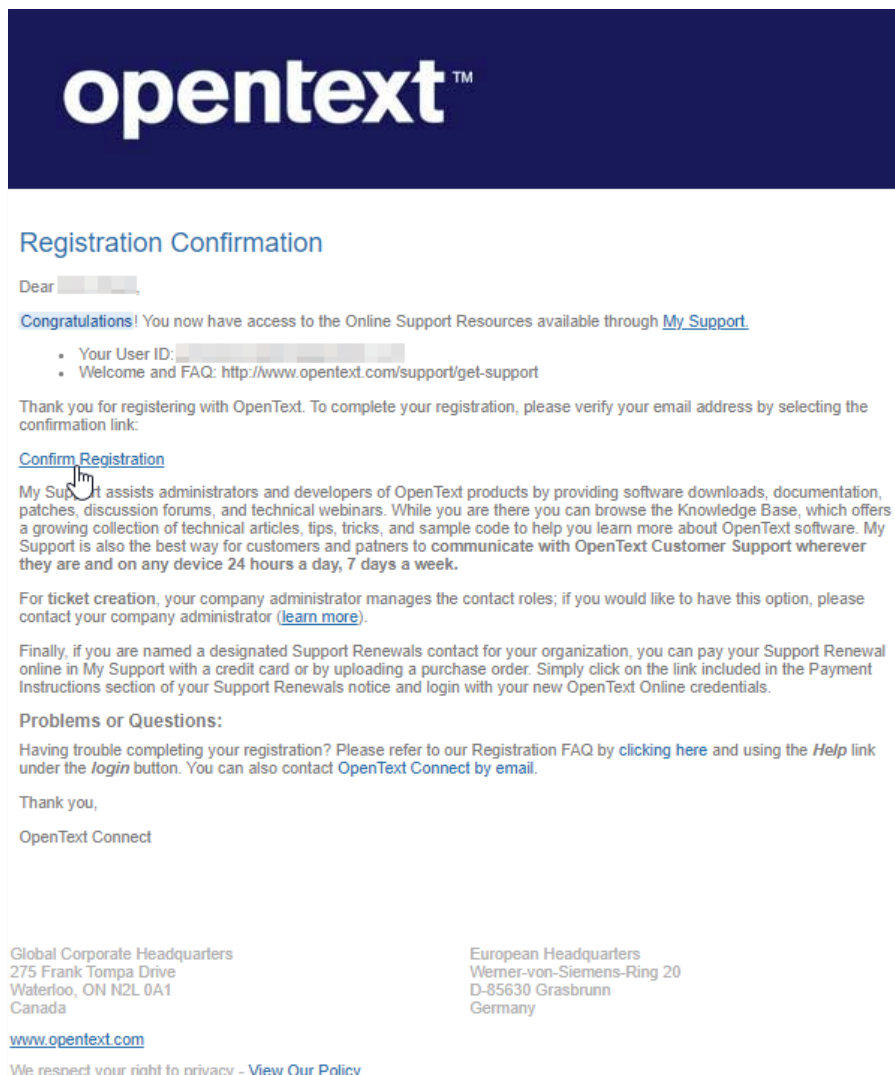
Collaborate

- Interact and share in OpenText blogs, wikis and forums.

Customize

- Manage your OpenText profile and preferences.
- Personalize your OpenText online experience.

Open your email client and click **Confirm Registration** from the registration confirmation email you have received (make sure it did not get blocked or moved to your spam folder).



opentext™

Registration Confirmation

Dear [Name],

Congratulations! You now have access to the Online Support Resources available through [My Support](#).

- Your User ID: [ID]
- Welcome and FAQ: <http://www.opentext.com/support/get-support>

Thank you for registering with OpenText. To complete your registration, please verify your email address by selecting the confirmation link:

[Confirm Registration](#)

My Support assists administrators and developers of OpenText products by providing software downloads, documentation, patches, discussion forums, and technical webinars. While you are there you can browse the Knowledge Base, which offers a growing collection of technical articles, tips, tricks, and sample code to help you learn more about OpenText software. My Support is also the best way for customers and partners to communicate with OpenText Customer Support wherever they are and on any device 24 hours a day, 7 days a week.

For ticket creation, your company administrator manages the contact roles; if you would like to have this option, please contact your company administrator ([learn more](#)).

Finally, if you are named a designated Support Renewals contact for your organization, you can pay your Support Renewal online in My Support with a credit card or by uploading a purchase order. Simply click on the link included in the Payment Instructions section of your Support Renewals notice and login with your new OpenText Online credentials.

Problems or Questions:

Having trouble completing your registration? Please refer to our Registration FAQ by [clicking here](#) and using the [Help](#) link under the [login](#) button. You can also contact [OpenText Connect](#) by email.

Thank you,

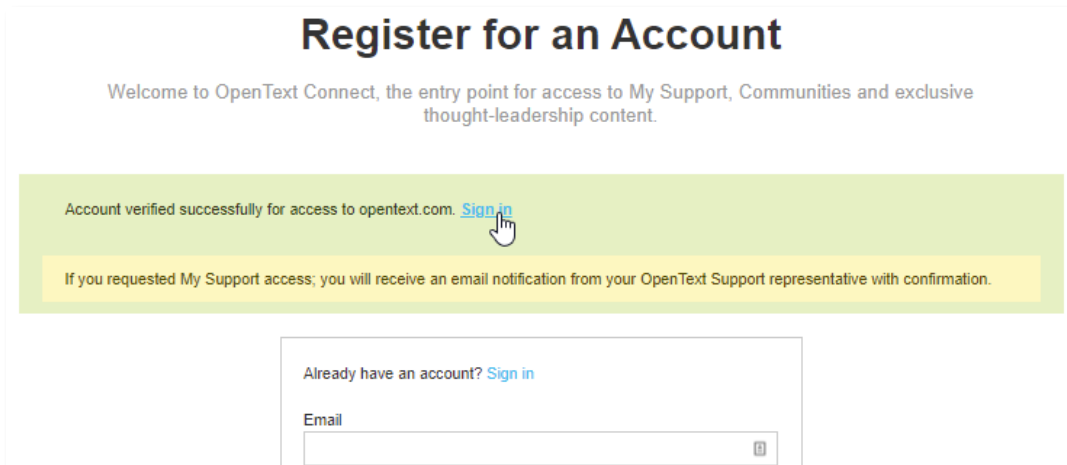
OpenText Connect

Global Corporate Headquarters
275 Frank Tompa Drive
Waterloo, ON N2L 0A1
Canada
www.opentext.com

European Headquarters
Werner-von-Siemens-Ring 20
D-85630 Grasbrunn
Germany

We respect your right to privacy - [View Our Policy](#)

The registration confirmation link should take you to a confirmation page from which you can click **Sign in** to sign in to your newly created OpenText Connect account.



Register for an Account

Welcome to OpenText Connect, the entry point for access to My Support, Communities and exclusive thought-leadership content.

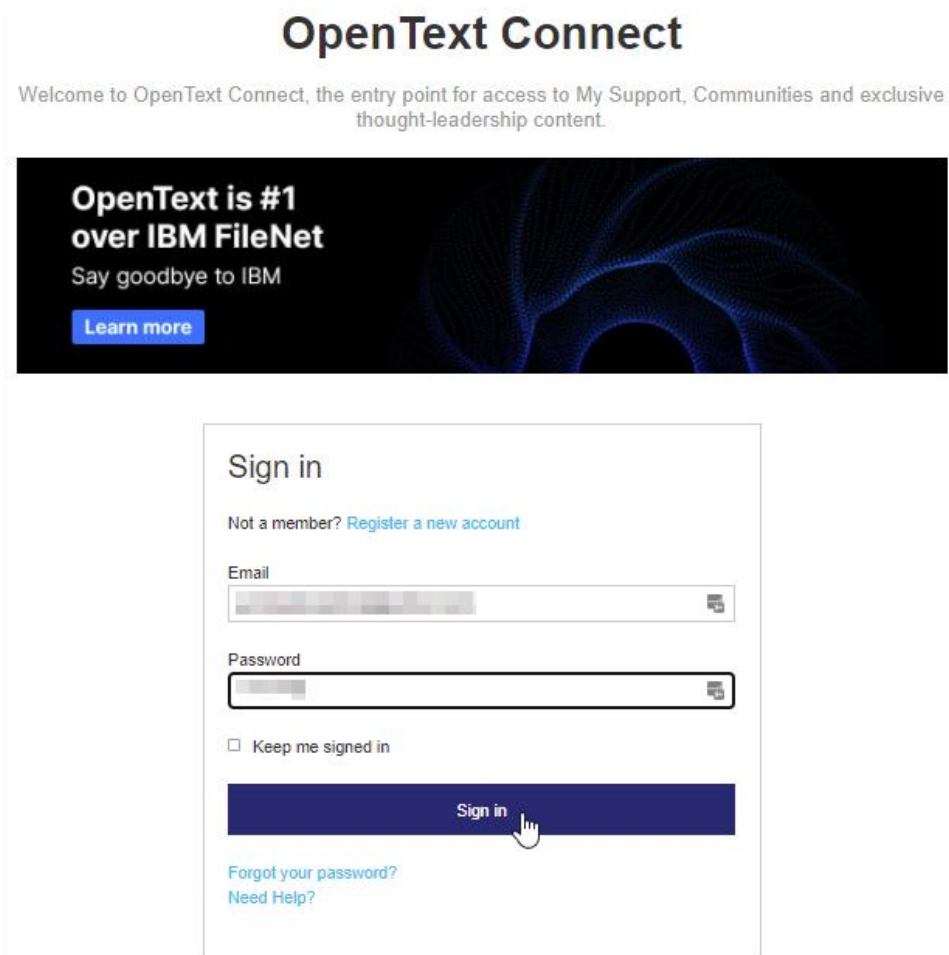
Account verified successfully for access to opentext.com. [Sign in](#)

If you requested My Support access; you will receive an email notification from your OpenText Support representative with confirmation.

Already have an account? [Sign in](#)

Email

Fill your email and (previously chosen) password and click **Sign in** to make sure signing in indeed works.



OpenText Connect

Welcome to OpenText Connect, the entry point for access to My Support, Communities and exclusive thought-leadership content.

OpenText is #1 over IBM FileNet
Say goodbye to IBM
[Learn more](#)

Sign in

Not a member? [Register a new account](#)

Email

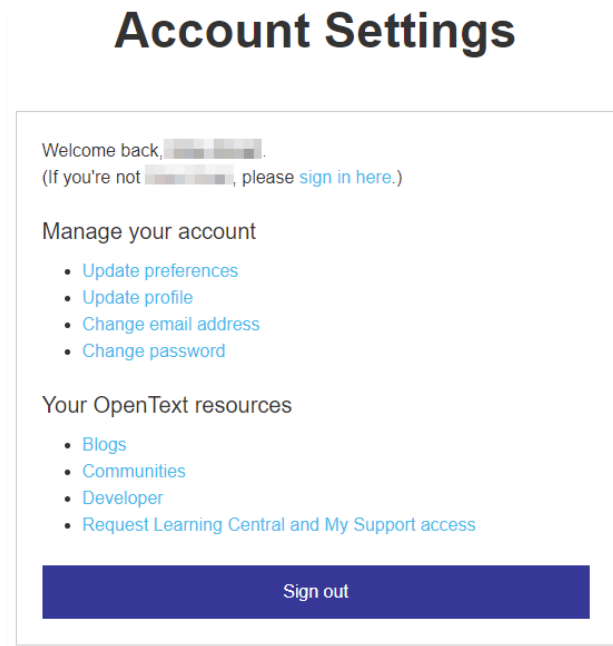
Password

☐ Keep me signed in

[Sign in](#)

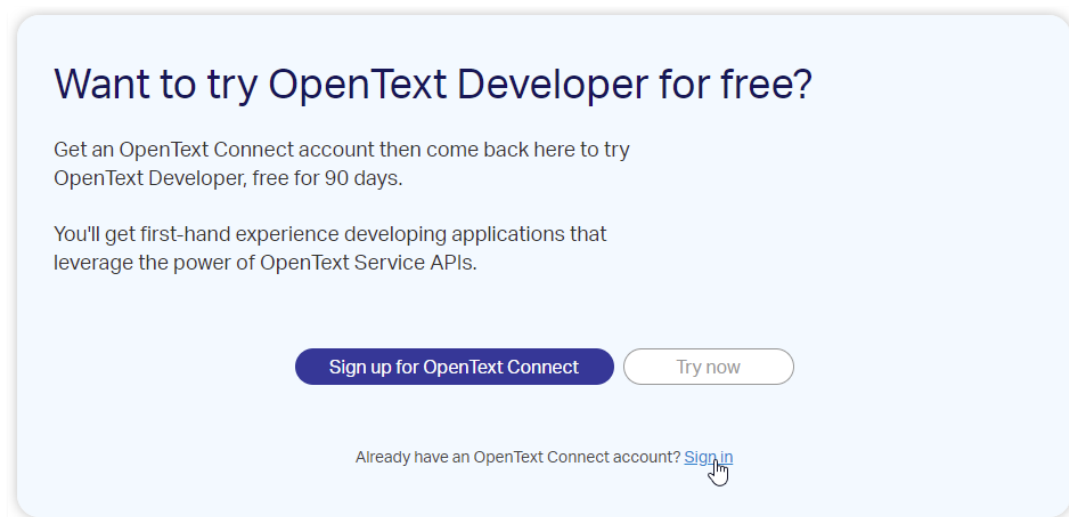
[Forgot your password?](#)
[Need Help?](#)

You are now signed in to your OpenText Connect account



Close (all open instances of) your web browser to disconnect from the session and navigate to <https://developer.opentext.com/plans/ot2-trial-signup> again so that you are ready to sign in with your newly created OpenText Connect account.

- To sign up for a trial account you need to sign in to your OpenText Connect account first. Click **Sign in**.



Fill your email and password and click **Sign in**.

Sign in

Not a member? [Register a new account](#)

Email

Password

☐ Keep me signed in

Sign in

[Forgot your password?](#)
[Need Help?](#)

REMARK:

If you already had an account (did not just create one) and did previously sign up for the Trial plan with that account, you will not get the screen from the next step. Instead, you will see the developer plans screen, with the **Sign Up** button of the **Free (Trial)** disabled (you cannot click it).

Developer Plans

Information Management as a Service (IMaaS) Developer Plans

Create your own hosted IMaaS application using OpenText services. All you need is Developer access to our platform and an active Developer plan.

[Click here](#) to learn more about the sign up process and what is included in these plans.

*Active Build & Test plan required to subscribe to add-ons

Free (Trial)	Build & Test	*Build & Test Add-ons	*Production Add-ons
<ul style="list-style-type: none"> Free for 90 days Sandbox Tenant 5 apps Unlimited developers 100,000 API calls 1 GB storage <p>Sign Up</p>	<ul style="list-style-type: none"> Annual term 3 Sandbox Tenants Unlimited apps Unlimited developers 500,000 API calls 100 GB storage <ul style="list-style-type: none"> \$5,400 USD Annually <p>Contact Us</p>	<p>Storage</p> <ul style="list-style-type: none"> Annual term 50 GB packs <ul style="list-style-type: none"> \$12 USD per Pack <p>API</p> <ul style="list-style-type: none"> Consumption model Packs of 1M API calls <ul style="list-style-type: none"> \$3,000 USD per 1M APIs <p>Contact Us</p>	<p>Storage</p> <ul style="list-style-type: none"> Annual term 50 GB packs <ul style="list-style-type: none"> \$12 USD per Pack <p>API</p> <ul style="list-style-type: none"> Consumption model Packs of 1M API calls <ul style="list-style-type: none"> \$3,000 USD per 1M APIs <p>Contact Us</p>

In this case, if your trial expired, you need to request an extension. If it did not expire (or if you got your extension), you are good to go and skip ahead to [Building the Contract Approval application](#). Beware though, that during the tutorial you will require the organization connection details from the **organization config file** that was provided during the Trial sign up process. If you don't have this anymore, you can get the required organization connection information through the developer console's [Organization information screen](#) (ID value and **Manage** button provide the organization id and organization client information).

On the OpenText Developer trial registration form, fill the following information:

- **Organization:** the name of your organization; this will be used in the developer environment as the actual name of your developer organization, so make sure to choose it correctly
- **Country:** your country
- **State/province:** depending on the selected country, you might have to fill your state or province as well
- **Purpose:** the purpose for which you are registering for the OpenText Developer free trial account
- You'll need to check both the **trial agreement** and **marketing communications and information regarding products, services and events** check boxes

Once the registration form is correctly filled, click **Try now** to start the 90-day trial.

Want to try OpenText Developer for free?

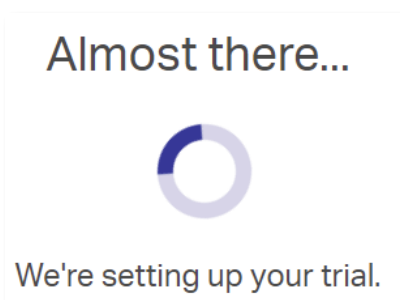
Our 90 day, **free** trial gives you first-hand experience developing applications that leverage the power of OpenText APIs.

33 / 150


☒ By checking this box, I confirm that I accept the [trial agreement](#) and authorize OpenText to contact me using my OT connect account email address and telephone number.

☒ By checking this box, I confirm that I would be interested in receiving marketing communications and information regarding products, services and events from Open Text. I understand that I may unsubscribe at any time. For additional details regarding how OpenText's shares, protects, retains, transfers and your rights, see the [OpenText Privacy Policy](#). Our [Cookie Policy](#) can be found here.


Try now




You are now presented with the confirmation that your 90-day free trial has started.

 Your 90 day free trial has started.

Your next steps:

- 1 Write down the password below - you'll need it to use the OT2 APIs. You can also reset it from the developer console
OT2 Service Password: 
- 2 Go create your first app.

[Download your organization config file](#) 


[Go to console](#)

IMPORTANT:


The 90-day trial confirmation screen contains important information, so make sure not to skip the next steps (under **Your next steps**) of copying the **OT2 Service Password** and downloading the **organization config file**.


You will need the organization configuration details later on in this tutorial (and won't be able to retrieve them beyond this point). The OT2 Service Password is not required in context of this tutorial, but if you ever need it when interacting with IMaaS APIs at the organization level, you will also not be able to retrieve it beyond this point (reset pwd is possible though).

As per the above important remark, make sure you click the **Download your organization config file** link.

 Your 90 day free trial has started.

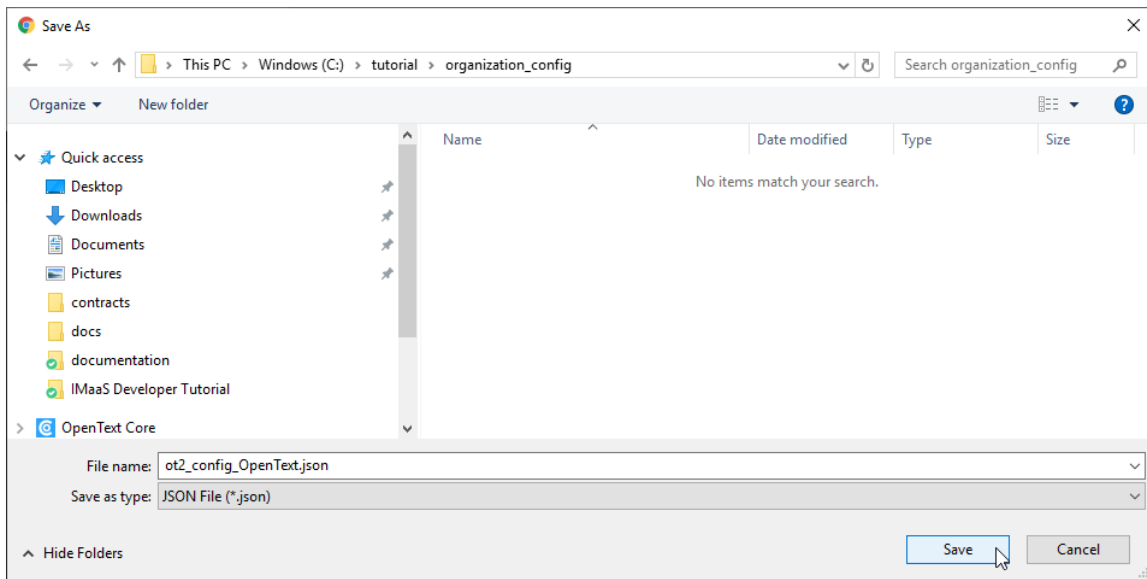
Your next steps:

- 1 Write down the password below - you'll need it to use the OT2 APIs. You can also reset it from the developer console
OT2 Service Password: 
- 2 Go create your first app.

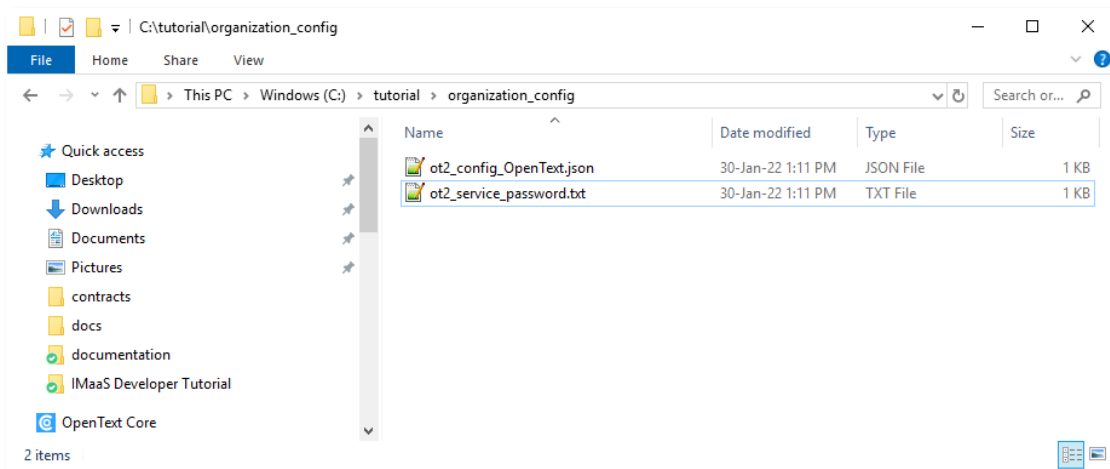
[Download your organization config file](#) 

[Go to console](#)

Create a folder (e.g.: **organization_config**) to store all organization related configuration for the IMaaS Developer tutorial and save the **ot2_config_<organization name, we are using OpenText>.json** organization configuration file in that newly created folder.




It is also very important you copy the **OT2 Service Password** from the trial confirmation screen and keep it for later use (even if we don't use it in this tutorial). We suggest you create an **ot2_service_password.txt** file next to the previously saved organization configuration file (inside of the new organization configuration folder) that contains the copied service password.



IMPORTANT:


Although we are using unencrypted/insecure text files to store the different key and password information for the purpose of this tutorial, it is of course recommended for real life scenarios to store any API key or password information in a secure way.

Once you have saved the OT2 Service Password and organization configuration file, click **Go to console** to go to the IMaaS Developer console.


 Your 90 day free trial has started.

Your next steps:

- 1 Write down the password below - you'll need it to use the OT2 APIs. You can also reset it from the developer console
OT2 Service Password:
- 2 Go create your first app.

[Download your organization config file](#) 

[Go to console](#)

 **opentext** | Developer


View apps by:

Organization

Tenants

Tenant 1


[Learn](#) [+ Create new app](#)


Create an app

My organization info

Name

ID

Organization service account 

Current region: North America Build and Test

Organization service client [Manage](#)

Tenants [Add](#)

My Plan [Upgrade](#)

Developer-Trial - Trial day 1 of 90

Calls made this month 0

1

0

-1

1

7

14

21

30

Storage Use

100%
Available

1 GB
Space with your plan

1 GB available

Service health

Last updated 1/30/22, 1:14 PM

Auth Service


Content Metadata Service


Content Storage Service


Viewer Service


Workflow Service


Workflow Service














[Show migration notice](#)

3 Building the Contract Approval application

This is the chapter throughout which you will be setting up your OpenText IMaaS development environment, and build, deploy and test the Contract Approval application.

It consists of 15 subsequent exercises that build on top of each other, so you cannot skip ahead, and it is very important you perform the exercises exactly as described.

That being said, if you are only interested in running the application and having a look at the finished project (incl. its IMaaS models and code), you can choose to skip to [Testing your application](#).

Of course, we recommend you go through all 15 exercises. In that case, you should start with the first exercise in this chapter (section 3.1) which details setting up your IMaaS Developer IDE.

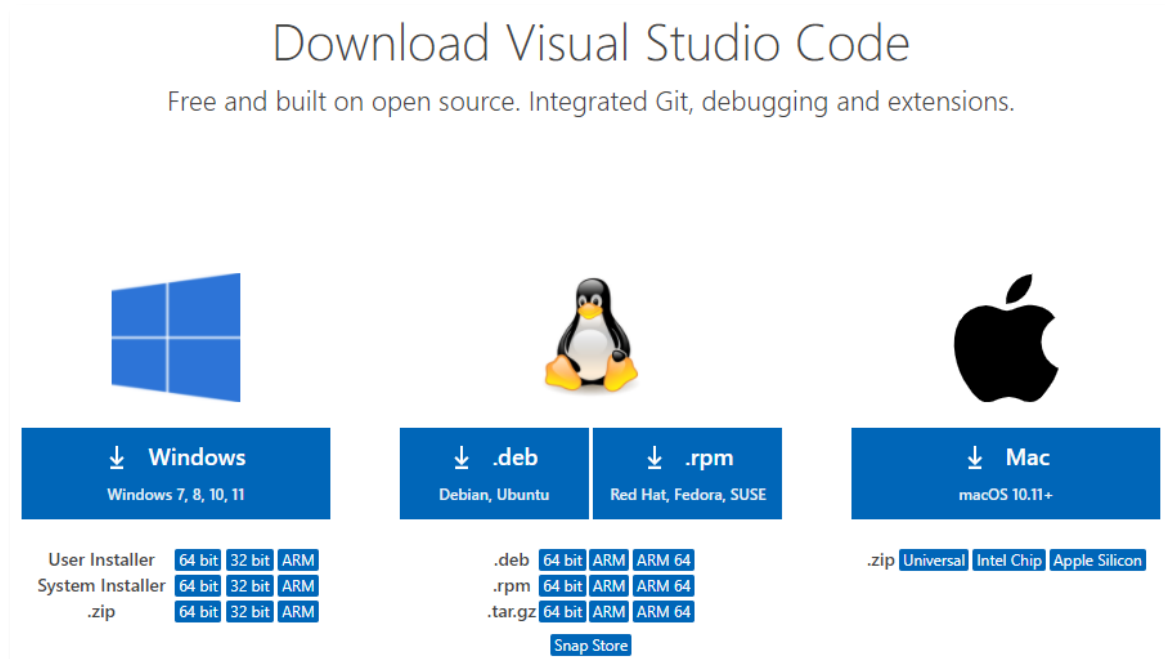
3.1 [25'] Setting up the IMaaS Developer IDE

This exercise will guide you through the downloading and installing of Microsoft Visual Studio Code (VS Code) and the adding of the IMaaS Tools VS Code extension to your VS Code installation.

Once you are done with this section, you will have set up your IMaaS Developer IDE, and you are ready to set up a connection to your IMaaS Developer organization.

To set up your IMaaS Developer IDE, proceed as follows:

- Navigate to <https://code.visualstudio.com/download> to download the Microsoft Visual Studio Code distribution that matches your system.



IMPORTANT:

Note that we have currently tested the OpenText IMaaS Tools for VS Code with Windows and macOS systems.

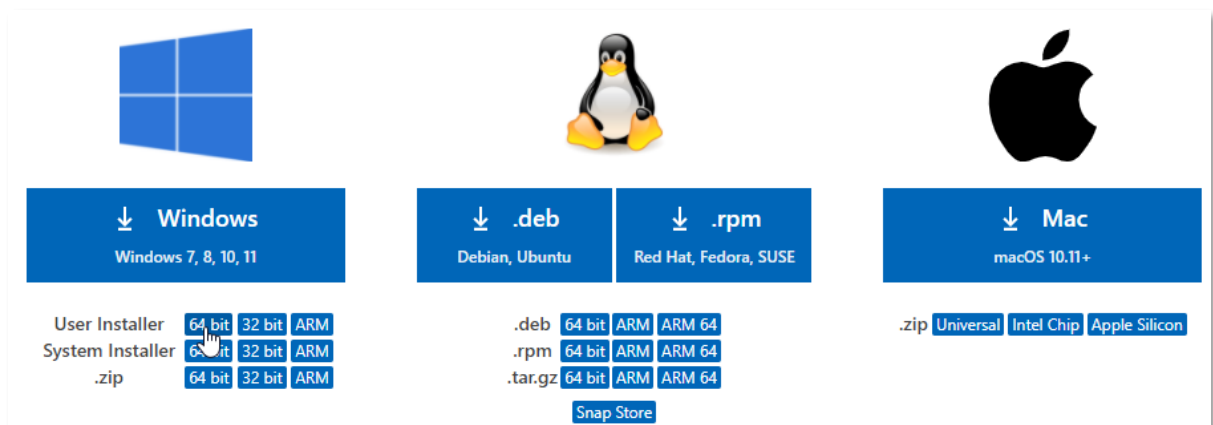
You can decide to use Linux, but if you run into problems on a Linux system, OpenText may not be able to provide you with a solution. Simply put, if you cannot perform the steps laid out in this tutorial when on a Linux system, you will have to switch to Windows or macOS.

In this tutorial, we will only provide you with the steps to perform the setting up of the (User Installer based) VS Code version for Windows 10.

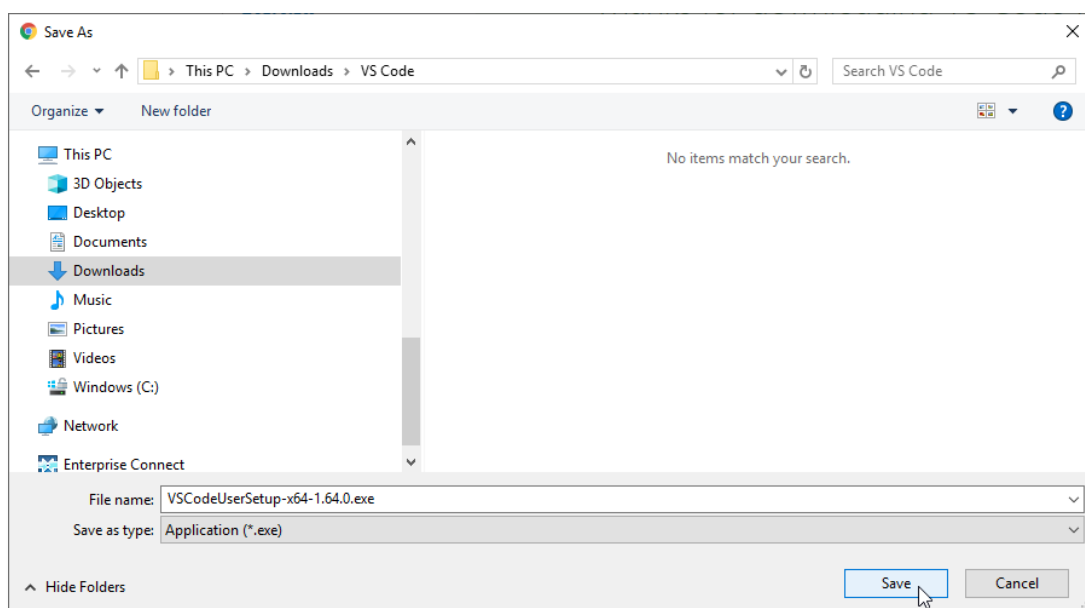
For other OS type systems or other versions of VS Code, you can follow the installation steps as outlined below as a guideline, but if the installation process is different, please refer to the VS Code documentation for further help.

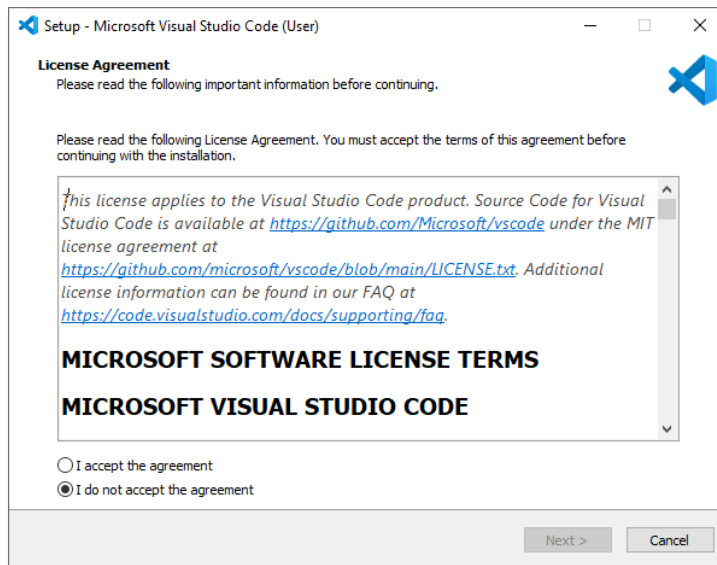
Also note that we recommend you install the latest version of VS Code, but that throughout the tutorial we will be using the current latest version for Windows 10 (1.64.0 64-bit), so if you have a different version and/or system, the screen shots might not always exactly match.

- If you are installing VS Code on a 64-bit Windows 10 system, choose to download the **64 bit User Installer for Windows**.

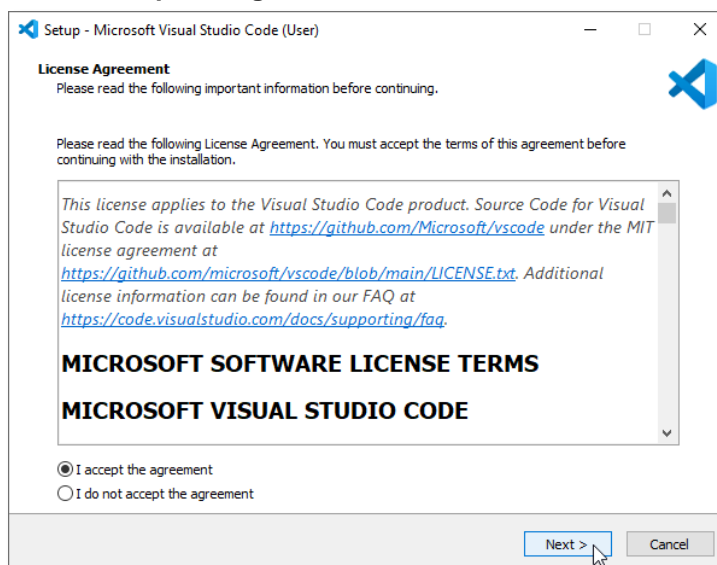


Save and run the installer.

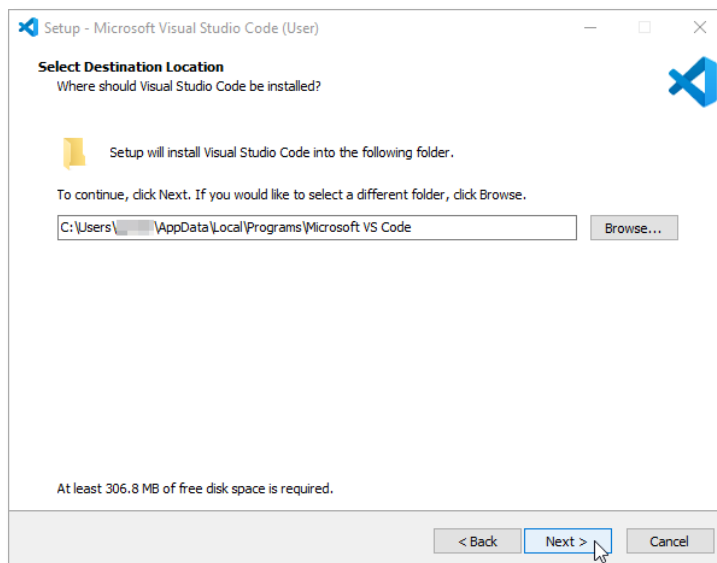




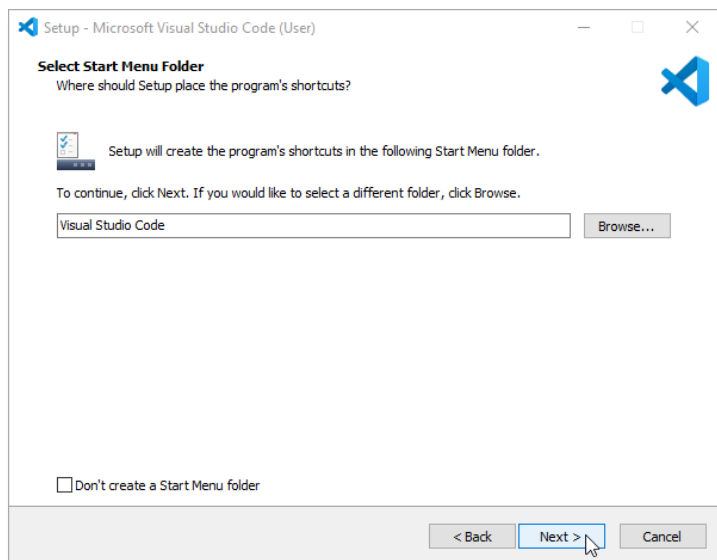
Select **I accept the agreement** and click **Next >** to continue.



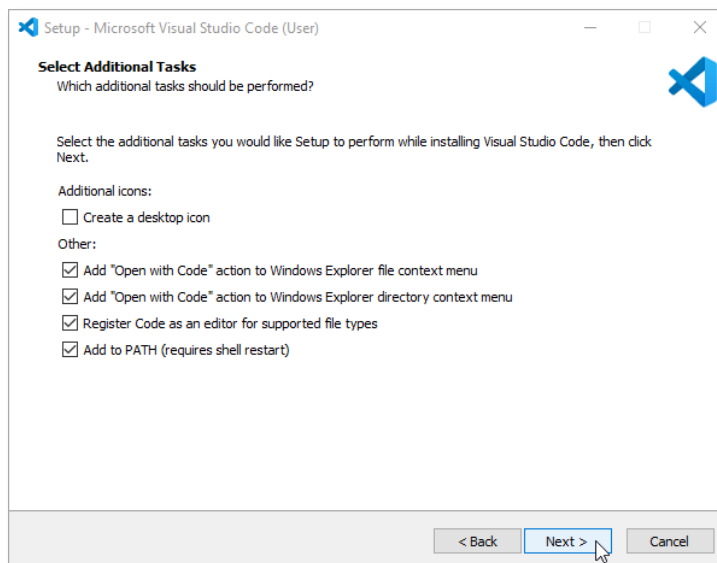
Select the installation destination location and click **Next >**. You can use the suggested default location.



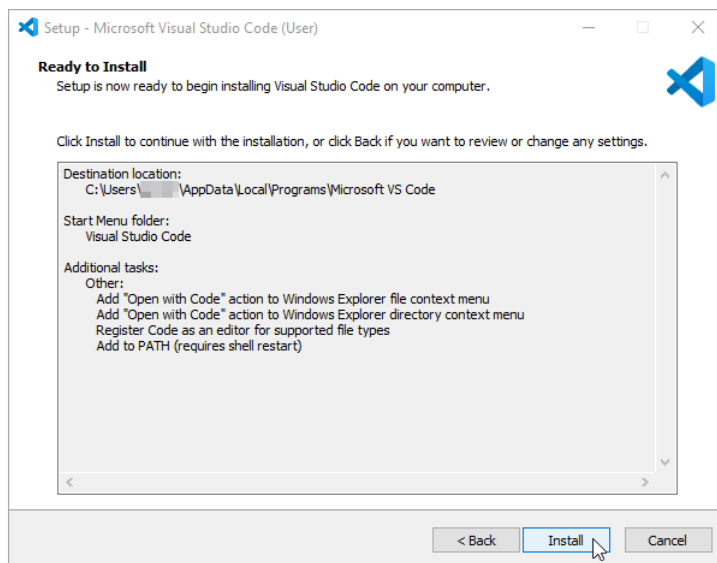
Keep the default setting for the selecting of the start menu folder and click **Next >**.



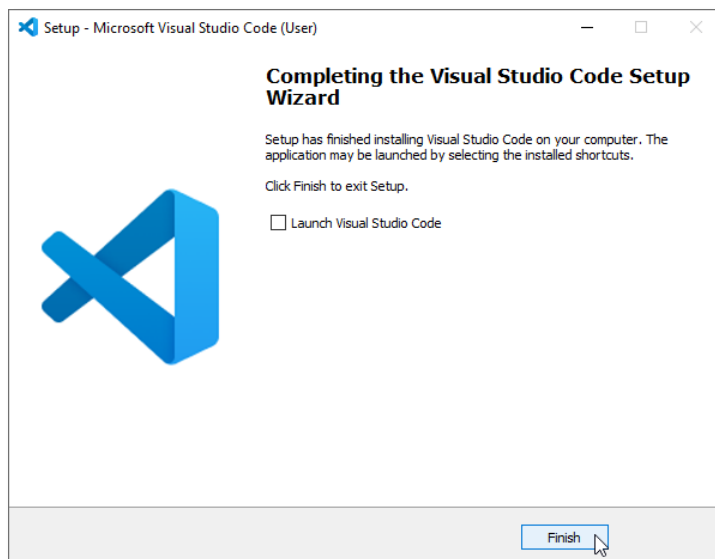
Select all additional tasks under **Other** and optionally select **Create a desktop icon**. Click **Next >** to continue.



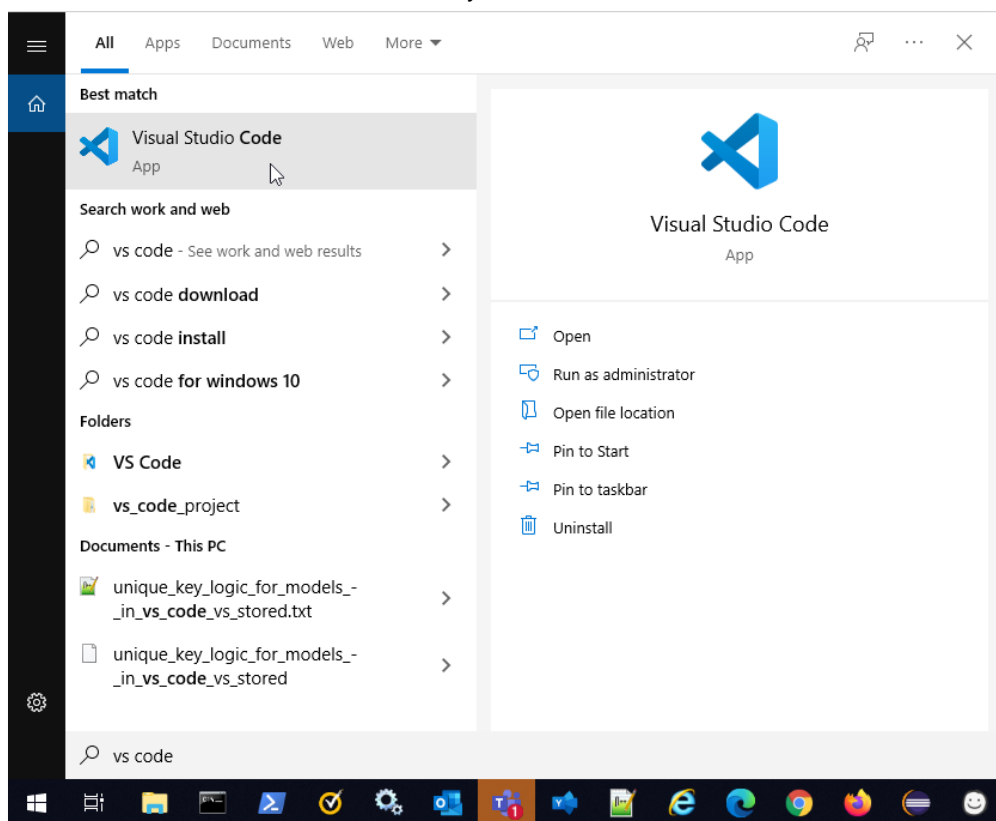
Verify all your choices and click **Install** to start the VS Code installation.



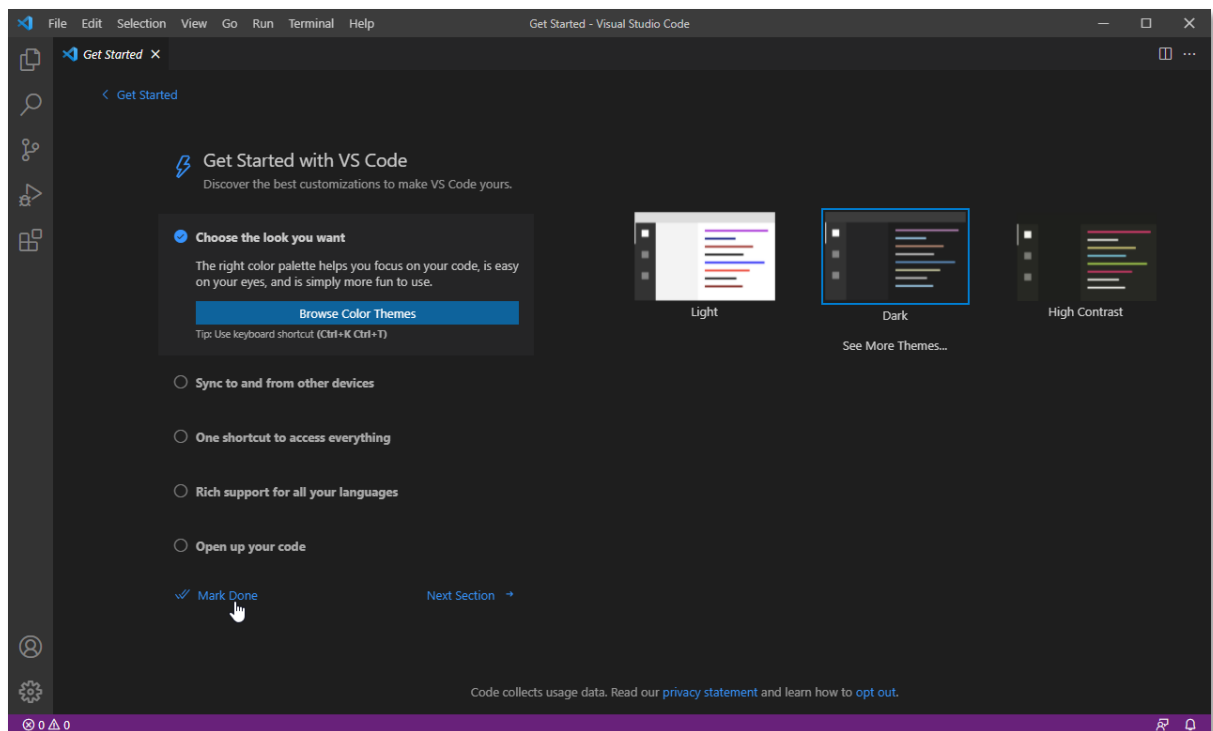
Once the installation is complete, you can click **Finish** to close the VS Code Setup Wizard (optionally leaving the **Launch Visual Studio Code** checkbox checked).



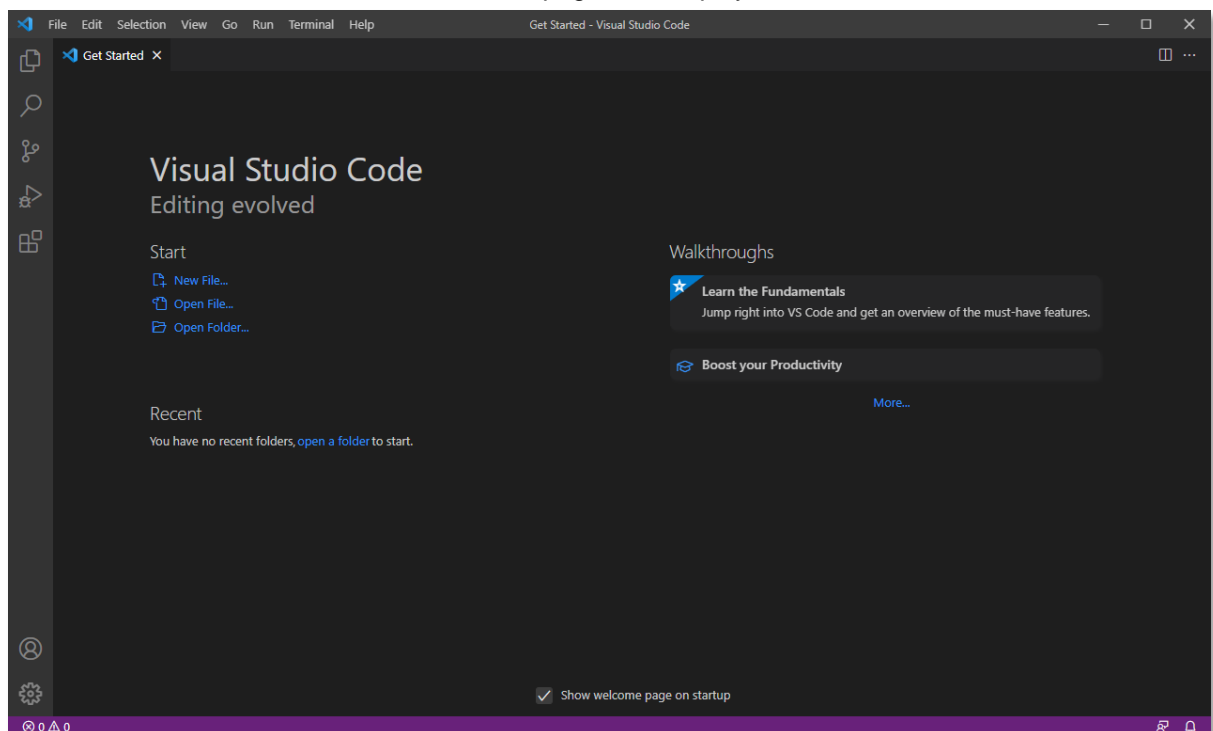
Once VS Code is installed, if not already open, you can open it by typing “vs code” in the Windows Start Menu search box and select the **Visual Studio Code** App. Feel free to **Pin to taskbar** for later easy access from the taskbar.



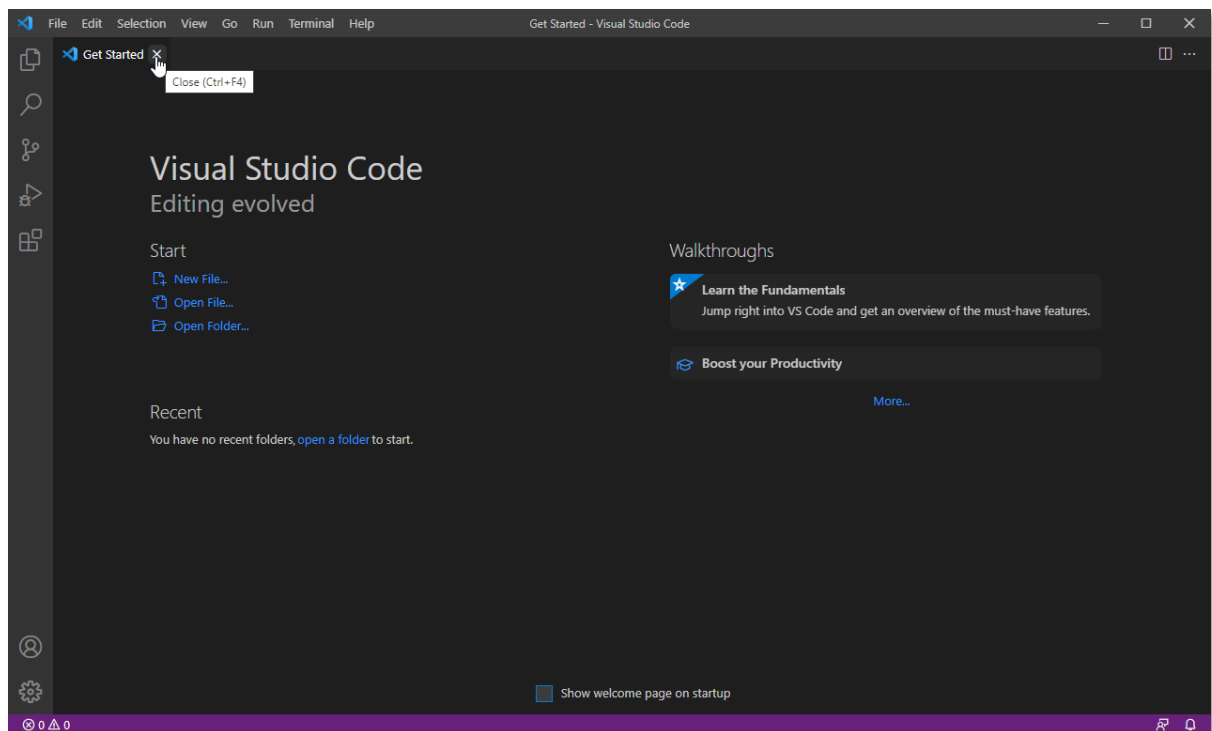
When opening VS Code for the first time, you are presented with the **Get Started with VS Code** wizard. Just perform the first activity of **Choose the look you want** by selecting the theme of your choosing. We are selecting the **Dark** theme. Click **Mark Done** to confirm your choice.



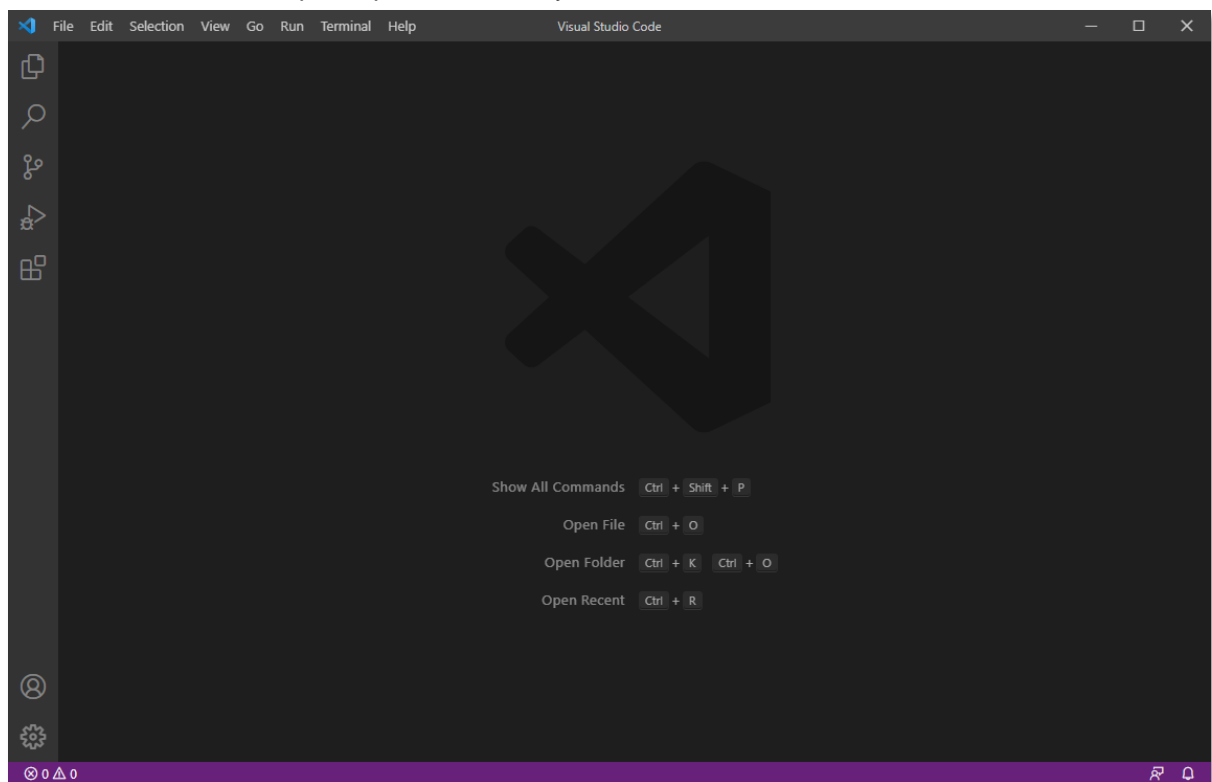
The standard VS Code **Get Started** welcome page now displays.



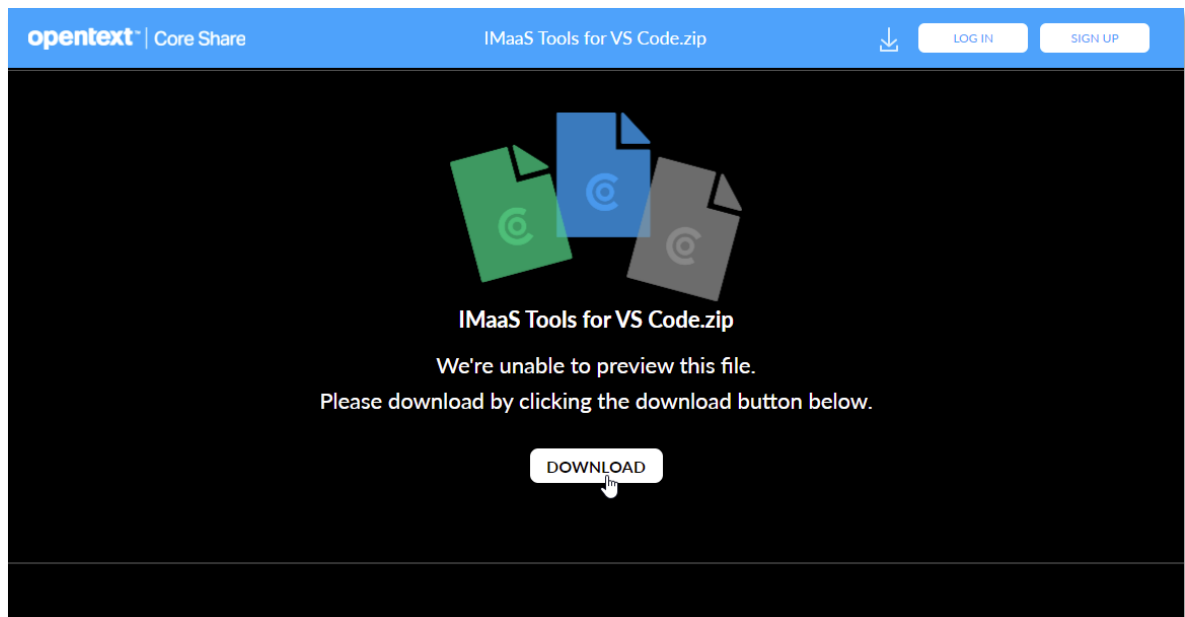
You can uncheck the **Show welcome page on startup** and close the **Get Started** page so that it will no longer display when you open VS Code.



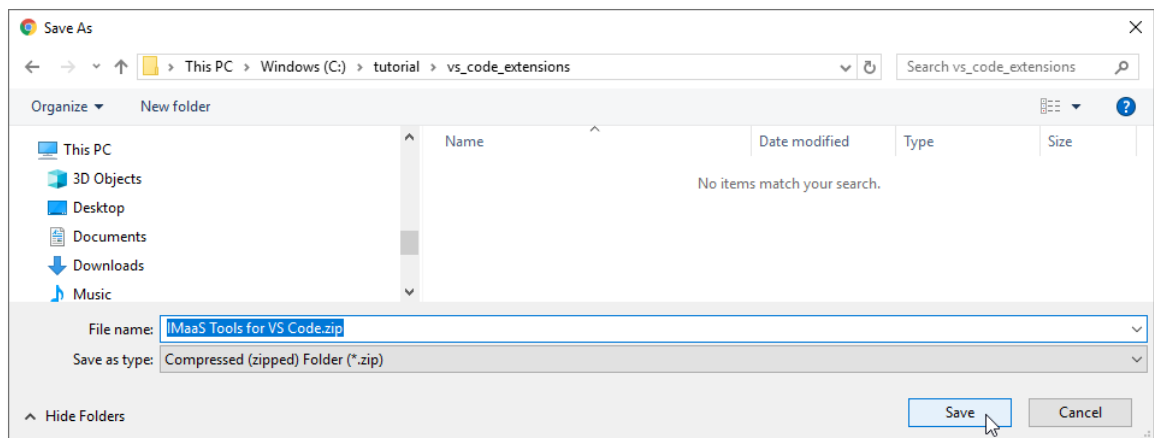
You are now ready to add the **OpenText IMaaS Tools for VS Code** to your VS Code IDE to enable the IMaaS Developer capabilities directly in VS Code.



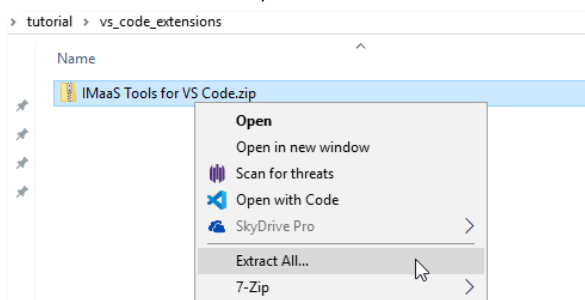
- Before you can install the **OpenText IMaaS Tools** VS Code extensions in VS Code, you first need to download them through this [link](#) (sign up for a Core Share account, if needed).

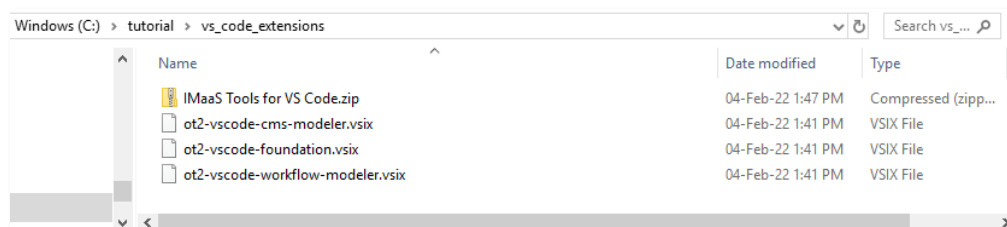
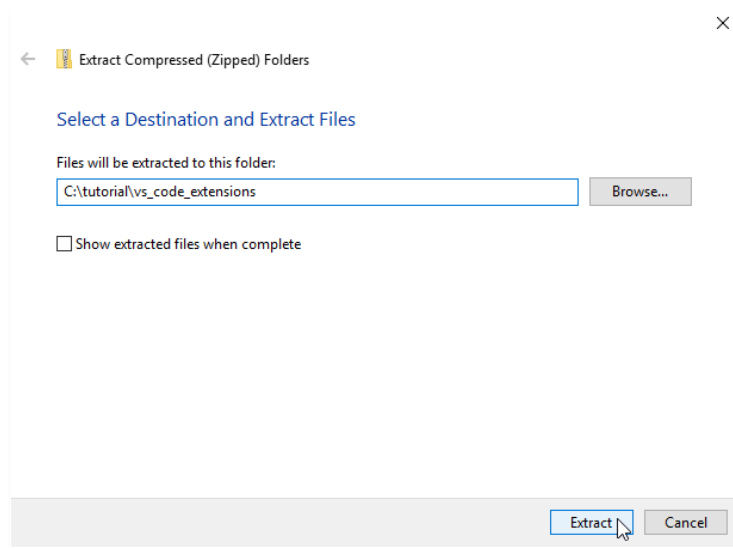


Save the **IMaaS Tools for VS Code.zip** file to a folder (e.g.: **vs_code_extensions**) on your local disk.

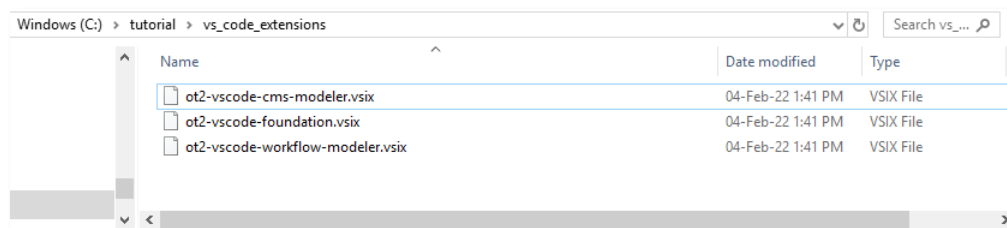


Choose to extract the contents of the **IMaaS Tools for VS Code.zip** file to the same (e.g.: **vs_code_extensions**) folder.





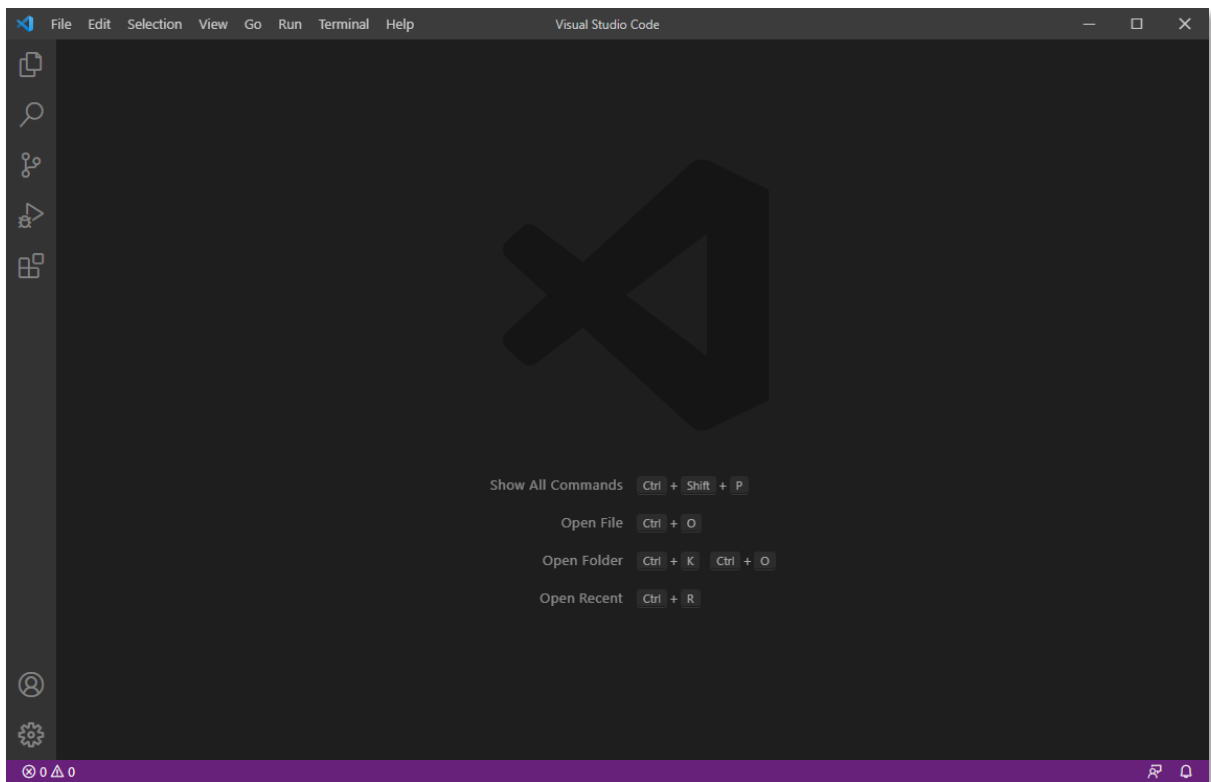
Once the extensions (.vsix files) are extracted, you can delete the **IMaaS Tools for VS Code.zip** file.




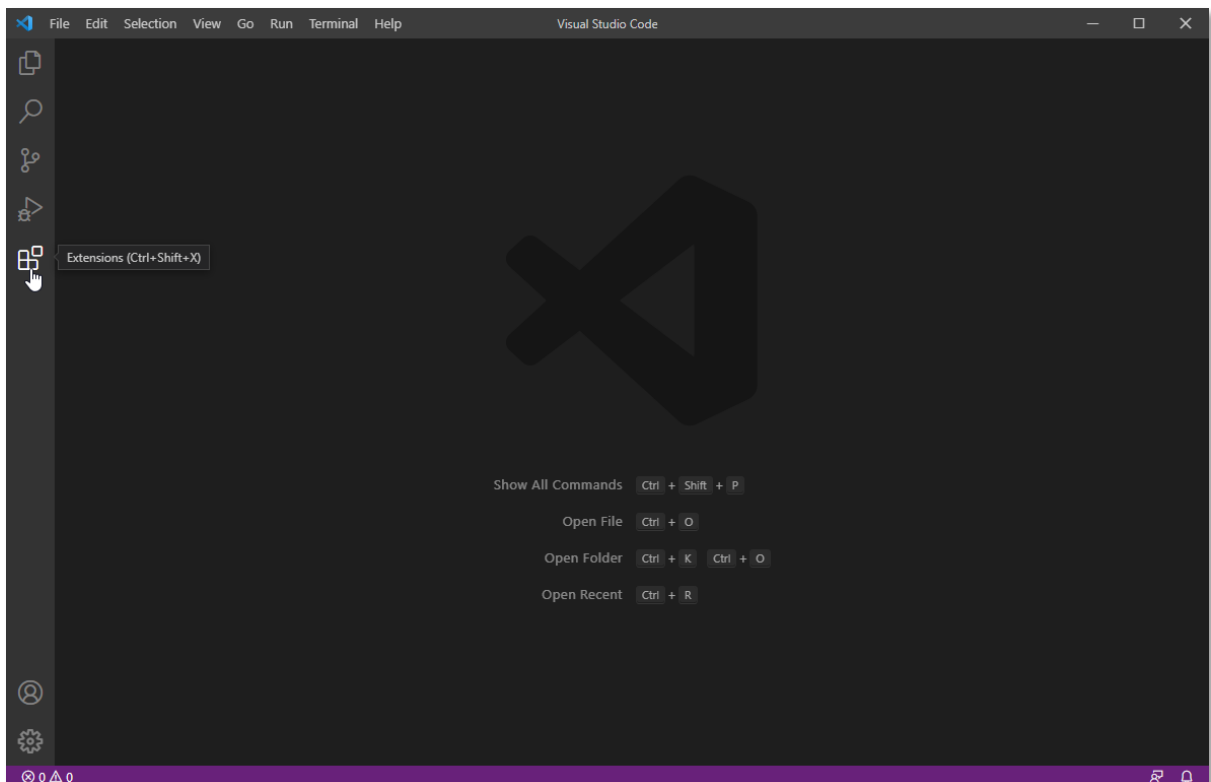
IMPORTANT:

Currently the OpenText IMaaS Tools for VS Code consist of 3 separate VS Code extensions. This is only applicable during the internal beta testing phase. As soon as the OpenText IMaaS Tools for VS Code are made public (to support the public beta testing phase), they will be available from the VS Code Marketplace as an extension pack (grouping all the different extensions together as one download).

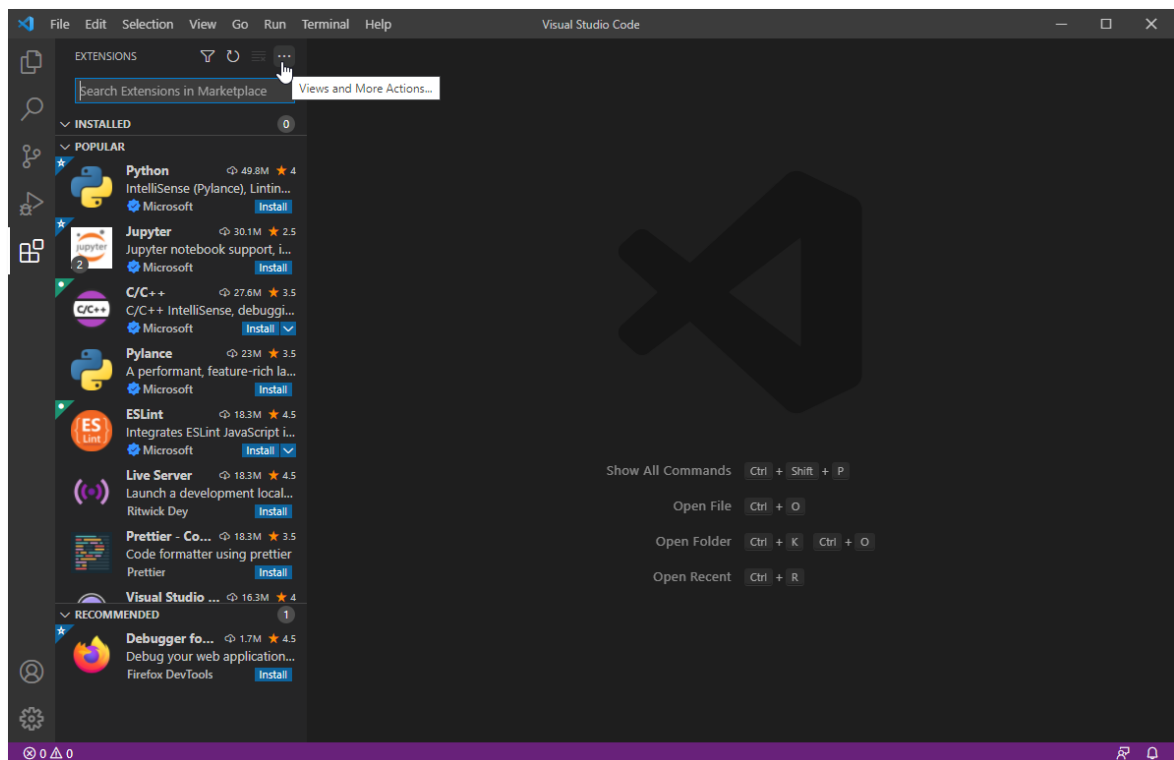
- The **OpenText IMaaS Tools** VS Code extensions are now downloaded, so you can go back to VS Code to install them.



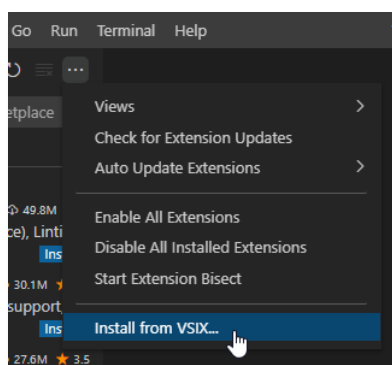
The VS Code Activity Bar on the left side lets you quickly switch between different views. Click  to switch to the **Extensions** view.



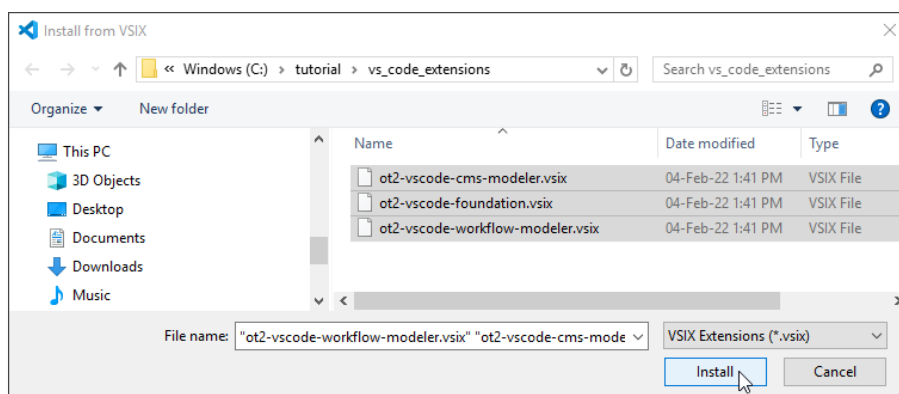
Select the [...] button above the **Search Extensions in Marketplace** search box to open the **Views and More Actions...** menu.



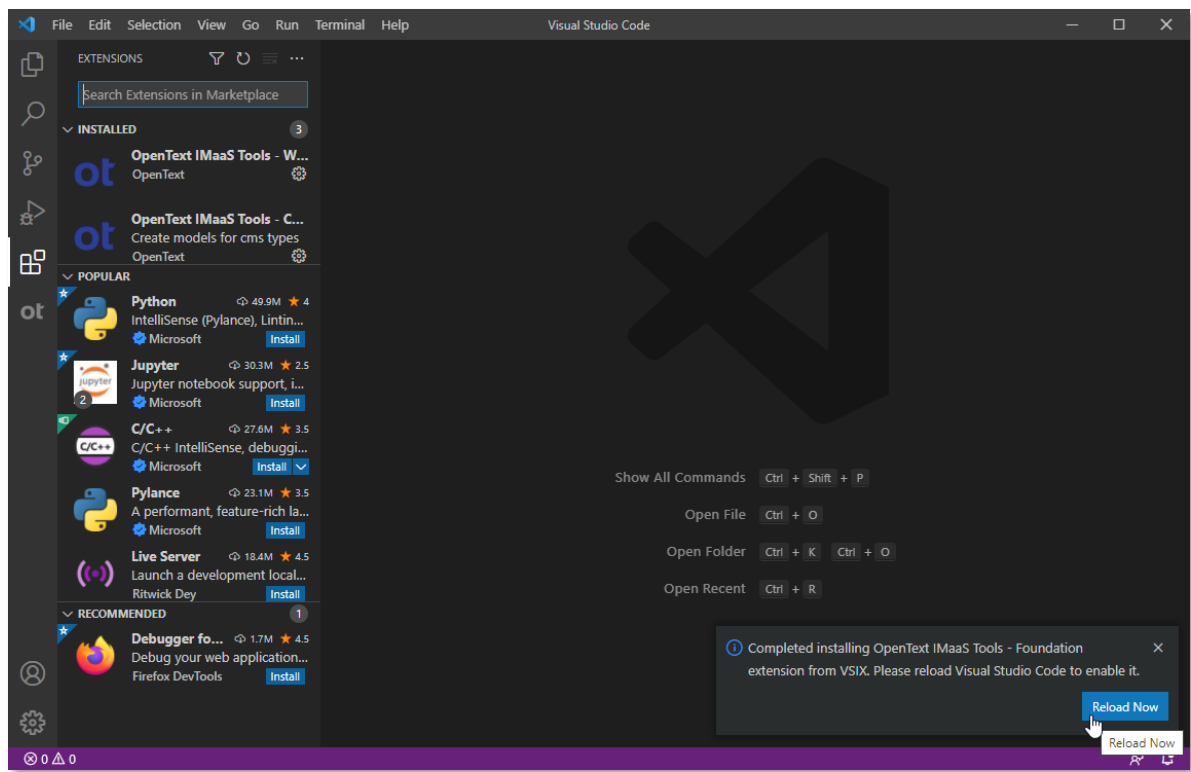
Select **Install from VSIX...**



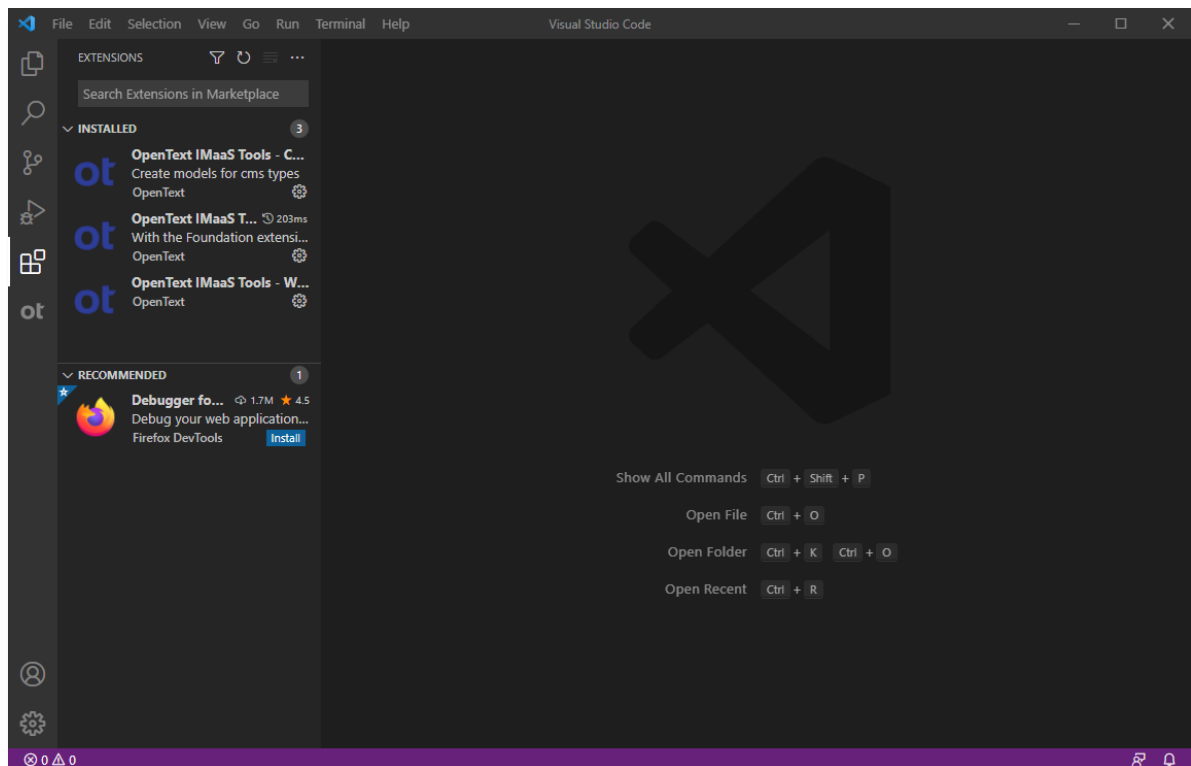
Select (all) the **OpenText IMaaS Tools** VS Code extensions you previously downloaded and click **Install** to install them into VS Code.



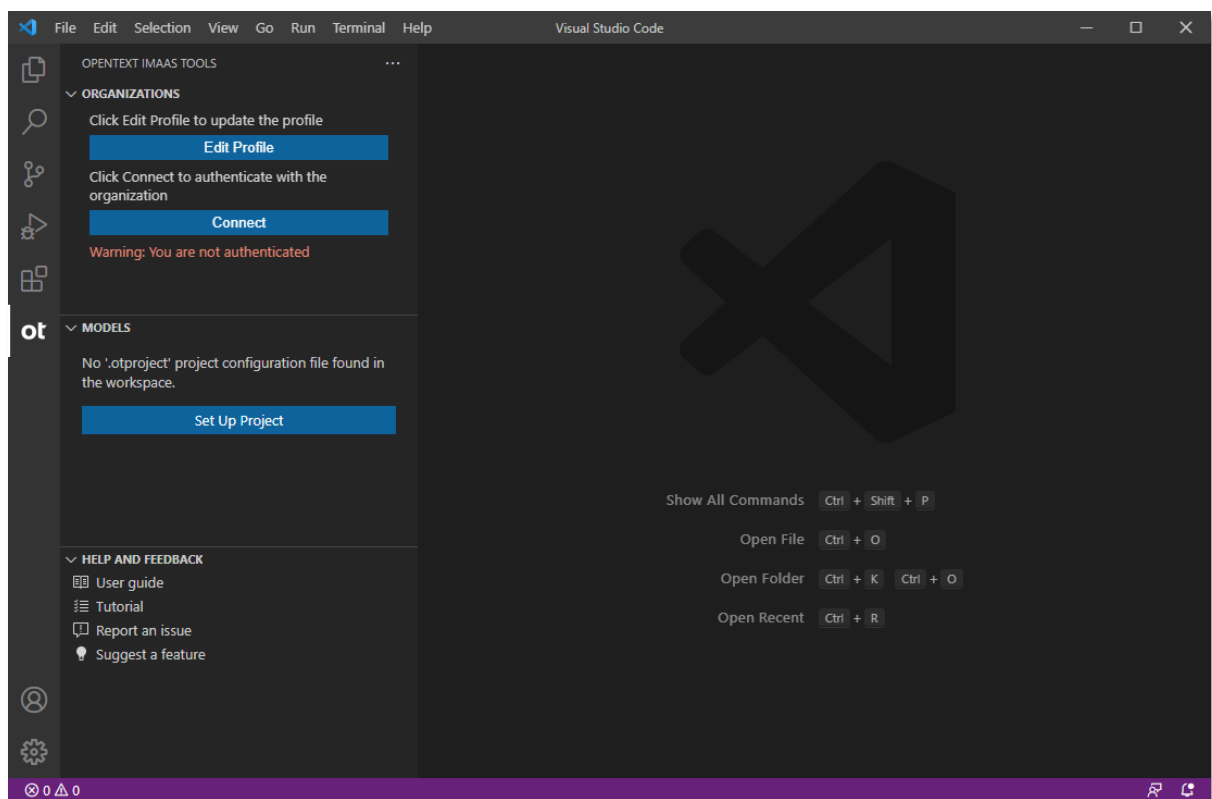
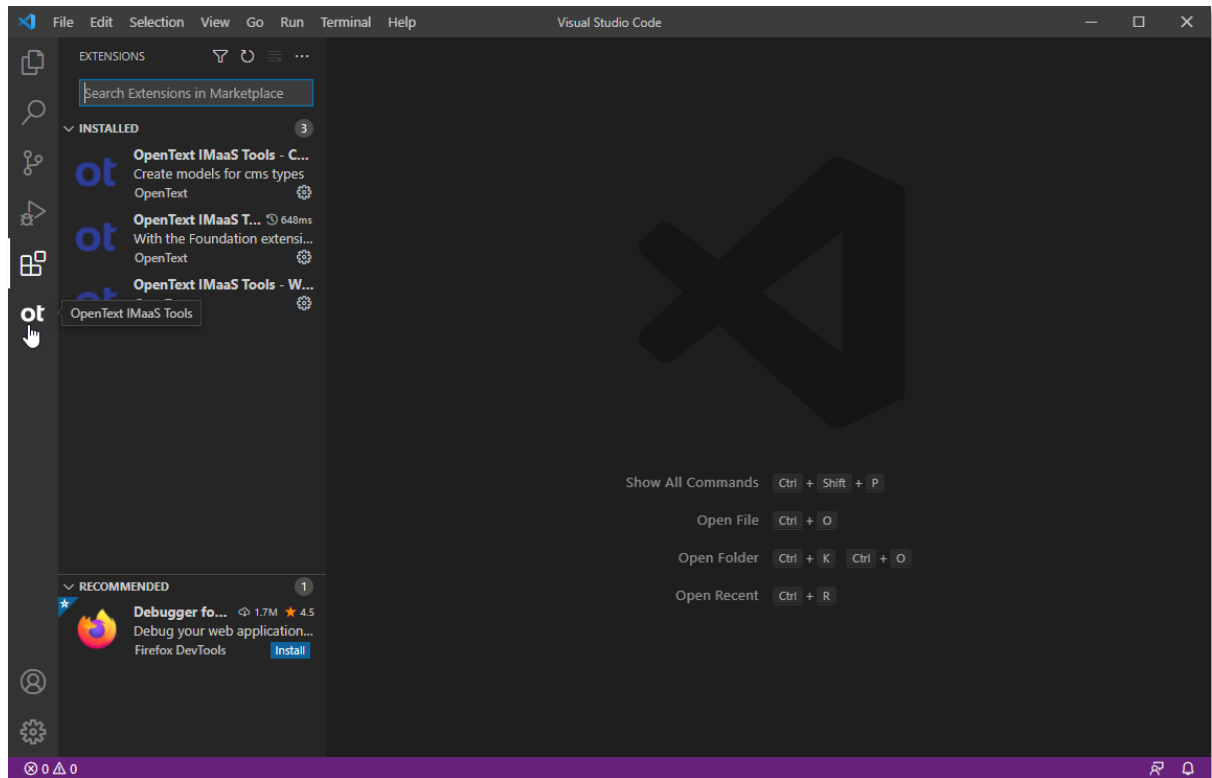
You will see some pop-up dialog boxes in the bottom right of VS Code. You can click **Reload Now** to indeed ensure the VS Code extensions you just installed get properly enabled.



Alternatively (e.g., if you missed the **Reload Now** button), you can also close and re-open VS Code, as this will certainly ensure the newly installed **OpenText IMaaS Tools** are enabled.



To confirm that the **OpenText IMaaS Tools for VS Code** are installed and working correctly, you can now switch to the **OpenText IMaaS Tools** view from the Activity Bar on the left side by clicking **ot**.



- Although you have now installed your VS Code IDE with the OpenText IMaaS Tools extensions, you also need to install Node.js 14.2.0 to support building and running the Contract Approval application specifically (as it is a Node.js and React application).

IMPORTANT:

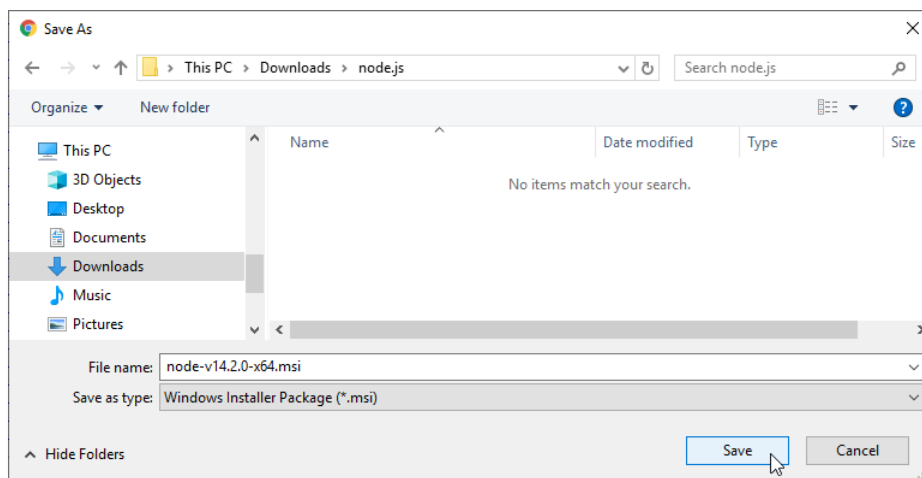
Make sure to install version 14.2.0 of Node.js, as otherwise you can (and most likely will) have problems running your Contract Approval application.

To install **Node.js 14.2.0**, navigate to <https://nodejs.org/download/release/v14.2.0/>, and download and run the correct installer for your OS.

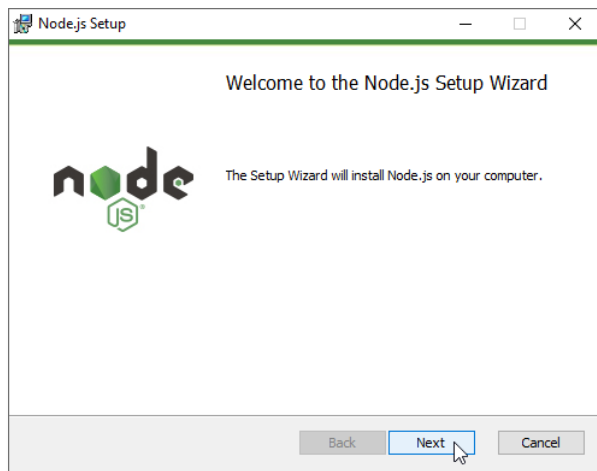
Like with the VS Code installation earlier, we will only go through the steps of the Node.js set up process for Windows 10.

Index of /download/release/v14.2.0/

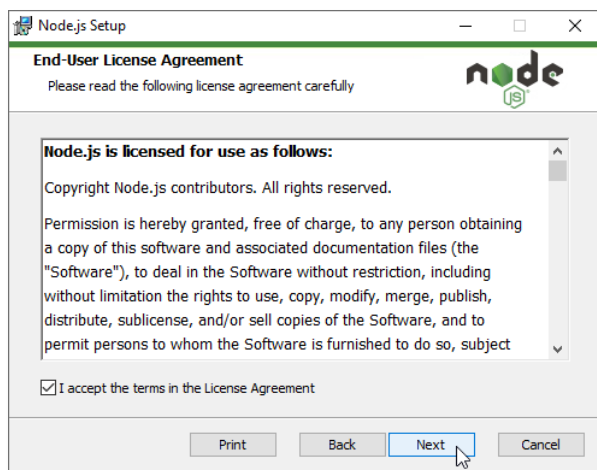
../		
docs/	05-May-2020 17:55	-
win-x64/	05-May-2020 17:47	-
win-x86/	05-May-2020 17:38	-
SHASUMS256.txt	05-May-2020 18:26	2929
SHASUMS256.txt.asc	05-May-2020 18:26	3811
SHASUMS256.txt.sig	05-May-2020 18:26	566
node-v14.2.0-aix-ppc64.tar.gz	05-May-2020 17:17	43279693
node-v14.2.0-darwin-x64.tar.gz	05-May-2020 17:27	31230592
node-v14.2.0-darwin-x64.tar.xz	05-May-2020 17:28	20042796
node-v14.2.0-headers.tar.gz	05-May-2020 17:55	554889
node-v14.2.0-headers.tar.xz	05-May-2020 17:55	370540
node-v14.2.0-linux-arm64.tar.gz	05-May-2020 17:18	33389128
node-v14.2.0-linux-arm64.tar.xz	05-May-2020 17:20	20270164
node-v14.2.0-linux-armv7l.tar.gz	05-May-2020 17:21	31651375
node-v14.2.0-linux-armv7l.tar.xz	05-May-2020 17:22	18780916
node-v14.2.0-linux-ppc64le.tar.gz	05-May-2020 17:20	35310510
node-v14.2.0-linux-ppc64le.tar.xz	05-May-2020 17:22	21923632
node-v14.2.0-linux-s390x.tar.gz	05-May-2020 17:20	33678519
node-v14.2.0-linux-s390x.tar.xz	05-May-2020 17:21	20393852
node-v14.2.0-linux-x64.tar.gz	05-May-2020 17:57	33355766
node-v14.2.0-linux-x64.tar.xz	05-May-2020 17:58	20822708
node-v14.2.0-win-x64.7z	05-May-2020 17:47	17100783
node-v14.2.0-win-x64.zip	05-May-2020 17:47	28385357
node-v14.2.0-win-x86.7z	05-May-2020 17:38	16000340
node-v14.2.0-win-x86.zip	05-May-2020 17:39	26837549
node-v14.2.0-x64.msi	05-May-2020 17:47	30150656
node-v14.2.0-x86.msi	05-May-2020 17:39	28508160
node-v14.2.0.pkg	05-May-2020 17:42	31574021
node-v14.2.0.tar.gz	05-May-2020 17:48	61511944
node-v14.2.0.tar.xz	05-May-2020 17:53	32884616



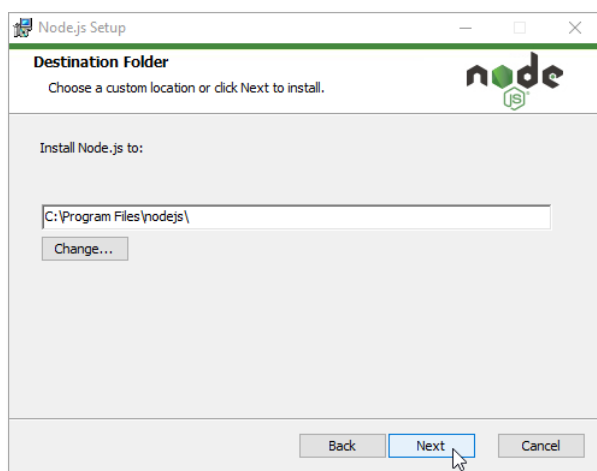
Click **Next** to continue.



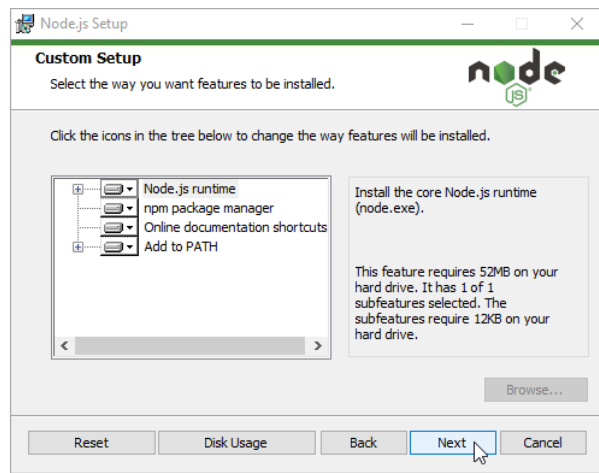
Select **I accept the terms in the License Agreement** and click **Next** to continue.



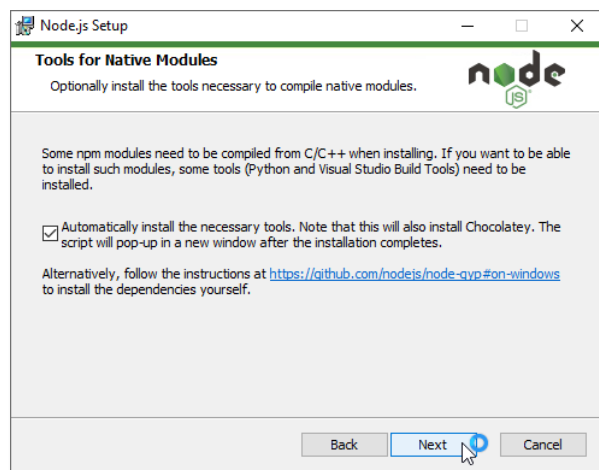
Select the installation destination location and click **Next**. You can use the suggested default location.



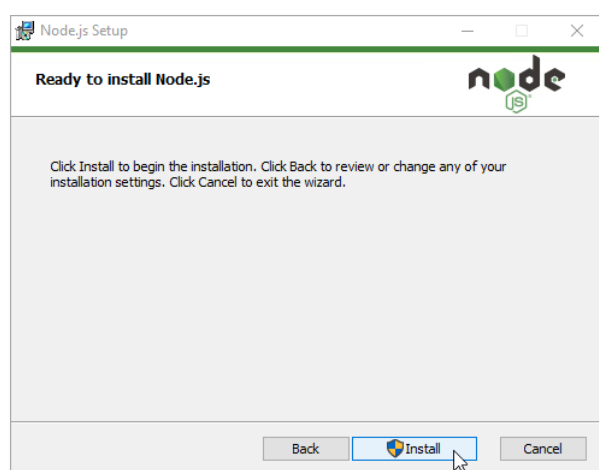
From the **Custom Setup** screen, we recommend you accept the default and just click **Next**.



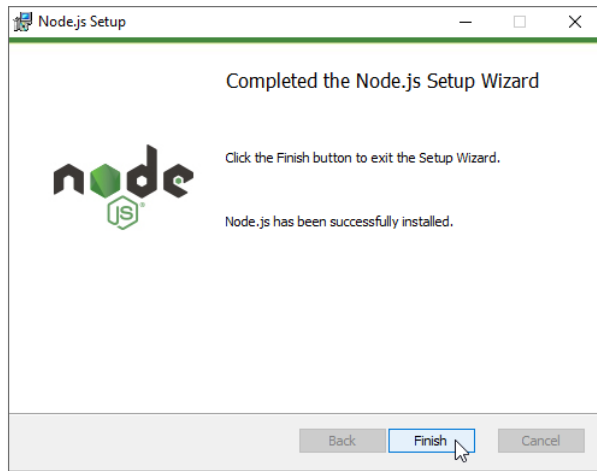
Select to **Automatically install the necessary tools** and click **Next**.



Click **Install** to start the installation.



Once the setup has completed, you can click **Finish** to close the Node.js Setup Wizard.



Press a key (e.g.: **space bar**) several times to run through the installation of the Additional Tools for Node.js.

```

Install Additional Tools for Node.js

=====
Tools for Node.js Native Modules Installation Script
=====

This script will install Python and the Visual Studio Build Tools, necessary
to compile Node.js native modules. Note that Chocolatey and required Windows
updates will also be installed.

This will require about 3 Gb of free disk space, plus any space necessary to
install Windows updates. This will take a while to run.

Please close all open programs for the duration of the installation. If the
installation fails, please ensure Windows is fully updated, reboot your
computer and try to run this again. This script can be found in the
Start menu under Node.js.

You can close this window to stop now. Detailed instructions to install these
tools manually are available at https://github.com/nodejs/node-gyp#on-windows

Press any key to continue . . .

```

```

Install Additional Tools for Node.js

Using this script downloads third party software

-----
This script will direct to Chocolatey to install packages. By using
Chocolatey to install a package, you are accepting the license for the
application, executable(s), or other artifacts delivered to your machine as a
result of a Chocolatey install. This acceptance occurs whether you know the
license terms or not. Read and understand the license terms of the packages
being installed and their dependencies prior to installation:
- https://chocolatey.org/packages/chocolatey
- https://chocolatey.org/packages/python
- https://chocolatey.org/packages/visualstudio2017-workload-vctools

This script is provided AS-IS without any warranties of any kind
-----
Chocolatey has implemented security safeguards in their process to help
protect the community from malicious or pirated software, but any use of this
script is at your own risk. Please read the Chocolatey's legal terms of use
as well as how the community repository for Chocolatey.org is maintained.

Press any key to continue . . .

```

Once done, press **Enter** to exit the PowerShell window that popped up.

```
Administrator: Windows PowerShell
WARNING: 'choco' was found at 'C:\ProgramData\chocolatey\bin\choco.exe'.
WARNING: An existing Chocolatey installation was detected. Installation will not continue.
For security reasons, this script will not overwrite existing installations.

Please use choco upgrade chocolatey to handle upgrades of Chocolatey itself.
Chocolatey v0.11.2
Upgrading the following packages:
python;visualstudio2017-workload-vctools
By upgrading, you accept licenses for the packages.

You have python v3.9.5 installed. Version 3.10.2 is available based on your source(s).
Progress: Downloading python3 3.10.2... 100%
Progress: Downloading python 3.10.2... 100%

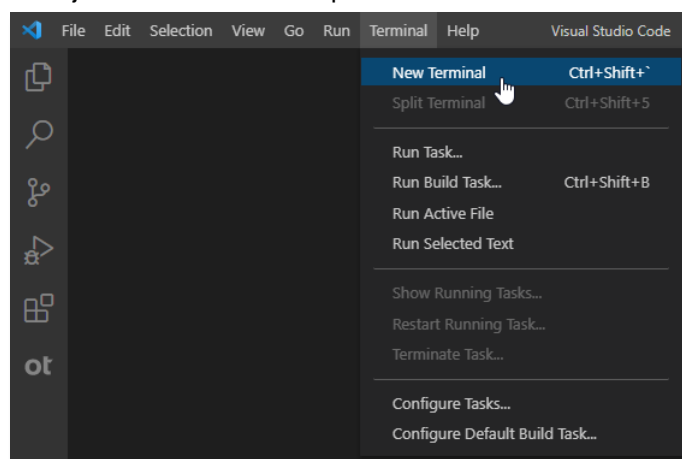
python3 v3.10.2 [Approved]
python3 package files upgrade completed. Performing other installation steps.
Installing 64-bit python3...
python3 has been installed.
Python installed to: 'C:\Python310'
Restricting write permissions to Administrators
python3 can be automatically uninstalled.
Environment Vars (like PATH) have changed. Close/reopen your shell to
see the changes (or in powershell/cmd.exe just type 'refreshenv').
The upgrade of python3 was successful.
Software installed as 'exe', install location is likely default.

python v3.10.2 [Approved]
python package files upgrade completed. Performing other installation steps.
The upgrade of python was successful.
Software install location not explicitly set, it could be in package or
default install location of installer.
visualstudio2017-workload-vctools v1.3.3 is the latest version available based on your source(s).

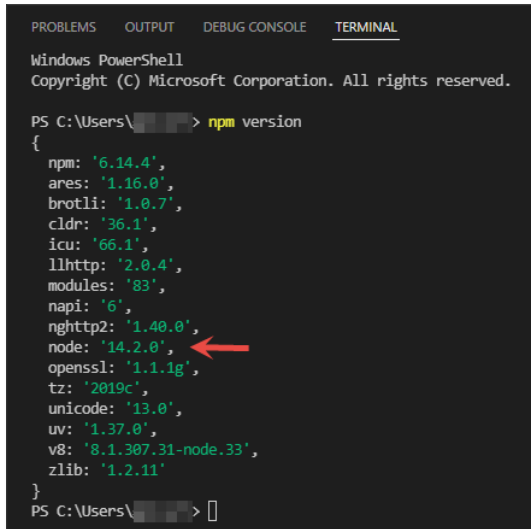
Chocolatey upgraded 2/3 packages.
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

Did you know the proceeds of Pro (and some proceeds from other
licensed editions) go into bettering the community infrastructure?
Your support ensures an active community, keeps Chocolatey tip-top,
plus it nets you some awesome features!
https://chocolatey.org/compare
Type ENTER to exit:
```

Node.js is now installed, let's quickly verify that VS Code is ready to use it when running your Node.js code. In VS Code open a new **Terminal**.



In the newly opened **Terminal**, type **npm version** and confirm node is indeed of version 14.2.0.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\> npm version
{
  npm: '6.14.4',
  ares: '1.16.0',
  brotli: '1.0.7',
  cldr: '36.1',
  icu: '66.1',
  llhttp: '2.0.4',
  modules: '83',
  napi: '6',
  nghttp2: '1.40.0',
  node: '14.2.0',
  openssl: '1.1.1g',
  tz: '2019c',
  unicode: '13.0',
  uv: '1.37.0',
  v8: '8.1.307.31-node.33',
  zlib: '1.2.11'
}
PS C:\Users\>
```

You are now ready to start building your Contract Approval Application project.

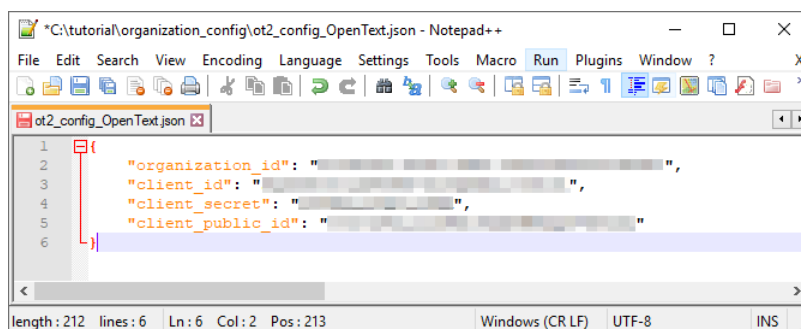
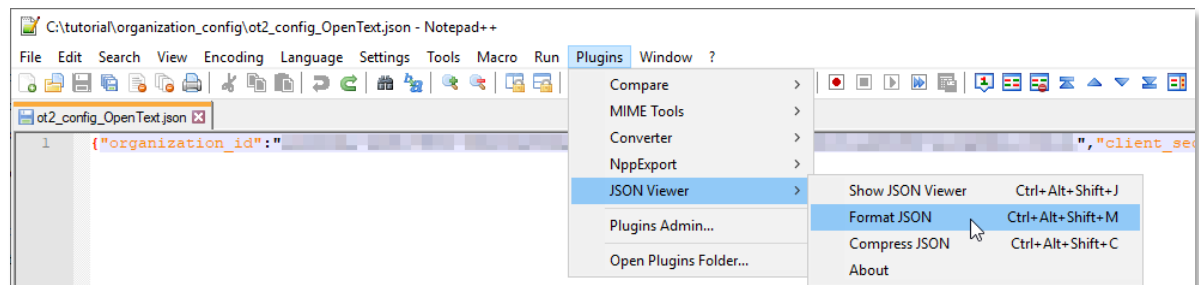
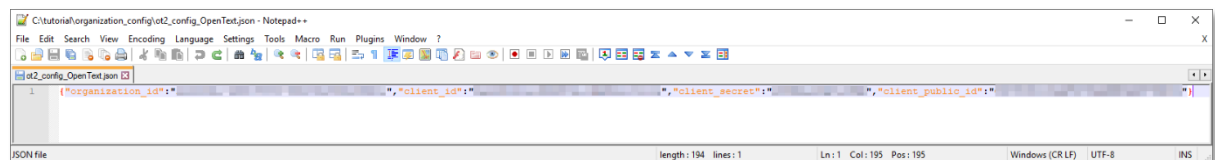
3.2 [10'] Adding an organization and testing the connection

During this exercise you will add the organization you previously created (when registering for an OpenText Developer free trial Account) to your VS Code IMaaS user settings. You will also test the connection to your organization, to ensure all is indeed set up correctly. The purpose of adding an organization is that you will later be able to deploy the application you have built into the organization, and more specifically the corresponding single developer tenant.

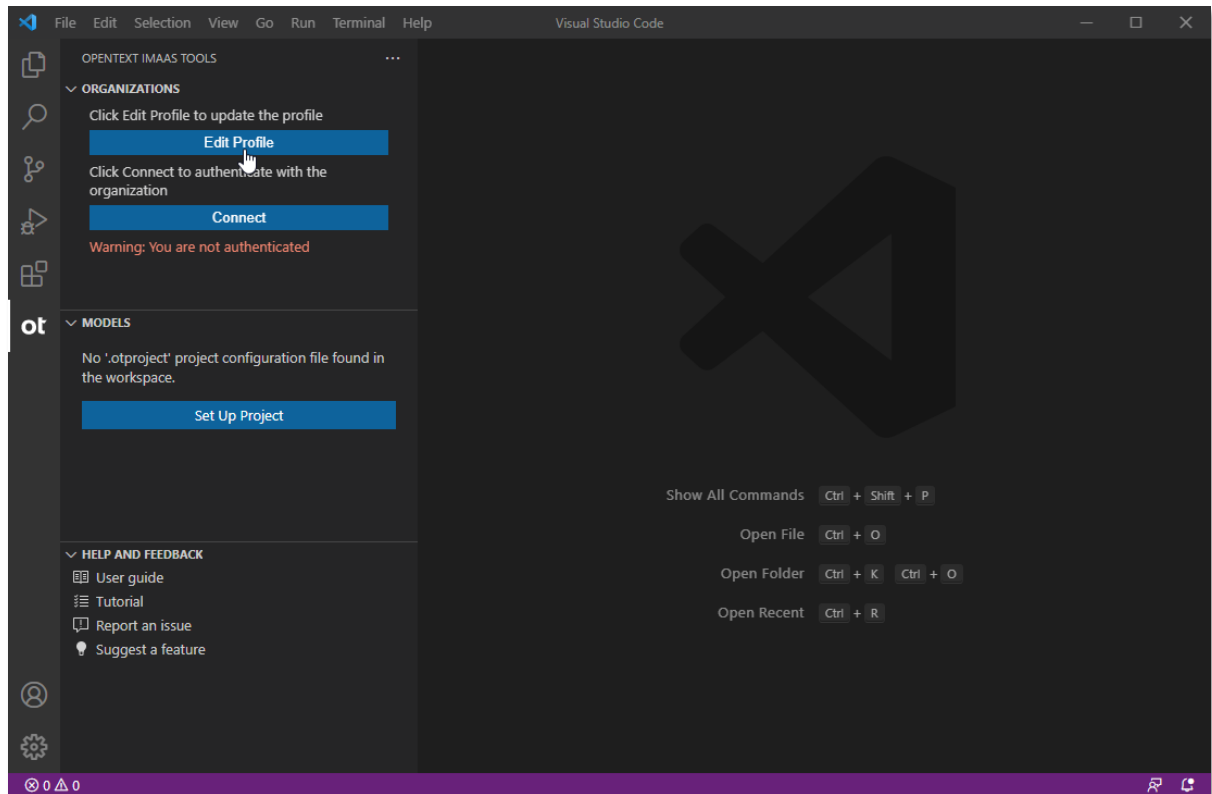
Once you are done with this section, you will have set up your IMaaS Developer organization connection, and you are ready to start creating your Contract Approval application project.

To add your organization to your IMaaS user settings and test the connection, proceed as follows:

- Open the **ot2_config_<organization name, we are using OpenText>.json** organization configuration file you previously saved in a text editor. We recommend you use **Notepad++** with the **JSON Viewer** plugin installed, so that you can format the **ot2_config_<organization name>.json** file to be more easily consumable (as shown in below screen shots). This is not required though to perform the steps in this exercise, as any text editor will allow you to copy the different required values.

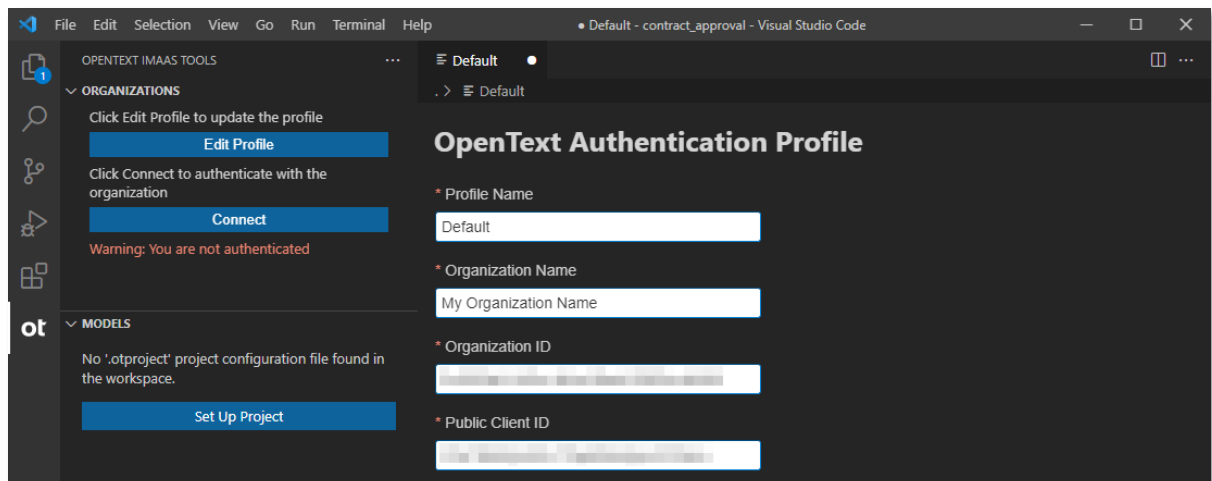


- You can add the organization connection configuration by editing the authentication profile. To do this, open VS Code, select the **OpenText IMaaS Tools** view from the Activity Bar on the left side, and click the **Edit Profile** button under the **ORGANIZATIONS** section.

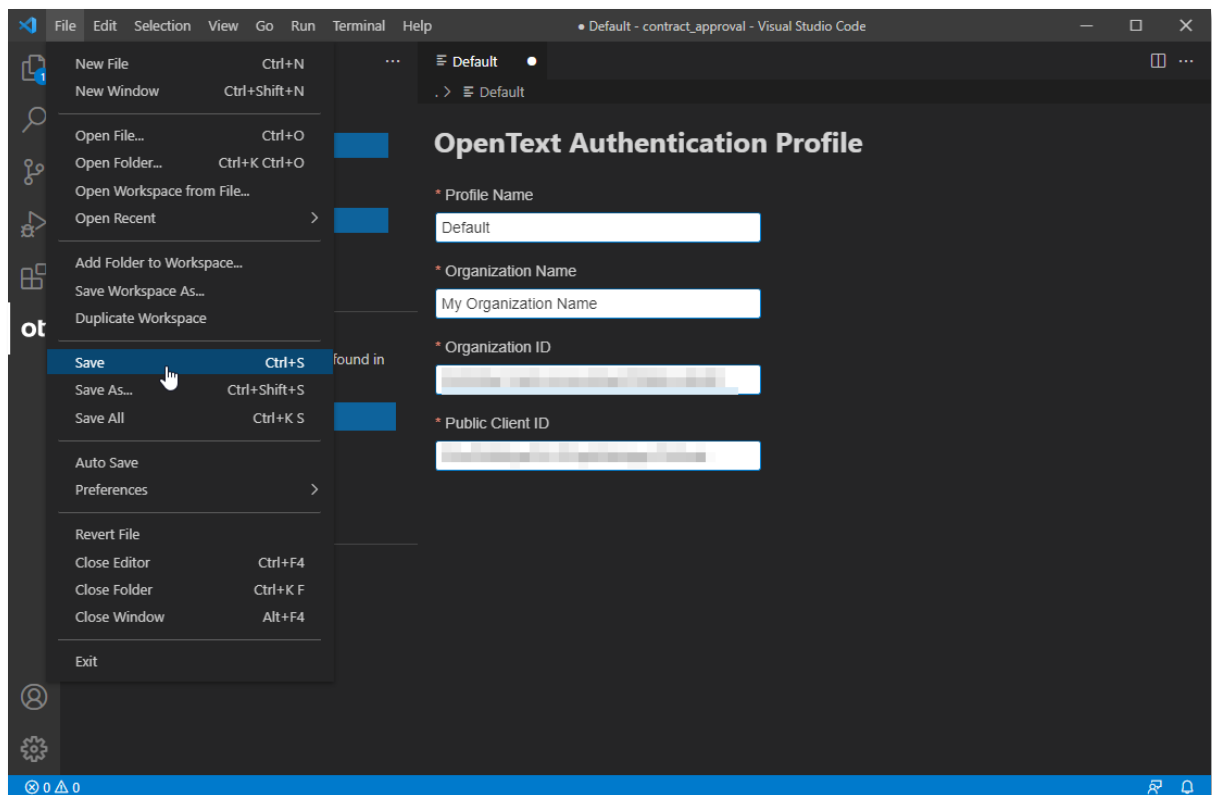


Fill the different authentication profile property values on the **OpenText Authentication Profile** form as follows:

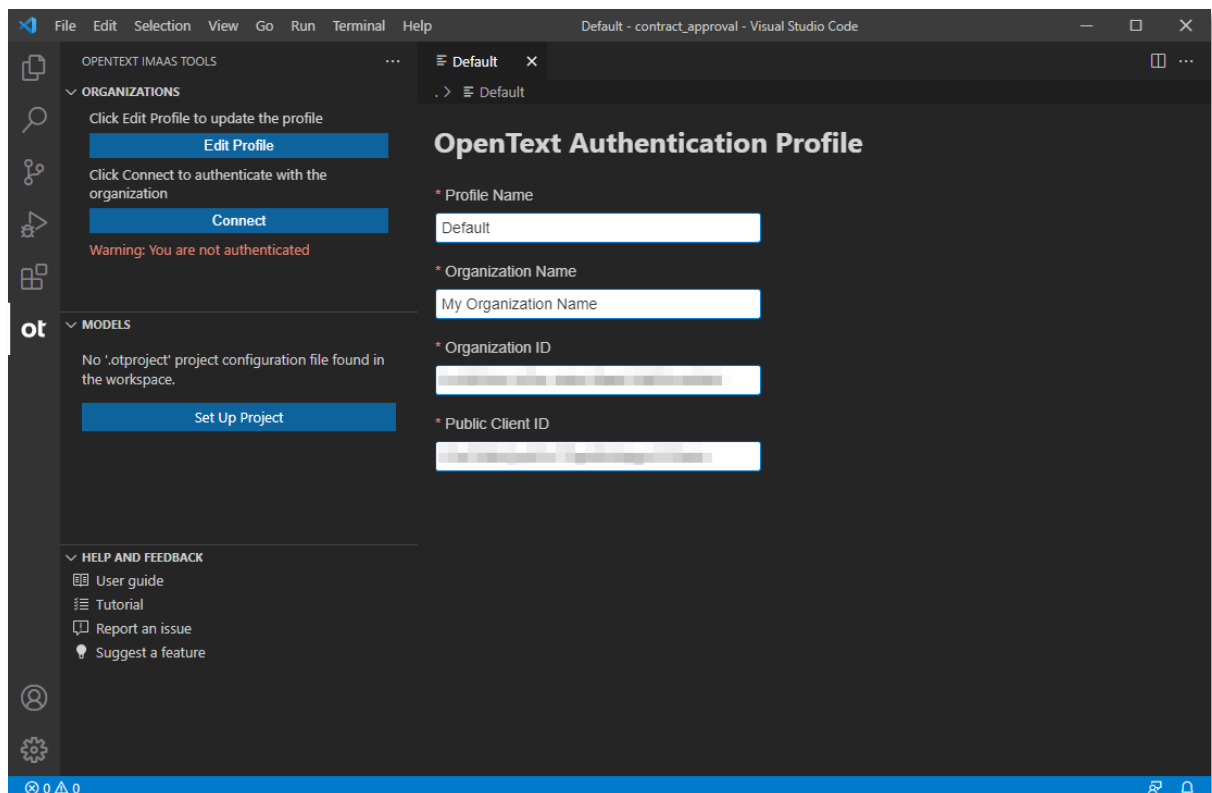
- Profile Name:** you can leave the **Default** value here
- Organization Name:** the name you chose for your organization
- Organization ID:** copy/paste the value of the **organization_id** JSON property from the `ot2_config_<organization name>.json` organization config file, which you previously opened in a text editor
- Public Client ID:** copy/paste the value of the **client_public_id** JSON property from the `ot2_config_<organization name>.json` organization config file, which you previously opened in a text editor



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



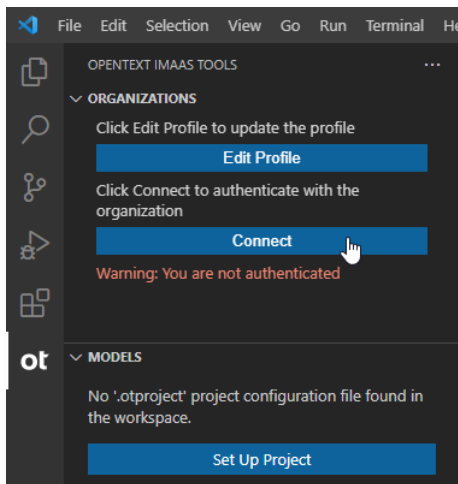
The **Default** tab now indeed indicates that the authentication profile configuration has been saved, as it no longer displays as unsaved.



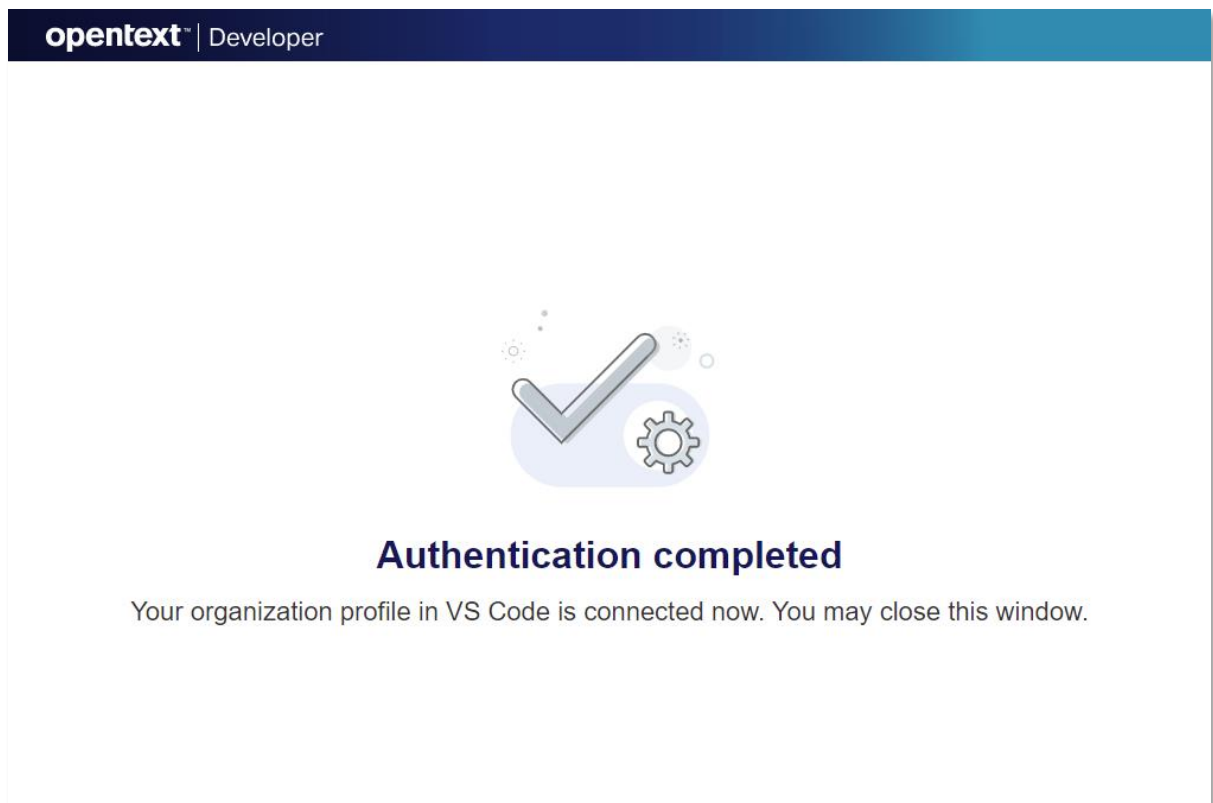
REMARK:

Note that all OpenText IMaaS Tools configuration artifacts, such as the previously set up organization connection, but also the project set up and different model configurations, use the standard VS Code file saving functionality. I.e.: to save your changes to any IMaaS configuration artifact, you can always press **Ctrl+S** on your keyboard (for Windows systems) or use the **Save** menu entry from the **File** menu. You will also be presented with a “Do you want to save the changes you made to ...?” dialog box when you try to close an unsaved configuration artifact.

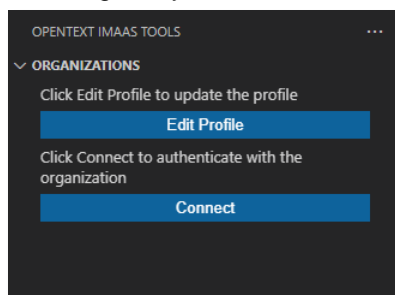
- You can now close the **OpenText Authentication Profile** form and click **Connect** from the **ORGANIZATIONS** section in the **OpenText IMaaS Tools** view to test the newly configured connection to your developer organization.



Fill your OpenText Connect account **username (email address)** and **password** and click **Sign in**.



The “Warning: You are not authenticated” warning message should now no longer appear, indicating that you indeed have successfully authenticated with your configured organization.



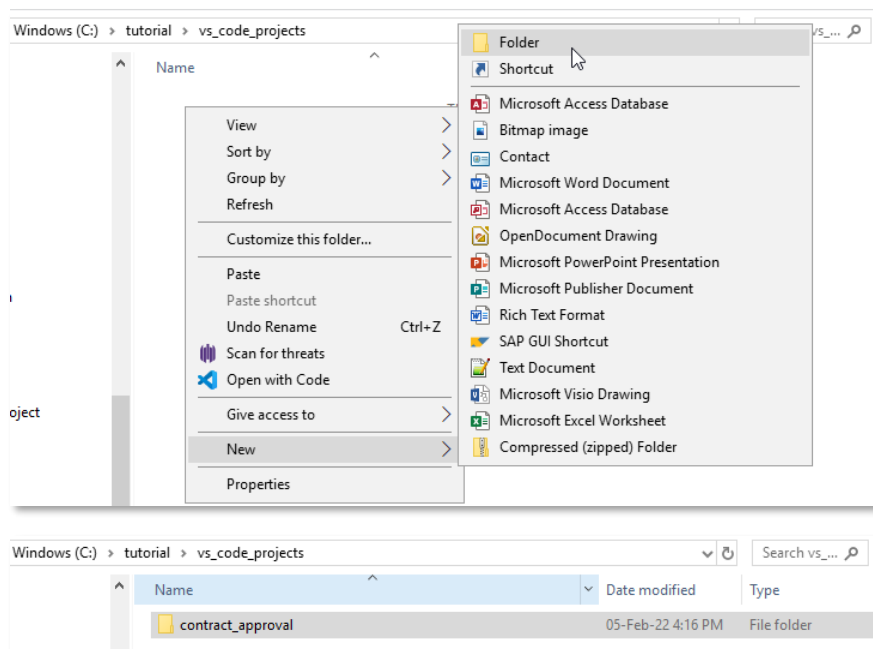
3.3 [15'] Creating an OpenText project

During this exercise you will create the folder on your file system in which you will be building your application. You will also set up your OpenText project. This includes filling the different IMaaS application properties, which will be used to create the **Contract Approval** application in your developer organization when deploying the project for the first time. It is required to set up an OpenText project before you can start building your IMaaS application components (or models).

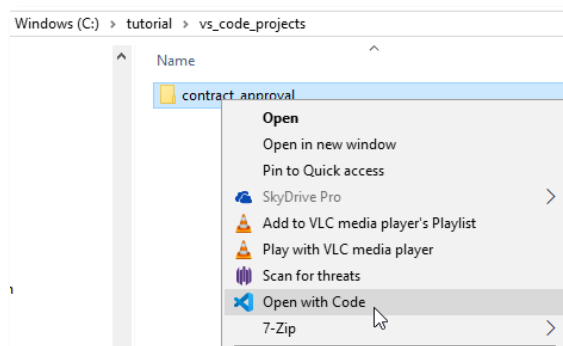
Once you are done with this section, you will have set up your OpenText project, and you are ready to start creating the Contract Approval (CMS) namespace.

To create an OpenText project for your **Contract Approval** application, proceed as follows:

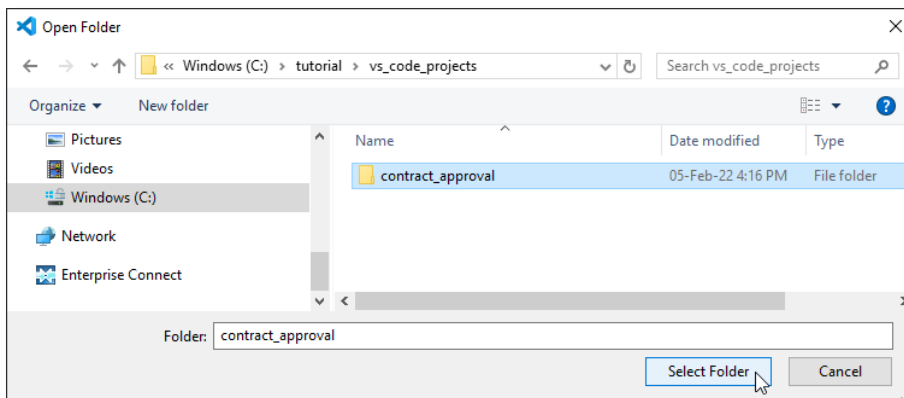
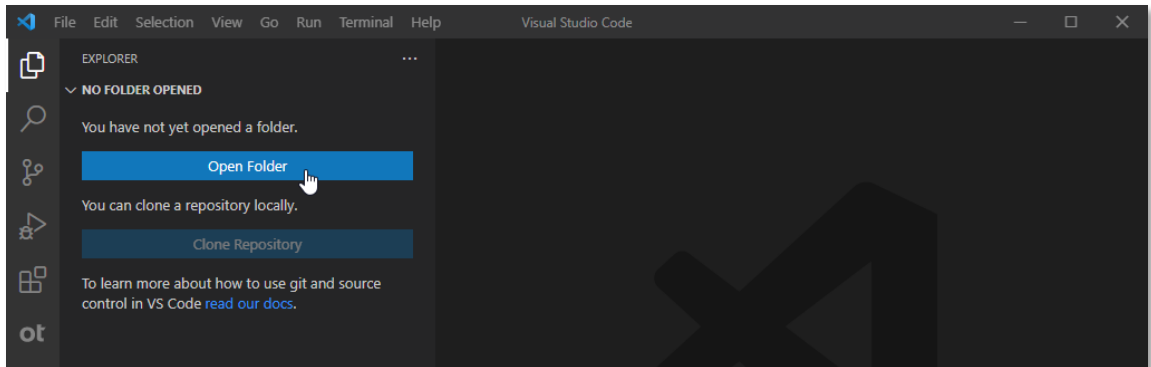
- Use your system's file system explorer (example shows Windows File Explorer) to create a new **contract_approval** folder.



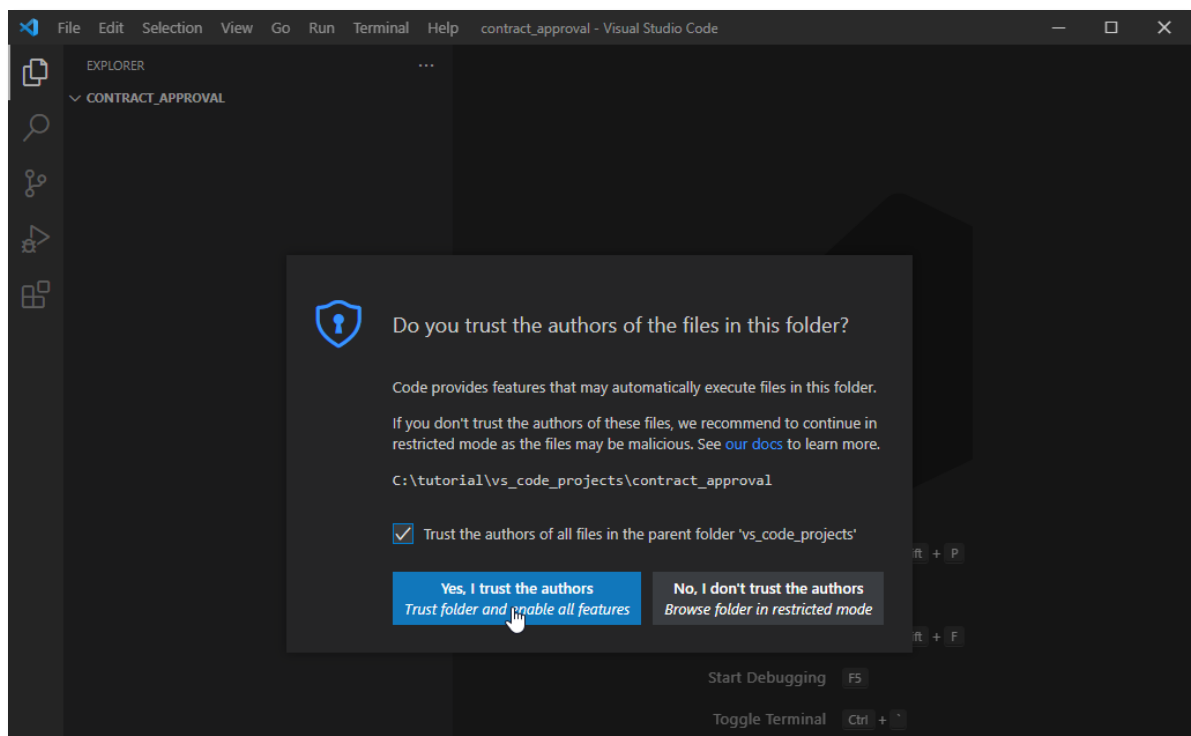
- Choose one of the following two options to open the Contract Approval project folder in VS Code:
 - **OPTION 1:**
If you are using Windows (and you followed the VS Code installation instructions exactly as described in this tutorial), in the Windows File Explorer, right-click the newly created **contract_approval** folder and select the **Open with Code** contextual menu item to open the Contract Approval project folder in VS Code.

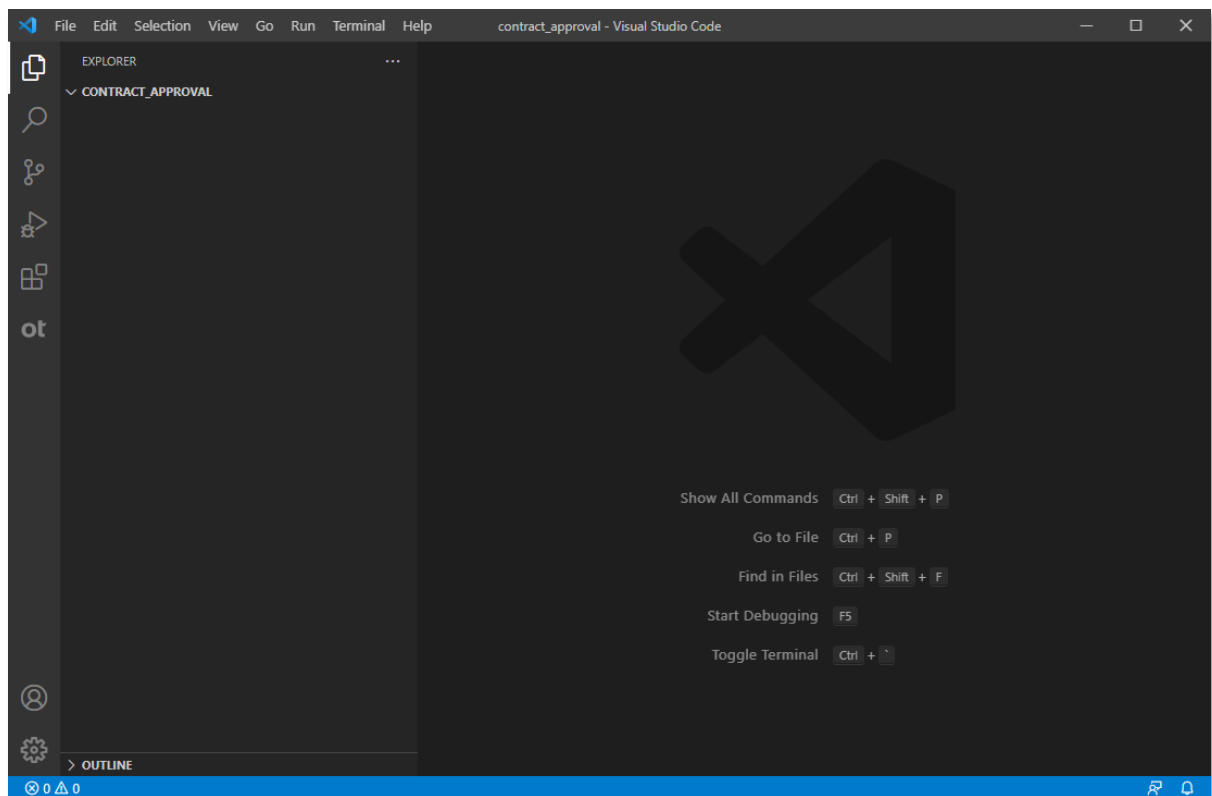


- **OPTION 2 (should work with all Operating Systems):**
Alternatively (if the contextual menu option is not available), you can use VS Code's **Open Folder** button from the **Explorer** view in the Activity Bar.

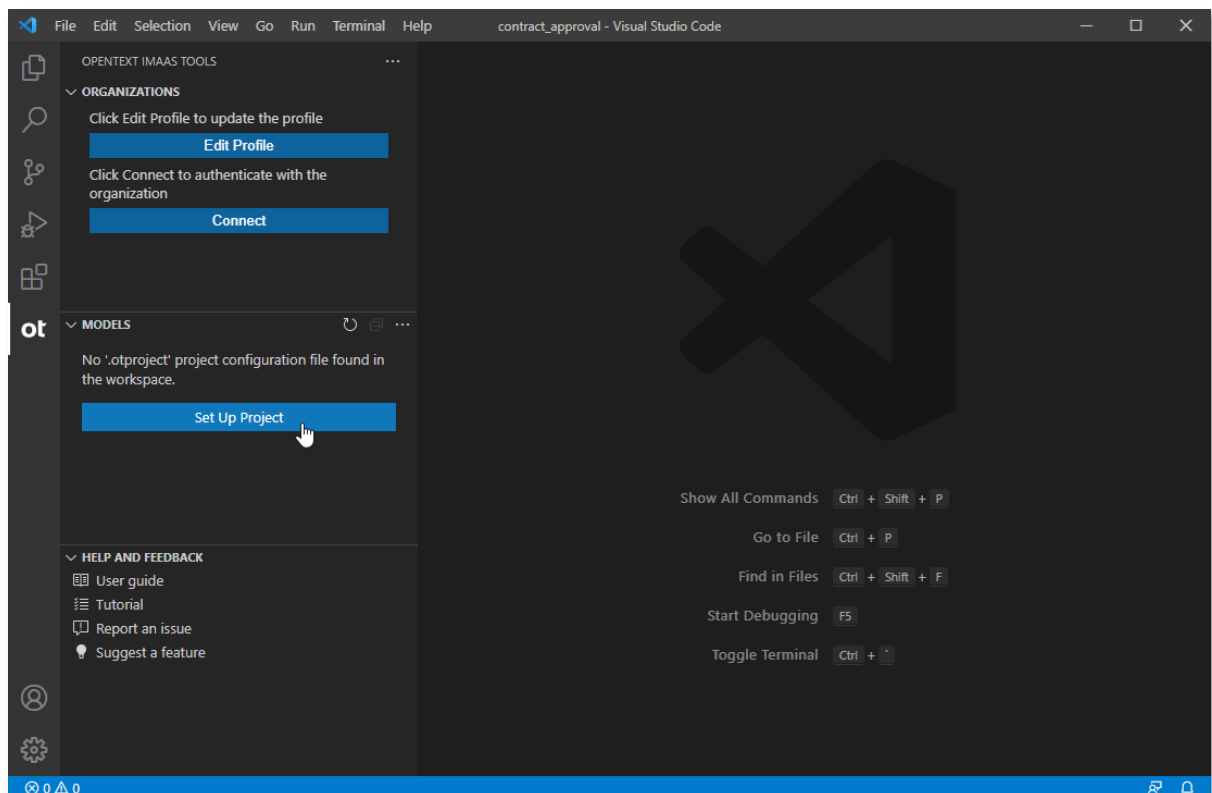


- Once the **contract_approval** folder is open in VS Code, choose to **Trust the authors of all files in the parent folder '...'** and click the **Yes, I trust the authors** button.



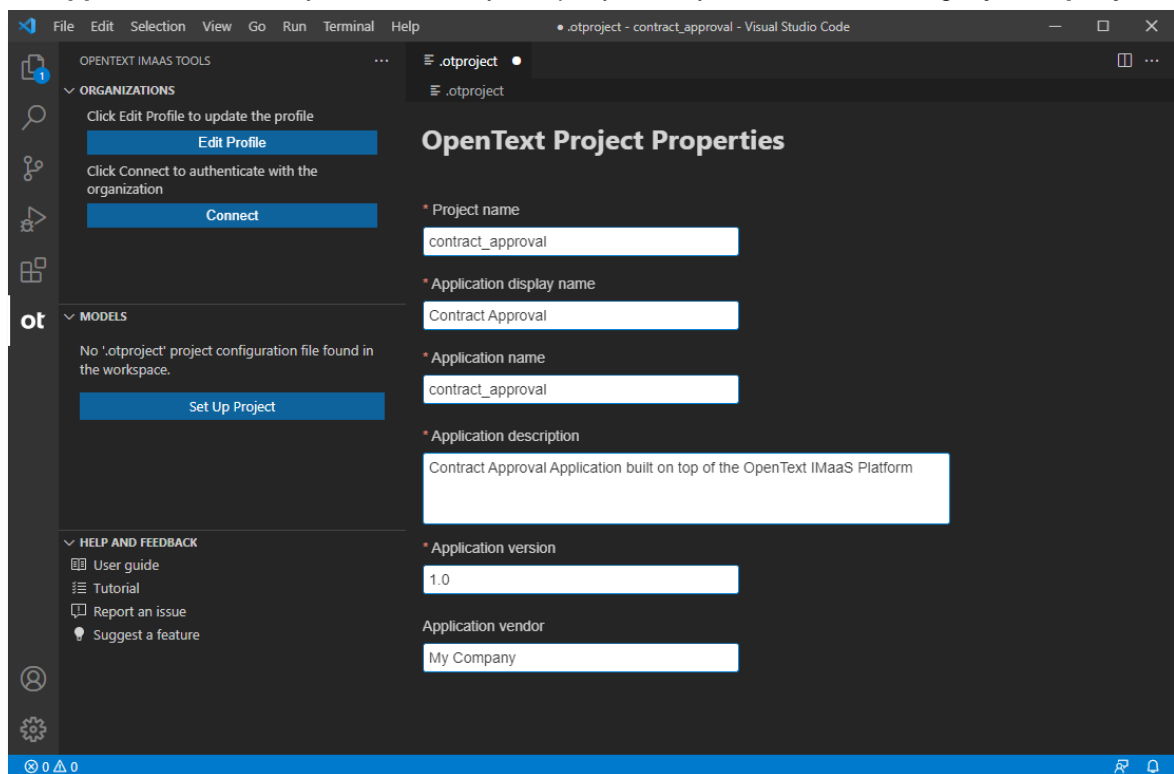


- Although you can now start creating files from VS Code in your Contract Approval application project folder, you still need to set up the OpenText project to be able to start building models. To set up the OpenText project, switch to the **OpenText IMaaS Tools** view and click the **Set Up Project** button.



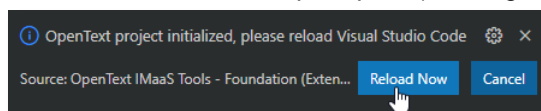
Fill the **OpenText Project Properties** as follows:

- **Project name:** the project name has been automatically populated from the project folder name (**contract_approval**) and does not require changing
- **Application display name:** **Contract Approval**
- **Application name:** when filling the application display name, the system will automatically populate the application name and you can leave the generated **contract_approval** value
- **Application description:** **Contract Approval Application built on top of the OpenText IMaaS Platform**
- **Application version:** **1.0**
- **Application vendor:** you can use any company name you like; we are using **My Company**

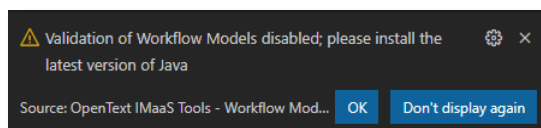


You can now save and close the **OpenText Project Properties** form.

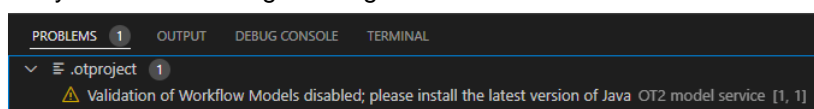
Click **Reload Now** when prompted (message on the bottom right of VS Code).



Depending on whether or not you have Java installed on your system, you will get the following warning message popping up in the bottom right of VS Code (when Java is not installed).



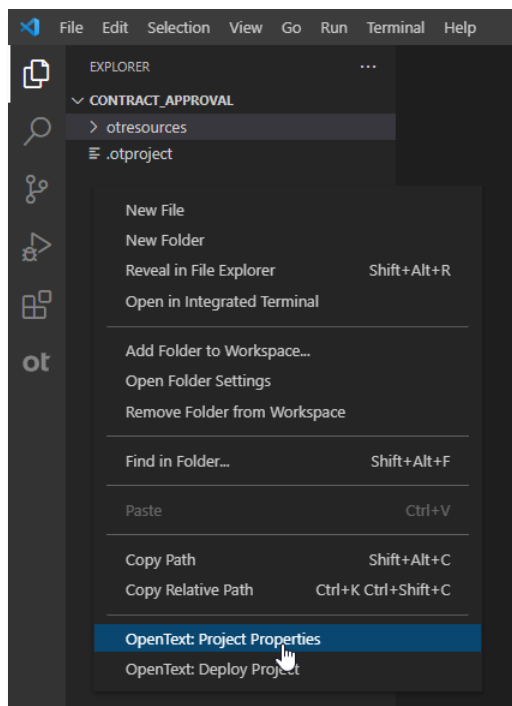
You can choose to simply close this message (clicking **OK**) or to not display it again (clicking **Don't display again**). Whichever option you chose, as long as Java is not installed you will always find the warning message in the **PROBLEMS** tab:



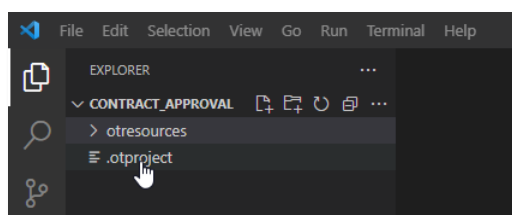
- If you previously got the **Validation of Workflow Models disabled; please install the latest version of Java** warning message, please continue with the next steps to install Java on your system.

If Java is already installed (you did not get any warning message), you can directly start with the next exercise section ([Creating a CMS namespace](#)) as your OpenText project has been set up and you are ready to start creating models.

Before you do though, note that 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:



Or



- Although you can choose to install any Java distribution (OpenJDK, Oracle or other) on your system, for your convenience we are providing you with the steps to install the Oracle Java version. To install Java on your system, first navigate to <https://www.oracle.com/java/technologies/downloads>.

Java 17 available now

Java 17 LTS is the latest long-term support release for the Java SE platform. JDK 17 binaries are free to use in production and free to redistribute, at no cost, under the [Oracle No-Fee Terms and Conditions](#).

JDK 17 will receive updates under these terms, until at least September 2024.

Java SE Development Kit 17.0.2 downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications and components using the Java programming language.

The JDK includes tools for developing and testing programs written in the Java programming language and running on the Java platform.

Linux **macOS** **Windows**

Product/file description	File size	Download
x64 Compressed Archive	171.34 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.zip (sha256 [2])
x64 Installer	152.43 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.exe (sha256 [2])
x64 MSI Installer	151.32 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.msi (sha256 [2])

At the time of writing of this tutorial, Java 17 LTS and the corresponding JDK 17 binaries are free to use in production and free to redistribute, at no cost, under [Oracle No-Fee Terms and Conditions](#) (feel free to read this for more information).

In that context we recommend you install the **Java SE Development Kit 17** version that corresponds with your OS on your system (currently 17.0.2).

For this tutorial we will be using the **x64 MSI Installer** for Windows. If you are also on a Windows machine, feel free to use the https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.msi link.

Java SE Development Kit 17.0.2 downloads

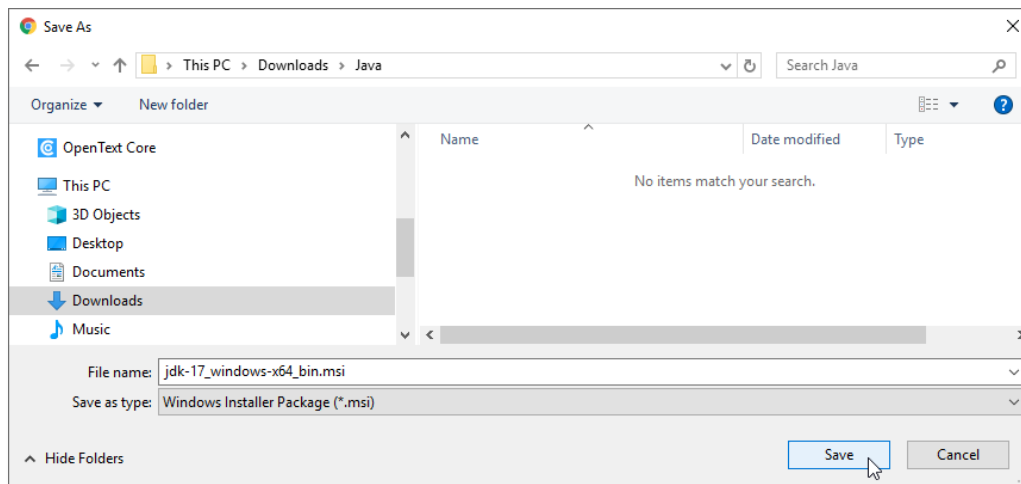
Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications and components using the Java programming language.

The JDK includes tools for developing and testing programs written in the Java programming language and running on the Java platform.

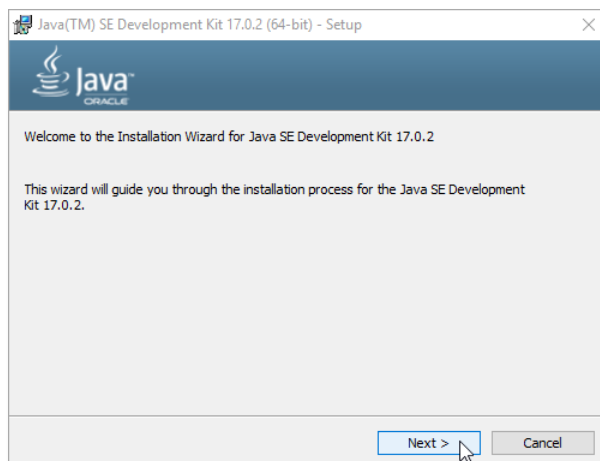
Linux **macOS** **Windows**

Product/file description	File size	Download
x64 Compressed Archive	171.34 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.zip (sha256 [2])
x64 Installer	152.43 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.exe (sha256 [2])
x64 MSI Installer	151.32 MB	https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.msi (sha256 [2])

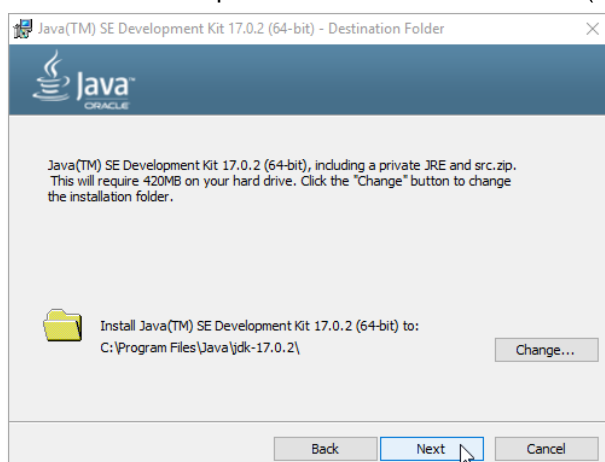
Save and run the **jdk-17_windows-x64_bin.msi** installer.



Choose **Next >** to start with the Java SE Development Kit installation process.



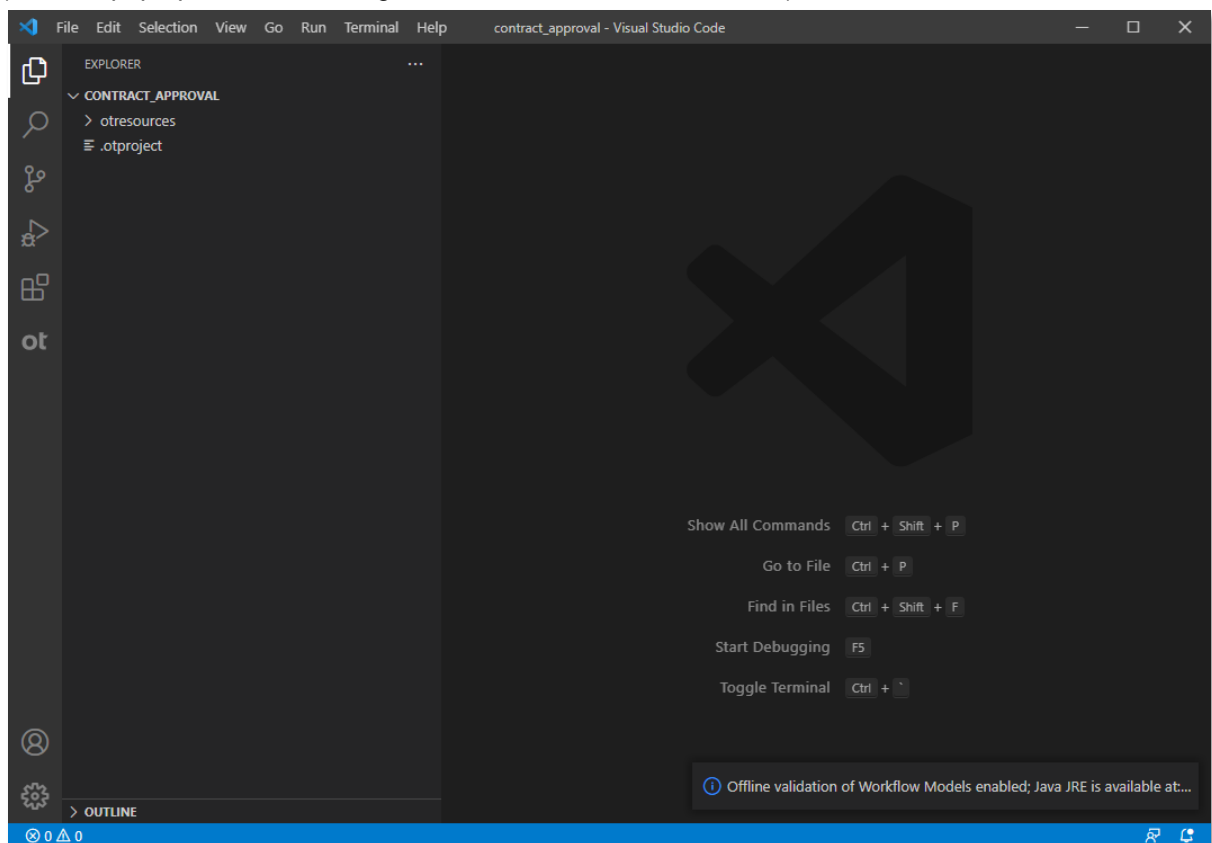
Click **Next** to accept the default installation folder (feel free to change it, if needed).



Once the installation is complete, click **Close** to exit the installation wizard.



Close and reopen VS Code, and you should now see that the **Validation of Workflow Models disabled; please install the latest version of Java** warning message does no longer appear (not as a pop up in the bottom right and not in the **PROBLEMS** tab).



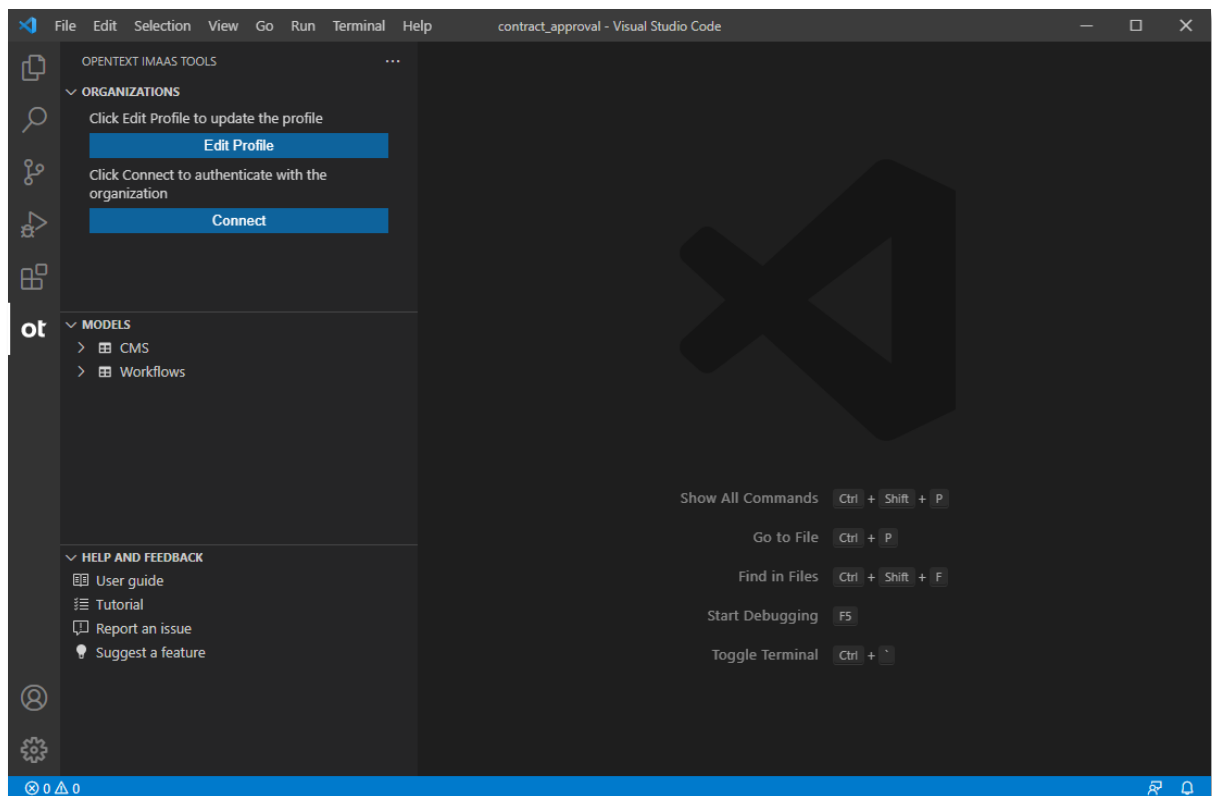
3.4 [10'] Creating a CMS namespace

During this exercise you will create the **Contract Approval** CMS namespace model. A namespace allows grouping the different CMS type and trait definitions together (e.g.: within the context of an application). For more information on CMS namespaces, you can refer to the [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.

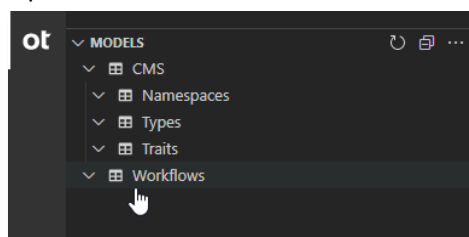
Once you are done with this section, you will have created the CMS namespace that corresponds with your Contract Approval application, and you are ready to start creating the Approval (CMS) trait definition.

To create the **Contract Approval** CMS namespace, proceed as follows:

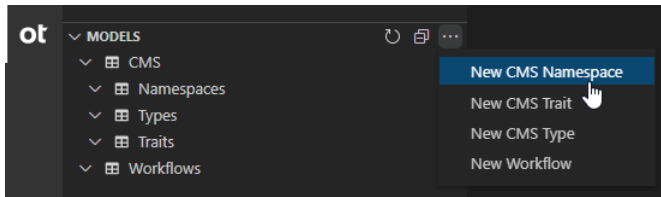
- In VS Code, switch to the **OpenText IMaaS Tools** view.



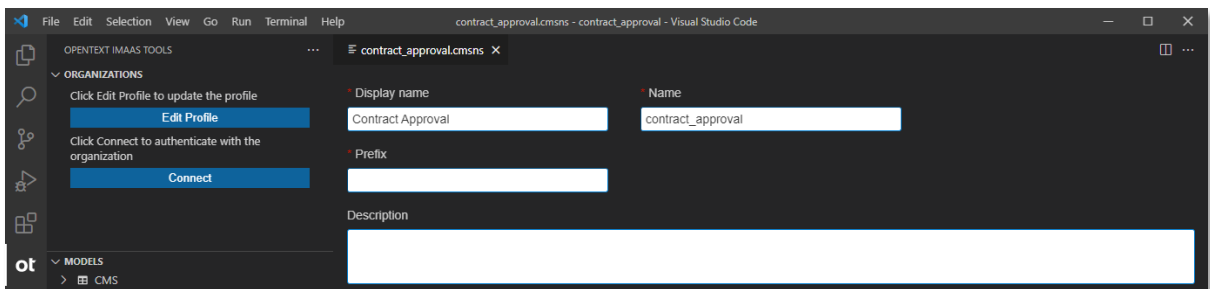
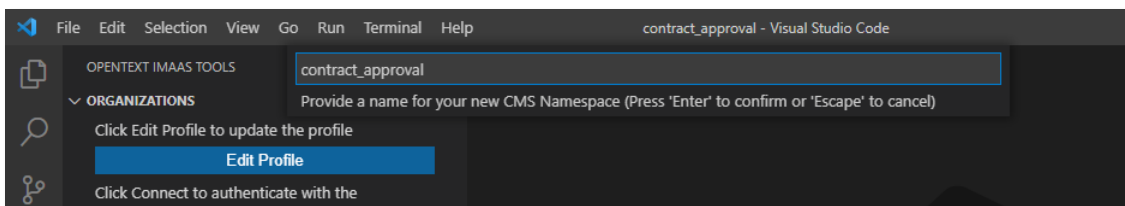
As you can see, since your OpenText project has been set up, the **MODELS** section shows a tree view that allows exploring the different models in your IMaaS application project. Feel free to expand all branches.



- To create a new CMS namespace, click [...] at the top right of the **MODELS** section and select **New CMS Namespace**.

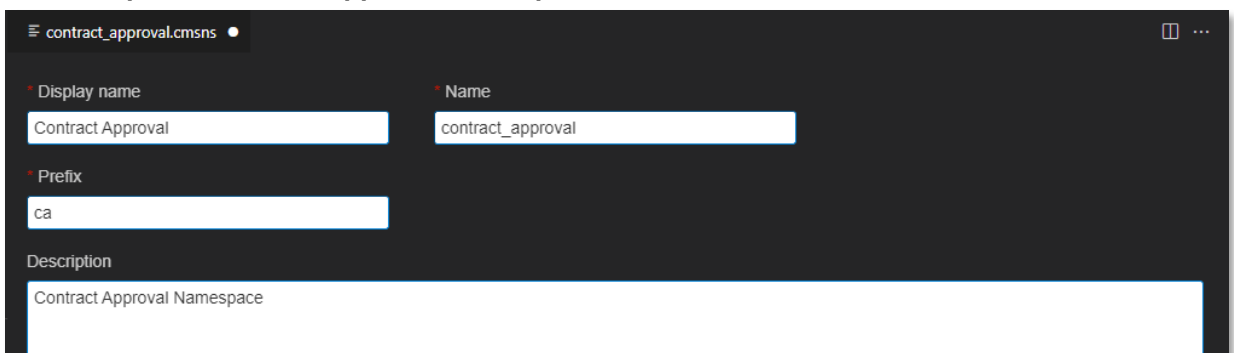


In the input box that appears at the top of VS Code, fill **contract_approval** as the name for the new CMS namespace and press **Enter**.



Fill the CMS namespace properties as follows:

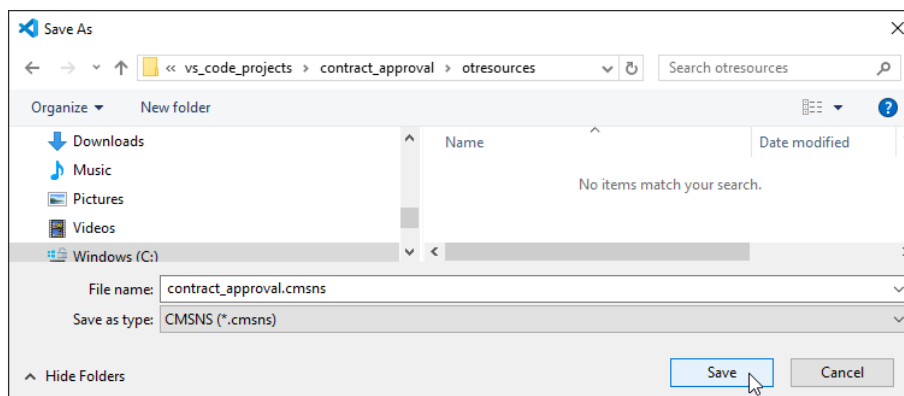
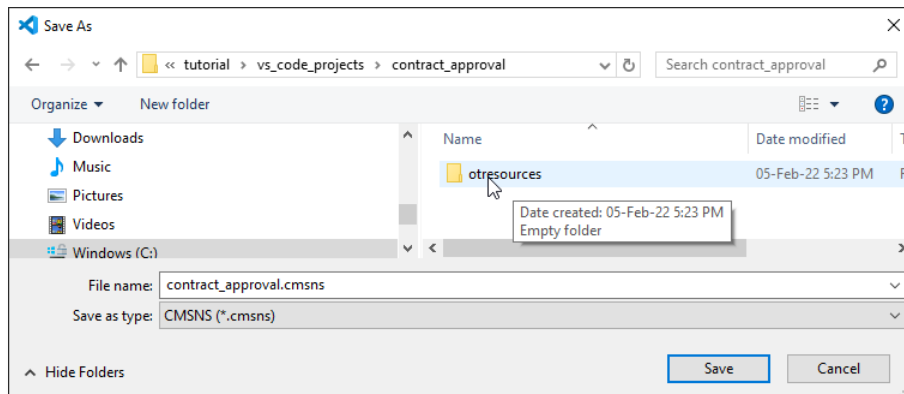
- **Display name:** the display name is the user-friendly name for the CMS namespace; this does not have to be unique, and it has been automatically populated for your convenience based on the previously chosen CMS namespace name (the model file name); you should best leave the value to be **Contract Approval** as it nicely aligns with the model file name and model name
- **Name:** the name is the technical name for the CMS namespace; this has to be unique (within your developer tenant), and it has been automatically populated for your convenience based on the previously chosen CMS namespace name (the model file name); you should best leave the value to be **contract_approval** as it nicely aligns with the model file name and model display name
- **Prefix:** the prefix is the prefix representing the CMS namespace (used in system naming of CMS traits and CMS types that are within that namespace); this has to be unique (within your developer tenant); please fill **ca** as its value
- **Description:** **Contract Approval Namespace**



REMARK:

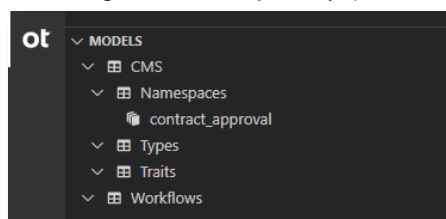
Note that for the current CMS namespace creation exercise we are using the [...] menu and the corresponding (**New CMS Namespace**) model creation menu item from the **MODELS** section in the **OpenText IMaaS Tools** view. There are two more ways to create models and those will be illustrated in the next tutorial exercises.

You can now save and close the **Contract Approval** CMS namespace model. Saving will pop up a **Save As** file saving dialog box. Make sure to select the **otresources** folder as target folder, leave the file name as is (**contract_approval.cmnsns**) and click **Save**.

**IMPORTANT:**

The **otresources** folder is the model folder and it was automatically generated during the project setup. It is important you create all models inside this **otresources** folder, as the system will refuse to store models anywhere else.

The model explorer tree in the **MODELS** section should now show your new **contract_approval** CMS namespace under **/CMS/Namespaces**. The model explorer shows the different models according to their unique key (which in context of a CMS namespace is the name property).



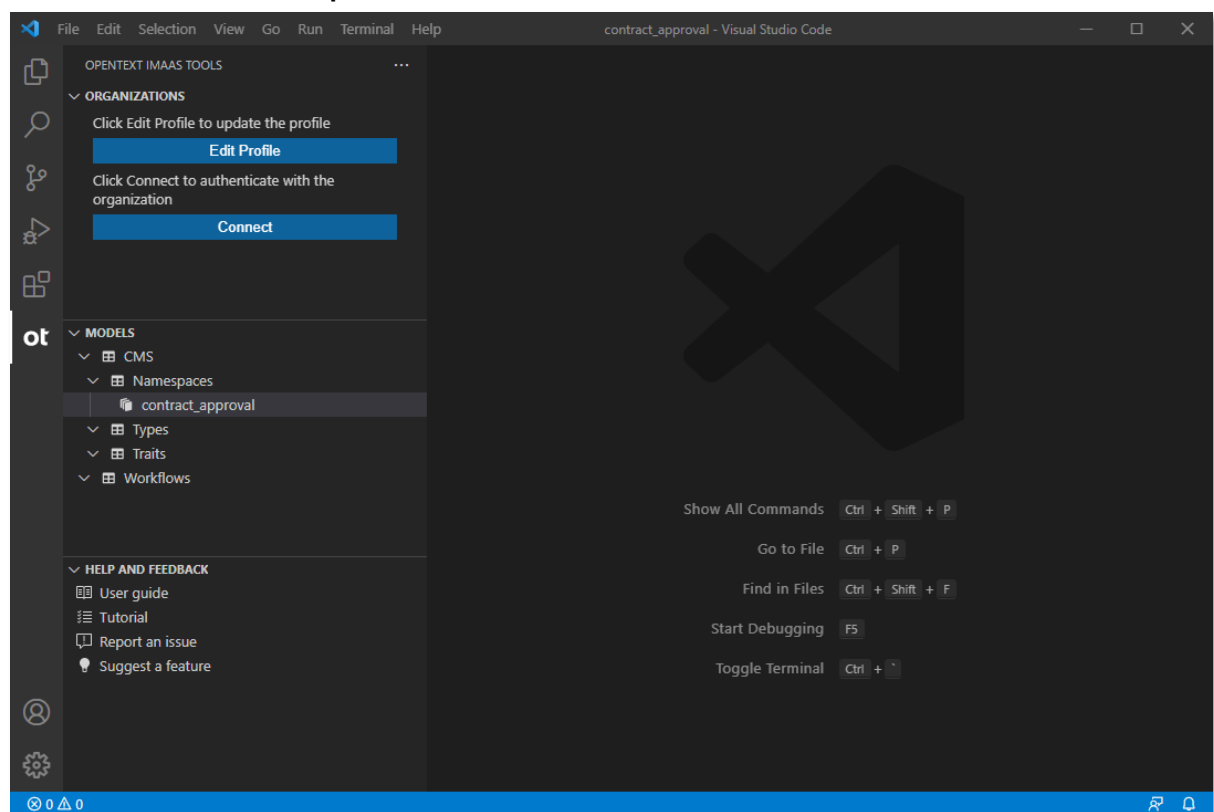
3.5 [15'] Creating a CMS trait definition

During this exercise you will create the **Approval** CMS trait definition model. A trait definition allows grouping several attributes into one more complex multi-attribute property. Trait instances can be dynamically added to a CMS type instance as part of the business process when using the application, but they can also be made mandatory as a required trait in a CMS type definition, so that they must always be added when creating a new CMS type instance. For the purpose of this tutorial, we will be using the concept of mandatory traits to represent the different approval steps on a contract (hence the Approval name of the CMS trait definition). For more information on CMS traits (definitions and instances), you can refer to the [Define a namespace, trait and "FILE" document type](#) and [Create instances using custom type with trait](#) sections in the Content Metadata Service product documentation or the [Trait](#) resource documentation in the Content Metadata Service API reference.

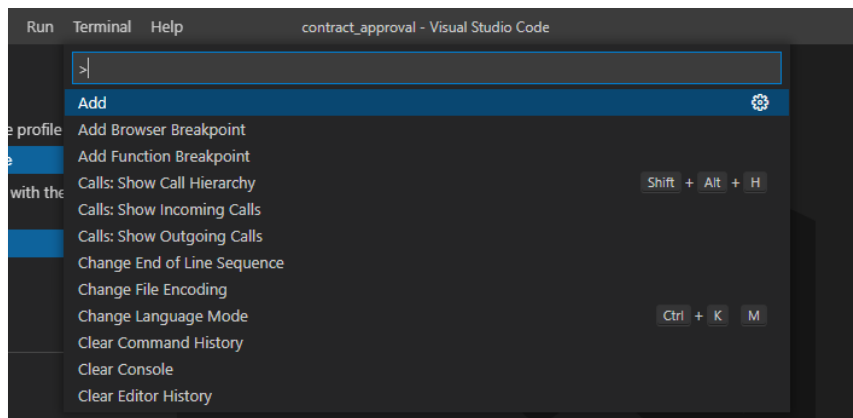
Once you are done with this section, you will have created the CMS trait definition that corresponds with your Contract Approval application's approval steps, and you are ready to start creating the Contract (CMS) type definition.

To create the **Approval** CMS trait definition, proceed as follows:

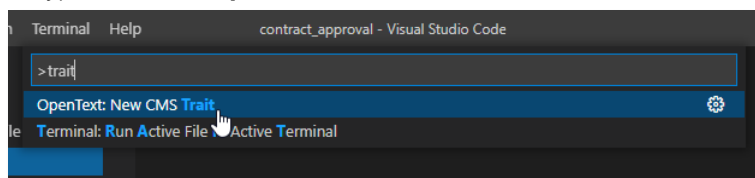
- In VS Code, switch to the **OpenText IMaaS Tools** view.



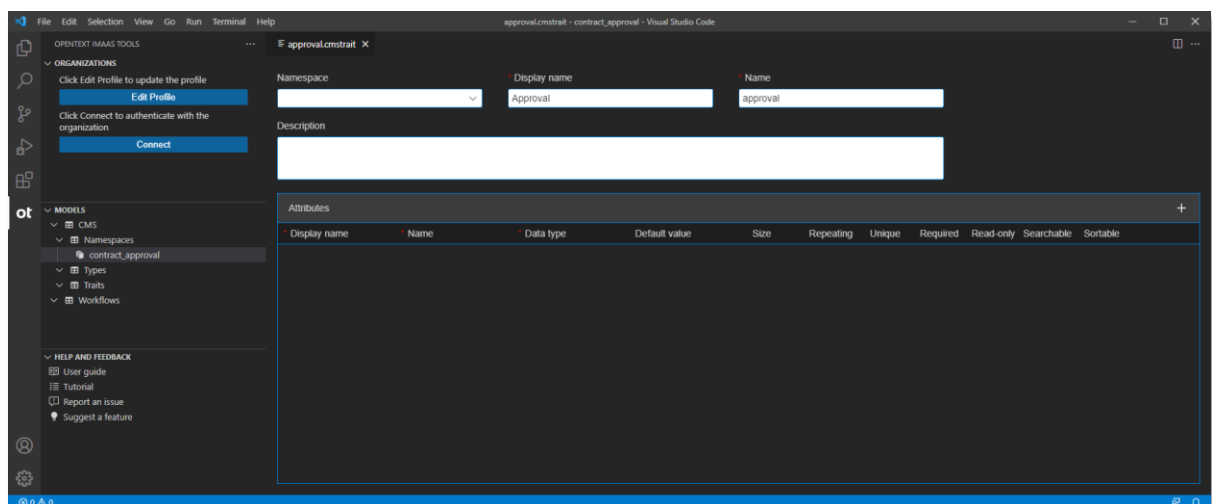
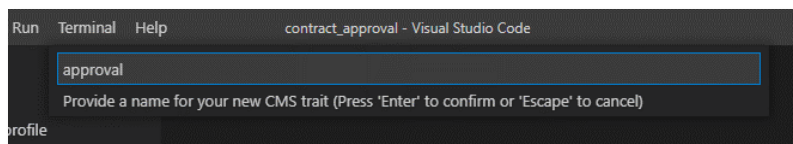
- To create a new CMS trait definition, press **F1** on your keyboard (or **Ctrl+Shift+P** if F1 doesn't work). This opens the Command Palette at the top of VS Code.



In the Command Palette, type **trait** and you should see the command list being filtered to show the **OpenText: New CMS Trait** command near the top (in the case of this tutorial it's the first entry). Select the **OpenText: New CMS Trait** command.



In the input box that appears, fill **approval** as the name for the new CMS trait definitions and press **Enter**.



Fill the CMS trait definition properties as follows:

- **Namespace:** the namespace to which this CMS trait definition belongs; as you previously created the **contract_approval** namespace, you can now select it as the namespace for your CMS trait definition


REMARK:

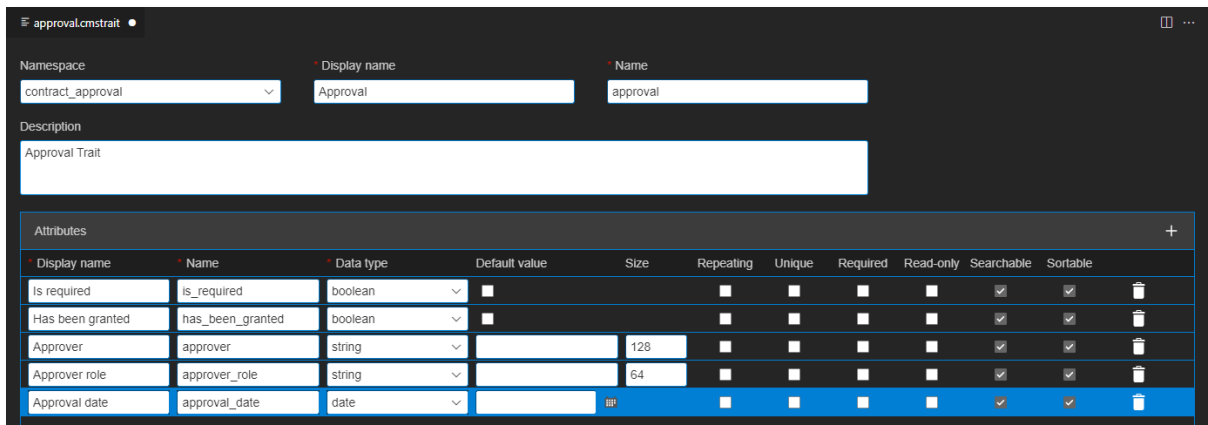
Note that the OpenText IMaaS Tools dynamically update the different model reference lists. In the context of this CMS trait definition, the list of available namespaces is dynamically updated when you add namespaces, but e.g., in context of CMS type definitions, the list of available traits and parent types will also update, based on the available CMS trait and type definitions.






- **Display name:** the display name is the user-friendly name for the CMS trait definition; this does not have to be unique, and it has been automatically populated for your convenience based on the previously chosen CMS trait definition name (the model file name); you should best leave the value to be **Approval** as it nicely aligns with the model file name and model name
- **Name:** the name is the technical name for the CMS trait definition; this has to be unique in context of the selected namespace (and the combination of namespace and trait definition name needs to be unique within your developer tenant), and it has been automatically populated for your convenience based on the previously chosen CMS trait definition name (the model file name); you should best leave the value to be **approval** as it nicely aligns with the model file name and model display name
- **Description: Approval Trait**
- **Attributes:** the attributes list defines the different attribute definitions of the CMS trait definition; each attribute definition has the following properties:
 - **Display name:** the display name is the user-friendly name for the attribute definition; this does not have to be unique, but this is recommended (to avoid confusion)
 - **Name:** the name is the technical name for the attribute definition; this has to be unique within the CMS trait definition, and it gets automatically populated for your convenience based on the display name you fill
 - **Data type:** the data type of the attribute; this is a pick list (bigint, boolean, date, double, integer, string and uuid)
 - **Default value:** you can assign a default value for the attribute (i.e.: the value that gets automatically assigned to the attribute when creating a new instance of the CMS trait definition); whether it is possible to assign a default value and how to assign it depends on the chosen data type (e.g.: a date data type gets a date picker, and a uuid data type doesn't allow for a default value)
 - **Size:** the size property only applies to the string data type and can thus only be chosen when picking the string data type; it represents the maximum length constraint for the string attribute
 - **Repeating:** whether or not the attribute is multi-valued (can have multiple values)
 - **Unique:** whether or not the attribute needs to be unique across all instances of the CMS trait definition
 - **Required:** whether or not the attribute must be filled upon creation
 - **Read-only:** whether or not the attribute can be modified after creation
 - **Searchable:** whether or not the attribute can be filtered against when performing a search
 - **Sortable:** whether or not the attribute can be used to sort a search result

In the context of this **Approval** CMS trait definition, the different attribute definitions represent an approval step, and the below table describes each attribute definition 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

Note that to add an attribute definition to a CMS trait definition, you need to use the  on the top right of the **Attributes** list.

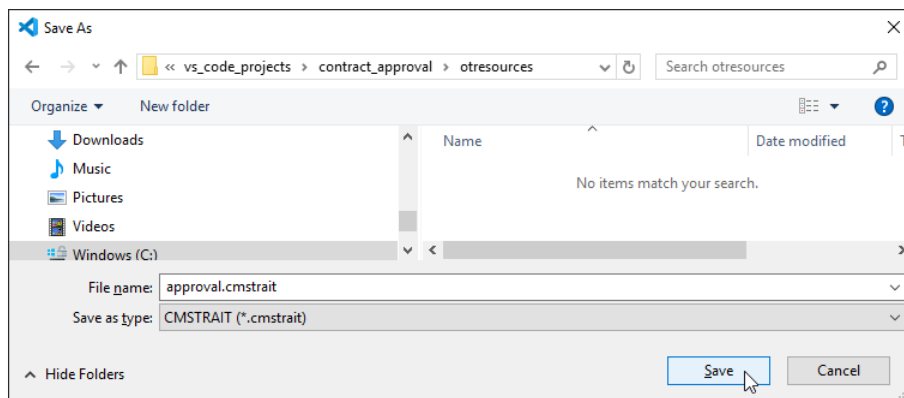
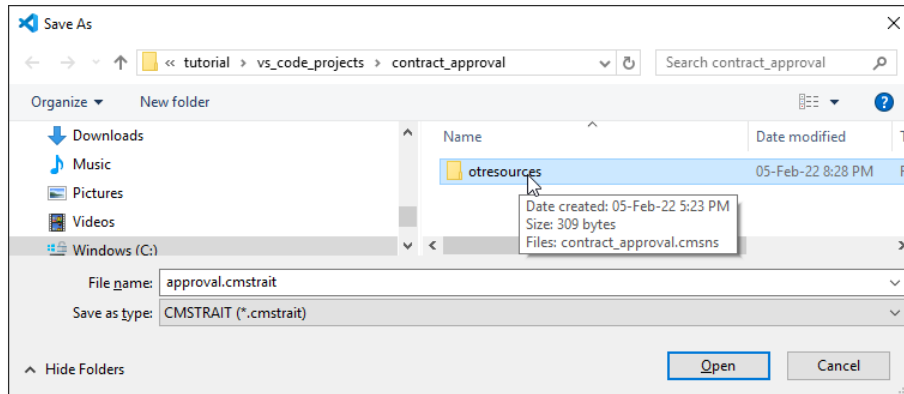


Display name	Name	Data type	Default value	Size	Repeating	Unique	Required	Read-only	Searchable	Sortable	
Is required	is_required	boolean			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Has been granted	has_been_granted	boolean			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Approver	approver	string		128	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Approver role	approver_role	string		64	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Approval date	approval_date	date			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

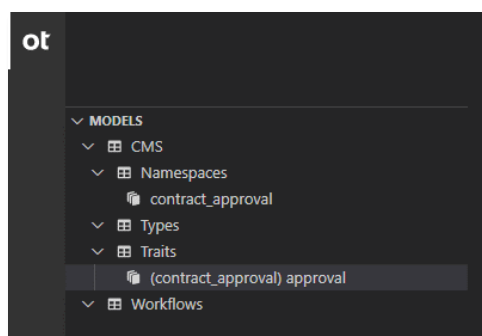
REMARK:

Note that for the current CMS trait definition creation exercise we are using the VS Code Command Palette and the corresponding (**OpenText: New CMS Trait**) model creation command. This is the second way to create models. There is one more way that remains, and this will be illustrated in the next tutorial exercise.

You can now save and close the **Approval** CMS trait definition model. Saving will pop up a **Save As** file saving dialog box. Make sure to select the **otresources** folder as target folder, leave the file name as is (**approval.cmstrait**) and click **Save**.



The model explorer tree in the **MODELS** section should now show your new **(contract_approval) approval** CMS trait definition under **/CMS/Traits**. The **contract_approval** value between brackets represents the CMS namespace to which the CMS trait definition belongs, as the model explorer shows the different models according to their unique key (which in context of a CMS trait definition is the combination of the namespace and name properties).



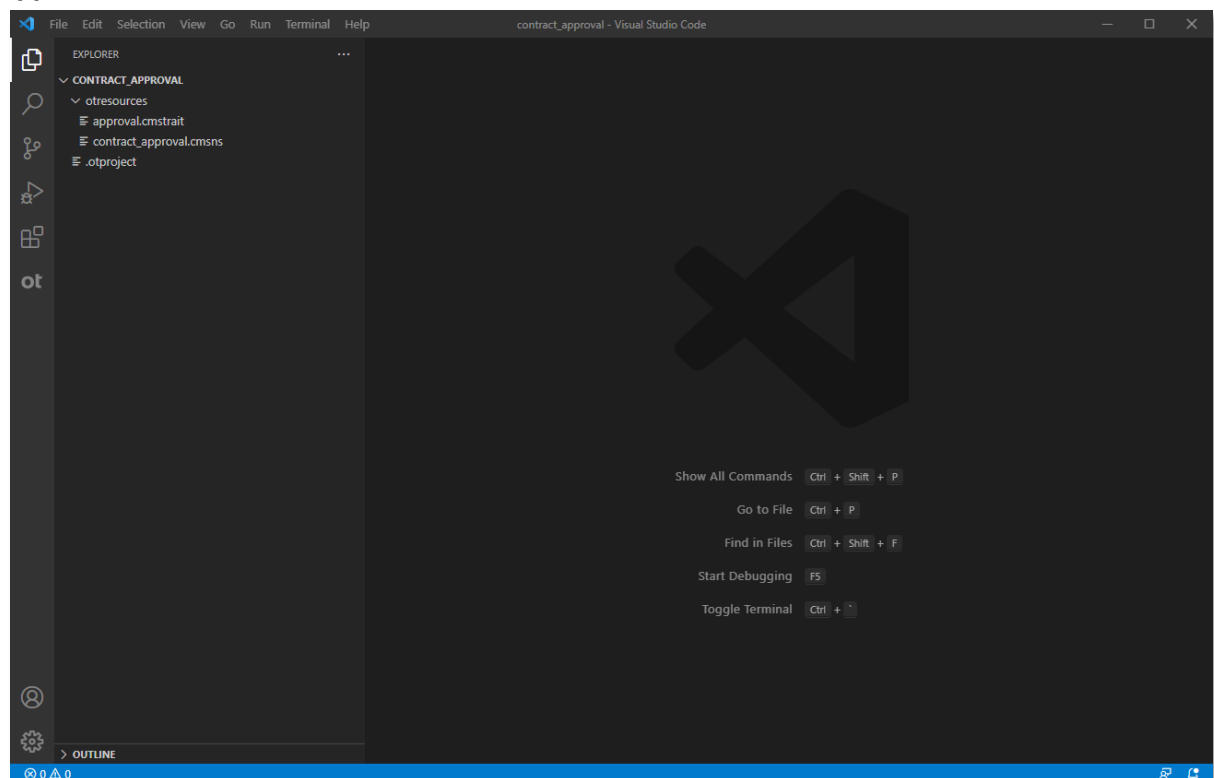
3.6 [20'] Creating a CMS file type definition

During this exercise you will create the **Contract** CMS type definition model. A type definition is the main component for building your application's (custom) data model. A type definition has its own attributes and required traits (i.e.: traits that are always added to the type instance upon creation). A CMS type definition can also be of two categories: file or folder. For the purpose of this tutorial, we will be creating two file type definitions and one folder type definition. This **Contract** CMS type definition is the first **file** type definition, and it will be the parent type of the Loan Contract CMS file type definition that we will create during the next exercise. For more information on CMS types (definitions and instances), you can refer to the [Define a namespace, trait and "FILE" document type](#) and [Create instances using custom type with trait](#) sections in the Content Metadata Service product documentation or the [Type](#) resource documentation in the Content Metadata Service API reference.

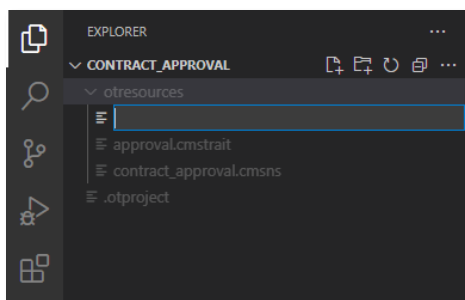
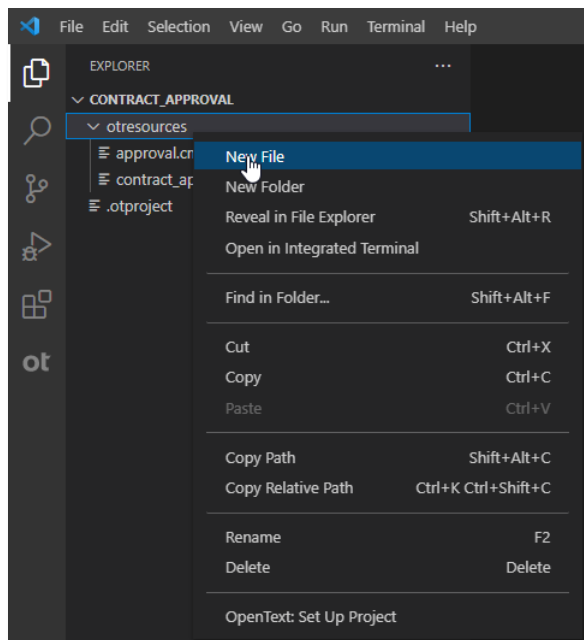
Once you are done with this section, you will have created the **Contract** CMS type definition that corresponds with your Contract Approval application's (parent) contract file type, and you are ready to start creating the (child) Loan Contract (CMS) type definition.

To create the **Contract** CMS type definition, proceed as follows:

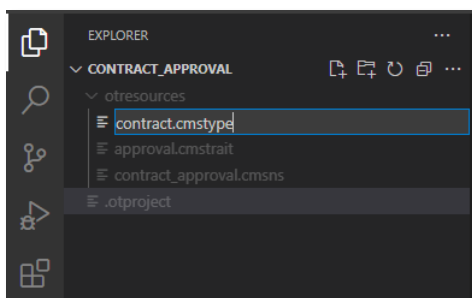
- In VS Code, switch to the **Explorer** view.
If you expand the **otresources** (model) folder under your **contract_approval** application root folder, you can indeed see the previously created **contract_approval** CMS namespace and **approval** CMS trait.

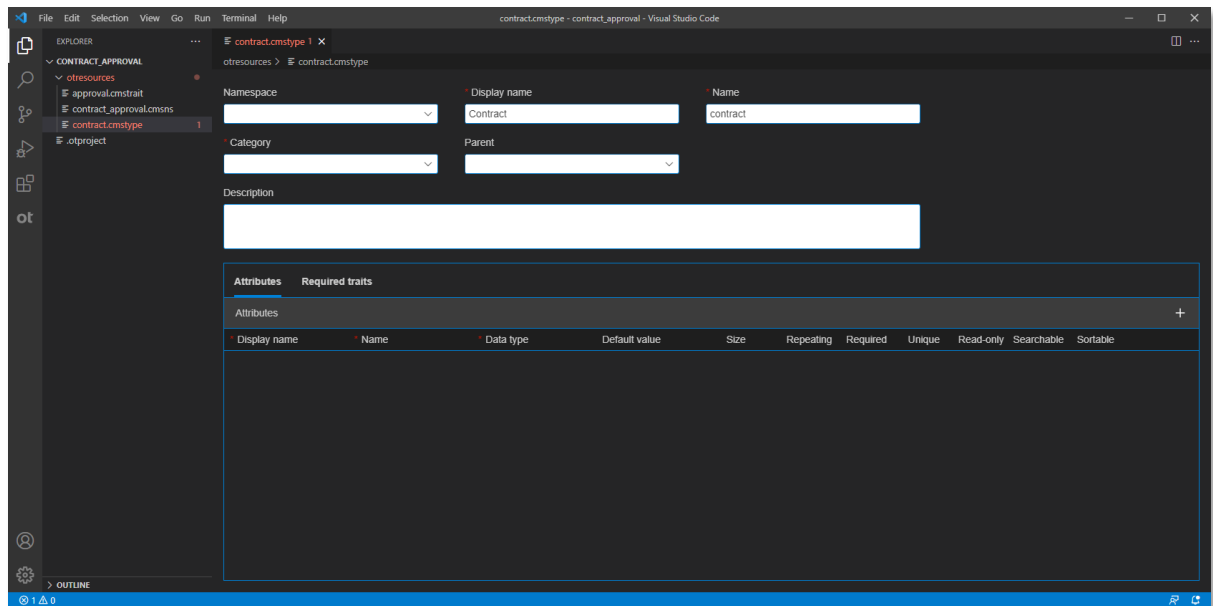


- To create a new CMS type definition, right-click the **otresources** (model) folder and select **New File**.



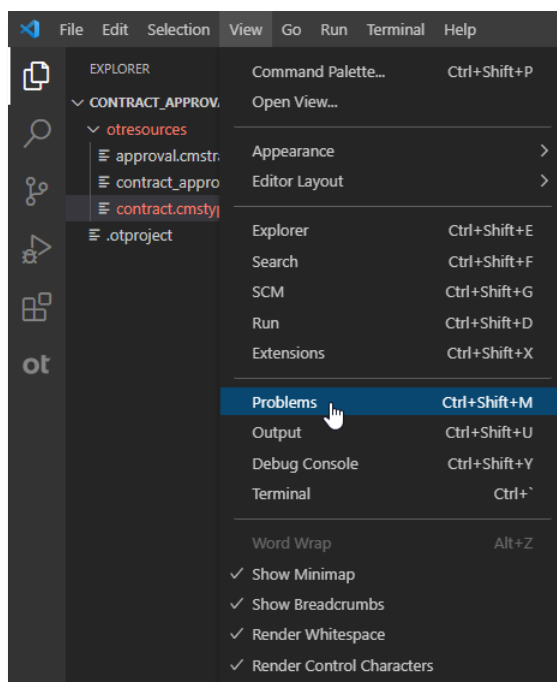
In the input box that appears under the **otresources** folder, type the new file name for you CMS type definition, **contract.cmstype**, and press **Enter** to confirm.



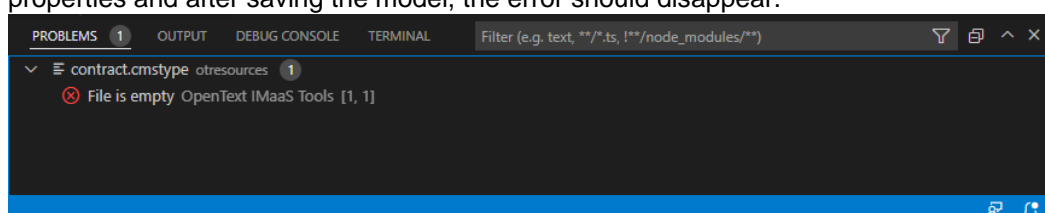


As you can see from both the red color of the file explorer tree and the CMS type definition model's tab, there seems to be a problem. Let's open the **PROBLEMS** tab to see what is happening.

Select **Problems** from the **View** menu.



The error message that displays is "File is empty", and this is indeed correct, as you created an empty model file via the file explorer tree. You will now fill the different CMS type definition model properties and after saving the model, the error should disappear.



Fill the CMS type definition properties as follows:

- **Namespace:** the namespace to which this CMS type definition belongs; select the **contract_approval** namespace
- **Display name:** the display name is the user-friendly name for the CMS type definition; this does not have to be unique, and it has been automatically populated for your convenience based on the previously chosen CMS type definition file name; you should best leave the value to be **Contract** as it nicely aligns with the model file name and model name
- **Name:** the name is the technical name for the CMS type definition; this has to be unique in context of the selected namespace (and the combination of namespace and type definition name needs to be unique within your developer tenant), and it has been automatically populated for your convenience based on the previously chosen CMS type definition file name; you should best leave the value to be **contract** as it nicely aligns with the model file name and model display name
- **Category:** the type category to which the CMS type definition belongs; this can be file or folder; the **Contract** CMS type definition is of the **file** category
- **Parent:** the parent CMS type definition for the type definition you are creating; the **Contract** CMS type has no parent
- **Description: Contract Type**
- **Attributes:** the attributes list defines the different attribute definitions of the CMS type definition; for a description of the attribute definition properties, please refer to the [previous exercise](#)


The below table describes each attribute definition 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

- **Required traits:** the required traits list defines the different mandatory traits for the CMS type definition; each required trait definition has the following properties:
 - **Instance name:** the instance name is the name of the required trait; this must be unique across the CMS type definition's required traits; in context of this tutorial where the required traits are all **Approval** traits, the instance name will represent the type of approval
 - **Trait name:** the trait name is the name of the selected CMS trait definition; in the context of this tutorial, all required traits will be **Approval** traits

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

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

Note that to add a required trait definition to a CMS type definition, you need to use the  on the top right of the **Required traits** list.

contract.cmstype 1

otresources > contract.cmstype

Namespace: contract_approval

Display name: Contract

Name: contract






Category: file

Parent:

Description: Contract Type

Attributes **Required traits**

Attributes

Display name	Name	Data type	Default value	Size	Repeating	Required	Unique	Read-only	Searchable	Sortable	
Requester email	requester_email	string		256	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Status	status	string		32	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Value	value	integer			<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Risk classification	risk_classification	integer			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Extracted terms	extracted_terms	string		256	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

contract.cmstype 1

otresources > contract.cmstype

Namespace: contract_approval

Display name: Contract

Name: contract




Category: file

Parent:

Description: Contract Type

Attributes **Required traits**

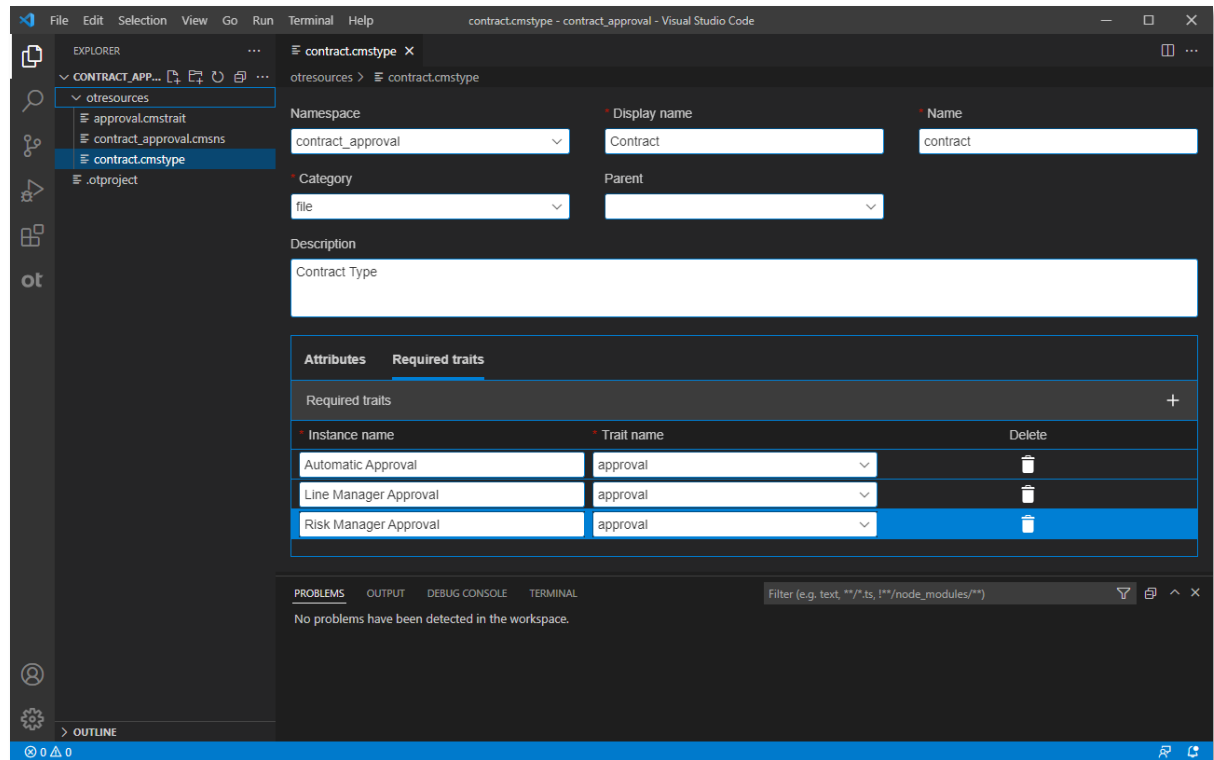
Required traits

Instance name	Trait name	Delete
Automatic Approval	approval	
Line Manager Approval	approval	
Risk Manager Approval	approval	

REMARK:

Note that for the current CMS type definition creation exercise we are using the **New File** menu item from the contextual menu on the **otresources** folder in the VS Code file explorer tree, and we are naming the file with the corresponding **contract.cmstype** file name and extension. This is the third and last way to create models we wanted to illustrate in this tutorial. In the next exercises, feel free to choose whichever of the three methods you prefer.

You can now save and close the **Contract** CMS type definition model. Saving will NOT pop up a **Save As** file saving dialog box because the file already exists in the **otresources** folder. However, you will note that the “File is empty” error has now disappeared from the **PROBLEMS** tab, since you indeed saved the model information for the first time, effectively no longer making it an empty file.

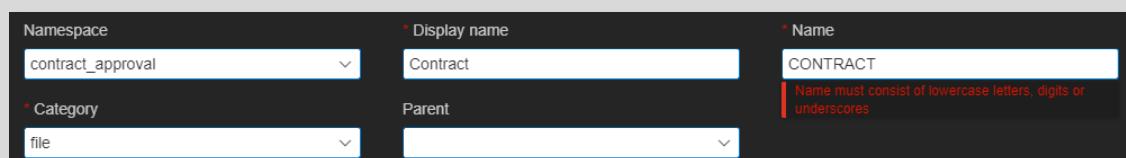


REMARK:

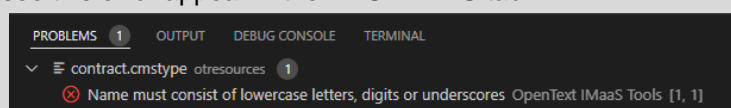
This is a good time to mention that all models get validated to enforce constraints/rules. So, very much like with the previous example of a model file not being allowed to be empty, the system enforces any constraints that have been defined for the type of model you are creating.

The validation for your model happens at two different moments:

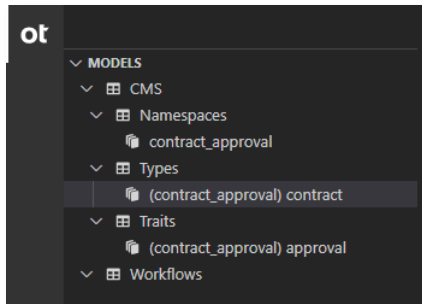
- **When you change a value of a model property.** Any constraint violation resulting from modifying a model property is immediately shown inline in your model editor. Try for example to put some upper-case letters in the **Name** property of your **contract** CMS type model. This will result in showing an “only lowercase, numbers and underscores allowed” error message directly under the **Name** field:



- **When you save the model file.** This additional validation looks at what is actually saved on disk, and it traps all constraint violations (including broken references) in your model folder (**/otresources**). The errors that reside inside your saved model files are shown in the **PROBLEMS** tab of VS Code (like with the previous empty file issue). As an additional example, feel free to save your model with the upper-case letters in the **Name** property. You will now also see this error appear in the **PROBLEMS** tab:



If you switch to the **OpenText IMaaS Tools** view, the model explorer tree in the **MODELS** section should now show your new **(contract_approval) contract** CMS type definition under **/CMS/Types**. The **contract_approval** value between brackets represents the CMS namespace to which the CMS type definition belongs, as the model explorer shows the different models according to their unique key (which in context of a CMS type definition is the combination of the namespace and name properties).



3.7 [10'] Creating a CMS file type definition that is a subtype

During this exercise you will create the **Loan Contract** CMS type definition model. This will be a subtype of the previously created **Contract** CMS type definition (i.e.: it will have **contract** for its **Parent** property). To allow inheriting (i.e.: subtyping) the **Contract** CMS type definition, the **Loan Contract** CMS type definition also has to be of the **file** category.

Once you are done with this section, you will have created the **Loan Contract** CMS type definition that corresponds with your Contract Approval application's (child) contract file type, and you are ready to start creating the Customer (CMS) type definition.

To create the **Loan Contract** CMS type definition, proceed as follows:

- In VS Code, using one of the three previously explained ways of creating a model, create a new CMS type definition and name it **loan_contract** when asked to enter the (file) name.

The screenshot shows a form for defining a CMS type. The fields are as follows:

- Namespace:** A dropdown menu.
- Display name:** A text field containing "Loan Contract".
- Name:** A text field containing "loan_contract".
- Category:** A dropdown menu.
- Parent:** A dropdown menu.
- Description:** A large text area.
- Attributes:** A table with the following columns: Display name, Name, Data type, Default value, Size, Repeating, Required, Unique, Read-only, Searchable, Sortable.

Fill the CMS type definition properties as follows:

- **Namespace:** **contract_approval**
- **Display name:** **Loan Contract**
- **Name:** **loan_contract**
- **Category:** **file**
- **Parent:** the parent CMS type definition for the type definition you are creating; choose **contract** as parent for the **Loan Contract** CMS type

REMARK:

Note that the **contract** parent CMS type is only available to be selected from the **Parent** drop-down list when the **file** **Category** is selected. If you empty the **Category** value or select **folder**, you will not be able to select **contract** as it is indeed a CMS file type definition.

- **Description:** **Loan Contract Type**

- **Attributes:** each attribute definition and the property values to assign is described in the table below

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	monthly_installments	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	yearly_income	integer		required

- **Required traits:** each required trait definition and the property values to assign is described in the table below

Required trait description	Instance name	Trait name
The automated approval step that checks whether or not the customer is solvent by checking that the monthly cost doesn't 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

loan_contract.cmstype

Namespace: contract_approval | Display name: Loan Contract | Name: loan_contract

Category: file | Parent: contract

Description: Loan Contract Type

Attributes

Display name	Name	Data type	Default value	Size	Repeating	Required	Unique	Read-only	Searchable	Sortable	
Monthly installments	monthly_installments	integer	12		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Yearly income	yearly_income	integer			<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

loan_contract.cmstype

Namespace: contract_approval | Display name: Loan Contract | Name: loan_contract

Category: file | Parent: contract

Description: Loan Contract Type

Required traits

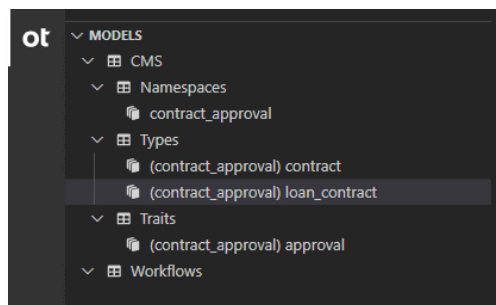
Instance name	Trait name	Delete
Solvency Check	approval	<input type="checkbox"/>

REMARK:

Although we only assigned two attributes and one required trait to the **Loan Contract** CMS type definition, being a subtype of the **Contract** CMS type definition implies that all the attributes and required traits of that CMS type definition are also present on the **Loan Contract** CMS type definition.

You can now save and close the **Loan Contract** CMS type definition model.

If you switch to the **OpenText IMaaS Tools** view, the model explorer tree in the **MODELS** section should now show your new **(contract_approval) loan_contract** CMS type definition under **/CMS/Types**.



3.8 [05'] Creating a CMS folder type definition

During this exercise you will create the **Customer** CMS type definition model. This will be a CMS type definition of the **folder** category as it represents a customer, and it will contain (as a folder) the different contracts of CMS file type **Contract** or **Loan Contract** related to that customer.

Once you are done with this section, you will have created the **Customer** CMS type definition that corresponds with your Contract Approval application's customer folder, and you are ready to start creating the Contract Approval workflow model.

To create the **Customer** CMS type definition, proceed as follows:

- In VS Code, using one of the three previously explained ways of creating a model, create a new CMS type definition and name it **customer** when asked to enter the (file) name.

The screenshot shows the 'customer.cmstype' file in VS Code. The form has the following fields:

- Namespace: (empty dropdown)
- Display name: Customer
- Name: customer
- Category: (empty dropdown)
- Parent: (empty dropdown)
- Description: (empty text area)

Below the form is a table for 'Attributes' and 'Required traits'.

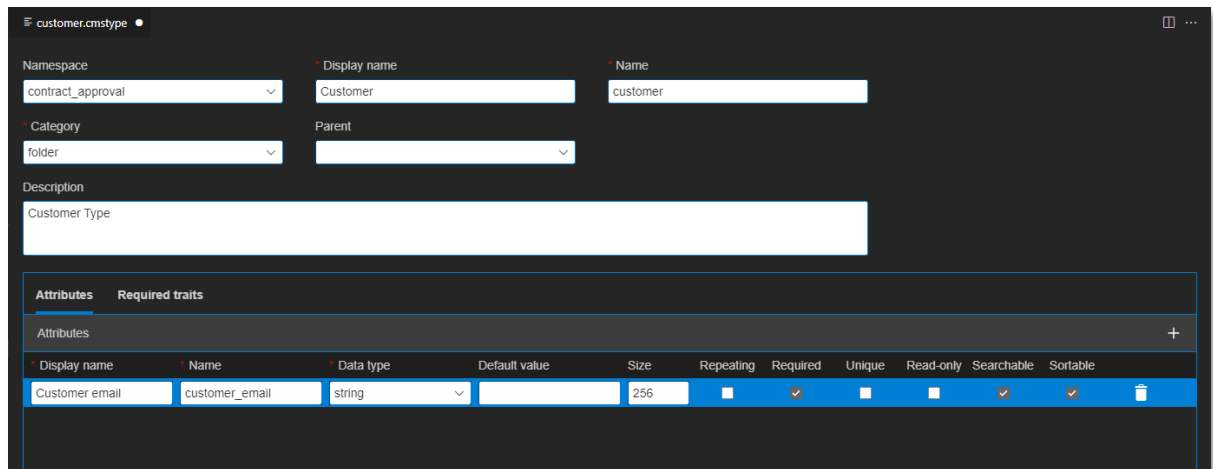
Display name	Name	Data type	Default value	Size	Repeating	Required	Unique	Read-only	Searchable	Sortable

Fill the CMS type definition properties as follows:

- **Namespace:** **contract_approval**
- **Display name:** **Customer**
- **Name:** **customer**
- **Category:** **folder**
- **Parent:** the **Customer** CMS type has no parent
- **Description:** **Customer Type**
- **Attributes:** each attribute definition and the property values to assign is described in the table below

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

- **Required traits:** the **Customer** CMS type has no required traits



customer.cmstype

Namespace: contract_approval

Display name: Customer

Name: customer

Category: folder

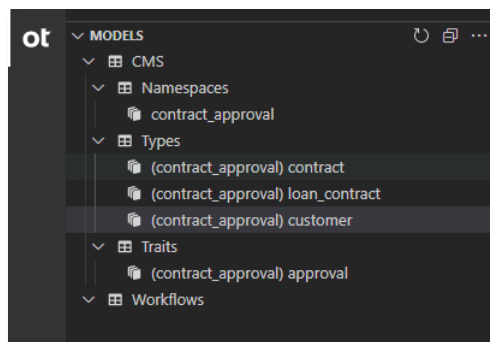
Parent:

Description: Customer Type

Display name	Name	Data type	Default value	Size	Repeating	Required	Unique	Read-only	Searchable	Sortable
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"/>

You can now save and close the **Customer** CMS type definition model.

If you switch to the **OpenText IMaaS Tools** view, the model explorer tree in the **MODELS** section should now show your new **(contract_approval) customer** CMS type definition under **/CMS/Types**.



3.9 [120'] Creating a workflow model

During this exercise you will create the **Contract Approval** workflow model. A workflow model represents an executable process model from which process instances will be created. The executable process model is stored as BPMN 2.0 encoded JSON. This **Contract Approval** workflow model will automate the contract approval process of your Contract Approval application. It consists of a number of automated and manual approval tasks. Not all approval tasks are required. They are conditional, based on the type, value, and risk of the contract. Throughout this exercise we will go into detail on how to configure the different events, activities, and conditional gateways (choices) of the **Contract Approval** workflow model. For more information on Workflow Service process models and process instances, you can refer to the [Workflow Service product documentation](#), the [Workflow Modeler product documentation](#) or the [Workflow Service API reference](#).

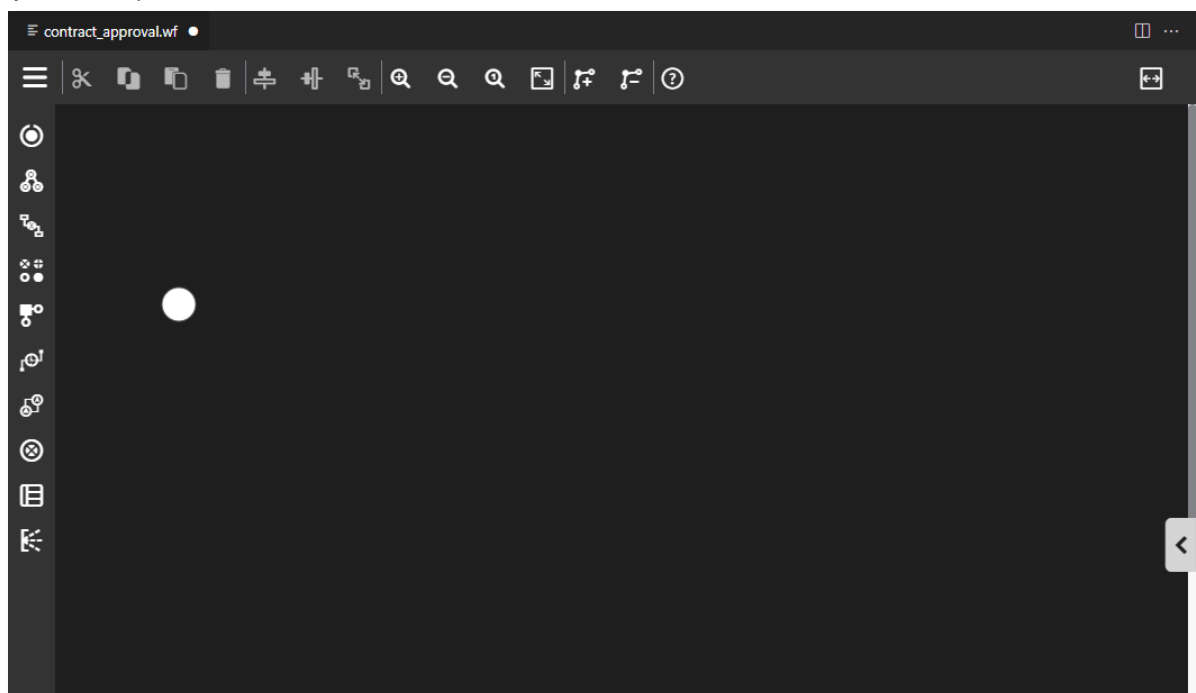
Once you are done with this section, you will have created the **Contract Approval** workflow model that corresponds with your Contract Approval application's approval process, and as this is the last IMaaS model for your application, you are ready to deploy your **Contract Approval** IMaaS application project (i.e.: its models) to the different IMaaS services.


REMARK:

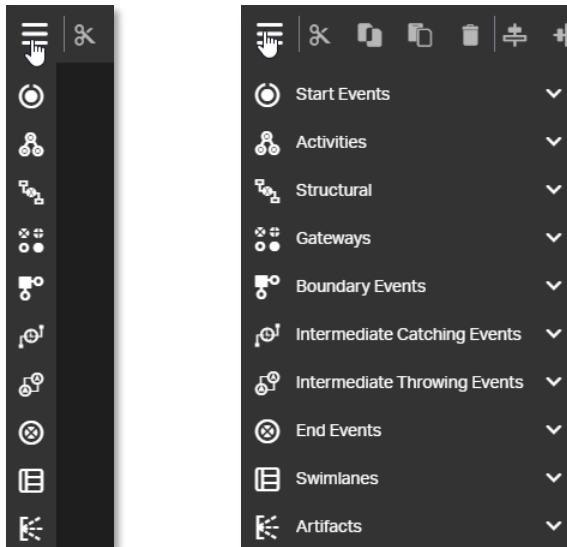
We certainly recommend you go through this entire exercise and build the workflow yourself. However, inherent to the complexity of the workflow model you are going to build is that there could be issues that require troubleshooting, and which you might have trouble fixing. To help with that, whether it is to compare your workflow model with a working version or just to use an existing working version instead of the one you built, you can download the finished Contract Approval application [here](#). You will find the **contract_approval.wf** workflow under the **otresources** folder.


To create the **Contract Approval** workflow model, proceed as follows:

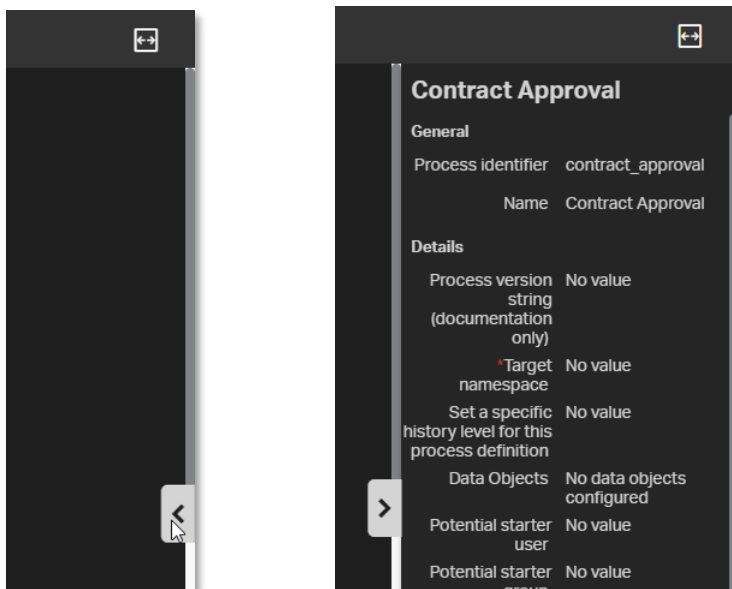
- In VS Code, using one of the three previously explained ways of creating a model, create a new workflow and name it **contract_approval** when asked to enter the (file) name. If you are using **New File** to create the new workflow, note that you should assign the **“.wf”** extension for the system to open the workflow model editor.




- Compared to the previously used CMS model editors, the workflow model editor is much more elaborate. It is intended to build entire business process definitions with all the complexity this brings. Throughout this exercise we will step by step build the **Contract Approval** workflow, and this will indeed illustrate this much higher degree of complexity. However, before getting started with building the actual workflow, we should first have a look at the model editor itself to understand its different features. Let's first expand the two collapsed side panes. On the top left of the editor, click the  button to expand the **palette** (i.e.: the left pane).



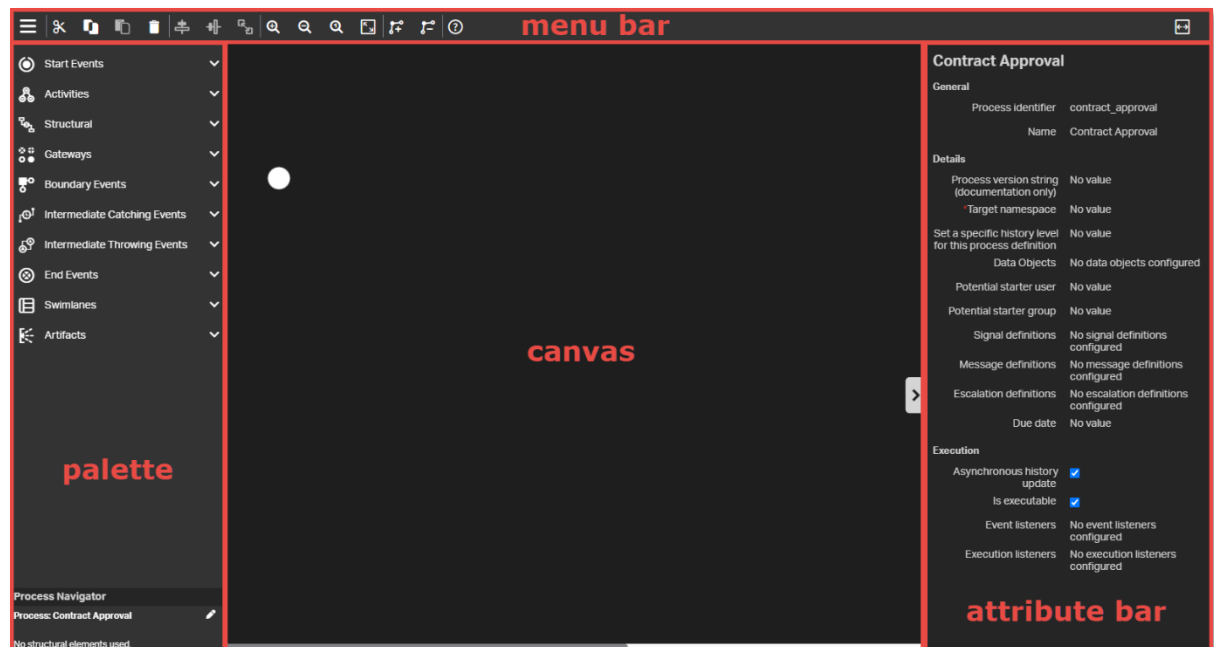
Now, click the  button on the middle right of the editor to expand the **attribute bar** (i.e.: the right pane).



Note that the  button on the top right allows to expand or collapse both side panes at once.

As you can now see, the workflow editor has four areas:

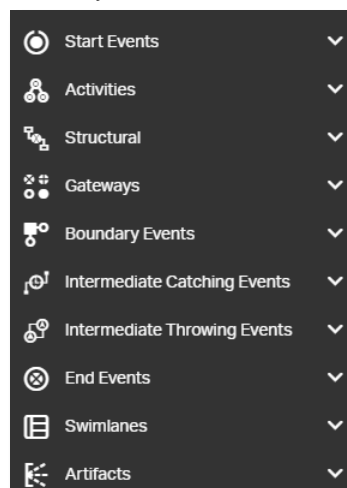
- The **menu bar** on top
- The **palette** on the left side
- The **canvas** in the middle
- The **attribute bar** on the right side



The **menu bar** contains the generic capabilities/buttons (e.g.: copy/paste, delete, align, zoom, help, etc.).



The **palette** contains the different workflow elements, which you can drag and drop on the canvas to build your workflow model.



The **canvas** is the area where you will be building/drawing your workflow model.



The **attribute bar** displays the attributes of the currently selected element. At the moment, no element is selected (which is the same as clicking on an empty area on the canvas), so the workflow model's attributes are displayed.

Contract Approval

General

Process identifier	contract_approval
Name	Contract Approval

Details

Process version string (documentation only)	No value
Target namespace	No value
Set a specific history level for this process definition	No value
Data Objects	No data objects configured
Potential starter user	No value
Potential starter group	No value
Signal definitions	No signal definitions configured
Message definitions	No message definitions configured
Escalation definitions	No escalation definitions configured
Due date	No value


Execution

Asynchronous history update	<input checked="" type="checkbox"/>
Is executable	<input checked="" type="checkbox"/>
Event listeners	No event listeners configured
Execution listeners	No execution listeners configured

- Now that you have an understanding of the different areas of the workflow model editor, let's start building our workflow model.

REMARK:

Although we will cover many features of the workflow model editor in this tutorial, providing you with a good and practical understanding of how to build business process definitions, we will not cover every single feature and capability. This means we will only use a subset of the buttons on the **menu bar** and the workflow elements in the **palette**. We will also not describe all attributes in the **attribute bar**.

Once you understand how to build workflow models, the [Workflow Modeler product documentation](#) (also available from the  button on the menu bar) provides you a more exhaustive and in depth explanation of the functionality and usage of the workflow model editor (or workflow modeler).

The first thing we will look at is filling the workflow model's own attributes, visible from the attribute bar when you have no element selected (i.e.: you click an empty area on the canvas to ensure the entire workflow model is selected). As per the above remark, we will only explain those attributes we consider relevant in context of this tutorial. Feel free to open the *Processes* section of the [Workflow Modeler product documentation](#) for more details on the different process model attributes.

Fill the workflow model attributes as follows:

- **Process identifier:** the process identifier is the technical name for the workflow model; this has to be unique (within your developer tenant), and it has been automatically populated for your convenience based on the previously chosen workflow name (the model file name); you should best leave the value to be **contract_approval** as it nicely aligns with the model file name and model name
- **Name:** the name is the user-friendly name for the workflow model; this does not have to be unique, and it has been automatically populated for your convenience based on the previously chosen workflow name (the model file name); you should best leave the value to be **Contract Approval** as it nicely aligns with the model file name and model process identifier
- **Target namespace:** The target namespace allows grouping the different workflow models together; you can fill **contract_approval** as value
- All other workflow model attributes can remain unchanged

Contract Approval

General

Process identifier	contract_approval
Name	Contract Approval

Details

Process version string (documentation only)	No value
*Target namespace	contract_approval
Set a specific history level for this process definition	No value
Data Objects	No data objects configured
Potential starter user	No value
Potential starter group	No value
Signal definitions	No signal definitions configured
Message definitions	No message definitions configured
Escalation definitions	No escalation definitions configured
Due date	No value

Execution

Asynchronous history update	<input checked="" type="checkbox"/>
Is executable	<input checked="" type="checkbox"/>
Event listeners	No event listeners configured
Execution listeners	No execution listeners configured

Since building a workflow model contains many steps, it is recommended that you regularly save your workflow model to avoid losing work. You can now save the workflow model.

- Now that the workflow model attributes have been set, we can start building the actual business process definition by drawing it on the canvas.

The first element to set is the start event. As this element is already on the canvas, we just need to set the attributes.

Select the **start event** element (on the canvas) and set the attributes as follows:

- **Name: Start**
- All other element attributes can remain unchanged

Start

General

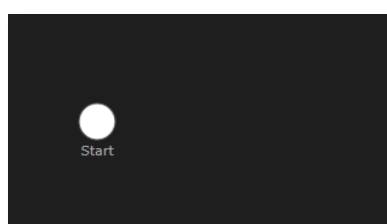
Id	No value
Name	Start

Details

Initiator	No value
-----------	----------

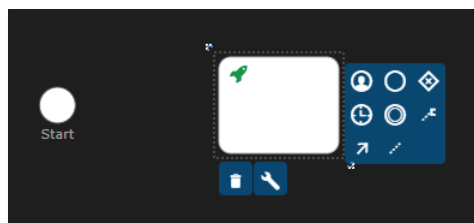
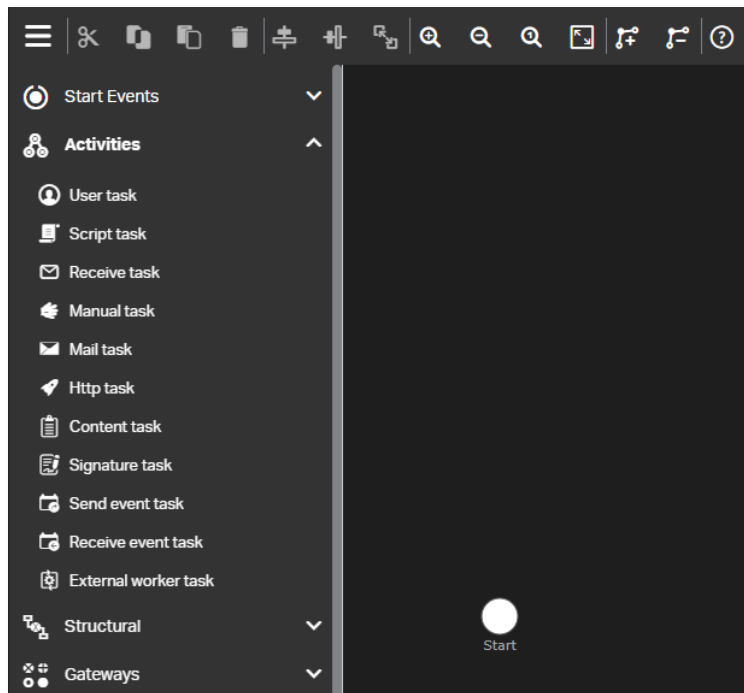
Execution

Execution listeners	No execution listeners configured
---------------------	-----------------------------------



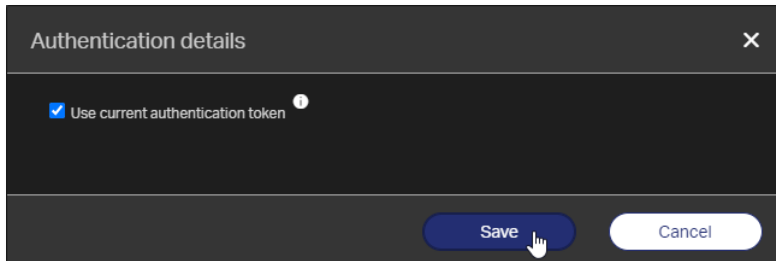
- The next element we are going to add is an http task (i.e.: REST API call). This task will perform a GET request to fetch the JSON object that holds the metadata of the contract to approve. This contract JSON object can then be used throughout the subsequent steps of the business process.

To add the new http task, expand the **Activities** section from the palette and drag and drop an **Http task** to the right of the **Start event**.



Select the **http task** element and set the element attributes as follows:

- **Name: Get contract from CMS**
- **Authentication details: Use current authentication token**

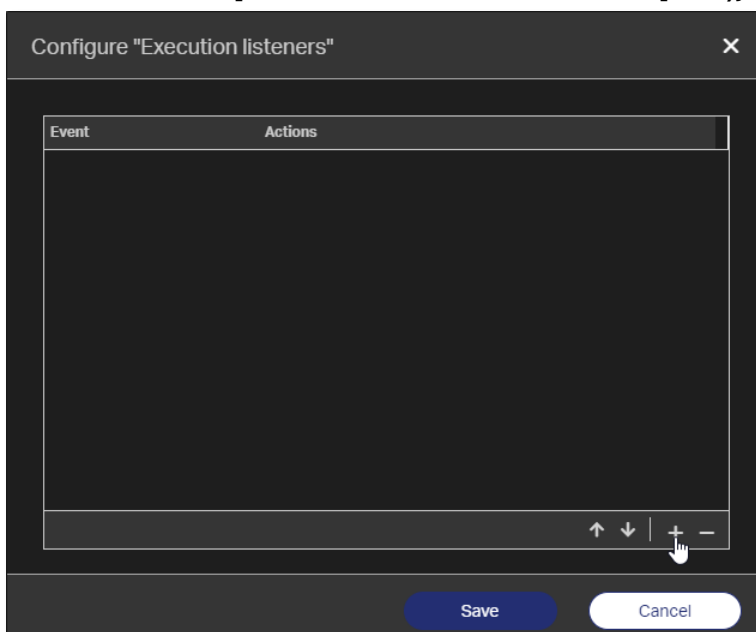


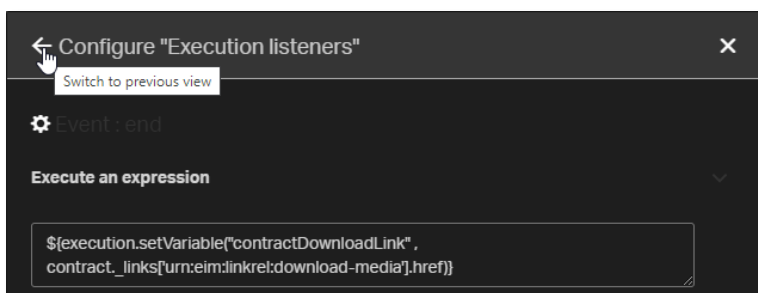
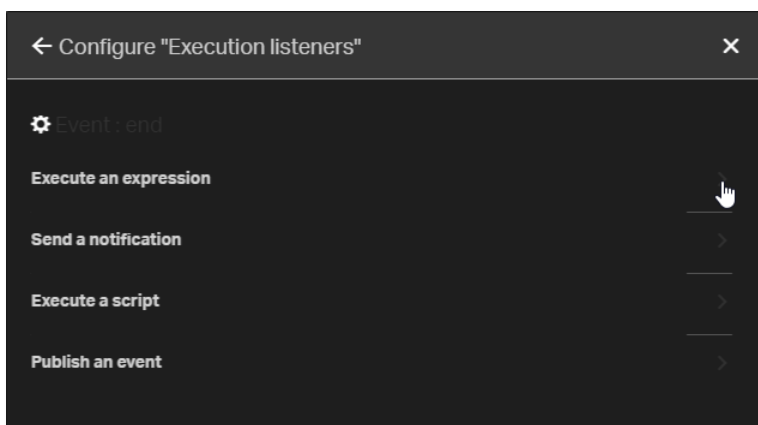
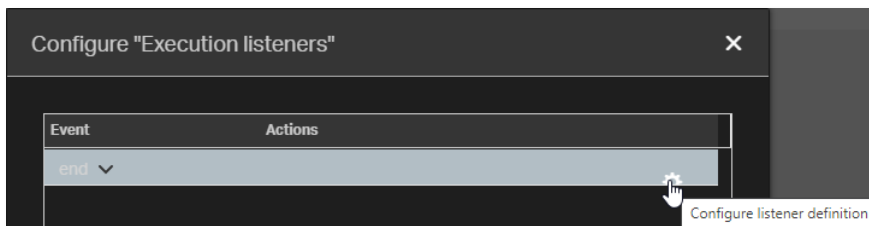
- **Request method: GET**
- **Request URL: `${base_url}/cms/instances/file/ca_contract/${contract_id}`**
 => The 'base_url' parameter is referring to the base URL of the IMaaS services for your trial account
 => The 'contract_id' parameter is referring to the unique identifier of the contract to approve (so that you can use it to fetch contract information)
 Both the 'base-url' and 'contract_id' are being passed to the workflow when initializing it from your application's code (this will be explained/shown later, under exercise [Building the application frontend](#)).

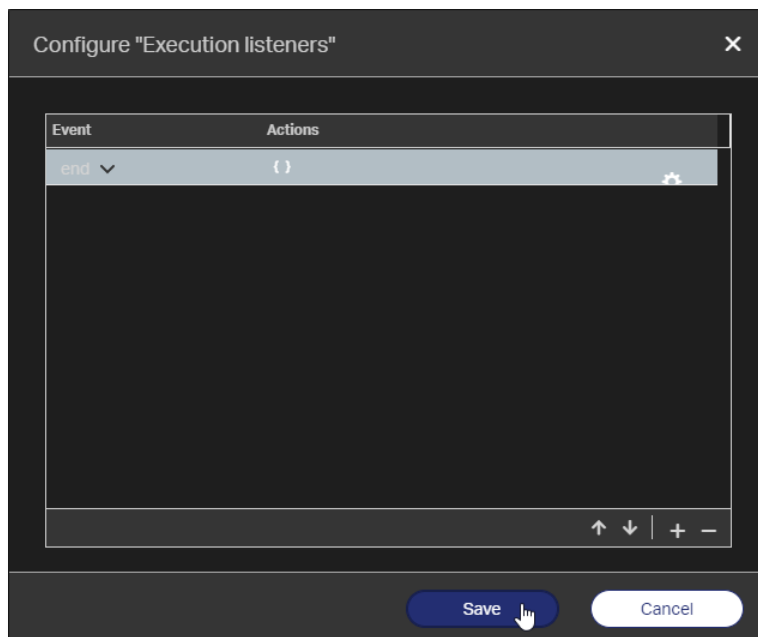
REMARK:

In the request URL you can see we are using `${}` to pass parameters. This is the standard mechanism to pass process variables as (string) parameters to expressions.

- **Response variable name: contract**
- **Save response as JSON: true**
- **Exclusive: true**
- **Execution listeners:**
 - **Event: end**
 - **Execute an expression: `${execution.setVariable("contractDownloadLink", contract._links["urn:eim:linkrel:download-media"].href)}`**







- All other element attributes can remain unchanged

Get contract from CMS

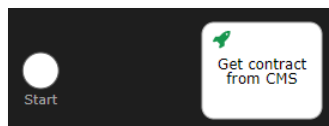
General

Id	No value
Name	Get contract from CM ...

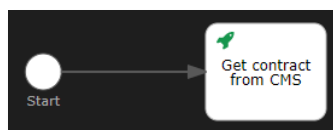
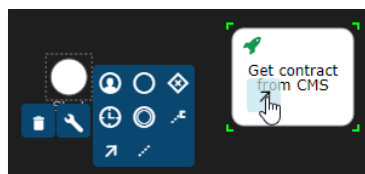
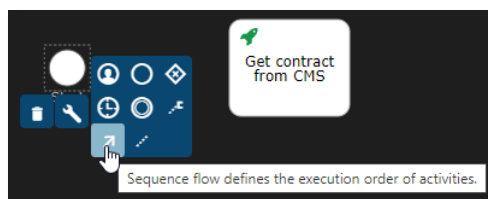
Details

Authentication details	Authentication configured
*Request method	GET
*Request URL	\${base_url}/cms/inst ...
Request headers	No value
Request body	No value
Request body encoding	No value
Request timeout	No value
Disallow redirects	No value
Fail status codes	No value
Handle status codes	No value
Ignore exception	No value
Response variable name	contract
Save request variables	No value
Save response status, headers	No value
Result variable prefix	No value
Save response as a transient variable	No value
Save response as JSON	true

Execution	
Asynchronous	<input type="checkbox"/>
Is for compensation	<input type="checkbox"/>
Exclusive	true
Skip expression	No value
Execution listeners	1 execution listeners
Multi-instance	
Type	None
Cardinality	No value
Collection	No value
Element variable	No value
Completion condition	No value



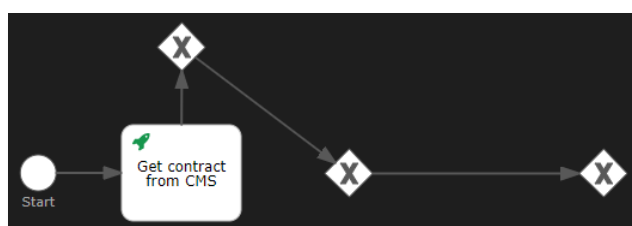
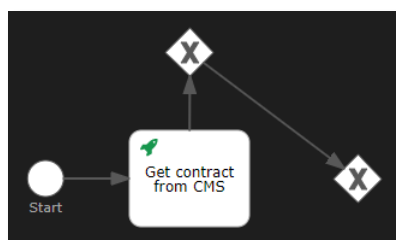
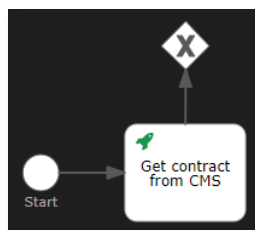
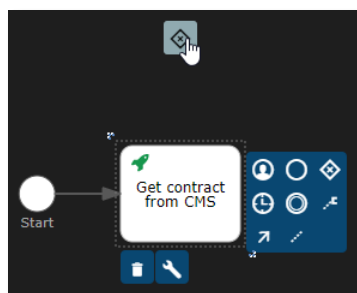
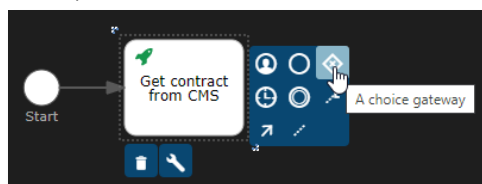
Drag a **sequence flow** (arrow) connector from the **Start event** element to the **Get contract from CMS http** task element.



There is no need to set any attributes for the sequence flow connector you just added.

- Let's now add three exclusive (or choice) gateways to, based upon certain conditions, route the business process execution in one of two directions. The three exclusive gateways we are going to add are going to have the following behavior:
 - Take a different route based on whether the contract is a **Standard contract** (of CMS type **contract**) or a **Loan Contract** (of CMS type **loan_contract**)
 - Take a different route based on whether the **value** (attribute) of the contract is **below or equal to 1000** or **above 1000**
 - Take a different route based on whether the **risk_classification** (attribute) is **MEDIUM (3) or lower** or **HIGH (4) or higher**

To create the three exclusive gateways, drag the **Exclusive gateway** element from the previous element (i.e.: for the first exclusive gateway, this is the **Get contract from CMS http task** and for the two subsequent exclusive gateways, it is the **Exclusive gateway** element you created before).

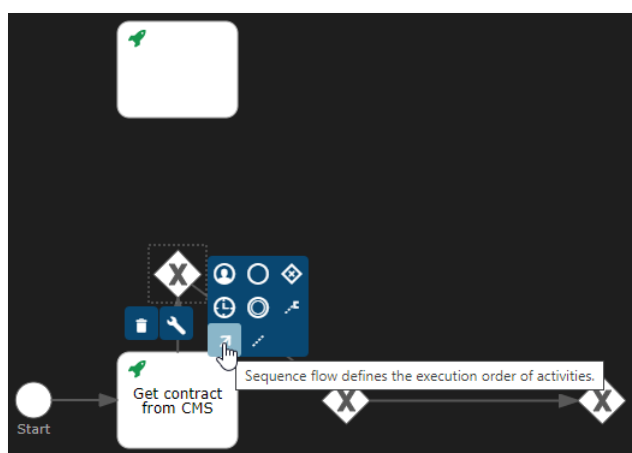
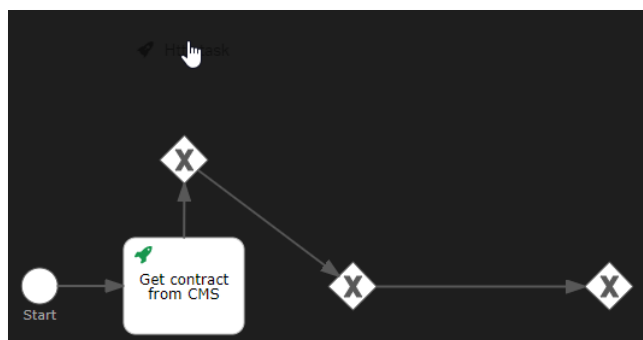
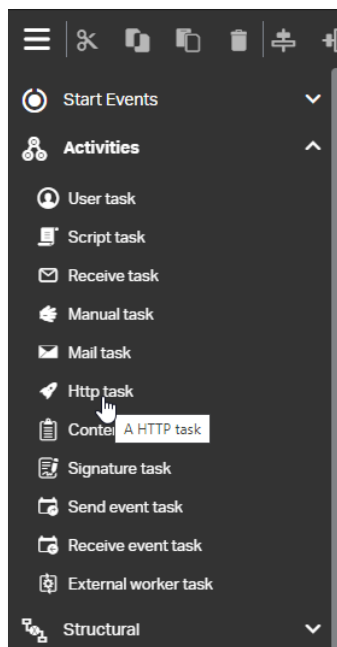


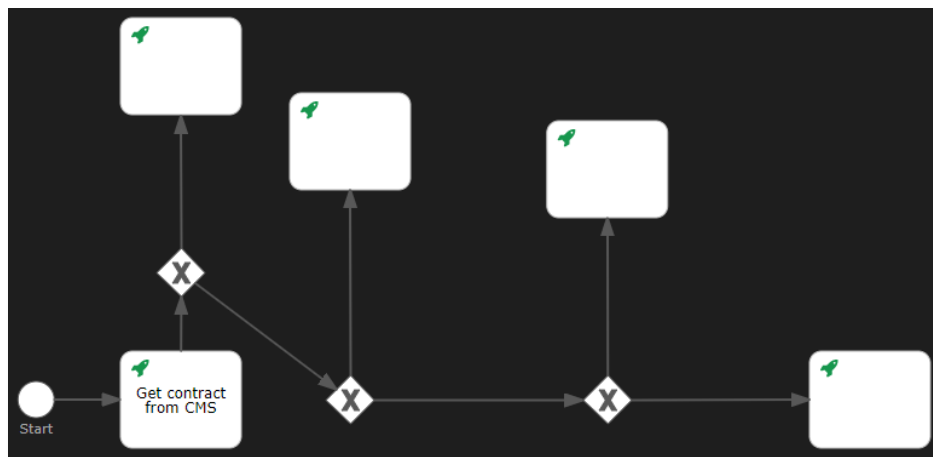
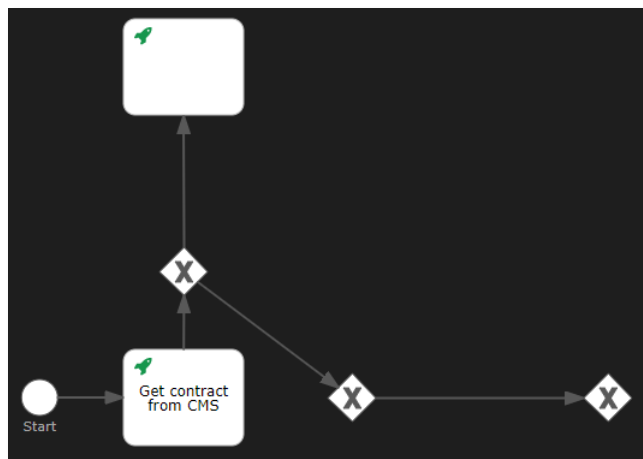
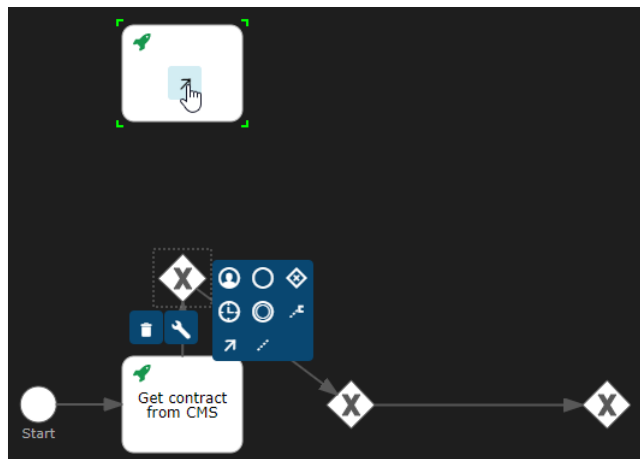
For all three exclusive gateway elements you just created, set the **Exclusive** attribute to **true**:

The screenshot shows a configuration window titled "My Process". It has three tabs: "General", "Details", and "Execution". The "Execution" tab is selected. It contains two settings: "Asynchronous" with a checkbox that is currently unchecked, and "Exclusive" with a text field containing the value "true".

- The next step is to add all activities that correspond with the different choices of the exclusive gateways. More specifically, for the first and the second exclusive gateway, we already have one sequence flow connector representing the (first) choice that moves the business process to the next exclusive gateway. Let's now add the second choice for the first and second exclusive gateways, and both choices for the last gateway. To explain what those choices are, let's revisit the logic of the exclusive gateways and add the activity/behavior for each choice:
 - First exclusive gateway:
 - In case the contract is a **Standard contract** (of CMS type **contract**), the process can advance to the second gateway
 - In case the contract is a **Loan Contract** (of CMS type **loan_contract**), the solvency (i.e.: the ability to pay back the loan) of the customer needs to be checked, and the process has to advance to a new **Http task** that updates (i.e.: **PATCH** request) the status of the contract to **SOLVENCY CHECK**
 - Second exclusive gateway:
 - In case the **value** (attribute) of the contract is **below or equal to 1000**, the process can advance to the third gateway
 - In case the **value** (attribute) of the contract is **above 1000**, the contract needs to be (manually) approved by a line manager, and the process has to advance to a new **Http task** that updates (i.e.: **PATCH** request) the status of the contract to **LINE MANAGER APPROVAL**
 - Third exclusive gateway:
 - In case the **risk_classification** (attribute) is **MEDIUM (3) or lower**, the process can advance to the (always required) automatic approval, and the process can advance to a new **Http task** that updates (i.e.: **PATCH** request) the status of the contract to **APPROVED**
 - In case the **risk_classification** (attribute) is **HIGH (4) or higher**, the contract needs to be (manually) approved by a risk manager, and the process has to advance to a new **Http task** that updates (i.e.: **PATCH** request) the status of the contract to **RISK MANAGER APPROVAL**

To add the four new (PATCH request) http tasks that update the status of the contract, drag and drop the **Http task** elements from the palette (under the **Activities** section) to the canvas, and connect the three exclusive gateways with these new http tasks using **sequence flow** (arrow) connectors.





Now that you have created the four different http tasks that update the status of the contract, you can set their respective attributes. From left to right (on the previous picture), please fill the http task attributes as follows:

- Http task to set contract status to SOLVENCY CHECK:
 - **Name:** Set contract status to SOLVENCY CHECK
 - **Authentication details:** Use current authentication token
 - **Request method:** PATCH
 - **Request URL:** \${base_url}/cms/instances/file/ca_contract/\${contract_id}
 - **Request headers:** Content-Type: application/json
 - **Request body:**

```
{
  "properties": {
    "status": "SOLVENCY CHECK"
  }
}
```
 - **Response variable name:** contract
 - **Save response as JSON:** true
 - **Exclusive:** true

Set contract status to SOLVENCY CH...

General

Id	No value
Name	Set contract status ...

Details

Authentication details	Authentication configured
Request method	PATCH
Request URL	\${base_url}/cms/inst ...
Request headers	Content-Type: applic ...
Request body	{ "properties": { ...
Request body encoding	No value
Request timeout	No value
Disallow redirects	No value
Fail status codes	No value
Handle status codes	No value
Ignore exception	No value
Response variable name	contract
Save request variables	No value
Save response status, headers	No value
Result variable prefix	No value
Save response as a transient variable	No value
Save response as JSON	true

Execution

Asynchronous	<input type="checkbox"/>
Is for compensation	<input type="checkbox"/>
Exclusive	true
Skip expression	No value

- Http task to set contract status to LINE MANAGER APPROVAL:
 - **Name:** Set contract status to LINE MANAGER APPROVAL
 - **Authentication details:** Use current authentication token
 - **Request method:** PATCH
 - **Request URL:** \${base_url}/cms/instances/file/ca_contract/\${contract_id}
 - **Request headers:** Content-Type: application/json
 - **Request body:**

```
{
  "properties": {
    "status": "LINE MANAGER APPROVAL"
  },
  "traits": {
    "ca_approval": {
      "Line Manager Approval": {
        "is_required": true,
        "has_been_granted": false,
        "approver": "${contract.updated_by.email}",
        "approver_role": "Line Manager"
      }
    }
  }
}
```
 - **Response variable name:** contract
 - **Save response as JSON:** true
 - **Exclusive:** true

Set contract status to LINE MANAGE...

General

Id	No value
Name	Set contract status ...

Details

Authentication details	Authentication configured
*Request method	PATCH
*Request URL	\${base_url}/cms/inst ...
Request headers	Content-Type: applic ...
Request body	{"properties": { ...
Request body encoding	No value
Request timeout	No value
Disallow redirects	No value
Fail status codes	No value
Handle status codes	No value
Ignore exception	No value
Response variable name	contract
Save request variables	No value
Save response status, headers	No value
Result variable prefix	No value
Save response as a transient variable	No value
Save response as JSON	true

Execution

Asynchronous	<input type="checkbox"/>
Is for compensation	<input type="checkbox"/>
Exclusive	true

- Http task to set contract status to RISK MANAGER APPROVAL:
 - **Name:** Set contract status to RISK MANAGER APPROVAL
 - **Authentication details:** Use current authentication token
 - **Request method:** PATCH
 - **Request URL:** \${base_url}/cms/instances/file/ca_contract/\${contract_id}
 - **Request headers:** Content-Type: application/json
 - **Request body:**

```
{
  "properties": {
    "status": "RISK MANAGER APPROVAL"
  },
  "traits": {
    "ca_approval": {
      "Risk Manager Approval": {
        "is_required": true,
        "has_been_granted": false,
        "approver": "${contract.updated_by.email}",
        "approver_role": "Risk Manager"
      }
    }
  }
}
```
 - **Response variable name:** contract
 - **Save response as JSON:** true
 - **Exclusive:** true

Set contract status to RISK MANAGE...

General

Id	No value
Name	Set contract status ...

Details

Authentication details	Authentication configured
*Request method	PATCH
*Request URL	\${base_url}/cms/inst ...
Request headers	Content-Type: applic ...
Request body	{ "properties": { ...
Request body encoding	No value
Request timeout	No value
Disallow redirects	No value
Fail status codes	No value
Handle status codes	No value
Ignore exception	No value
Response variable name	contract
Save request variables	No value
Save response status, headers	No value
Result variable prefix	No value
Save response as a transient variable	No value
Save response as JSON	true

Execution

Asynchronous	<input type="checkbox"/>
Is for compensation	<input type="checkbox"/>
Exclusive	true

- Http task to set contract status to APPROVED:
 - **Name:** Set contract status to APPROVED
 - **Authentication details:** Use current authentication token
 - **Request method:** PATCH
 - **Request URL:** \${base_url}/cms/instances/file/ca_contract/\${contract_id}
 - **Request headers:** Content-Type: application/json
 - **Request body:**

```
{
  "properties": {
    "status": "APPROVED"
  }
}
```
 - **Response variable name:** contract
 - **Save response as JSON:** true
 - **Exclusive:** true

Set contract status to APPROVED

General

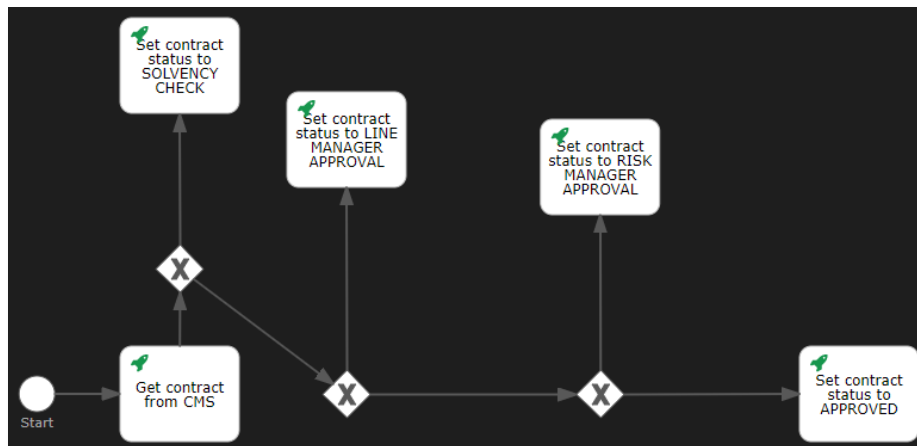
Id	No value
Name	Set contract status ...

Details

Authentication details	Authentication configured
Request method	PATCH
Request URL	\${base_url}/cms/inst ...
Request headers	Content-Type: applic ...
Request body	{ "properties": { ...
Request body encoding	No value
Request timeout	No value
Disallow redirects	No value
Fail status codes	No value
Handle status codes	No value
Ignore exception	No value
Response variable name	contract
Save request variables	No value
Save response status, headers	No value
Result variable prefix	No value
Save response as a transient variable	No value
Save response as JSON	true

Execution

Asynchronous	<input type="checkbox"/>
Is for compensation	<input type="checkbox"/>
Exclusive	true
Skip expression	No value



The last step to completing the exclusive gateway (choices) logic is to configure the different sequence flows (i.e.: set the attributes of the arrow connectors). Per exclusive gateway, for the outgoing sequence flow arrow connectors, set the attributes as follows:

- First exclusive gateway:
 - Sequence flow going to the “Set contract status to SOLVENCY CHECK” http task:
 - **Name:** Loan contract
 - **Flow condition:** `${contract.type == "ca_loan_contract"}`
 - **Default flow:** <not checked>

Loan contract	
General	
Id	No value
Name	Loan contract
Details	
One * property is required	
*Flow condition	<code>\${contract.type == "</code>
*Default flow	<input type="checkbox"/>
Skip expression	No value
Execution	
Execution listeners	No execution listeners configured

- Sequence flow going to the second exclusive gateway:
 - **Name:** Standard contract
 - **Default flow:** <checked>

Standard contract	
General	
Id	No value
Name	Standard contract
Details	
One * property is required	
*Flow condition	No condition set
*Default flow	<input checked="" type="checkbox"/>
Skip expression	No value
Execution	
Execution listeners	No execution listeners configured

- Second exclusive gateway:
 - Sequence flow going to the “Set contract status to LINE MANAGER APPROVAL” http task:
 - **Name: Contract value > 1000**
 - **Flow condition: `${contract.properties.value > 1000}`**
 - **Default flow: <not checked>**

Contract value > 1000

General

Id

No value

Name

Contract value > 100 ...

Details

One * property is required

*Flow condition

`${contract.propertie`

*Default flow

☐

Skip expression

No value

Execution

Execution listeners

No execution listeners configured

- Sequence flow going to the third exclusive gateway:
 - **Name: Contract value <= 1000**
 - **Default flow: <checked>**

Contract value <= 1000

General

Id

No value

Name

Contract value <= 10 ...

Details

One * property is required

*Flow condition

No condition set

*Default flow

☒

Skip expression

No value

Execution

Execution listeners

No execution listeners configured

- Third exclusive gateway:
 - Sequence flow going to the “Set contract status to RISK MANAGER APPROVAL” http task:
 - **Name: Contract risk > 3**
 - **Flow condition: `${contract.properties.risk_classification > 3}`**
 - **Default flow: <not checked>**

Contract risk > 3

General

Id	No value
Name	Contract risk > 3

Details

One * property is required

*Flow condition	<code>\${contract.properties.risk_classification > 3}</code>
*Default flow	<input type="checkbox"/>
Skip expression	No value

Execution

Execution listeners	No execution listeners configured
---------------------	-----------------------------------

- Sequence flow going to the “Set contract status to APPROVED” http task:
 - **Name: Contract risk <= 3**
 - **Default flow: <checked>**

Contract risk <= 3

General

Id	No value
Name	Contract risk <= 3

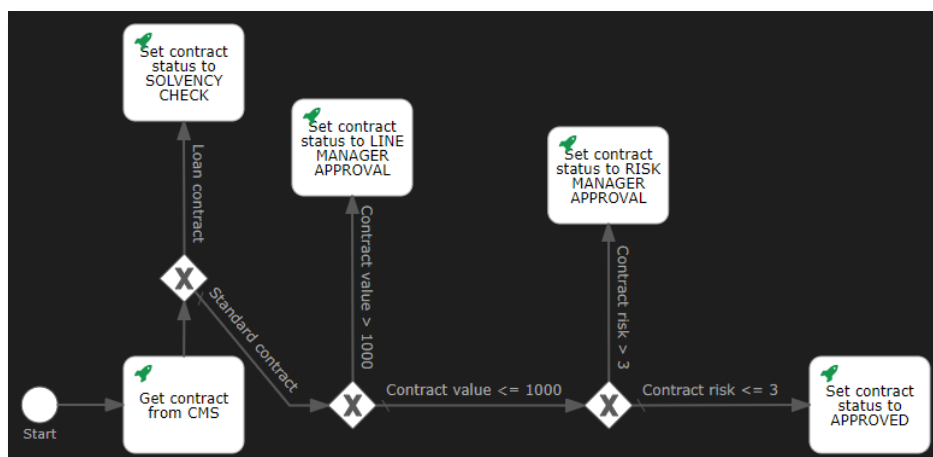
Details


One * property is required

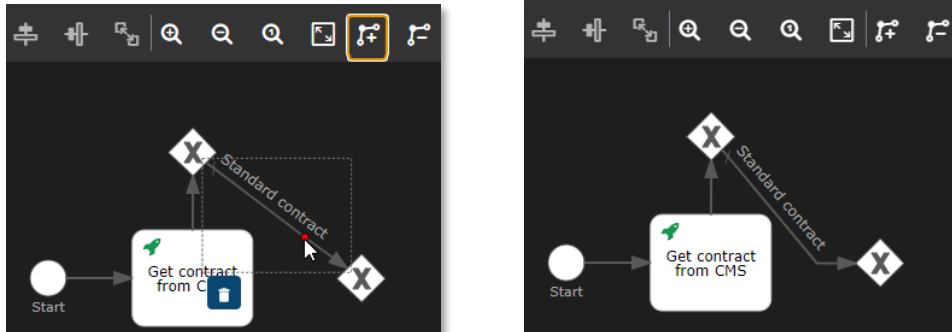
*Flow condition	No condition set
*Default flow	<input checked="" type="checkbox"/>
Skip expression	No value

Execution

Execution listeners	No execution listeners configured
---------------------	-----------------------------------



You might have noticed that the “Standard contract” sequence flow arrow connector now has a bend point (angle). This can be useful for clarity/layout purposes. To add bend points to the arrow connectors, you should use the  button from the button bar.



If you did not recently, this is a good time to save your workflow model again.

REMARK:

As is the case with all models, if your workflow model contains validation errors these will be shown under the **PROBLEMS** tab from VS Code. Note that, unlike with the other models, a workflow model doesn't show inline errors right away. That would be too distracting (annoying even) for the user building the workflow. To see any validation issue while you are building the workflow, you indeed need to save the model to trigger the validation.

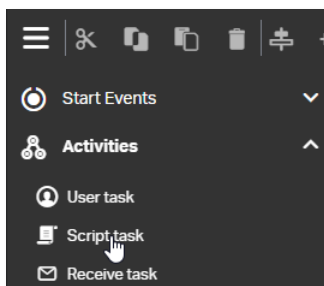
- Now that we have created all activities that set the status of the contract to indicate that the different approval activities are in progress, we can add the actual approval steps themselves. Except for the automatic approval that just sets the contract to be approved, there are indeed three approval steps requiring an approval activity:
 - Calculate solvency:

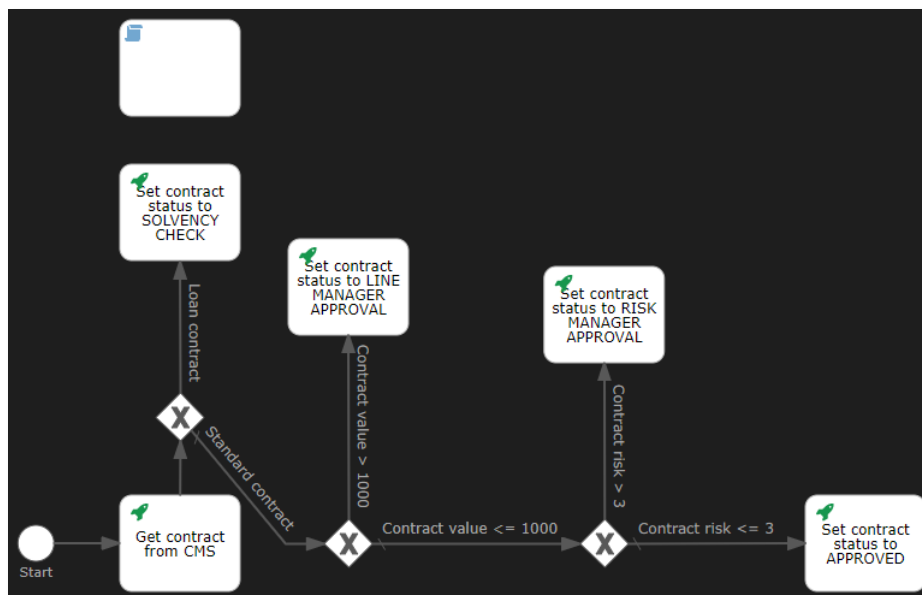
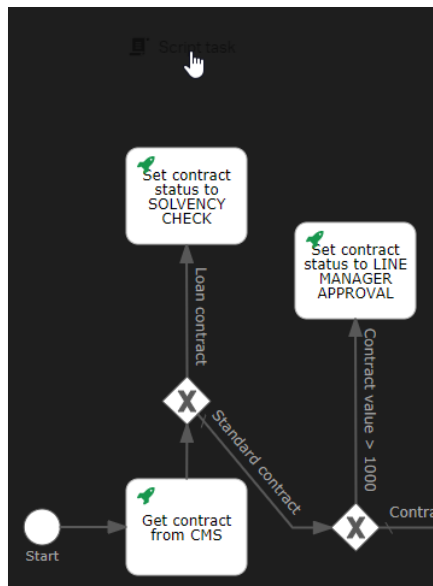
This is an automated check that calculates whether or not the person requesting the approval of the loan contract has enough monthly cash flow to pay back the loan. The logic it follows is that it compares the monthly available cash (based on dividing the yearly income by 12) with the monthly payments (based on the total monthly installments count and the value of the contract). If the monthly payment/cost exceeds 25% of the monthly available cash, the customer is considered not to be solvent (enough) and the loan contract approval will be automatically rejected.
 - Line Manager Approval:

This is a manual approval task by a Line Manager. The Line Manager can choose to approve or reject the contract.
 - Risk Manager Approval:

This is a manual approval task by a Risk Manager. The Risk Manager can choose to approve or reject the contract.

Let's start with the “Calculate solvency” approval task. From the palette (under the **Activities** section), drag and drop a **Script task** onto the canvas above the “Set contract status to SOLVENCY CHECK” http task.





Select the **script task** element you just added and set the element attributes:

- **Name:** Calculate solvency
- **Script format:** JavaScript
- **Script:**

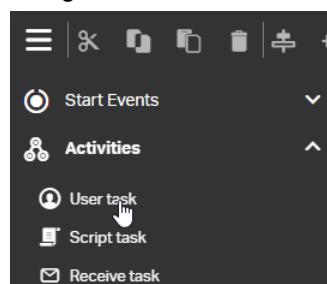
```
var contractDetails = JSON.parse(execution.getVariable("contract"));
var monthlyPayments = contractDetails.properties.value /
contractDetails.properties.monthly_installments;
var monthlyBudget = contractDetails.properties.yearly_income / 12 / 4;

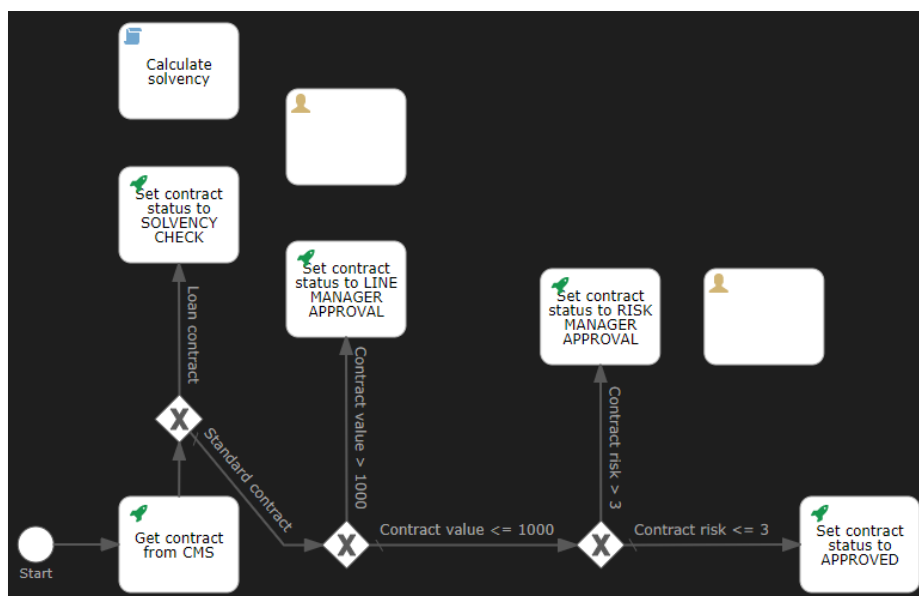
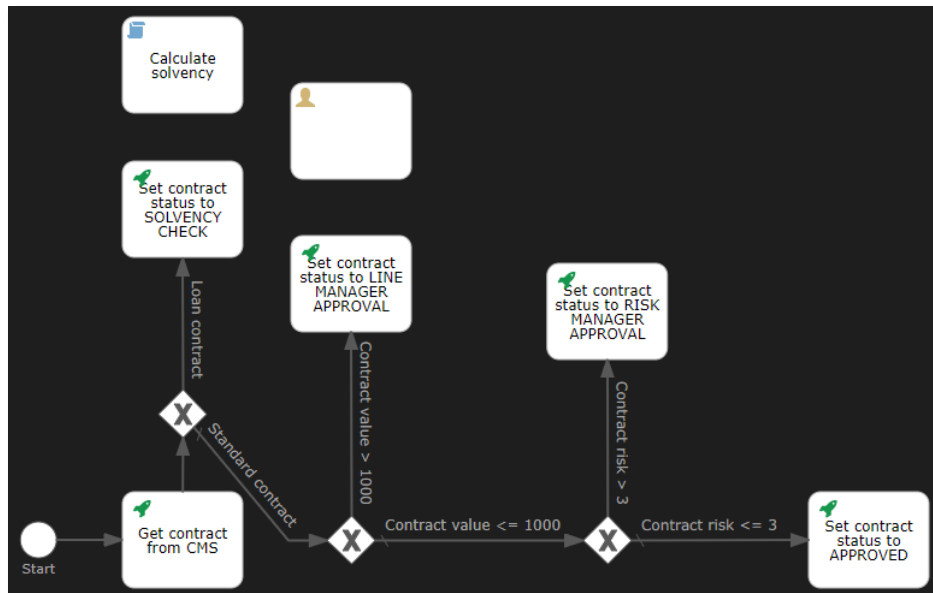
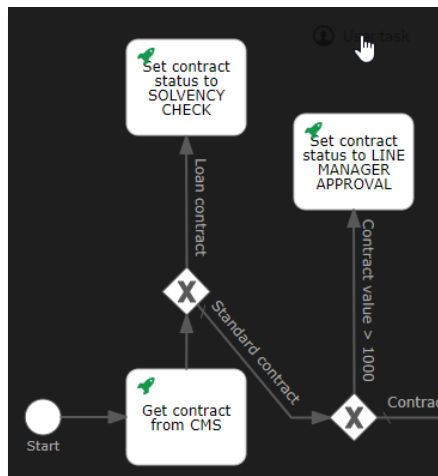
execution.setVariable("solvent", monthlyBudget >= monthlyPayments);
```

The screenshot shows the configuration dialog for a task named "Calculate solvency". The dialog is divided into several sections:

- General:**
 - Name:** Calculate solvency
- Details:**
 - Script format:** JavaScript
 - Script:** var contractDetails ...
 - Auto Store Variables:** ☐
- Execution:**
 - Asynchronous:** ☐
 - Is for compensation:** ☐
 - Exclusive:** false
 - Execution listeners:** No execution listeners configured
- Multi-instance:**
 - Type:** None
 - Cardinality:** No value
 - Collection:** No value
 - Element variable:** No value
 - Completion condition:** No value

Let's now add both the "Line Manager Approval" and "Risk Manager Approval" manual approval tasks. From the palette (under the **Activities** section), drag and drop a **User task** onto the canvas twice. One above the "Set contract status to LINE MANAGER APPROVAL" http task, and one to the right of the "Set contract status to RISK MANAGER APPROVAL" http task.





From left to right, set the element attributes for both manual approval tasks:

- User task for the Line Manager Approval step:
 - **Name: Line Manager Approval**
 - **Delivery options**

REMARK:

The delivery options screen has two tabs, but they are not very clear to read due to the text being in black on a dark background (known issue). If you hover over the tab when it is not selected, you'll see the text more clearly as it would highlight in blue.

Note that overall, this screen requires improving of the clarity. This is a known issue and will be addressed in a future update.

Assignments tab:

- **Task type: Approval**
- **Assignee: \$INITIATOR** (this is an automatically populated process variable that points to the person initiating the workflow, which obviously only works for demo/tutorial purposes, as the assignee often is NOT the workflow initiator)

When you have filled the assignee, you can select the **Outcomes** tab.

Delivery options

Assignments Outcomes Task type: Approval

#	Assignee	Candidate Users	Candidate Groups
1	\$INITIATOR		

☐ Assign to process initiator
 ☐ Dynamic task assignment using a variable
 ☐ Allow process initiator to complete task

Save Cancel

Outcomes tab:

- Modify the customized value for both possible outcomes:

- 1 **Possible outcomes: Approve**
Customized value: approved
- 2 **Possible outcomes: Reject**
Customized value: rejected

- **Task outcome response variable name: approvalStatus**

Once you have filled the **Outcomes** tab, you can click **Save** to save the delivery options and continue filling the manual task properties.

Delivery options ×

Assignments **Outcomes** Task type Approval

#	Possible outcomes	Customized value
1	Approve	<input type="text" value="approved"/>
2	Reject	<input type="text" value="rejected"/>

Task outcome response variable name

Save Cancel

▪ **Exclusive: true**

Line Manager Approval

General

Id No value

Name Line Manager Approva ...

Details

*Delivery options Configured

Task nature No task nature selected

Due date No value

Priority No value

Execution

Asynchronous ☐

Is for compensation ☐

Exclusive true

Skip expression No value

Task listeners No task listeners configured

Execution listeners No execution listeners configured

- User task for the Risk Manager Approval step:
 - **Name: Risk Manager Approval**
 - **Delivery options:**
 - Assignments tab:**
 - **Task type: Approval**
 - **Assignee: \$INITIATOR**
 - Outcomes tab:**
 - Modify the customized value for both possible outcomes:
 - 1 **Possible outcomes: Approve**
Customized value: approved
 - 2 **Possible outcomes: Reject**
Customized value: rejected
 - **Task outcome response variable name: approvalStatus**

Once you have filled the **Outcomes** tab, you can click **Save** to save the delivery options and continue filling the manual task properties.

- **Exclusive: true**

Risk Manager Approval

General

Id

No value

Name

Risk Manager Approva ...

Details

Delivery options

Configured

Task nature

No task nature selected

Due date

No value

Priority

No value

Execution

Asynchronous

☐

Is for compensation

☐

Exclusive

true

Skip expression

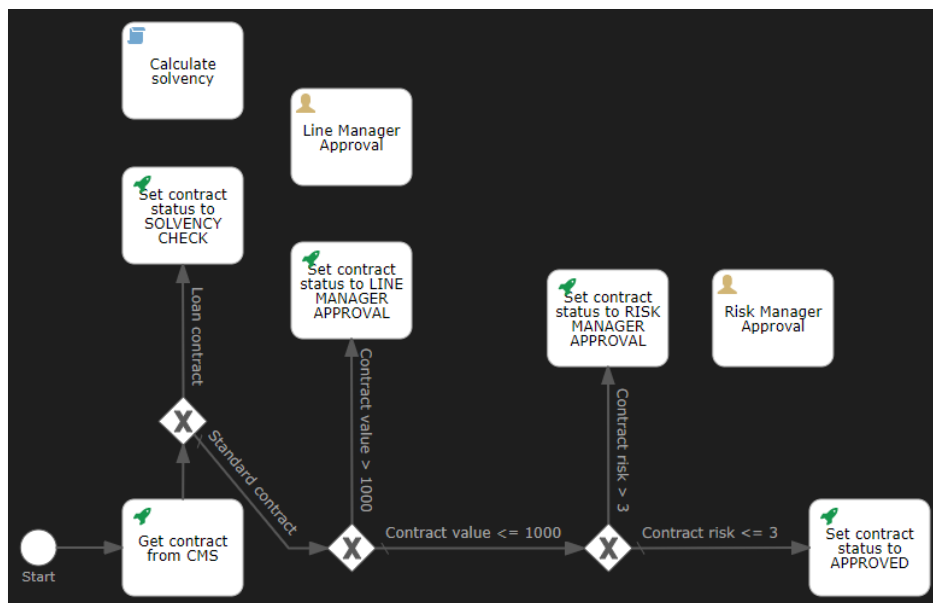
No value

Task listeners

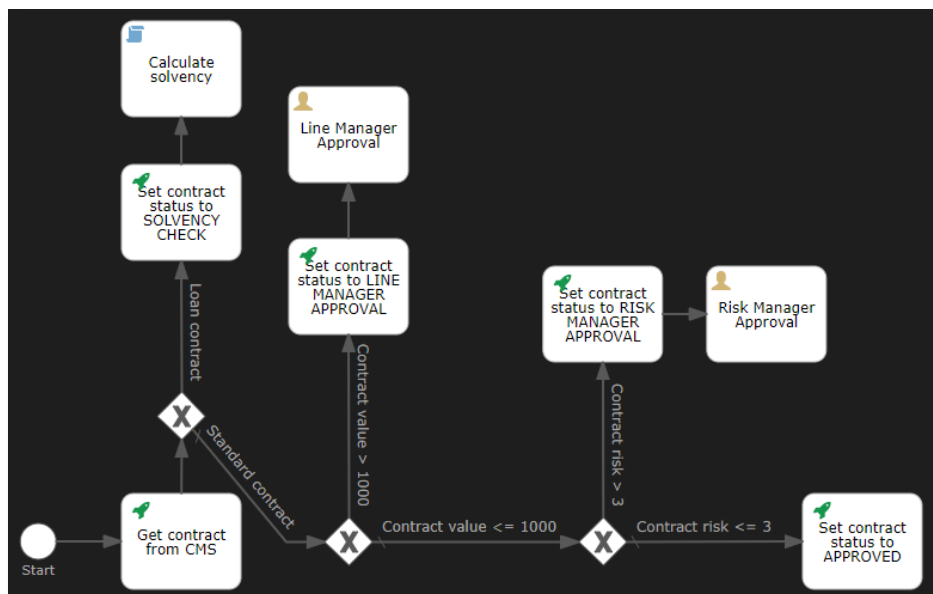
No task listeners configured

Execution listeners

No execution listeners configured



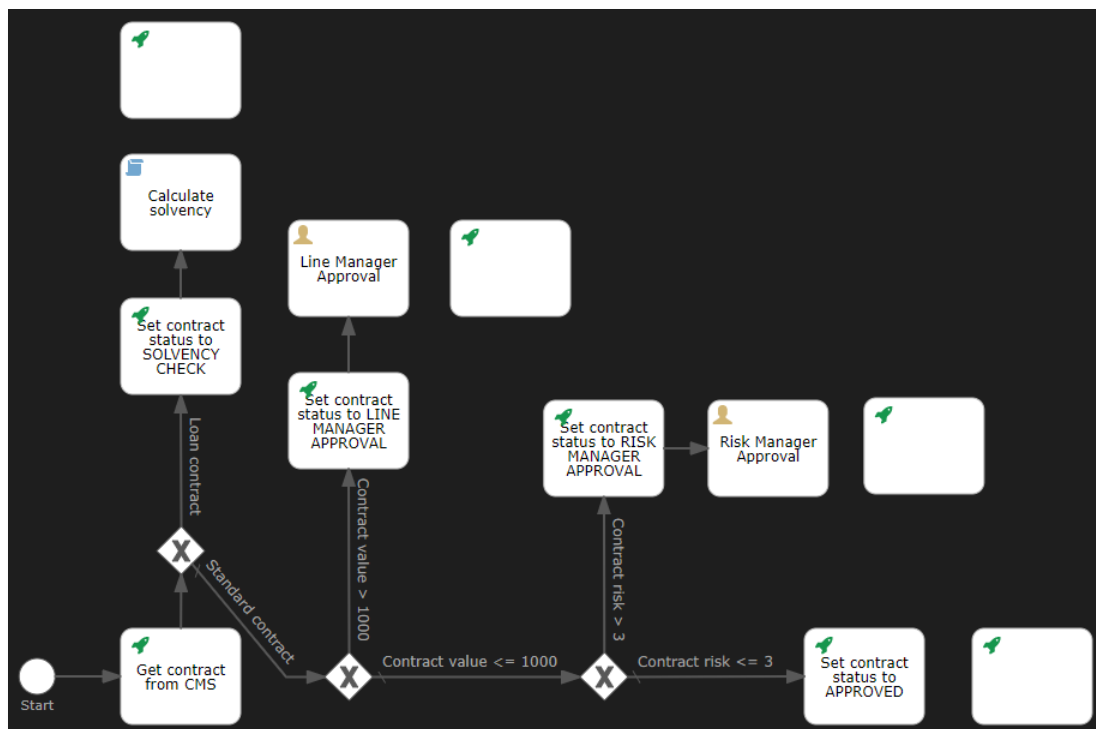
To finish creating the three approval tasks (Calculate solvency, Line Manager Approval, and Risk Manager Approval), you can now connect them from their respective preceding tasks by adding the three corresponding sequence flow arrow connectors:



There is no need to set any attributes for the sequence flow connectors you just added. Please save your workflow model to make sure not to lose your work.

- Now that we have added all approval steps, we can add the http tasks that update the contract with the results of the approval. More specifically, each approval step has its own corresponding (required) CMS trait on the contract instance that is being approved, and we will create an “Update trait” http task for each approval activity.

Note that we have actually already done this before. You might have previously noticed that for the manual approval steps, the corresponding CMS traits have been already partially updated (during the “Set contract status to ...” http tasks) to signal that a manual approval is required. Let’s now add all four “Update trait” (PATCH request) http tasks. Drag and drop four **Http task** elements onto the canvas. One above the “Calculate solvency” script task, one to the right of the “Line Manager Approval” user task, one to the right of the “Risk Manager Approval” user task, and one to the right of the “Set contract status to APPROVED” http task.



Set the attributes of the four new "Update trait" (PATCH request) http tasks as follows:

- Http task to update the Solvency Check trait:
 - **Name:** Update Solvency Check trait
 - **Authentication details:** Use current authentication token
 - **Request method:** PATCH
 - **Request URL:** \${base_url}/cms/instances/file/ca_contract/\${contract_id}
 - **Request headers:** Content-Type: application/json
 - **Request body:**

```
{
  "traits": {
    "ca_approval": {
      "Solvency Check": {
        "is_required": true,
        "has_been_granted": ${solvent},
        "approver": "SYSTEM",
        "approver_role": "Solvency Check",
        "approval_date": "${contract.update_time}"
      }
    }
  }
}
```
 - **Response variable name:** contract
 - **Save response as JSON:** true
 - **Exclusive:** true

Update Solvency Check trait

General

Id	No value
Name	Update Solvency Chec ...

Details

Authentication details	Authentication configured
*Request method	PATCH
*Request URL	\${base_url}/cms/inst ...
Request headers	Content-Type: applic ...
Request body	{ "traits": { ...
Request body encoding	No value
Request timeout	No value
Disallow redirects	No value
Fail status codes	No value
Handle status codes	No value
Ignore exception	No value
Response variable name	contract
Save request variables	No value
Save response status, headers	No value
Result variable prefix	No value
Save response as a transient variable	No value
Save response as JSON	true

Execution

Asynchronous	<input type="checkbox"/>
Is for compensation	<input type="checkbox"/>
Exclusive	true

- Http task to update the Line Manager Approval trait:
 - **Name:** Update Line Manager Approval trait
 - **Authentication details:** Use current authentication token
 - **Request method:** PATCH
 - **Request URL:** \${base_url}/cms/instances/file/ca_contract/\${contract_id}
 - **Request headers:** Content-Type: application/json
 - **Request body:**

```
{
  "traits": {
    "ca_approval": {
      "Line Manager Approval": {
        "is_required": true,
        "has_been_granted": "${approvalStatus == "approved"}",
        "approver": "${contract.updated_by.email}",
        "approver_role": "Line Manager",
        "approval_date": "${contract.update_time}"
      }
    }
  }
}
```
 - **Response variable name:** contract
 - **Save response as JSON:** true
 - **Exclusive:** true

Update Line Manager Approval trait

General

Id	No value
Name	Update Line Manager ...

Details

Authentication details	Authentication configured
Request method	PATCH
Request URL	\${base_url}/cms/inst ...
Request headers	Content-Type: applic ...
Request body	{ "traits": { ...
Request body encoding	No value
Request timeout	No value
Disallow redirects	No value
Fail status codes	No value
Handle status codes	No value
Ignore exception	No value
Response variable name	contract
Save request variables	No value
Save response status, headers	No value
Result variable prefix	No value
Save response as a transient variable	No value
Save response as JSON	true

Execution

Asynchronous	<input type="checkbox"/>
Is for compensation	<input type="checkbox"/>
Exclusive	true

- Http task to update the Risk Manager Approval trait:
 - **Name:** Update Risk Manager Approval trait
 - **Authentication details:** Use current authentication token
 - **Request method:** PATCH
 - **Request URL:** \${base_url}/cms/instances/file/ca_contract/\${contract_id}
 - **Request headers:** Content-Type: application/json
 - **Request body:**

```
{
  "traits": {
    "ca_approval": {
      "Risk Manager Approval": {
        "is_required": true,
        "has_been_granted": "${approvalStatus == "approved"}",
        "approver": "${contract.updated_by.email}",
        "approver_role": "Risk Manager",
        "approval_date": "${contract.update_time}"
      }
    }
  }
}
```
 - **Response variable name:** contract
 - **Save response as JSON:** true
 - **Exclusive:** true

Update Risk Manager Approval trait

General

Id	No value
Name	Update Risk Manager ...

Details

Authentication details	Authentication configured
Request method	PATCH
Request URL	\${base_url}/cms/inst ...
Request headers	Content-Type: applic ...
Request body	{"traits": { ...
Request body encoding	No value
Request timeout	No value
Disallow redirects	No value
Fail status codes	No value
Handle status codes	No value
Ignore exception	No value
Response variable name	contract
Save request variables	No value
Save response status, headers	No value
Result variable prefix	No value
Save response as a transient variable	No value
Save response as JSON	true

Execution

Asynchronous	<input type="checkbox"/>
Is for compensation	<input type="checkbox"/>
Exclusive	true

- Http task to update the Automatic Approval trait:
 - **Name:** Update Automatic Approval trait
 - **Authentication details:** Use current authentication token
 - **Request method:** PATCH
 - **Request URL:** \${base_url}/cms/instances/file/ca_contract/\${contract_id}
 - **Request headers:** Content-Type: application/json
 - **Request body:**

```
{
  "traits": {
    "ca_approval": {
      "Automatic Approval": {
        "has_been_granted": true,
        "approver": "SYSTEM",
        "approver_role": "Automatic Approval",
        "approval_date": "${contract.update_time}"
      }
    }
  }
}
```
 - **Response variable name:** contract
 - **Save response as JSON:** true
 - **Exclusive:** true

Update Automatic Approval trait

General

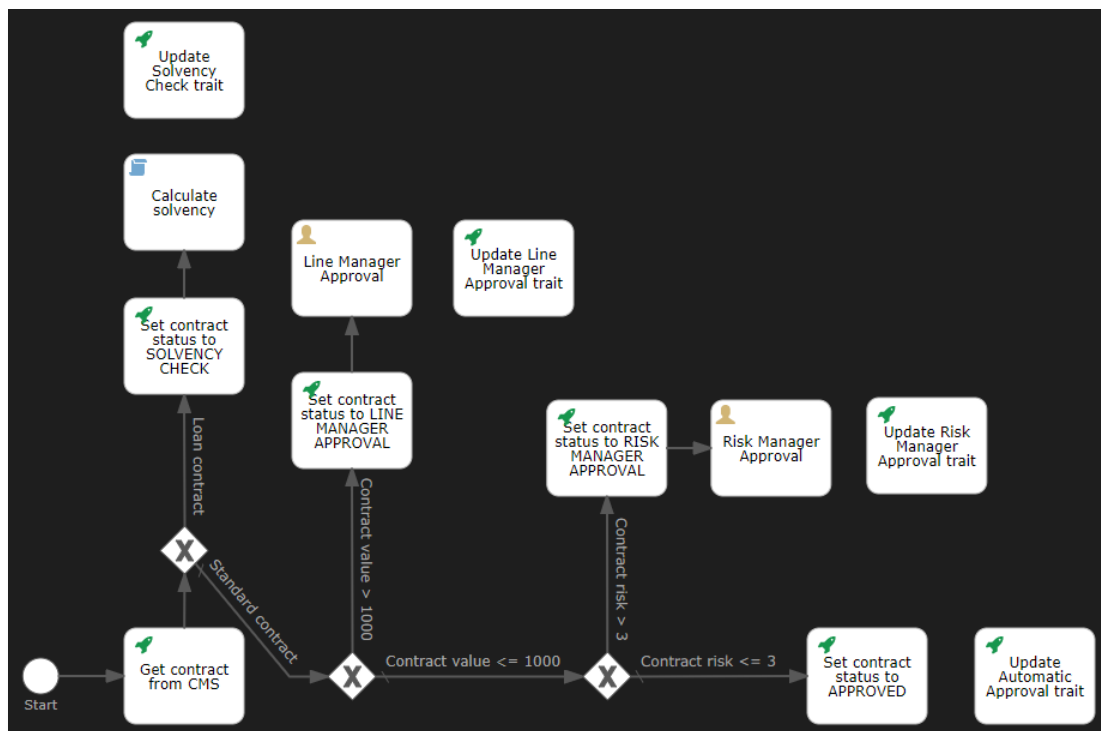
Id	No value
Name	Update Automatic App ...

Details

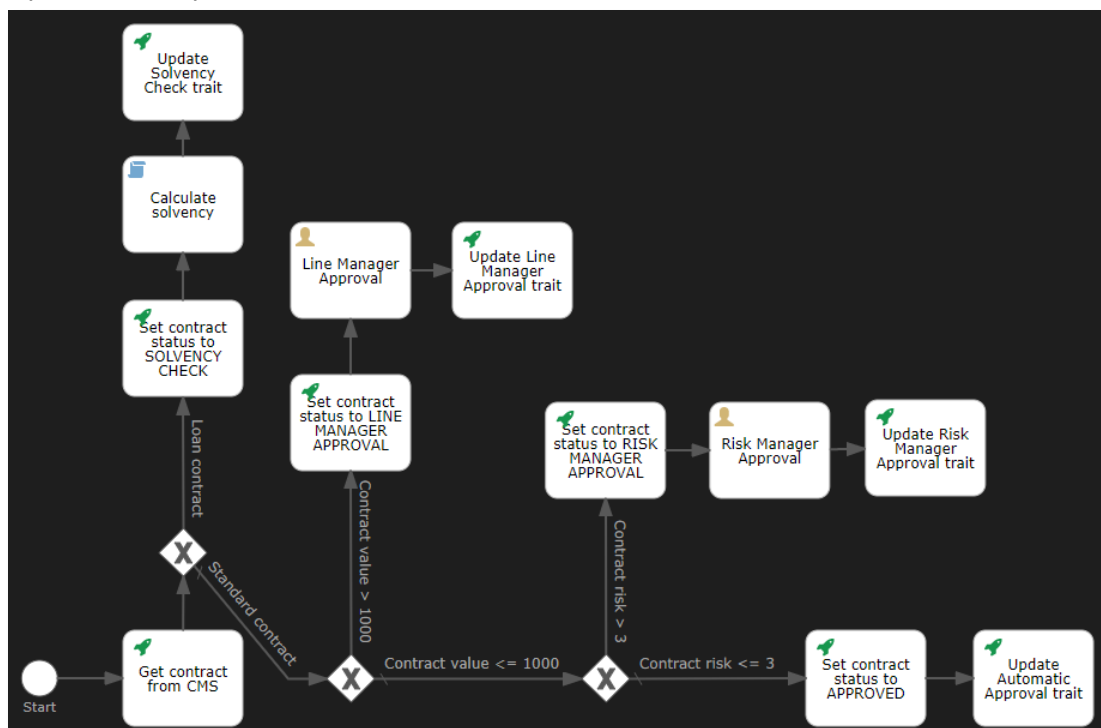
Authentication details	Authentication configured
Request method	PATCH
Request URL	\${base_url}/cms/inst ...
Request headers	Content-Type: applic ...
Request body	{ "traits": { ...
Request body encoding	No value
Request timeout	No value
Disallow redirects	No value
Fail status codes	No value
Handle status codes	No value
Ignore exception	No value
Response variable name	contract
Save request variables	No value
Save response status, headers	No value
Result variable prefix	No value
Save response as a transient variable	No value
Save response as JSON	true

Execution

Asynchronous	<input type="checkbox"/>
Is for compensation	<input type="checkbox"/>
Exclusive	true



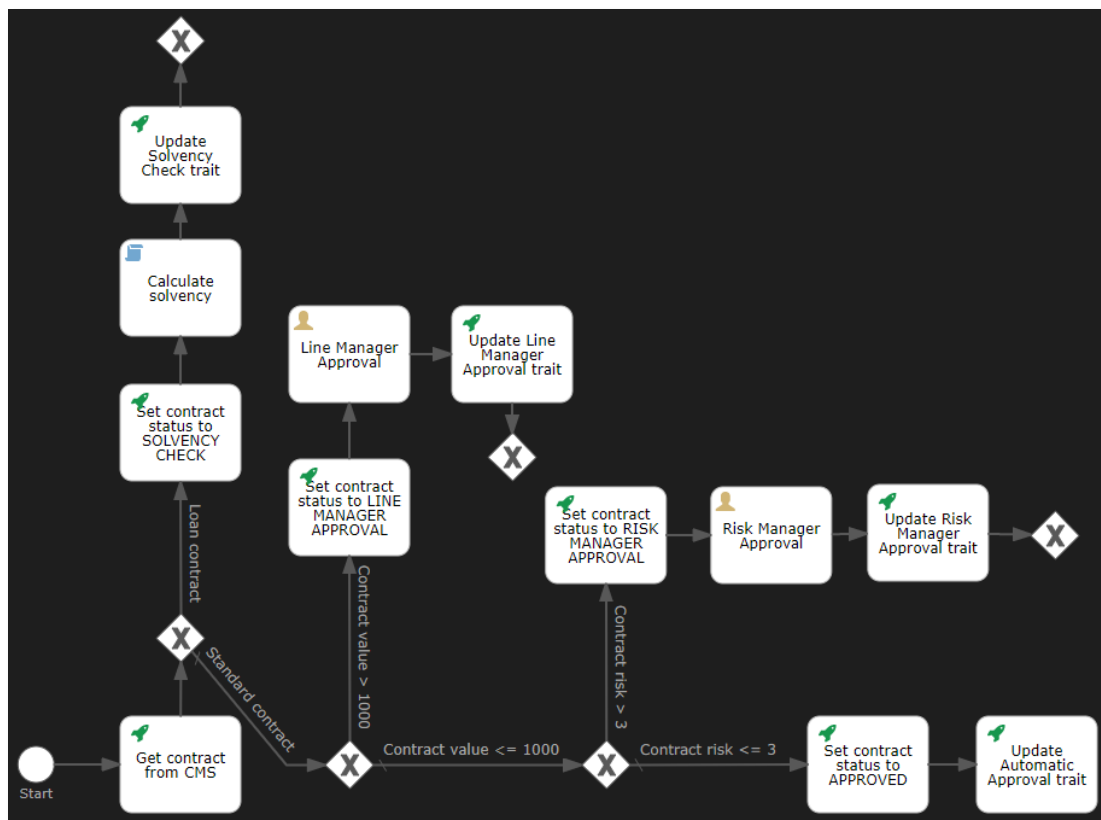
You can now add the four sequence flow arrow connectors that link the previous tasks to the “Update trait” http tasks:



There is no need to set any attributes for the sequence flow connectors you just added. Please save your workflow model to make sure not to lose your work.

- The result of the three approval tasks that have the option to approve or reject the contract (Solvency check, Line Manager Approval, and Risk Manager Approval) is now known. So, it is now possible, depending on whether the approval step resulted in approval being granted or not, to advance to the next step in the approval process when approved, or terminate to the process via the rejection flow when rejected.

To implement this logic of choosing the correct process route, based on the results of the approval steps, we will add (you might have guessed it already) three new **Exclusive gateway** elements. In the same way as with the previously created exclusive gateways, you can also already add the sequence flow arrow connectors coming from the previous (approval) task elements.



There's no need to set any attributes for the sequence flow arrow connectors.

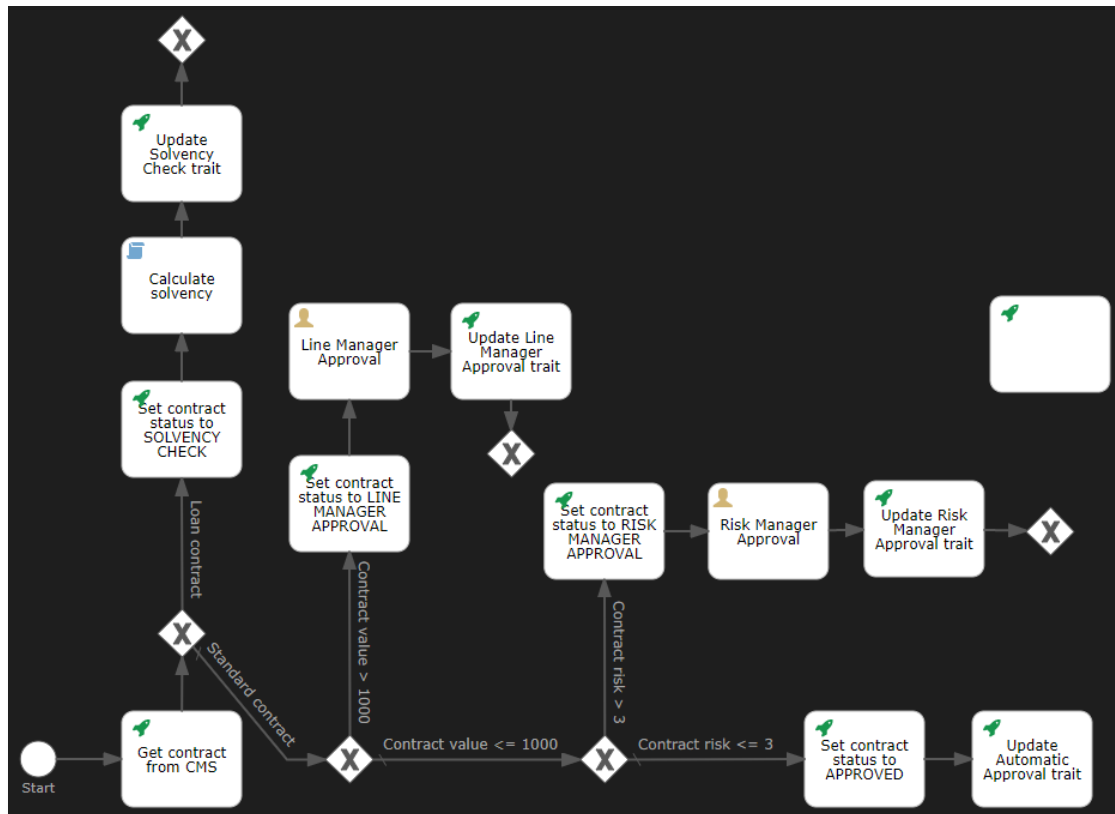
For all three new exclusive gateway elements, set the **Exclusive** attribute to **true**:

My Process

General	
Id	No value
Name	No value
Details	
Flow order	No sequence flow order determined
Execution	
Asynchronous	<input type="checkbox"/>
Exclusive	<input checked="" type="checkbox"/> true

- Before we can connect the exclusive gateways to the two possible next process steps (depending on being approved or rejected), we need to create the activity that represents what needs to happen when the approval task resulted in a rejection, as this missing activity is one of the two outcomes/options. More specifically, we need to create the http task that sets the contract's status to REJECTED.

Proceed by dragging and dropping a new **Http task** element onto the canvas right above the exclusive gateway element that is most to the right, out of the three new exclusive gateway elements.



Very much like with the other “Set contract status to ...” tasks, set the attributes of this new “Set contract status to REJECTED” http task as follows:

- **Name: Set contract status to REJECTED**
- **Authentication details: Use current authentication token**
- **Request method: PATCH**
- **Request URL: \${base_url}/cms/instances/file/ca_contract/\${contract_id}**
- **Request headers: Content-Type: application/json**
- **Request body:**

```
{
  "properties": {
    "status": "REJECTED"
  }
}
```
- **Response variable name: contract**
- **Save response as JSON: true**
- **Exclusive: true**

Set contract status to REJECTED

General

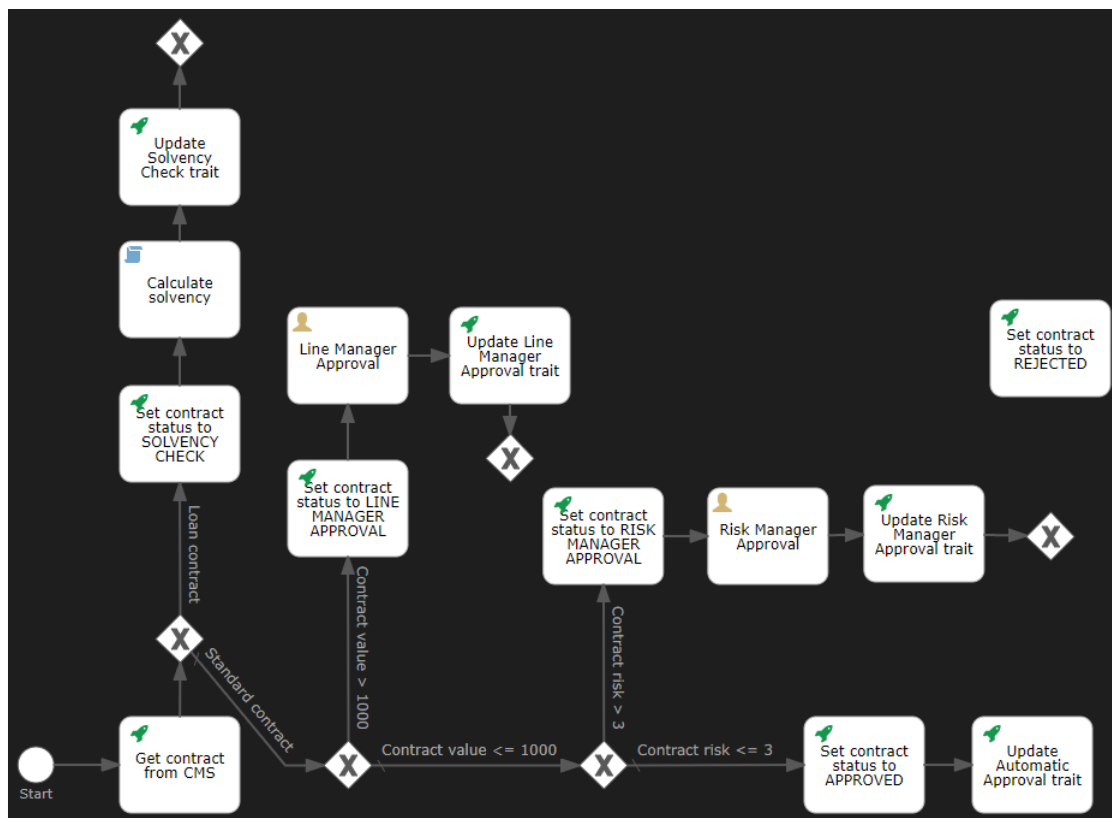
Id	No value
Name	Set contract status ...


Details

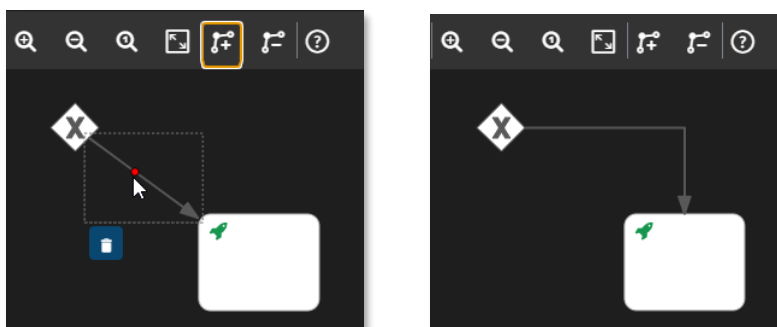
Authentication details	Authentication configured
Request method	PATCH
Request URL	\${base_url}/cms/inst ...
Request headers	Content-Type: applic ...
Request body	{ "properties": { ...
Request body encoding	No value
Request timeout	No value
Disallow redirects	No value
Fail status codes	No value
Handle status codes	No value
Ignore exception	No value
Response variable name	contract
Save request variables	No value
Save response status, headers	No value
Result variable prefix	No value
Save response as a transient variable	No value
Save response as JSON	true

Execution

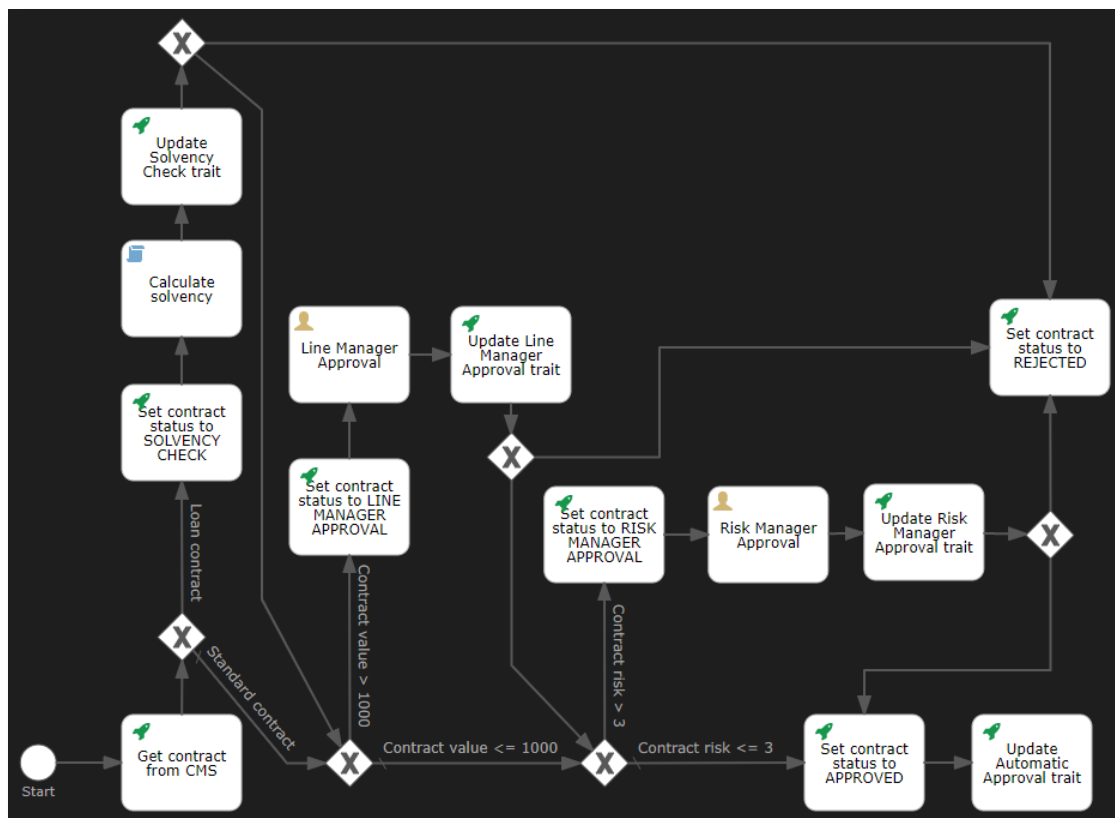
Asynchronous	<input type="checkbox"/>
Is for compensation	<input type="checkbox"/>
Exclusive	true
Skip expression	No value



- We can now connect the three approved/rejected exclusive gateways to their two possible next process steps. For each of the three new exclusive gateway elements, use two sequence flow arrow connectors to connect them with their respective subsequent process step elements. Remember that to add bend points (angles) to the arrow connectors, you should use the  button from the button bar.



Once done, you should have created 6 new sequence flow arrow connectors, as shown in the next screen shot.



What's left for the three approved/rejected exclusive gateways is to set the attributes of the different outgoing sequence flows. From left to right, set the sequence flow attributes as follows:

- First exclusive gateway:
 - Sequence flow going to the "Set contract status to REJECTED" http task:
 - **Name:** Not solvent
 - **Flow condition:** `#{!solvent}`
 - **Default flow:** `<not checked>`

Not solvent

General

Id

No value

Name

Not solvent

Details

One * property is required

Flow condition

`#{!solvent}`

Default flow

☐

Skip expression

No value

Execution

Execution listeners

No execution listeners configured

- Sequence flow going to the contract value checking exclusive gateway:
 - **Name: Solvent**
 - **Default flow: <checked>**

Solvent

General

Id	No value
Name	Solvent

Details

One * property is required

*Flow condition	No condition set
*Default flow	<input checked="" type="checkbox"/>
Skip expression	No value

Execution

Execution listeners	No execution listeners configured
---------------------	-----------------------------------

- Second exclusive gateway:
 - Sequence flow going to the “Set contract status to REJECTED” http task:
 - **Name: Rejected**
 - **Flow condition: \${approvalStatus == "rejected"}**
 - **Default flow: <not checked>**

Rejected

General

Id	No value
Name	Rejected

Details

One * property is required

*Flow condition	\${approvalStatus ==
*Default flow	<input type="checkbox"/>
Skip expression	No value

Execution

Execution listeners	No execution listeners configured
---------------------	-----------------------------------

- Sequence flow going to the contract risk checking exclusive gateway:
 - **Name: Approved**
 - **Default flow: <checked>**

Approved

General

Id	No value
Name	Approved

Details

One * property is required

*Flow condition	No condition set
*Default flow	<input checked="" type="checkbox"/>
Skip expression	No value

Execution

Execution listeners	No execution listeners configured
---------------------	-----------------------------------

- Third exclusive gateway:
 - Sequence flow going to the “Set contract status to REJECTED” http task:
 - **Name: Rejected**
 - **Flow condition: `${approvalStatus == "rejected"}`**
 - **Default flow: <not checked>**

Rejected

General

Id	No value
Name	Rejected

Details

One * property is required

*Flow condition	<code>\${approvalStatus ==</code>
*Default flow	<input type="checkbox"/>
Skip expression	No value

Execution

Execution listeners	No execution listeners configured
---------------------	-----------------------------------

- Sequence flow going to the “Set contract status to APPROVED” http task:
 - **Name: Approved**
 - **Default flow: <checked>**

Approved

General

Id	No value
Name	Approved

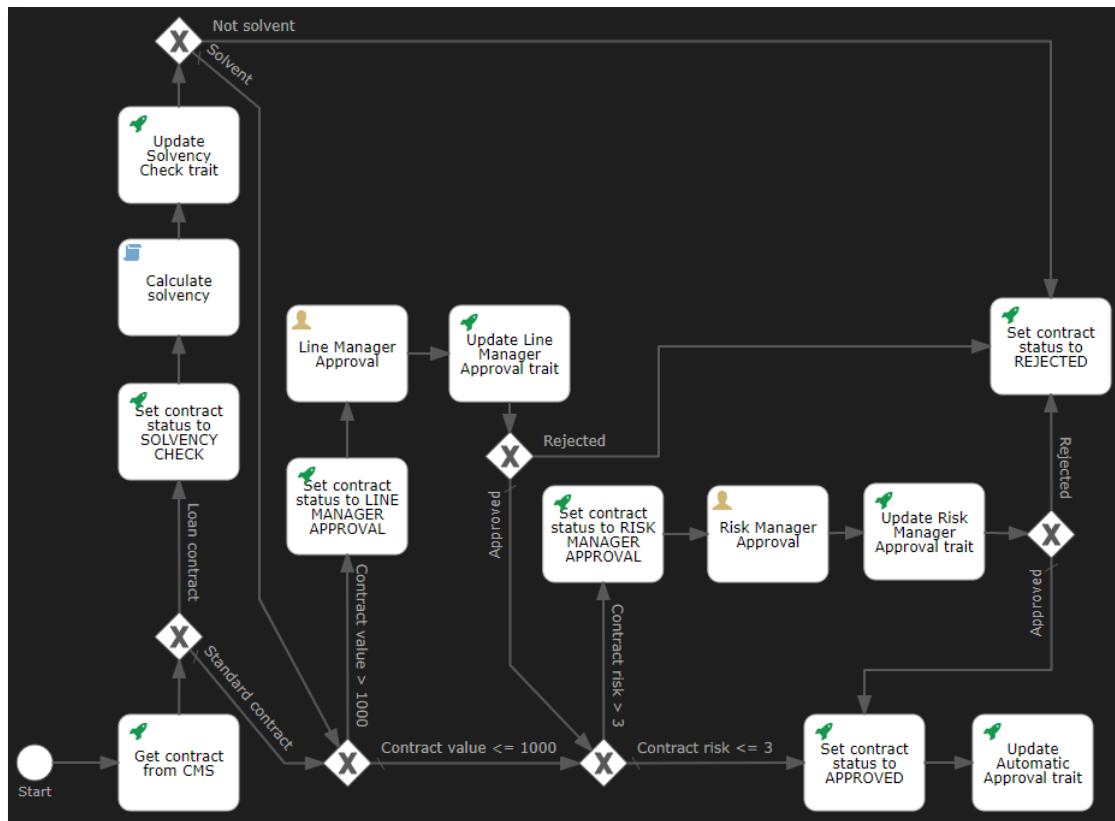
Details

One * property is required

*Flow condition	No condition set
*Default flow	<input checked="" type="checkbox"/>
Skip expression	No value

Execution

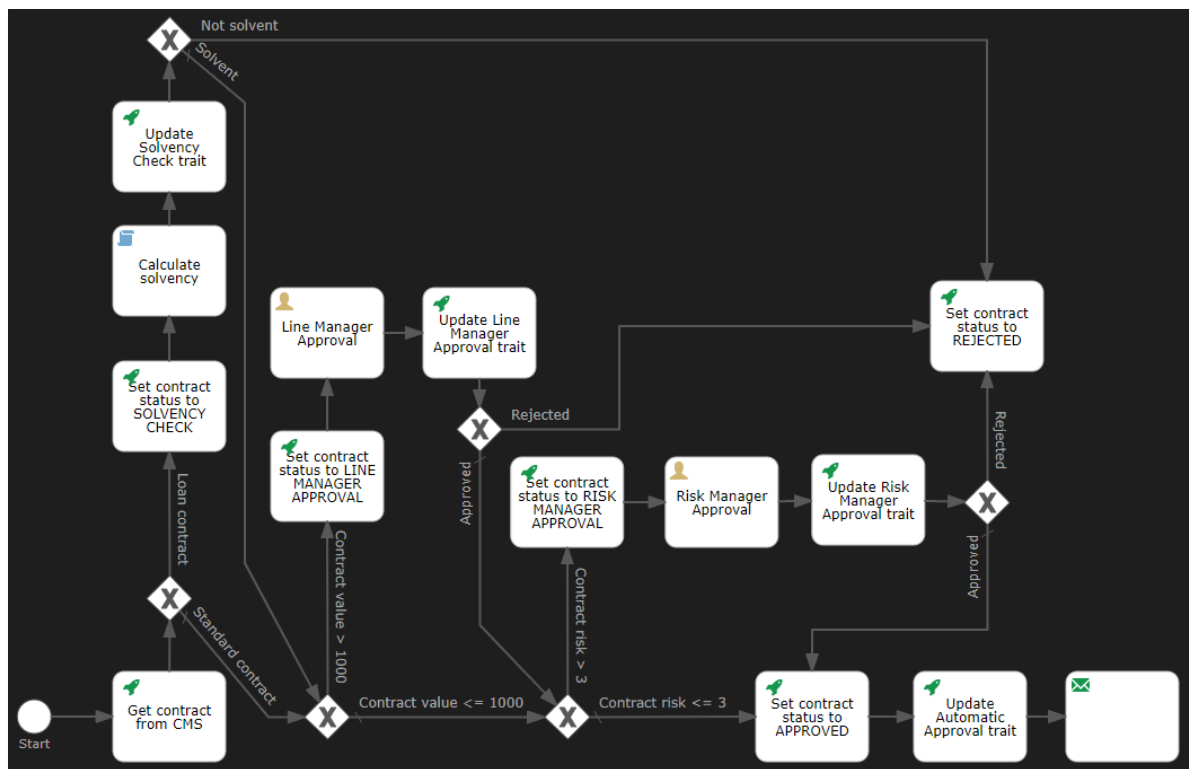
Execution listeners	No execution listeners configured
---------------------	-----------------------------------



This is again a good time to save your work.

- We have now built the main flow of the contract approval business process. The only thing left, before adding an end event to the workflow model, is to add an email task that sends an email to the person having requested the approval of the contract (i.e.: the customer) to inform them about the approval status (APPROVED or REJECTED) of the contract.

To do this, drag and drop a **Mail task** from the palette (under the **Activities** section) onto the canvas to the right of the “Update Automatic Approval trait” http task. You can also immediately add the sequence flow arrow connector, as there are no specific sequence flow attributes to set.



Fill the mail task attributes as follows:

- Name: Send Email on contract status
- To: \${contract.properties.requester_email}
- From: noreply@mycompany.com
- Subject: Contract Approval Status
- Text:
 - Contract: \${contract.name}
 - Status: \${contract.properties.status}
- Exclusive: true

Send Email on contract status

General

Id No value

Name Send Email on contra ...

Details

Headers No value

To \${contract.propertie ...}

From noreply@mycompany.co ...

Subject Contract Approval St ...

Cc No value

Bcc No value

Text Contract: \${contract ...}

TextVar No value

Html No value

HtmlVar No value

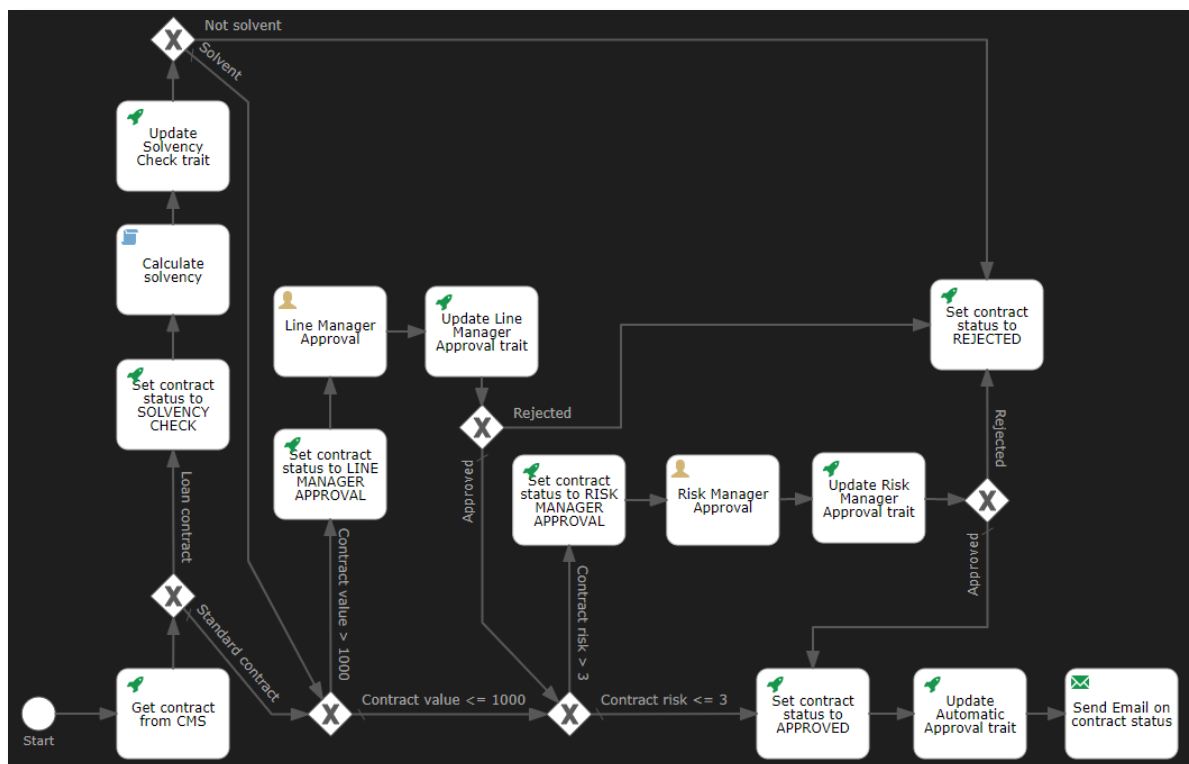
Charset No value

Execution

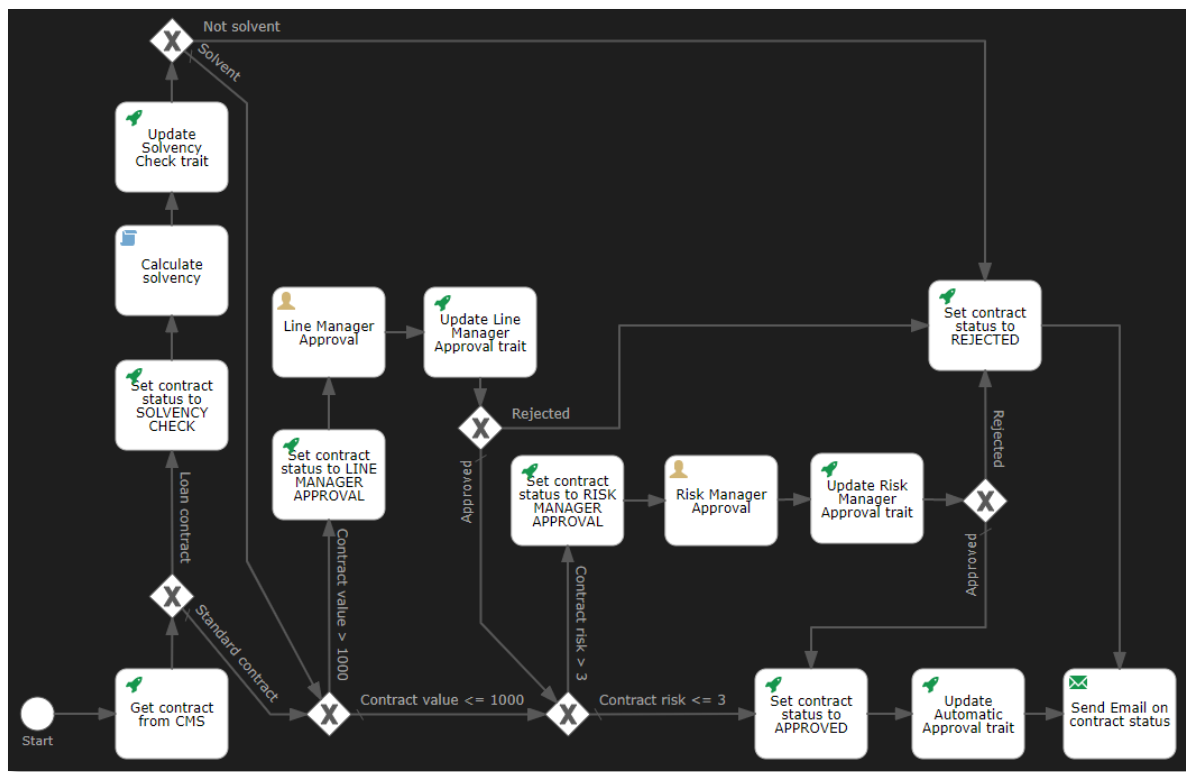
Asynchronous ☐

Is for compensation ☐

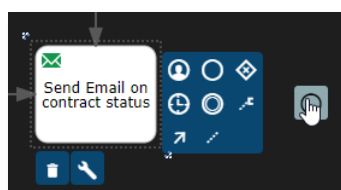
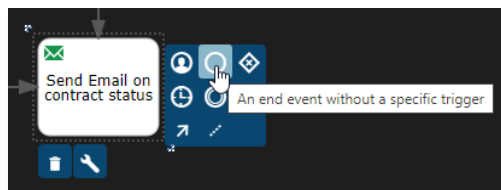
Exclusive true

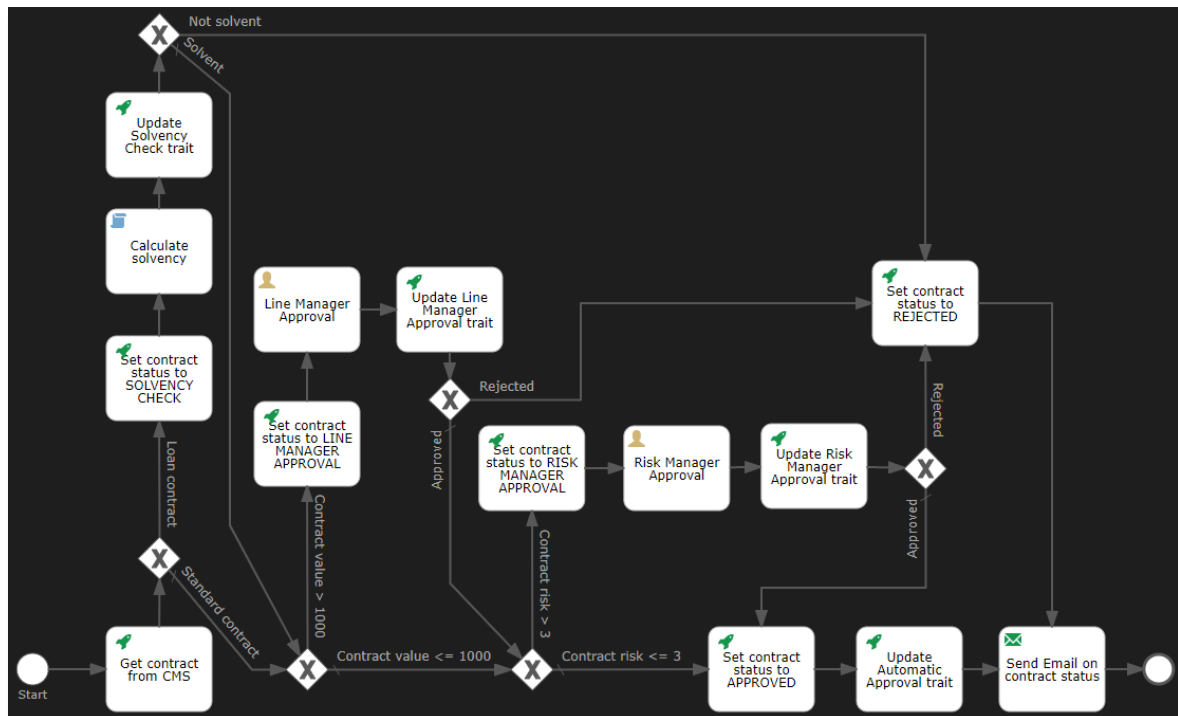


To ensure the REJECTED contract approval status email gets sent as well, connect the “Set contract status to REJECTED” http task to the new “Send Email on contract status” task.



- You can now add the end event as termination event of your business process. Do this by simply selecting the “Send Email on contract status” **email task** and drag and dropping the **end event** (middle top icon) to the right of the email task.

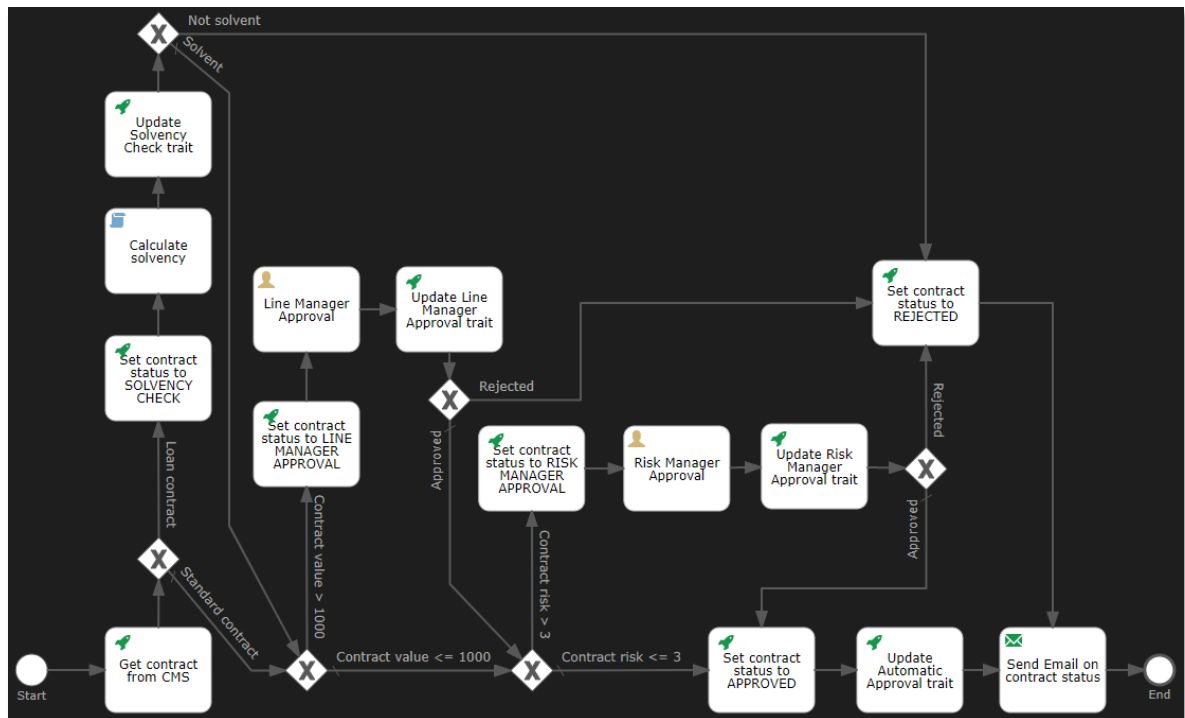




Set the attributes for the end event as follows:

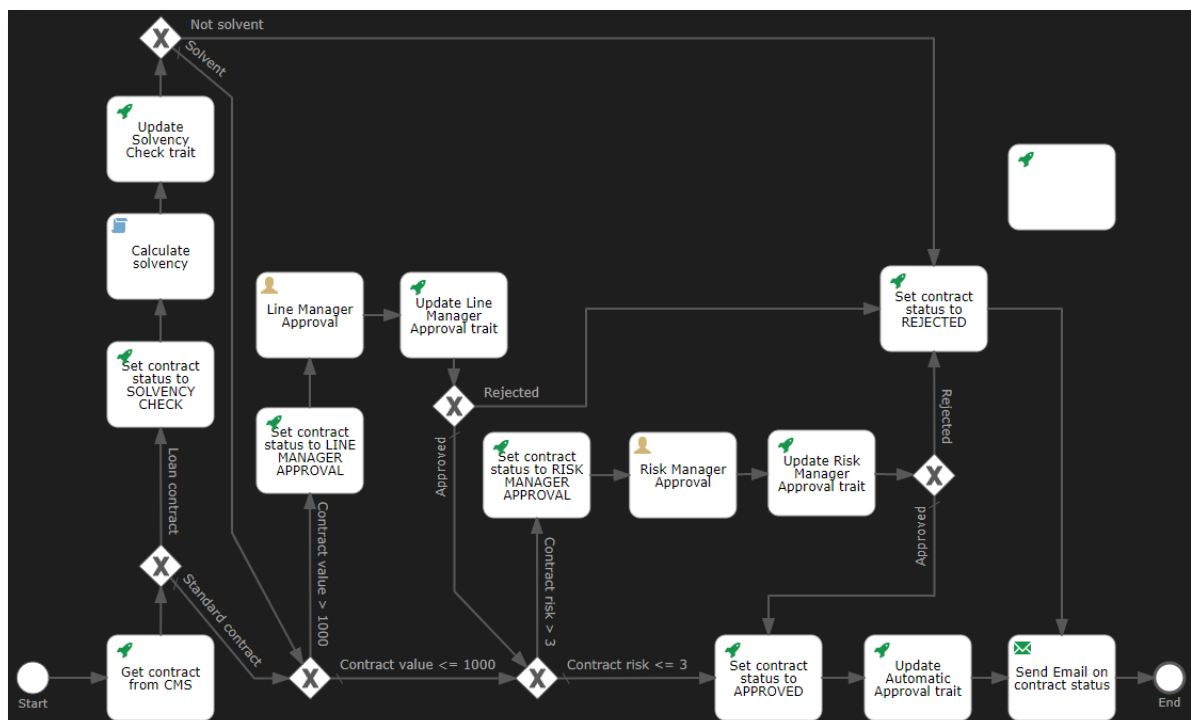
- **Name:** End

End	
General	
Id	No value
Name	End
Execution	
Execution listeners	No execution listeners configured



Again, please save the workflow model to ensure you don't lose any work.

- You have now implemented the main logic of the contract approval business process. Let's add one more activity to address a special case. More specifically, we want the manual approval tasks (by Line Manager and Risk Manager) to expire if a certain wait period has been exceeded, so that approval tasks don't get stuck forever in the approver's inbox. For that, we will add one more http task that marks the contract as EXPIRED (as special exception status). So, please drag and drop a new **Http task** element onto the canvas above the "Send Email on contract status" email task at the top end of the canvas (so that it is higher than the "Set contract status to REJECTED" http task).



Fill the http task attributes as follows:

- **Name: Set contract status to EXPIRED**
- **Authentication details: Use current authentication token**
- **Request method: PATCH**
- **Request URL: \${base_url}/cms/instances/file/ca_contract/\${contract_id}**
- **Request headers: Content-Type: application/json**
- **Request body:**

```
{
  "properties": {
    "status": "EXPIRED"
  }
}
```
- **Response variable name: contract**
- **Save response as JSON: true**
- **Exclusive: true**

Set contract status to EXPIRED

General

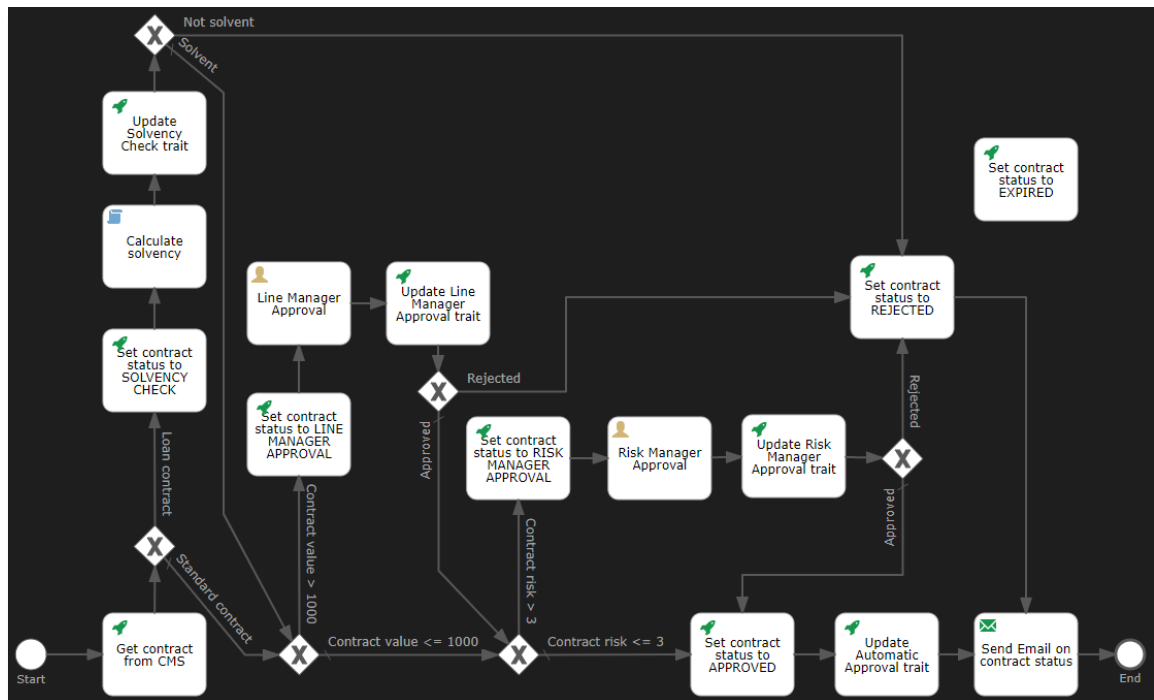
Id	No value
Name	Set contract status ...

Details

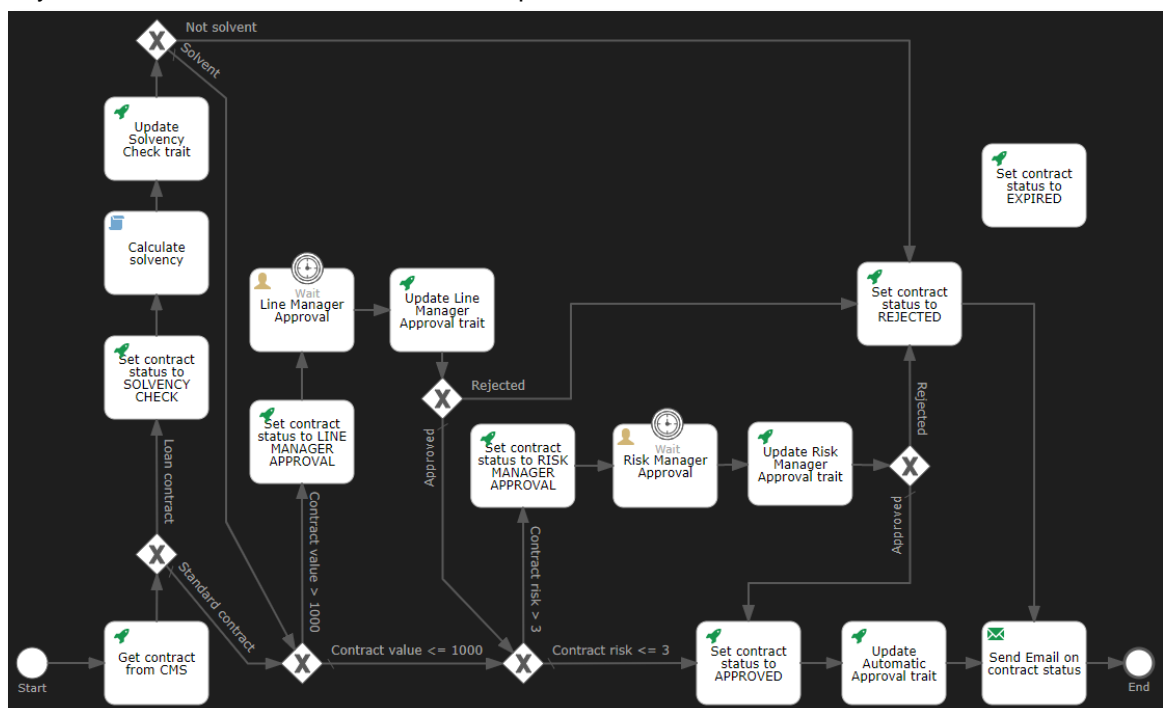
Authentication details	Authentication configured
Request method	PATCH
Request URL	\${base_url}/cms/inst ...
Request headers	Content-Type: applic ...
Request body	{ "properties": { ...
Request body encoding	No value
Request timeout	No value
Disallow redirects	No value
Fail status codes	No value
Handle status codes	No value
Ignore exception	No value
Response variable name	contract
Save request variables	No value
Save response status, headers	No value
Result variable prefix	No value
Save response as a transient variable	No value
Save response as JSON	true

Execution

Asynchronous	<input type="checkbox"/>
Is for compensation	<input type="checkbox"/>
Exclusive	true
Skip expression	No value



Before you can connect the sequence flow connector arrows from the two manual approval tasks to the “Set contract status to EXPIRED” http task, there is one more thing you need to consider. This is not a normal sequence flow, as it only gets triggered when the manual approval task times out. More specifically, you need to add a boundary timer event to both approval tasks. To add the “Wait for timeout” timer event to the “Line Manager Approval” and “Risk Manager Approval” user tasks, drag and drop (from the **Boundary Events** section on the palette) a **Boundary timer event** on top of both the **Line Manager Approval** and **Risk Manager Approval** http tasks (which will show green to indicate when the boundary event can be released/dropped to be linked to them). Both start timer events should be aligned to the top of the http tasks, so that they sit half inside and half outside of the http task element.



Set the attributes for both start timer events as follows:

- **Name:** Wait
- **Time duration (e.g. PT5M):** PT5M
- **Cancel activity:** <checked>

Wait

General

Id

No value

Name

Wait

Details

One * property is required

Time cycle (e.g. R3/PT10H)

No value

Time date in ISO-8601

No value

Time duration (e.g. PT5M)

PT5M

Time End Date in ISO-8601

No value

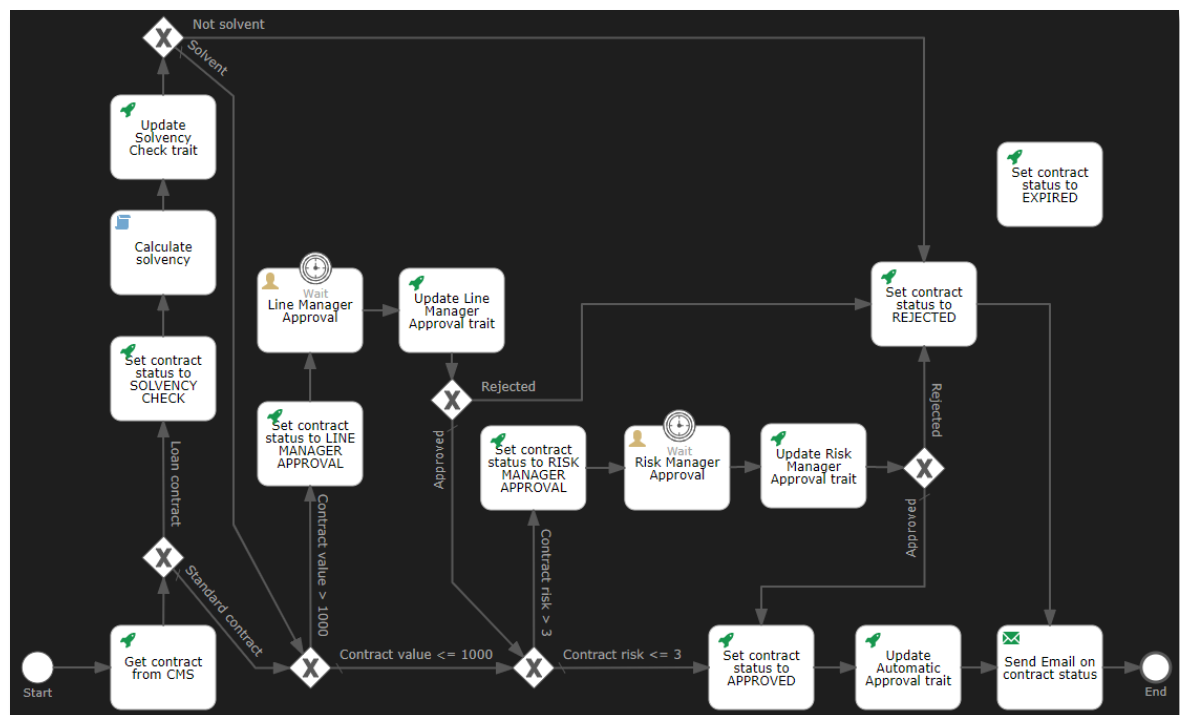
Cancel activity

☒

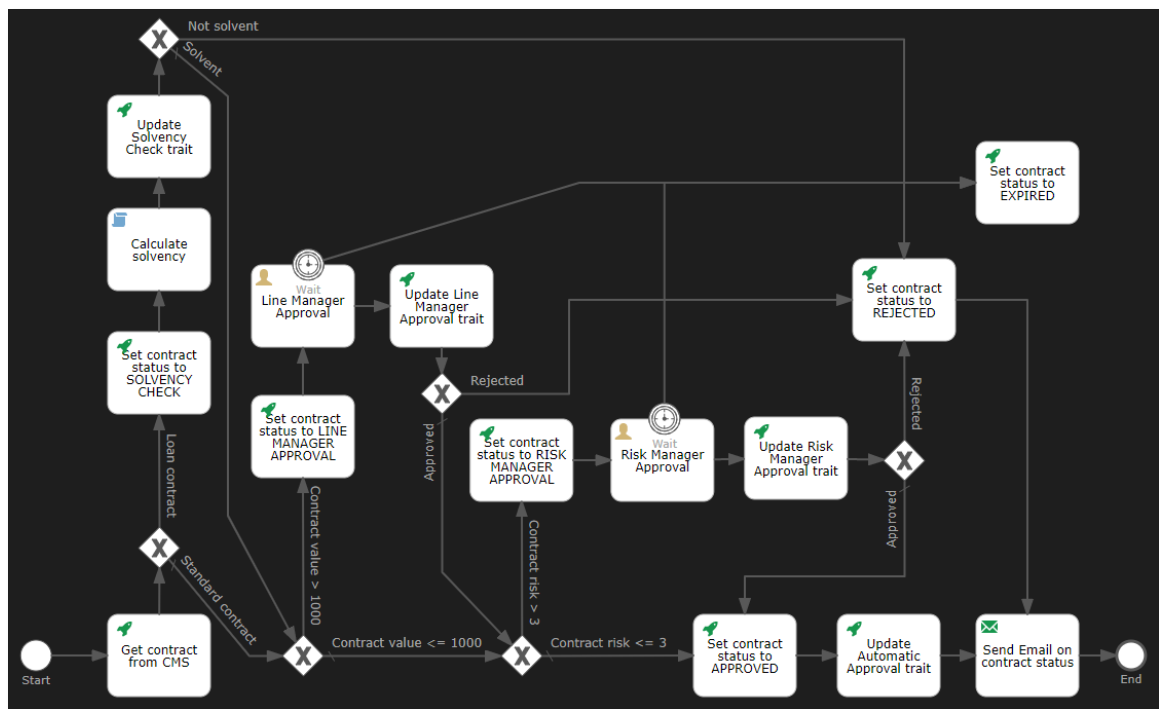
Execution

Execution listeners

No execution listeners configured



You can now connect both “Wait” start timer events to the “Set contract status to EXPIRED” http task with sequence flow arrow connectors.



Set the attributes for the two new sequence flow arrow connectors to the following:

- For the sequence flow originating from the “Line Manager Approval” http task:
 - **Name:** Wait timeout exceeded
 - **Flow condition:** `${contract.properties.status == "LINE MANAGER APPROVAL"}`

Wait timeout exceeded

General

Id

No value

Name

Wait timeout exceede ...

Details

One * property is required

Flow condition

`${contract.propertie`

Default flow

☐

Skip expression

No value

- For the sequence flow originating from the “Risk Manager Approval” http task:
 - **Name:** Wait timeout exceeded
 - **Flow condition:** `${contract.properties.status == "RISK MANAGER APPROVAL"}`

Wait timeout exceeded

General

Id

No value

Name

Wait timeout exceede ...

Details

One * property is required

Flow condition

`${contract.propertie`

Default flow

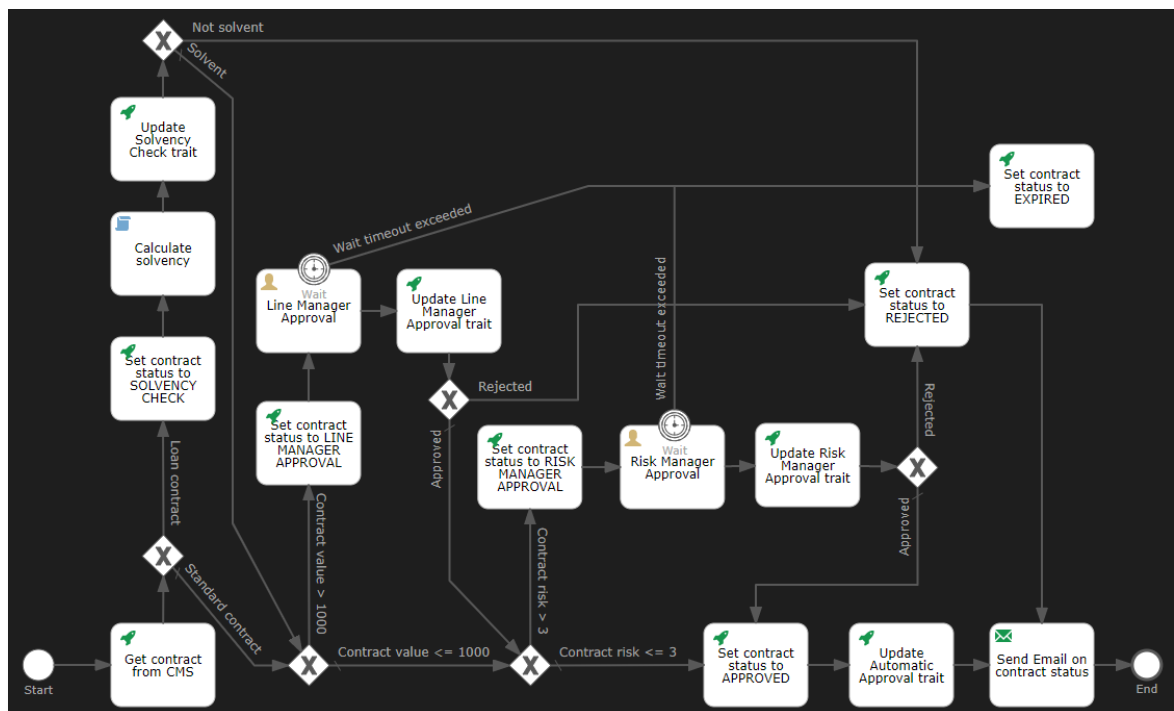
☐

Skip expression

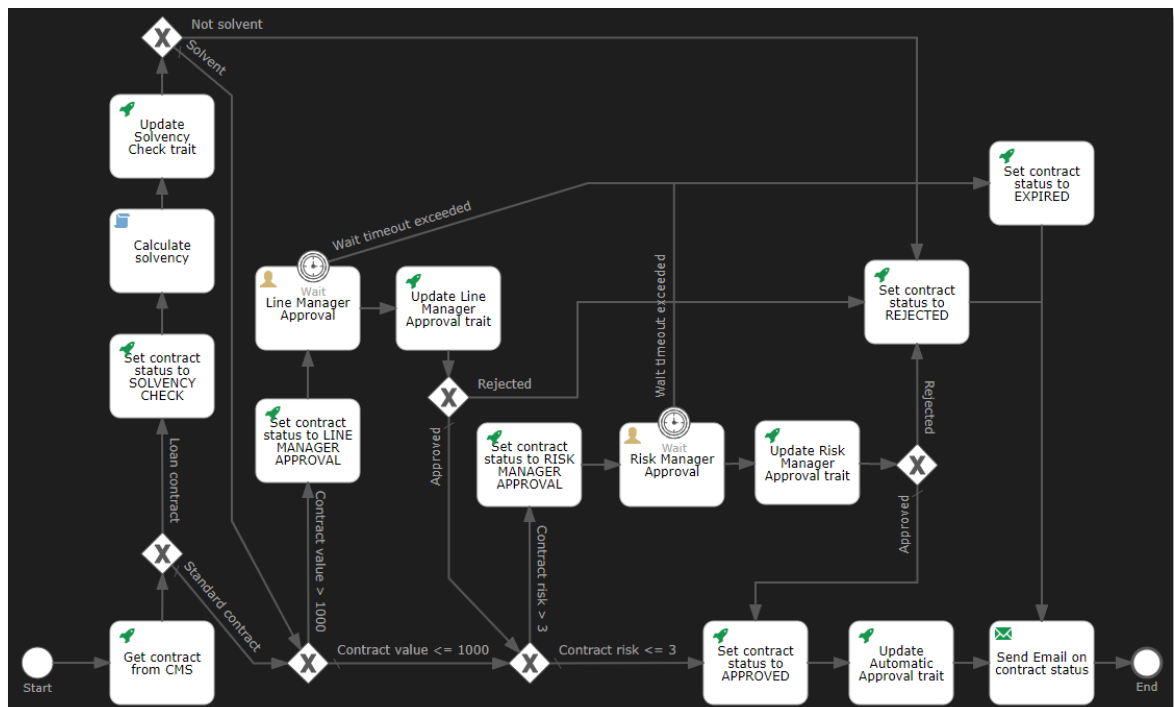
No value

Copyright © 2022 Open Text. All rights reserved. Trademarks owned by Open Text.

121



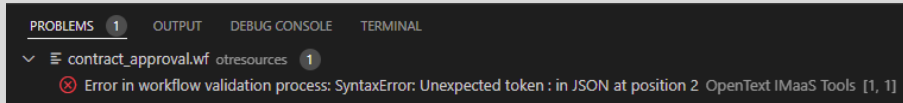
There's one last sequence flow arrow connector to add. To ensure the EXPIRED contract approval status email gets sent as well, connect the "Set contract status to EXPIRED" http task to the "Send Email on contract status" task.



Please save the workflow and verify that there are no errors in the **PROBLEMS** tab.

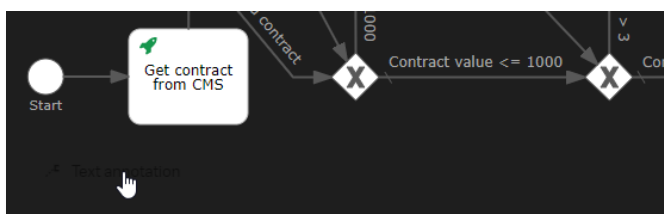
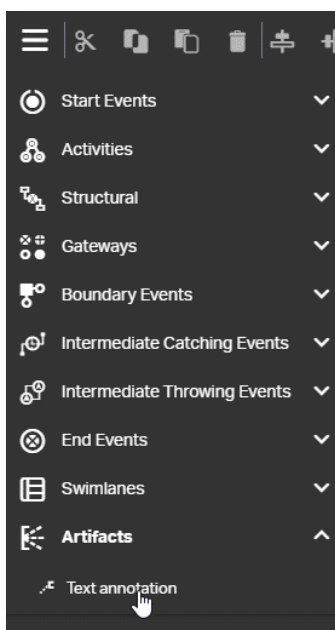
WARNING:

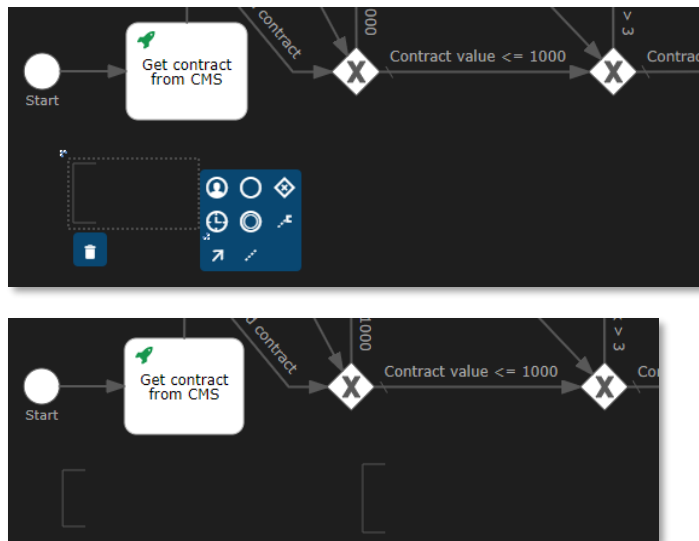
You might get the following error message in the **PROBLEMS** tab:



This is most likely caused by an error in the Contract Approval workflow model, where the **Boundary timer event** is not properly connected to the related **User task** (i.e.: the **Line Manager Approval** and/or **Risk Manager Approval** task). To solve this, open the workflow model and move the **Boundary timer event** on top of the **User task** and make certain the **User task** highlights green before you release it. You can even wiggle the **Boundary timer event** around a little after you dropped it to make sure the **User task** indeed lights up as green again (it has to still show a green border, if it doesn't show green the **Boundary timer event** is not properly connected). Save the changes to the workflow and confirm the **PROBLEMS** tab does no longer contain the error.

You don't have to do this last step, but as the proverbial cherry on the workflow model cake, we'd like to add some additional information to the workflow that describes the workflow input and, specifically in case of this contract approval workflow model, the different contract states. Drag and drop two **Text annotation** elements from the **Artifacts** section of the palette onto the bottom left of the canvas.



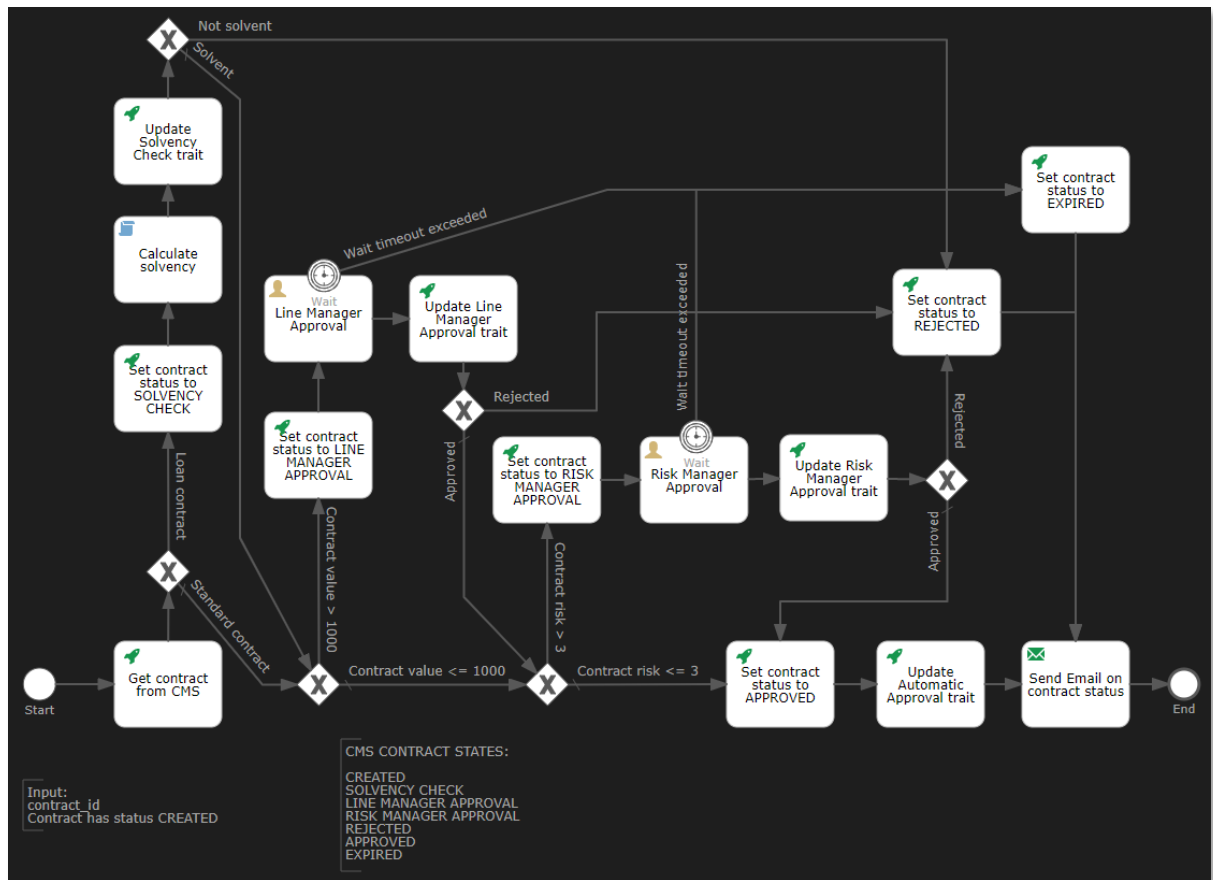


From left to right, double-click and fill the text annotations with the following text (i.e.: information for the developer looking at the workflow model):

- First text annotation:
Input:
contract_id
Contract has status CREATED
- Second text annotation:
CMS CONTRACT STATES:

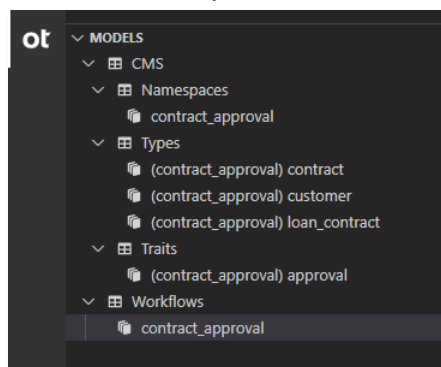
CREATED
SOLVENCY CHECK
LINE MANAGER APPROVAL
RISK MANAGER APPROVAL
REJECTED
APPROVED
EXPIRED

If the text doesn't fit, you can drag the **text annotation** element's bottom right or top left corner to expand it.



Your **Contract Approval** workflow model is now complete. Please, save it one last time, and close the workflow editor.

If you switch to the **OpenText IMaaS Tools** view, the model explorer tree in the **MODELS** section should now show your new **contract_approval** workflow model under **Workflows**.



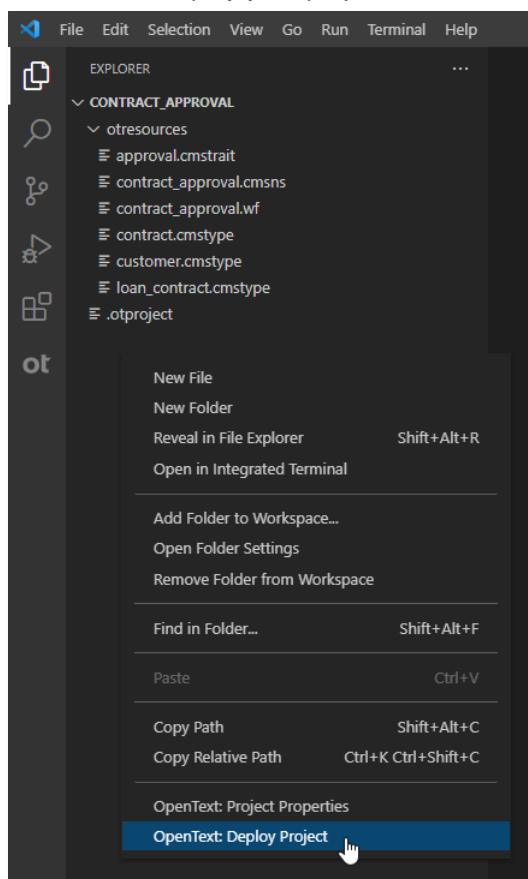
3.10[10'] Deploying the application to the IMaaS services

During this exercise you will deploy your **Contract Approval** IMaaS application project (i.e.: its models) to the different IMaaS services. Thanks to the organization connection you previously created, this is as simple as selecting the “Deploy Project” contextual menu from your project root folder.

Once you are done with this section, you will have deployed the **Contract Approval** application into your developer organization's single tenant, and you are ready to verify the deployment of the different models, using the IMaaS APIs via Postman.

To deploy the **Contract Approval** IMaaS application project, proceed as follows:

- In the **Explorer** view of VS Code, right-click inside your **contract_approval** project root folder (in the empty area to avoid selecting a sub folder or file) and click the **OpenText: Deploy Project** menu item to deploy your project.



REMARK:

Since you have tested your connection at the beginning of this tutorial, it can be that the connection (i.e.: authentication token) has expired. You will most likely have to log in again when clicking **OpenText: Deploy Project**.

After successfully deploying, the system pops up a “Deploy Project returned success” message, and the **Output** view automatically opens to display the **OT Deployment** output. It contains the tenant ID and (application) API key data.

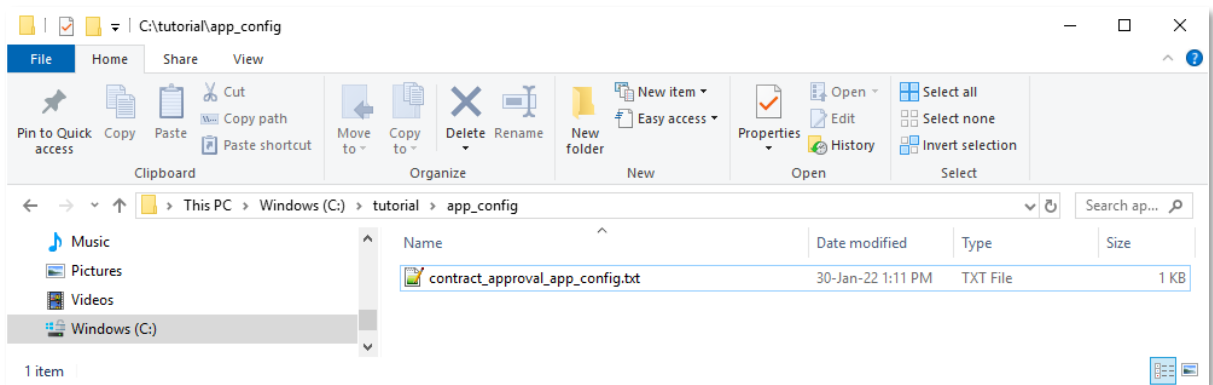
OpenText: Deploy Project returned success

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
deployment of project 'contract_approval' in tenant '...' created application 'contract_approval'.
Store below API key data in a secure way.
Public Client ID: ...
Confidential Client ID: ...
Confidential Client Secret: ...

```

- You need to store the **OT Deployment** information, as you will need it when verifying that your application models are correctly deployed. You will also need this when developing the actual frontend and backend (pro-code) parts of your application. Copy the text from the **Output** view window into a **contract_approval_app_config.txt** text file and save it (e.g.: under a new **app_config** folder).

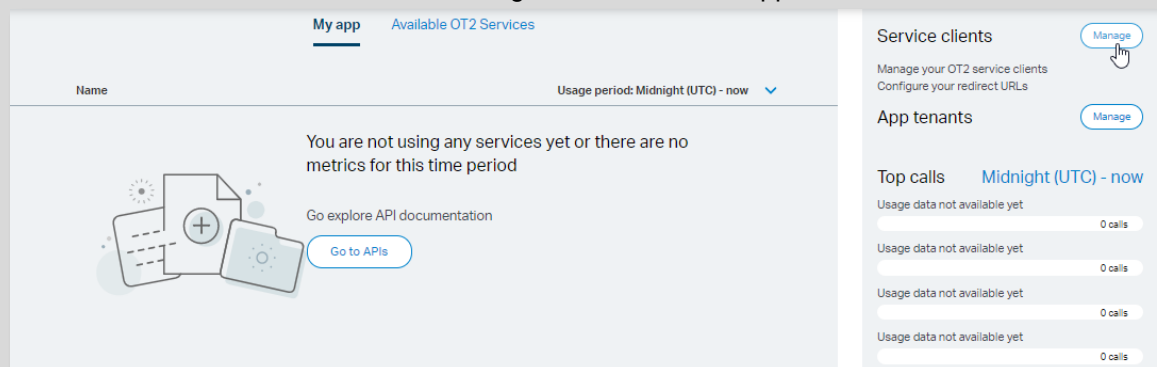


IMPORTANT:

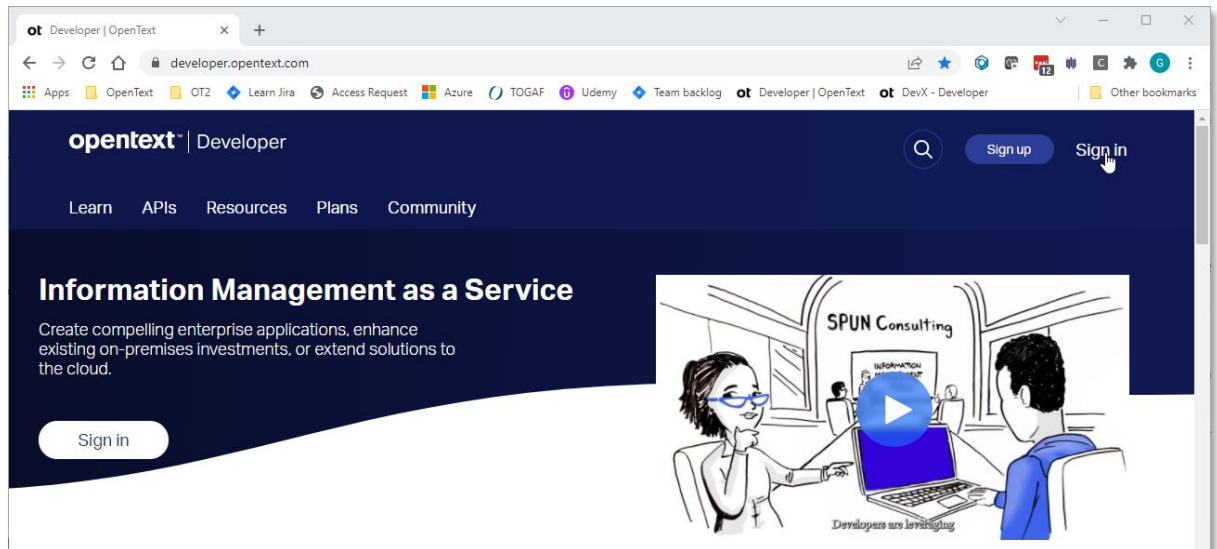
Although we are using unencrypted/insecure text files to store the different key and password information for the purpose of this tutorial, it is of course recommended for real life scenarios to store any API key or password information in a secure way.

REMARK:

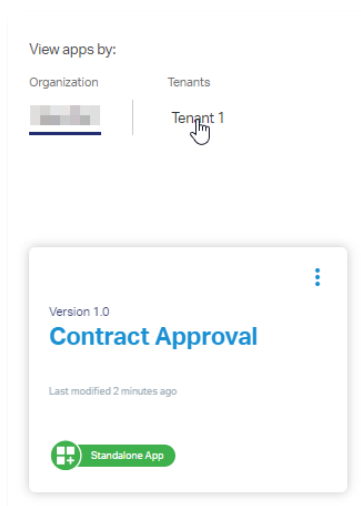
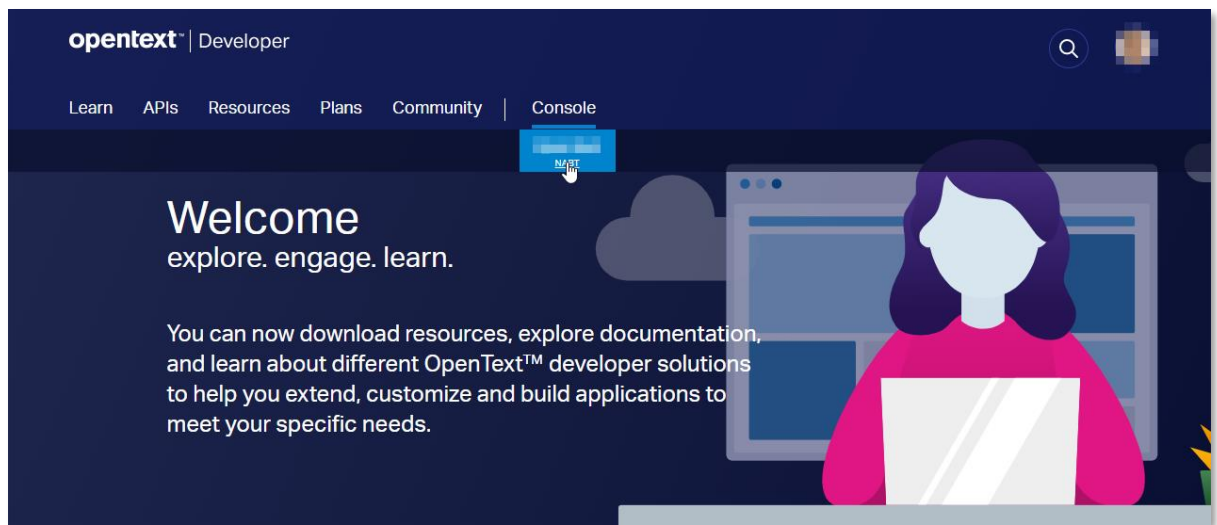
If you somehow missed the opportunity to copy/save the **OT Deployment** information (Tenant ID and client credentials), you can always retrieve the Tenant ID from your developer organization's console (see [here](#) how to access the console). The Tenant ID can be copied from the Tenant Info screen, and the client credentials can be regenerated from the App Details:



- To validate that deploying your application has successfully added an application entry in your developer organization, use your web browser to navigate to **developer.opentext.com** and log in with your developer (trial) account user.

A screenshot of the 'Sign in' form on the OpenText Developer website. The form has a white background and a dark blue header with the text 'Sign in'. Below the header is a link for 'Not a member? Register a new account'. The form contains two input fields: 'Email' and 'Password', both with placeholder text and a small 'Show/Hide' icon on the right. Below the password field is a checkbox labeled 'Keep me signed in'. At the bottom of the form is a large dark blue button labeled 'Sign in'. Below the button are two links: 'Forgot your password?' and 'Need Help?'.

Open the **Console** for your developer organization and confirm you can find your application deployed under your organization's single tenant: **Tenant 1**.



The screenshot shows the OpenText Developer portal interface. At the top, there's a header with the OpenText logo and 'Developer'. Below the header, there's a navigation bar with 'View apps by:' and 'Tenants'. Under 'Tenants', 'Tenant 1' is selected. To the right, there's a 'My Tenant info' sidebar showing 'Name: Tenant 1', 'ID', 'Tenant service account', 'Status: ACTIVE', and 'My Plan: Developer-Trial'. The main area displays a card for 'Contract Approval' (Version 1.0) with a 'Standalone App' badge. Buttons for 'Learn' and 'Create new app' are visible.

This close-up shows the 'Contract Approval' app card with a context menu open. The menu options are: 'App details', 'Add to tenant', 'Remove from tenant', and 'Delete app'. The 'App details' option is highlighted by the mouse cursor.



The screenshot shows the 'Contract Approval 1.0' app details page. The header includes a back arrow, the app name 'Contract Approval 1.0', and an edit icon. Below the header, there's a table with app details:

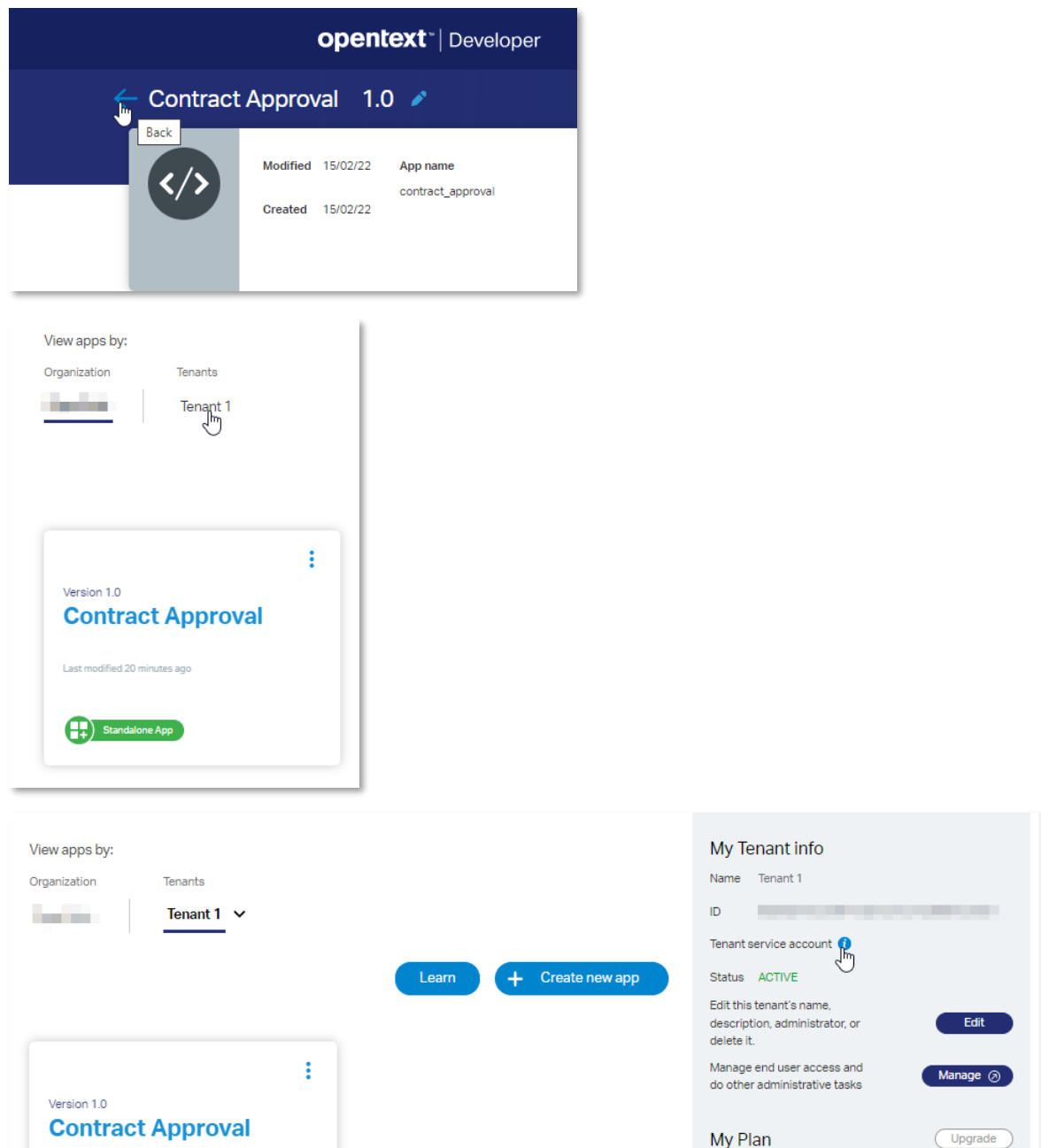
	Modified	15/02/22	App name	App description	
			contract_approval	Contract Approval Application built on top of the OpenText IMaaS Platform	
Created	15/02/22				

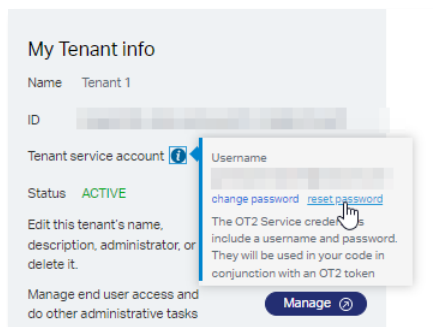
This confirms your application has been deployed and is now available in your **Tenant 1** developer tenant.

Resetting the Tenant password

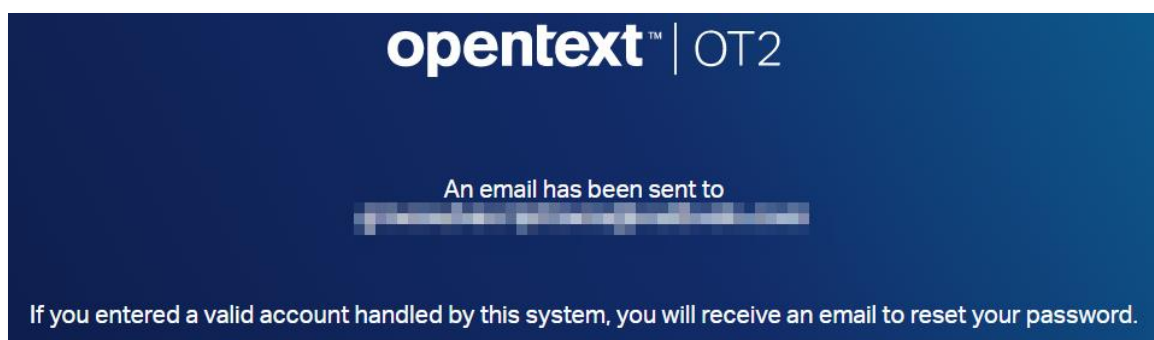
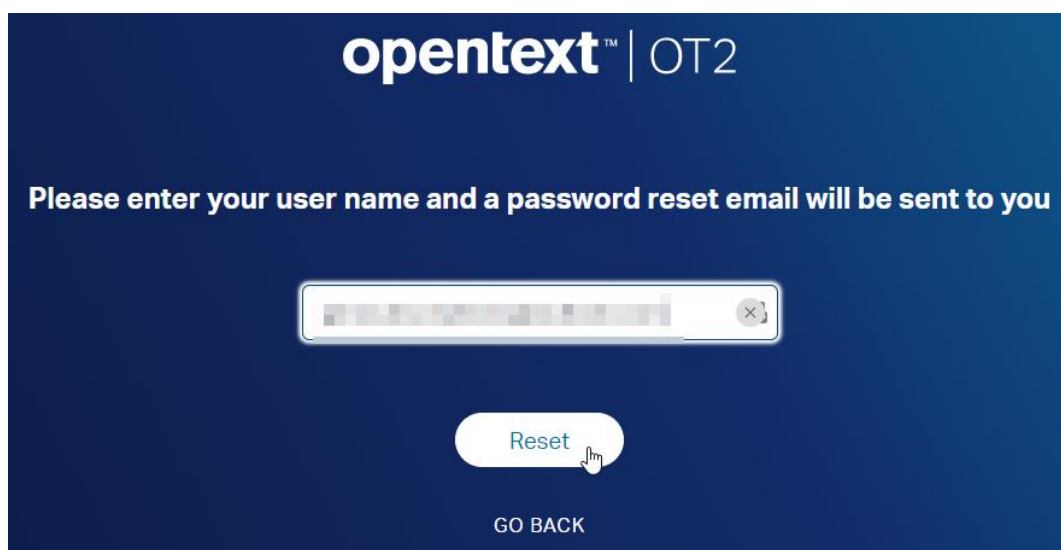
- This is also the right time to (re)set your **Tenant 1** password. You need to do this, as you have never actually set this password. Although the developer tenant has been generated alongside your developer organization, for security reasons, you need to set a password specifically at the tenant level as well.

Click the  to go back to your organization view, make sure to select the tenant view, and click  next to **Tenant service account** from the **My Tenant info** pane to request a password reset.

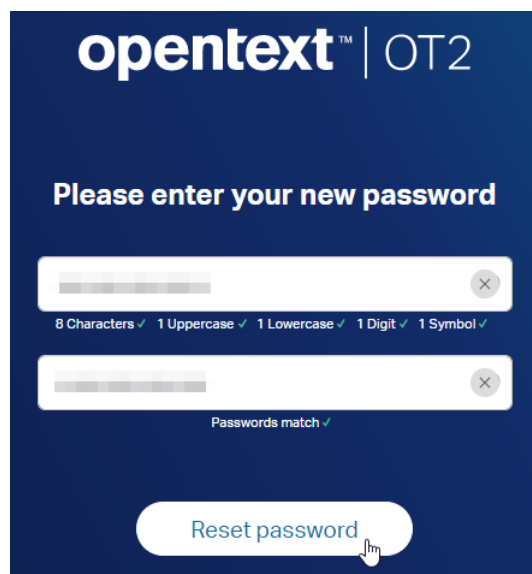
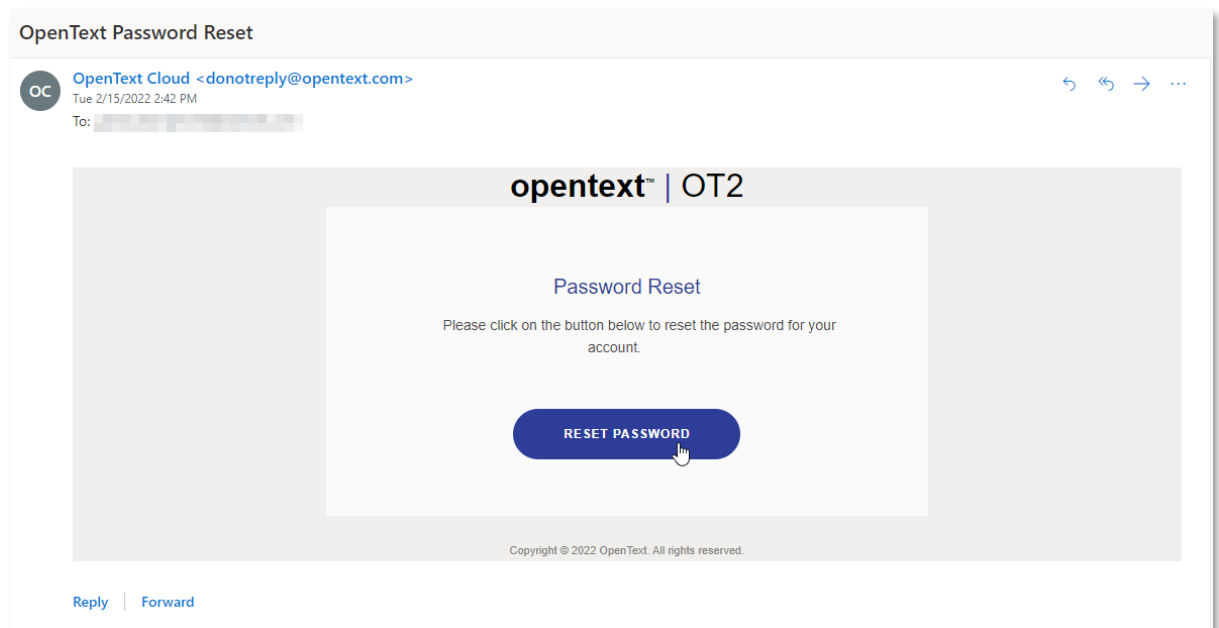




Fill the email address corresponding with the (tenant service account) **Username** and click **Reset**.



Go to your email client and open the **Password Reset** email you just received from **OpenText Cloud** and choose your new password.



- Once you have chosen your new password, please save it for later use. You can simply add this new tenant password to the previously created **contract_approval_app_config.txt** text file.

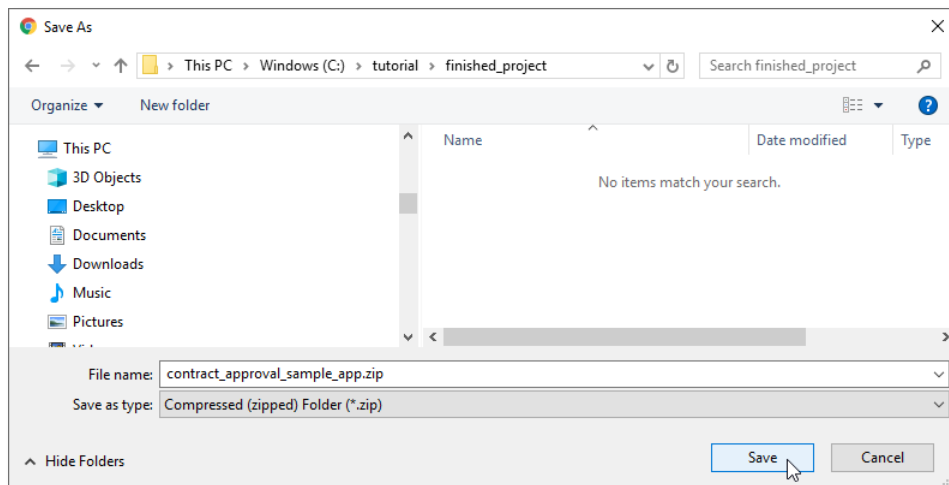
3.11[25'] Working with the IMaaS APIs

During this exercise you will use the CMS and Workflow Service IMaaS APIs (via Postman) to verify that your application models have been correctly deployed. For more information on the CMS and Workflow Service APIs, you can refer to their API reference documentation, respectively [CMS API reference](#) and [Workflow Service API reference](#).

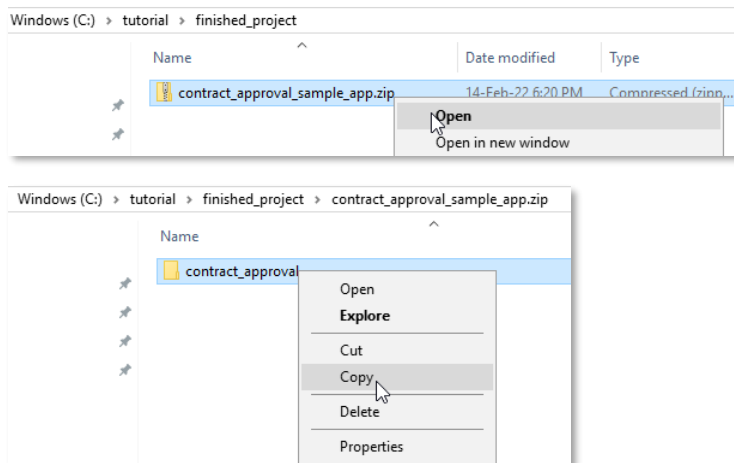
Once you are done with this section, you will have confirmed that your application models have indeed been correctly deployed and you will have a good understanding of how to use the IMaaS APIs, so that you are ready to start building the backend (services) part of your Contract Approval application.

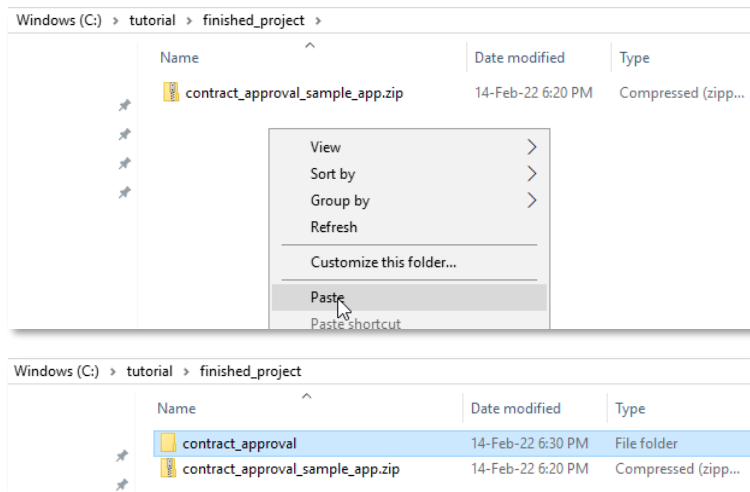
To verify the deployment of your application models, proceed as follows:

- Postman is a very popular API testing tool, and it is available for download from the following link: <https://www.postman.com/downloads>. If you don't already have it installed on your computer, please download and install the version that fits your system.
- You will also need to download the finished version of the application you are building, as it contains the Postman collection and Postman environment that you will be using to test that your models have been correctly deployed. You can download the full Contract Approval application (ZIP file) through this [link](#) (e.g. into a **finished_project** folder).



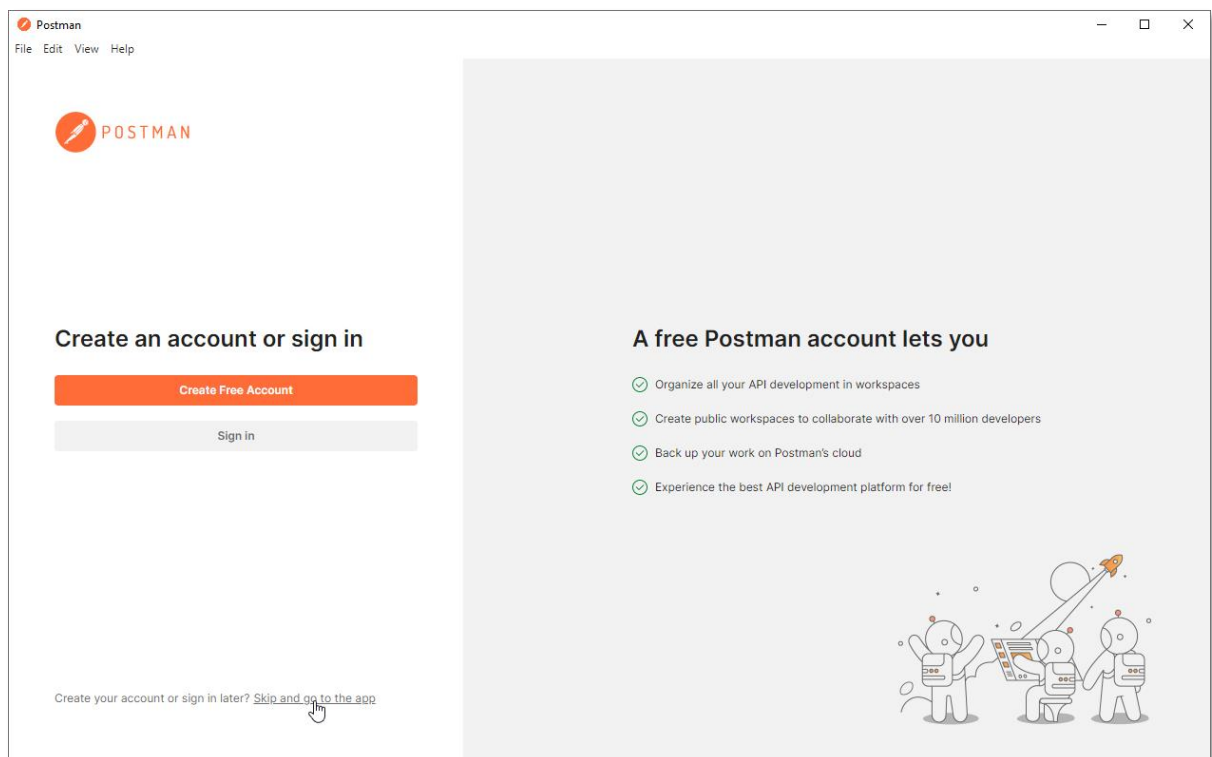
Extract (copy from within the ZIP file) the **contract_approval** application project folder.

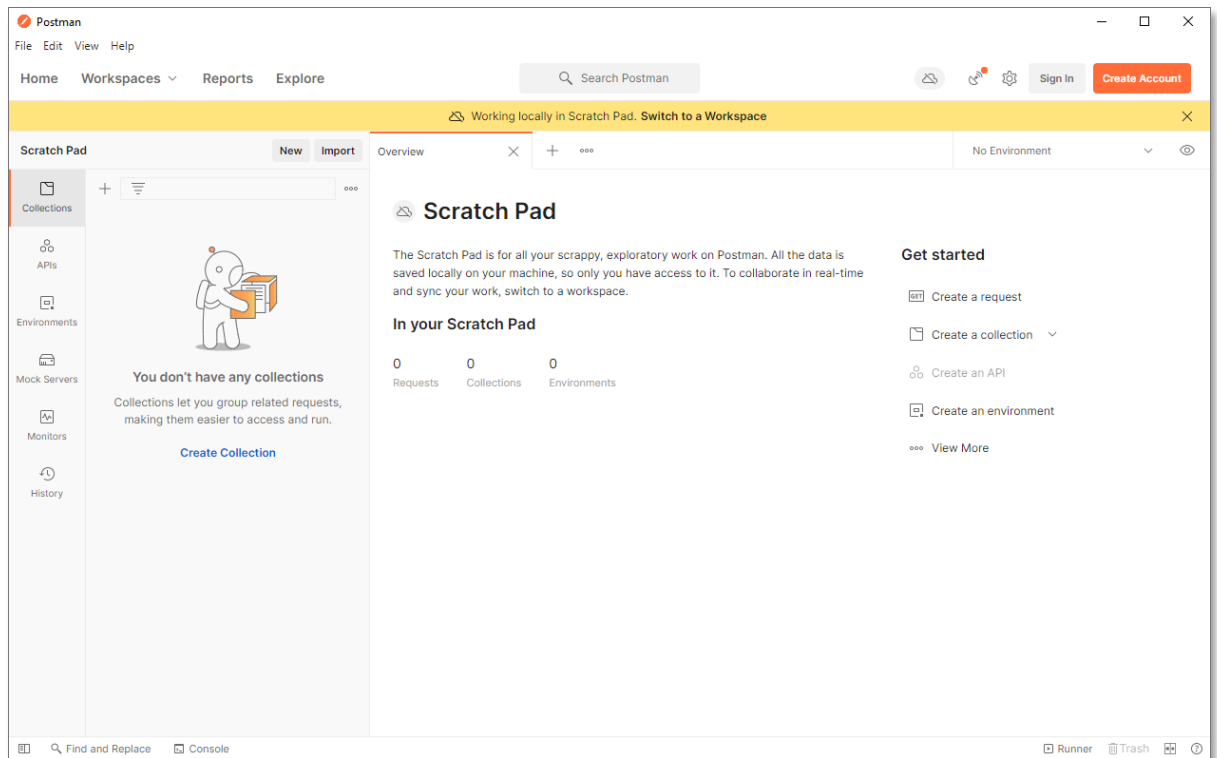




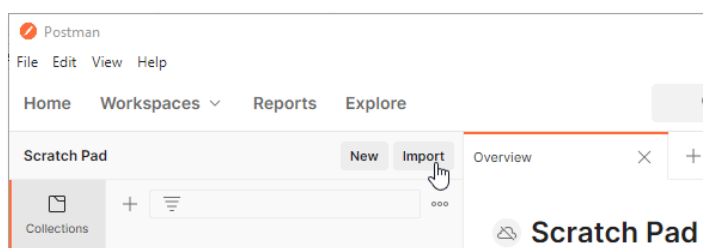
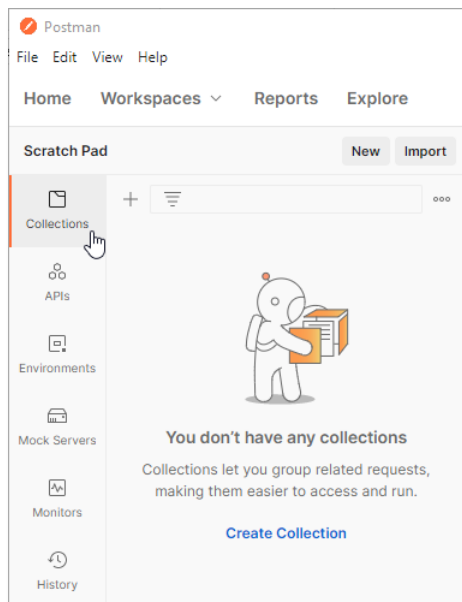
Feel free to delete the ZIP file after extracting the contract_approval folder.

- After downloading and extracting the finished Contract Approval application project, we can now configure Postman to work with the OpenText CMS & Workflow Service APIs. This is done by importing the Postman collection and Postman environment into your Postman application. First, open Postman (the screen shots are for a freshly downloaded and installed Windows 64-bit version).

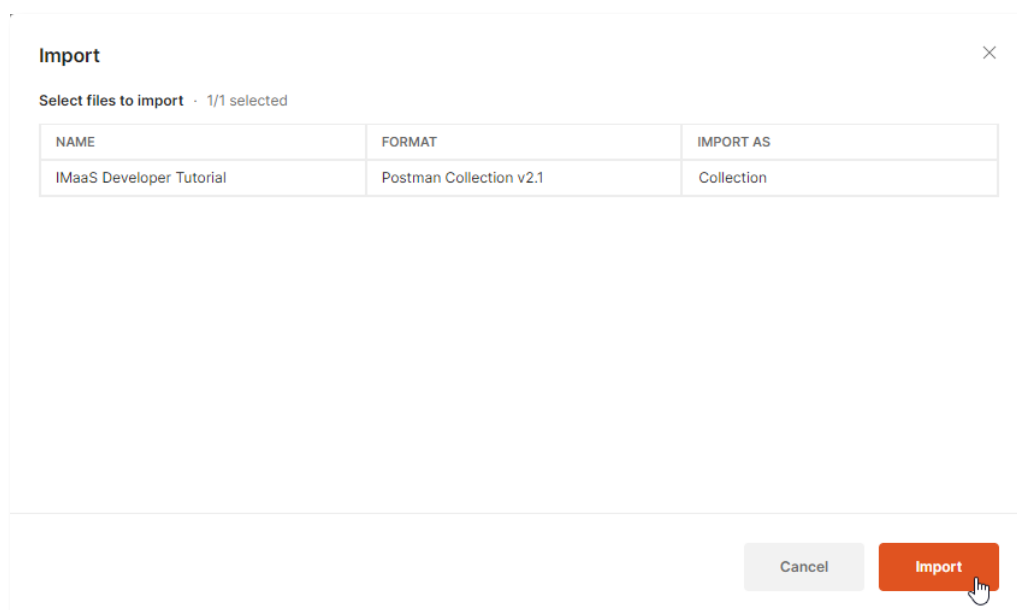
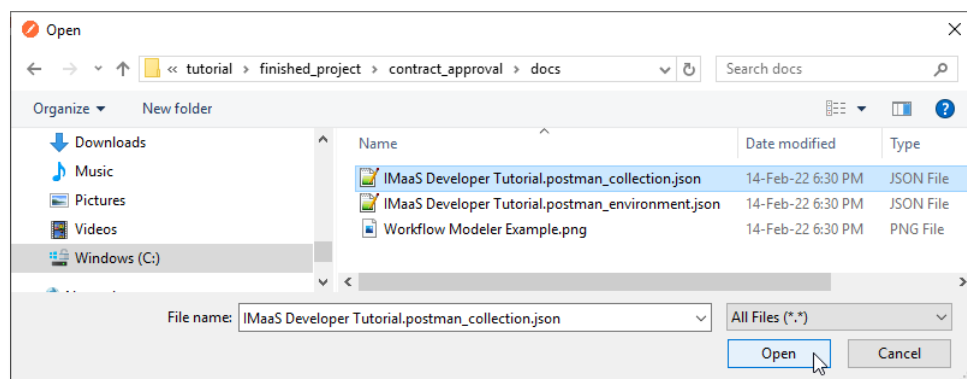
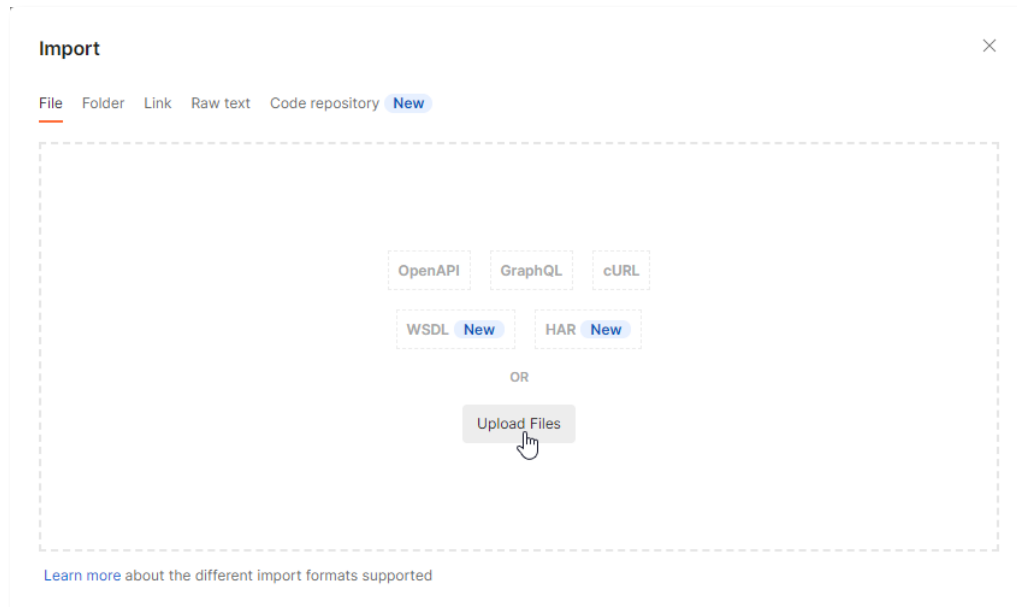


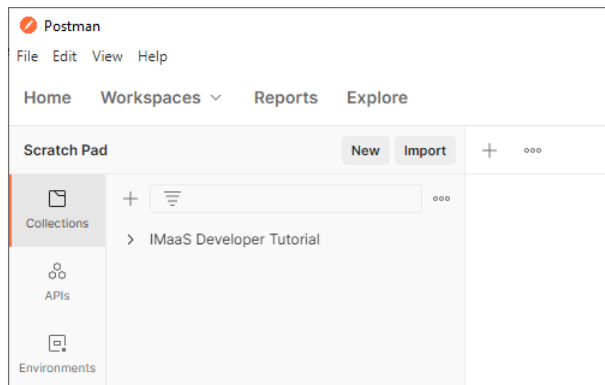


To import the “IMaaS Developer Tutorial” collection, make sure to select **Collections** from the left sidebar and click **Import**.

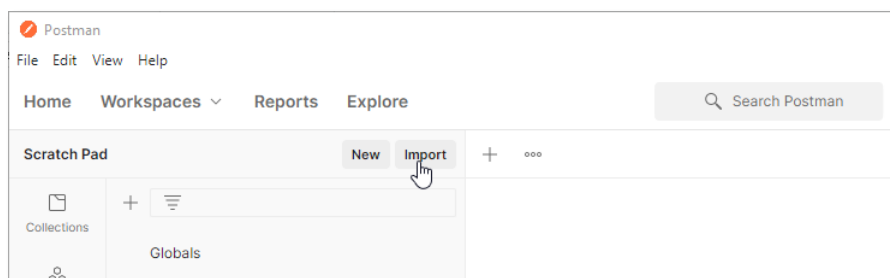
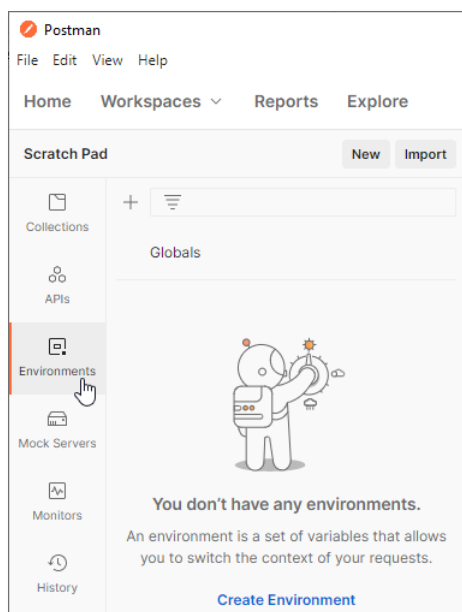


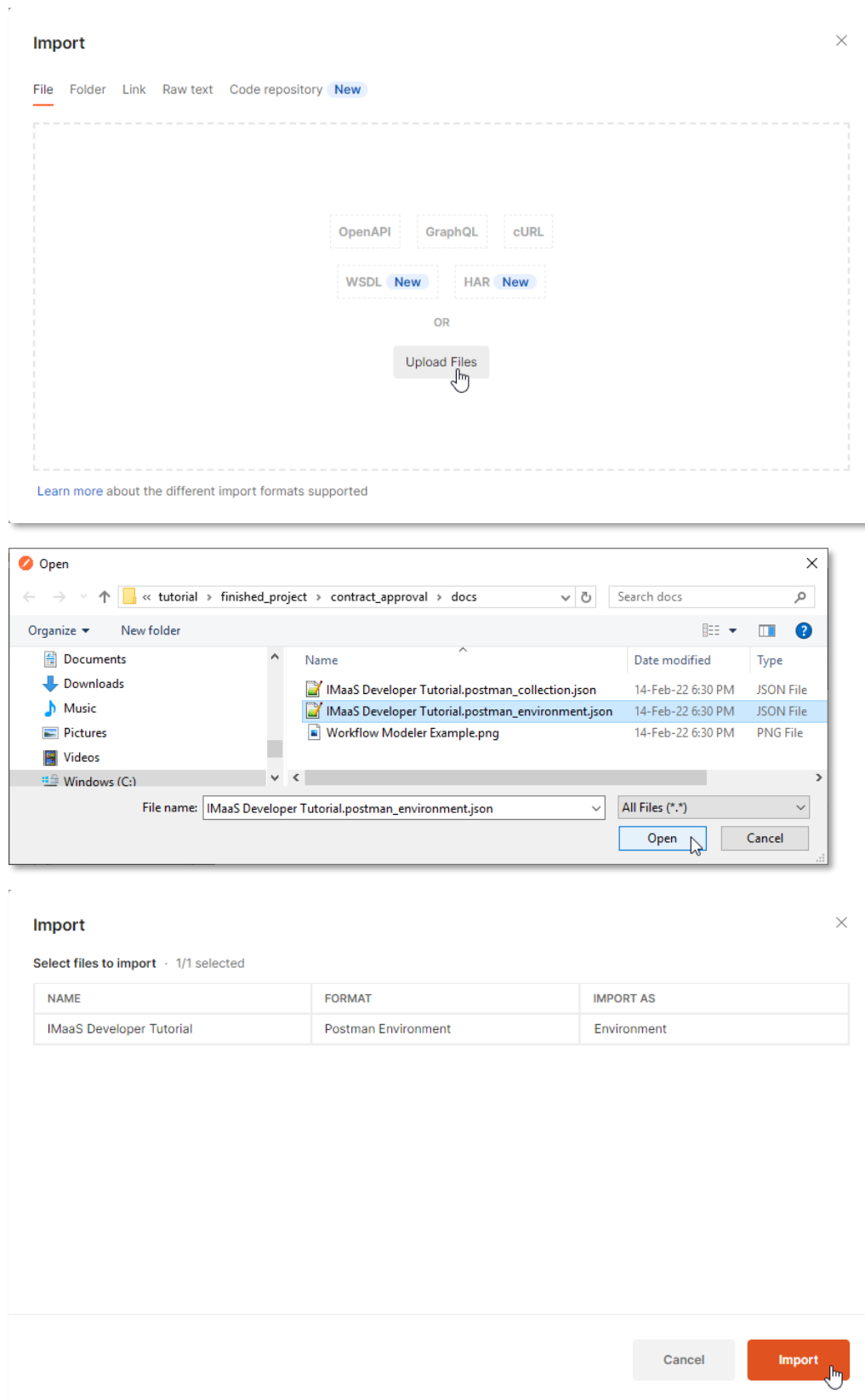
From the **Upload Files** dialog, navigate to the **/docs** folder in the **contract_approval** folder you just extracted and choose to import the **IMaaS Developer Tutorial.postman_collection.json** collection file.

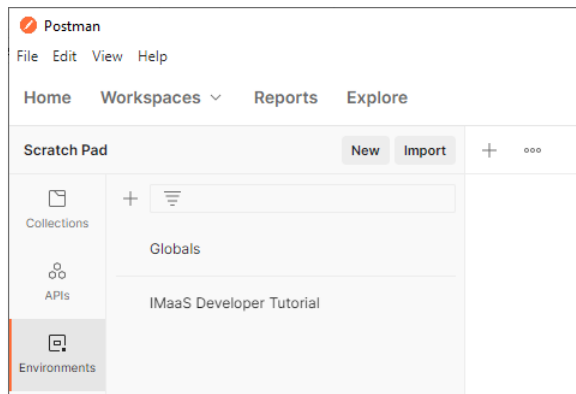




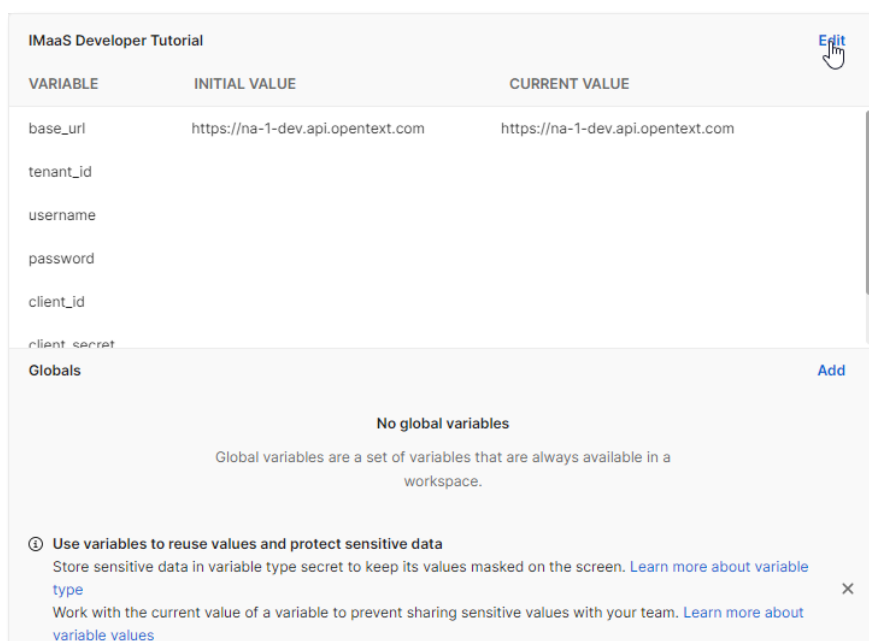
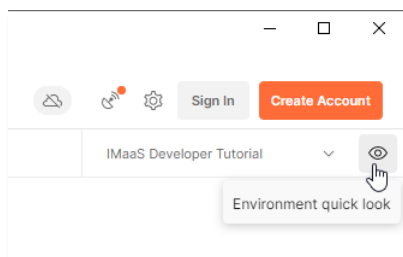
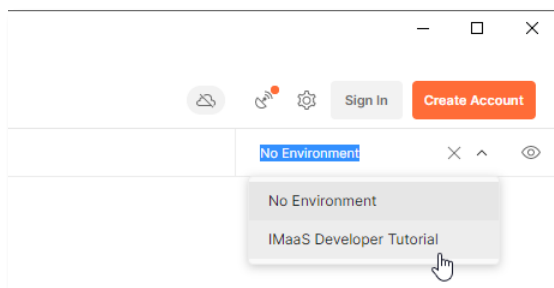
Similar to importing the “IMaaS Developer Tutorial” collection, import the “IMaaS Developer Tutorial” environment by clicking **Import** after having selected **Environments** from the left sidebar.

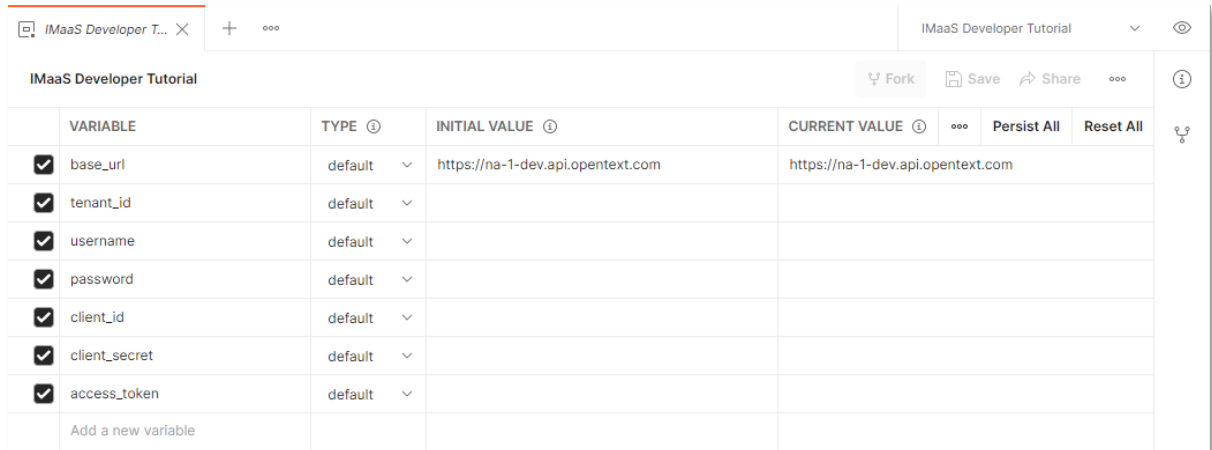






- Before you can call the IMaaS APIs to verify your deployment, you must fill the IMaaS Developer Tutorial environment's environment variables with the values that correspond with your developer organization/tenant and deployed application.



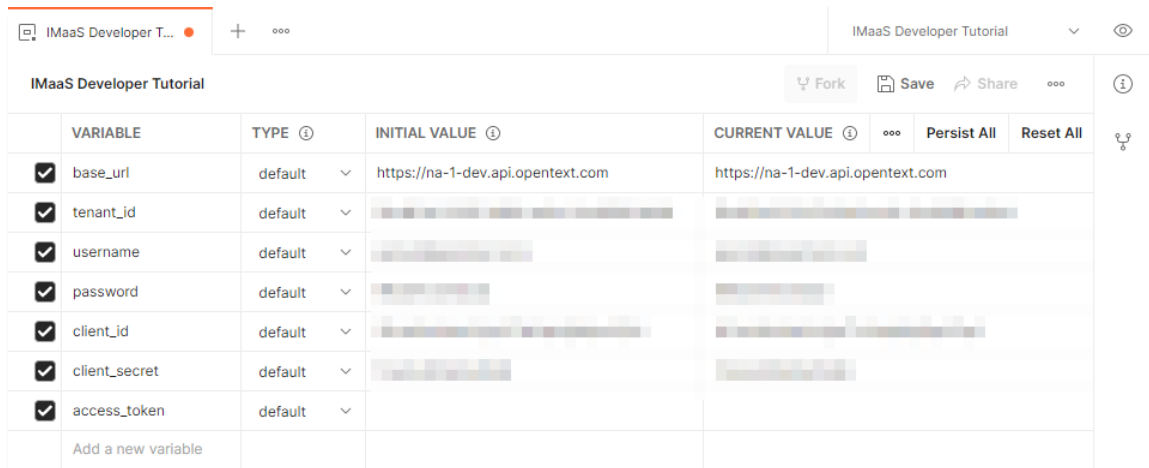


	VARIABLE	TYPE	INITIAL VALUE	CURRENT VALUE		Persist All	Reset All
<input checked="" type="checkbox"/>	base_url	default	https://na-1-dev.api.opentext.com	https://na-1-dev.api.opentext.com			
<input checked="" type="checkbox"/>	tenant_id	default					
<input checked="" type="checkbox"/>	username	default					
<input checked="" type="checkbox"/>	password	default					
<input checked="" type="checkbox"/>	client_id	default					
<input checked="" type="checkbox"/>	client_secret	default					
<input checked="" type="checkbox"/>	access_token	default					
	Add a new variable						

Fill both the INITIAL VALUE and CURRENT VALUE columns for each of the following environment variables (the ones that are not mentioned, you must leave as is):

- **tenant_id**: use the tenant id (from text in **tenant** '<tenant id>') you saved after deploying the IMaaS application project
- **username**: use the email address you used for your tenant service account
- **password**: use the password you saved for the tenant service account (after having reset it)
- **client_id**: use the **Confidential Client ID** you saved after deploying the IMaaS application project
- **client_secret**: use the **Confidential Client Secret** you saved after deploying the IMaaS application project

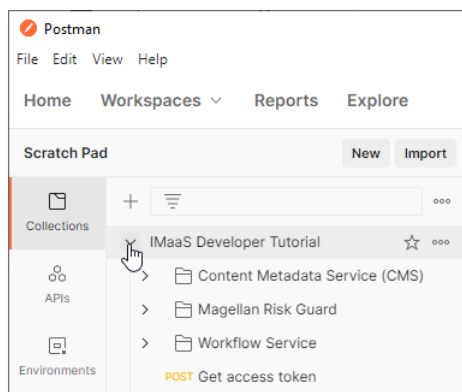
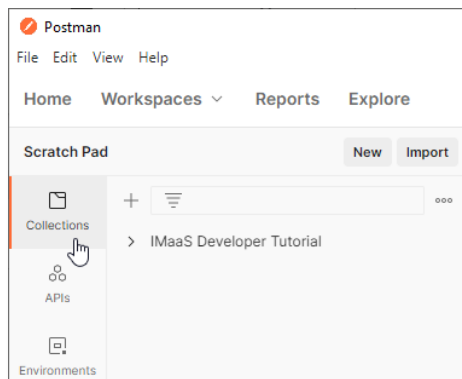
Note that if you followed the tutorial exactly and used the **contract_approval_app_config.txt** file to save the application configuration information (see: [Deploying the application to the IMaaS services](#)), all above values should be available from that file (except for the email address, which is the standard email address you used to create your organization).



	VARIABLE	TYPE	INITIAL VALUE	CURRENT VALUE		Persist All	Reset All
<input checked="" type="checkbox"/>	base_url	default	https://na-1-dev.api.opentext.com	https://na-1-dev.api.opentext.com			
<input checked="" type="checkbox"/>	tenant_id	default	[Masked]	[Masked]			
<input checked="" type="checkbox"/>	username	default	[Masked]	[Masked]			
<input checked="" type="checkbox"/>	password	default	[Masked]	[Masked]			
<input checked="" type="checkbox"/>	client_id	default	[Masked]	[Masked]			
<input checked="" type="checkbox"/>	client_secret	default	[Masked]	[Masked]			
<input checked="" type="checkbox"/>	access_token	default	[Masked]	[Masked]			
	Add a new variable						

You can now save and close the **IMaaS Developer Tutorial** environment configuration screen.

- Let's now switch to the **Collections** view from the left side bar and have a first look at the **IMaaS Developer Tutorial** Postman collection.



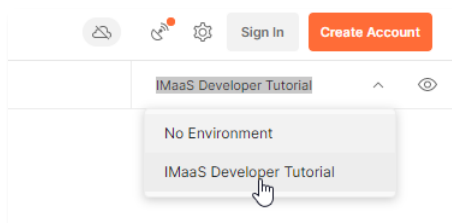
When you expand the **IMaaS Developer Tutorial** collection, you can see the three folders representing the services (CMS, Magellan Risk Guard and Workflow Service) for which the collection has example requests.

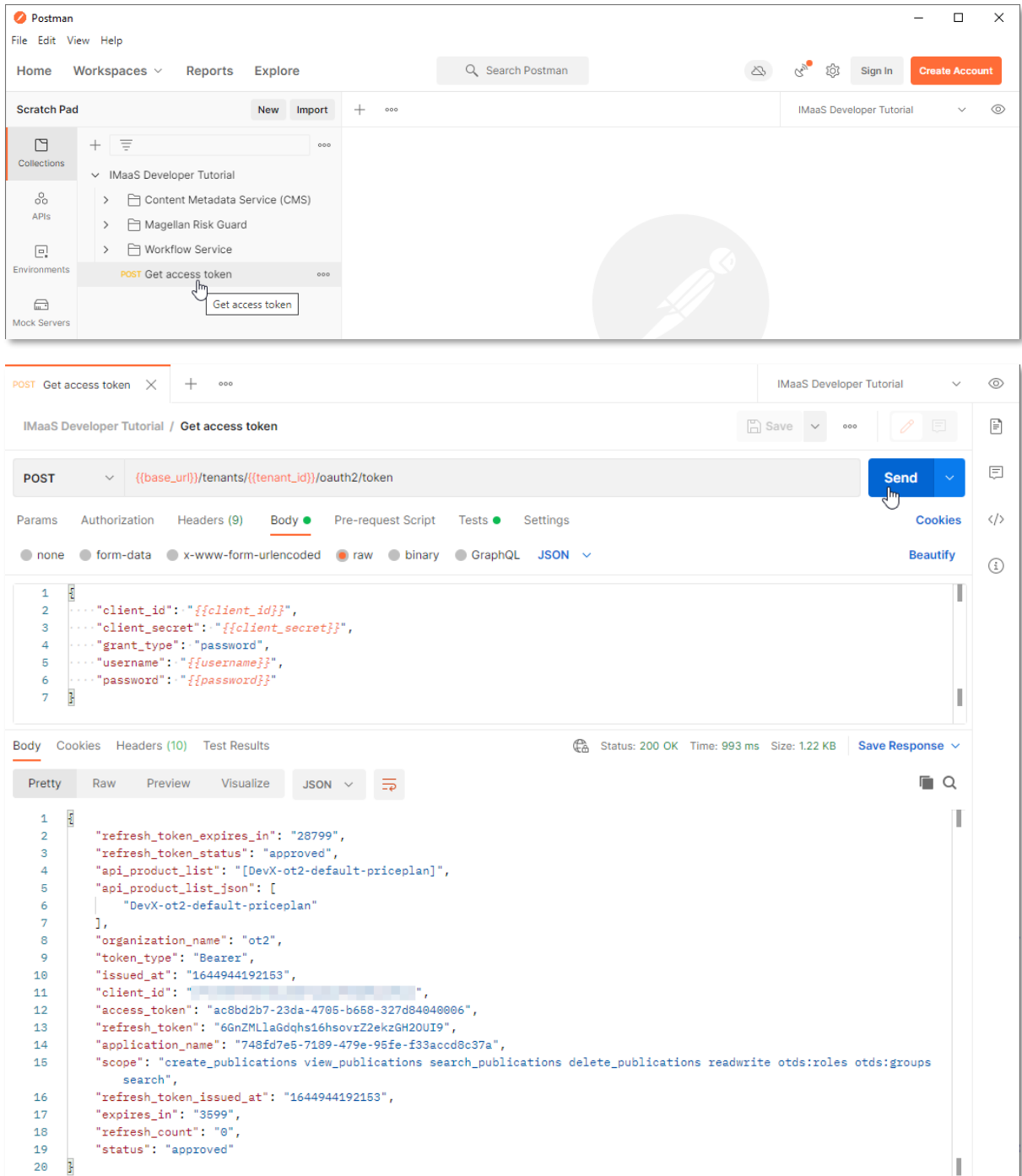
There's also a **Get access token** POST request in the root of the collection, as this is the single request you will be using to get an access token to use for all other requests. Once you run this request successfully the **access_token** environment variable gets populated and can be used in every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

REMARK:

The **access_token** can expire. If this happens your API requests will start failing, pointing to the token not being valid. When that happens, just re-run the **Get access token** request and you should be able to continue (perform the other requests again).

- To get started, let's now indeed get a token by running the **Get access token** POST request. Make sure you have selected the **IMaaS Developer Tutorial** environment, open the **Get access token** request and click **Send**.





Postman interface showing the 'Get access token' endpoint in the 'IMaaS Developer Tutorial' workspace.

The endpoint is a POST request to `{{base_url}}/tenants/{{tenant_id}}/oauth2/token`.

The request body is a JSON object with the following fields:

```

1  {
2    "client_id": "{{client_id}}",
3    "client_secret": "{{client_secret}}",
4    "grant_type": "password",
5    "username": "{{username}}",
6    "password": "{{password}}"
7  }

```

The response is a JSON object with the following fields:

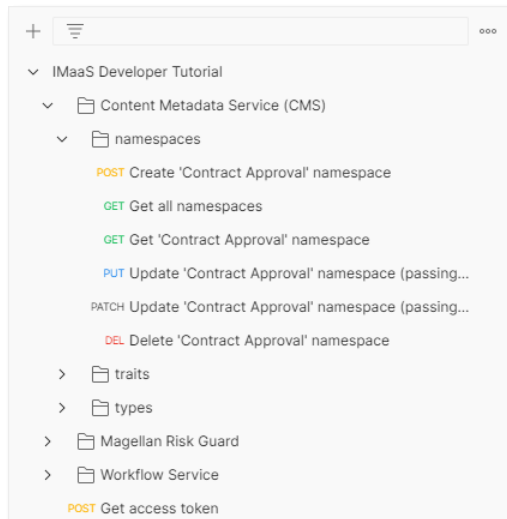
```

1  {
2    "refresh_token_expires_in": "28799",
3    "refresh_token_status": "approved",
4    "api_product_list": "[DevX-ot2-default-priceplan]",
5    "api_product_list_json": [
6      "DevX-ot2-default-priceplan"
7    ],
8    "organization_name": "ot2",
9    "token_type": "Beaxer",
10   "issued_at": "1644944192153",
11   "client_id": "ac8bd2b7-23da-4705-b658-327d84040006",
12   "access_token": "ac8bd2b7-23da-4705-b658-327d84040006",
13   "refresh_token": "6GnZMLlaGdohs16hsovz22ekzGH20UI9",
14   "application_name": "748fd7e5-7189-479e-95fe-f33accd8c37a",
15   "scope": "create_publications view_publications search_publications delete_publications readwrite otds:roles otds:groups search",
16   "refresh_token_issued_at": "1644944192153",
17   "expires_in": "3599",
18   "refresh_count": "0",
19   "status": "approved"
20 }

```

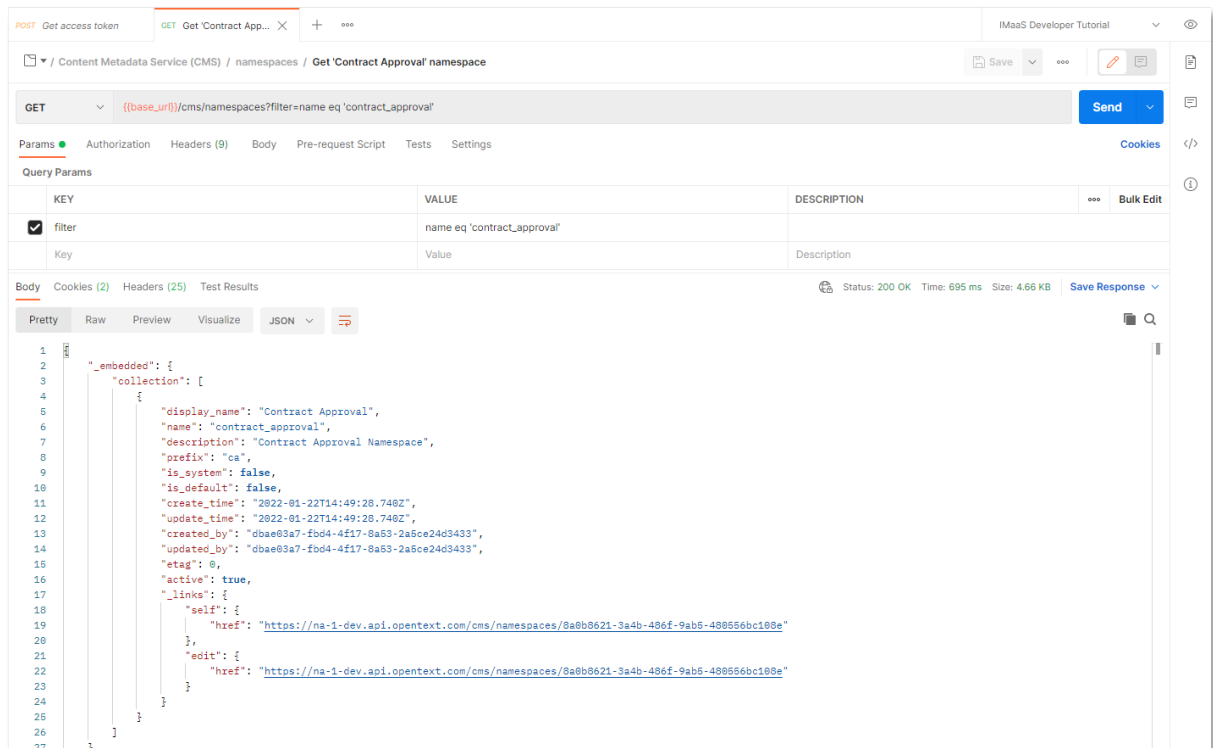
- Let's first verify that the **contract_approval** namespace has been correctly deployed/created in CMS.

Expand the **Content Metadata Service (CMS)** folder from the IMaaS Developer Tutorial collection, followed by expanding the **namespaces** folder.



What you see here are the different example requests relating to CMS namespaces. We'll have a look at the one that allows to retrieve the namespace we created.

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



As you can see, the Contract Approval namespace is being returned (i.e.: it has been correctly deployed) with its **display_name**, **name**, **description**, and **prefix** attributes.

Note that the request we are using, `{{base_url}}/cms/namespaces?filter=name eq 'contract_approval'` is using a filter to retrieve the namespaces with a name equal to 'contract_approval'.

All the requests are based on the API reference documentation, so if you want to look at the related documentation, just go to the [CMS API reference](#) on developer.opentext.com and select the appropriate request explanation. This applies to all requests in the collection (note that for Workflow Service you need to refer to the [Workflow Service API reference](#) and for Magellan Risk Guard to the [Magellan Risk Guard API reference](#)).

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

Back to APIs **GET** **Get list of Namespaces**
The Namespace's end point lists the list of namespaces

AUTHORIZATIONS: tenant

QUERY PARAMETERS

Parameter	Type	Description
page	integer <int32>	The page number required
items-per-page	integer <int32>	Number of items per page
include-total	string	whether to include total number of results in response or not
filter	string	The query search filter to filter the list of namespaces
sortBy	string	The sort by query parameter

Responses

- > 200 List of namespaces
- 401 Unauthorized to perform the operation
- 403 Forbidden error

POST create a new Namespace

GET /namespaces

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

GET **get Namespace Details**
The Namespace's end point to get a namespace details using id of namespace

AUTHORIZATIONS: tenant

PATH PARAMETERS

Parameter	Type	Description
namespaceId	string <uuid>	Id of the namespace

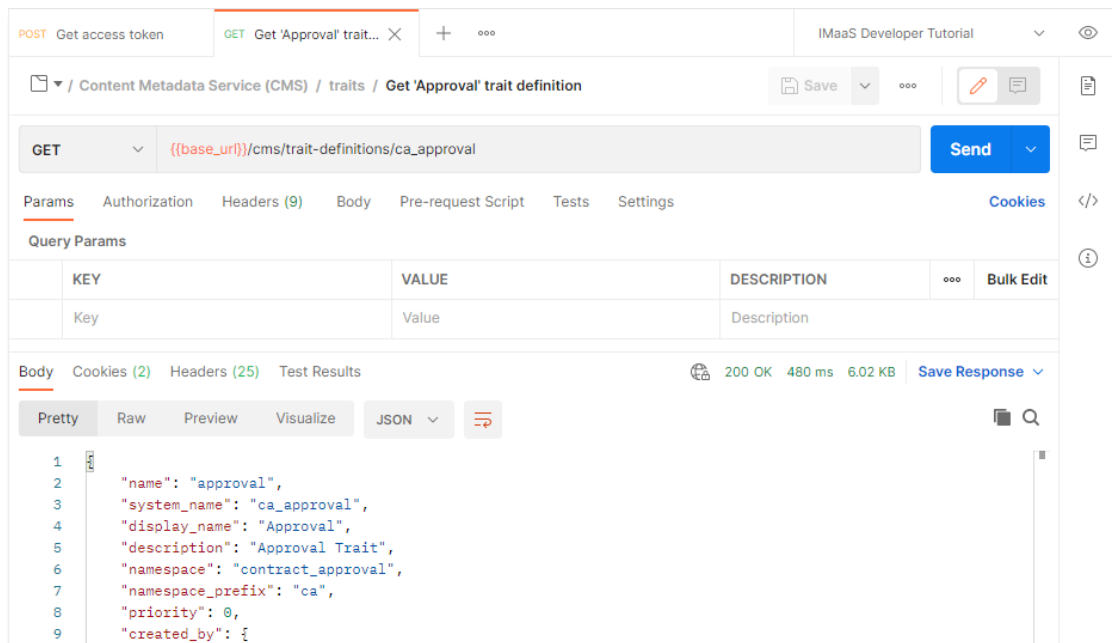
GET /namespaces/{namespaceId}

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

REMARK:

Throughout the Postman collections you'll find UUIDs representing unique IDs of resources as parameters for GET, PUT, PATCH and DELETE requests. In case you want to use those requests, you of course need to replace the UUID values with the value of the resource ID you want to work with.

- The next deployed model to verify is the **approval** CMS trait definition. Similar to how we verified the namespace, expand the **traits** folder under the **Content Metadata Service (CMS)** folder in the collection, and perform the **Get 'Approval' trait definition** GET request.



- To verify the deployment of the **contract**, **loan_contract** and **customer** CMS type definitions, respectively perform the following requests:
 - /Content Metadata Service (CMS)/types/file/Contract/type definitions/Get 'Contract' type definition**

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

- Method:** GET
- URL:** `{{base_url}}/cms/type-definitions/ca_contract`
- Query Params Table:**

KEY	VALUE	DESCRIPTION
Key	Value	Description
- 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,

```
- Response Status:** 200 OK, 1124 ms, 6.77 KB

- /Content Metadata Service (CMS)/types/file/Loan Contract/type definitions/Get 'Loan Contract' type definition**

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

- Method:** GET
- URL:** `{{base_url}}/cms/type-definitions/ca_loan_contract`
- Query Params Table:**

KEY	VALUE	DESCRIPTION
Key	Value	Description
- Body:**

```

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,

```
- Response Status:** 200 OK, 469 ms, 6.87 KB

- **/Content Metadata Service (CMS)/types/folder/Customertype definitions/Get 'Customer' type definition**

The screenshot shows a REST client interface with a GET request to `{{base_url}}/cms/type-definitions/ca_customer`. The response is a JSON object with the following structure:

```

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,

```

As you will probably have noticed, for CMS type definitions, we added some levels to the collection folder structure to make the distinction between the **file** and **folder** CMS type category, and the actual CMS type definitions. We also provided **type instances** related requests (on top of the **type definitions** request) to allow retrieving and manipulating (including a “delete all instances” request) the CMS type instances that you create when testing the application.

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

The screenshot shows a REST client interface with a GET request to `{{base_url}}/workflow/v1/runtime/models?modelType=json&key=contract_approval`. The response is a JSON object with the following structure:

```

1  {
2    "_embedded": {
3      "models": [
4        {
5          "id": "6afee5ae-8bf9-11ec-be30-eeee0aff71c7",
6          "name": "Contract Approval",
7          "key": "contract_approval",
8          "tenantId": " ",
9          "version": 1,
10         "category": "contract_approval",
11         "processDefinitionId": "contract_approval:1:6b0f8781-8bf9-11ec-be30-eeee0aff71c7",
12         "subscriptionId": " ",
13         "deploymentSpace": "TENANT"
14       }

```


You have now verified the deployment of all your models, and you have seen how to use the API reference documentation and the **IMaaS Developer Tutorial** Postman collection in the process.

It is now time to start with the actual code writing part of the tutorial. Don't worry, we'll provide you with the sample code, so that you can import it into your VS Code project. The next two chapters will take you through this sample code; first for the backend (the services layer), and then for the frontend (the UI).

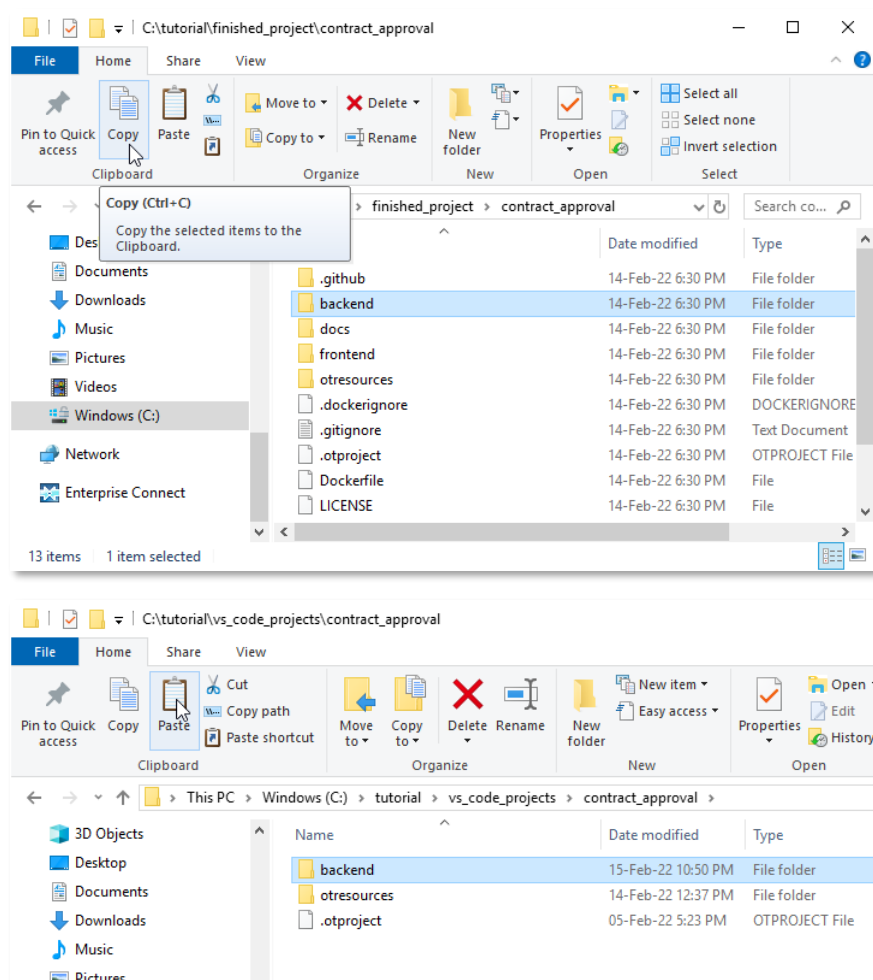
3.12[15'] Building the application backend

During this exercise we will be going through the backend code (the REST API services) of the Contract Approval application. Although we will import and not actually write the Node.js code, we will go over its structure, logic and how it calls the different IMaaS services (CMS, Workflow Service and Magellan Risk Guard). It is not the intent to go over every single detail of how the code was written, and which file does what, but we will touch upon some key aspects of how the backend services have been developed. For more information on the CMS, Workflow Service and Magellan Risk Guard APIs, you can refer to their API reference documentation, respectively [CMS API reference](#), [Workflow Service API reference](#) and [Magellan Risk Guard API reference](#).

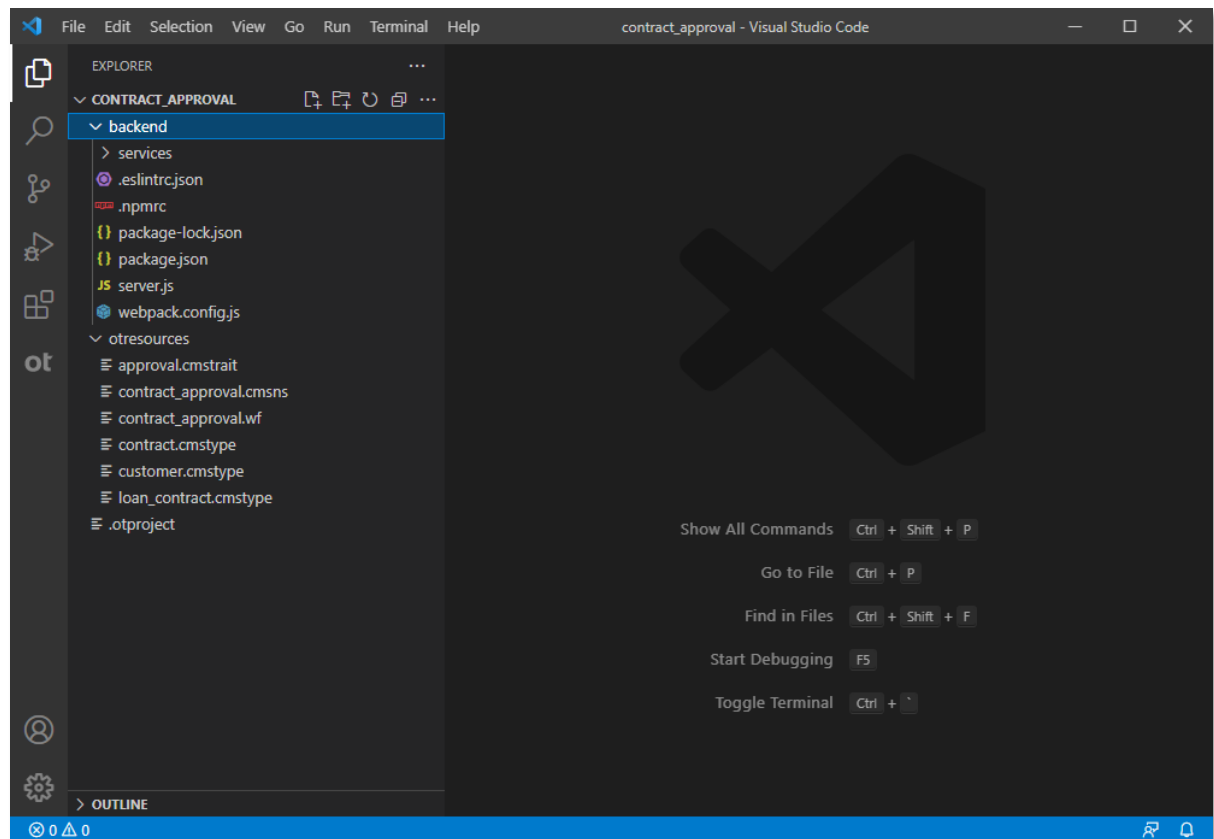
Once you are done with this section, you will have a better understanding of how the backend code of the Contract Approval application has been written, how it consumes the deployed models, and how it calls the different IMaaS APIs. You are then ready to start building the frontend (UI) part of your Contract Approval application.

To import and go over the backend code of the Contract Approval application, proceed as follows:

- The first step in this exercise is indeed to import the backend code into your VS Code project. This is very easy, as you just need to copy the **backend** folder from the previously extracted finished version of the Contract Approval application project. If you performed the step in the previous exercise of extracting the **contract_approval** project folder (containing the finished application), navigate into that **contract_approval** folder and copy the **backend** folder into the root of your Contract Approval project.



You can now open VS Code. The backend folder should be visible in the **Explorer** view.



- The main entry point for the backend services is the **server.js** file, directly available from the **backend** folder. Let's start with opening it and having a look at the JavaScript code inside.

```

JS server.js
backend > JS server.js > saveConfiguration
1  const express = require("express");
2  const session = require("express-session");
3  const request = require("request");
4  const fileUpload = require("express-fileupload");
5  const replace = require('replace-in-file');
6  const fs = require('fs')
7
8  const { tasksGetObjects, tasksUpdate } = require("../services/Tasks");
9  const { cmsGetObjects, cmsGetObject, cmsCreateInstance } = require("../services/CMS");
10 const { cssDownloadContent, cssUploadContent } = require("../services/CSS");
11 const { rgGetToken, rgProcessDoc } = require("../services/RiskGuard");
12 const { workflowCreateInstance } = require("../services/Workflow");
13
14 const bodyParser = require("body-parser");
15 const cookieParser = require('cookie-parser');
16
17 const app = express();
18 require("dotenv").config();
19
20 app.set("port", process.env.PORT || 3001);
21 app.use(cookieParser());
22 app.use(session({
23   key: 'session_id',
24   secret: 'my secret string',
25   resave: false,
26   unset: 'destroy',
27   saveUninitialized: true,
28   cookie: {
29     maxAge: 20 * 60 * 1000 // 20 minutes session timeout
30   }
31 }));
32
33 app.disable('x-powered-by');
```

Directly at the top of the file (line **20**), you can see that the backend (RESTful) services will be listening on port 3001 by default (but it can be configured otherwise through the PORT environment variable).

Let's now immediately have a look at how we expose the different backend services to the frontend. From line **125** until line **312** you can see how this is done, as each REST endpoint is exposed using express.js (as **app.<HTTP method>**).

As an example, go to line **253**. This shows a post request endpoint to create a new type instance in the Content Metadata Service.

```

250  /**
251   * Creates a new type instance in the Content Metadata Service.
252   */
253  app.post("/api/cms/instances/:category/:type", async (req, res) => {
254    try {
255      let responseBody = await cmsCreateInstance(req, req.params.category, req.params.type, getAuthorizationWithToken());
256      res.send(responseBody);
257    } catch (err) {
258      res.status(err.status).send(err.description);
259    }
260  });

```

An example of an incoming post request that would be handled by this endpoint could be the creation of a new loan contract: **/api/cms/instances/file/ca_loan_contract**.

In the above code snippet, you can see that **cmsCreateInstance** is called to perform the type instance creation. We'll have a look at what that means next. Before we do, please note the **getAuthorizationWithToken** method that provides the access token as a parameter for the **cmsCreateInstance** method. It basically fetches the previously stored/set access (Bearer) token.

```

42  /**
43   * Returns the Authorization header
44   */
45  const getAuthorizationWithToken = () => {
46    return `Bearer ` + app.get("access_token");
47  }

```

The code for the endpoint to request and set an access token can be found at line **359** and the method that actually calls the OT2 platform's authentication endpoint is on line **319**.

```

356  /**
357   * Requests and sets an access token.
358   */
359  app.post("/api/token", async (req, res) => {
360    try {
361      let responseBody = await getToken(req);
362      let response = JSON.parse(responseBody.body);
363      app.set("access_token", response.access_token);
364      res.sendStatus(200);
365    } catch (err) {
366      console.log('Error getting token. Error: ' + JSON.stringify(err));
367      const errorMessage = isNaN(err.status) ? err.status + ' ' + err.description : err.description;
368      res.status(isNaN(err.status) ? 500 : err.status).send(errorMessage);
369    }
370  });
371
372  app.listen(app.get("port"), () => {
373    console.log('Find the server at: http://localhost:${app.get("port")}'); // eslint-disable-line no-console
374  });
375

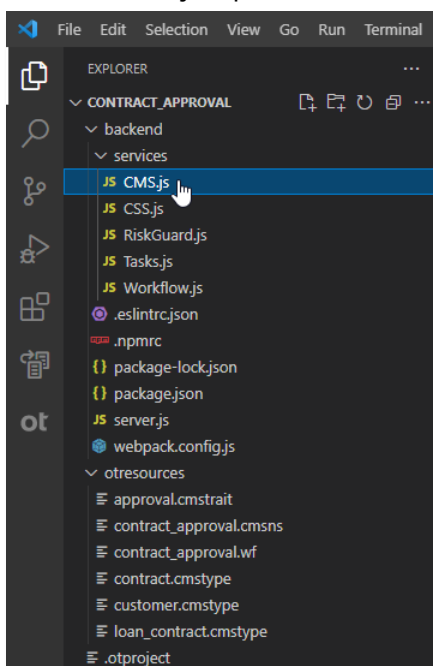
```

```

316  /**
317   * Gets an authorization token from OT2.
318   */
319  const getToken = async (req) => {
320    const username = req.body.username;
321    const password = req.body.password;
322    let postRequest = {
323      method: "post",
324      url: process.env.BASE_URL + "/tenants/" + process.env.TENANT_ID + "/oauth2/token",
325      headers: {
326        "Content-Type": "application/json"
327      },
328      body: JSON.stringify({
329        client_id: process.env.CLIENT_ID,
330        client_secret: process.env.CLIENT_SECRET,
331        username: username,
332        password: password,
333        grant_type: "password"
334      }),
335    };
336
337    return new Promise((resolve, reject) => {
338      request(postRequest, (error, response) => {
339        if (error) {
340          console.log('Authentication with ot2 failed, error: ' + error);
341          return reject({status: response ? response.statusCode : error.code, description: error.message ? error.message : error});
342        }
343        if (response.statusCode !== 200) {
344          let responseBody = JSON.parse(response.body);
345          console.log('Authentication with ot2 failed: ', responseBody);
346          return reject({ status: response.statusCode, description: responseBody.error_description });
347        }
348        req.session.user = {
349          email: username
350        };
351        resolve(response);
352      });
353    });
354  };

```

- The **cmsCreateInstance** method, which is being called from the **server.js** code to create a new type instance, is not part of **server.js**. Instead, you will find it inside the **CMS.js** file (line 54) under the **services** folder. The **services** folder contains a JavaScript file for each of the different API domains. **CMS.js** represents the Content Metadata Service (CMS) domain.



```

54 const cmsCreateInstance = async (req, category, type, authorization) => {
55   let postRequest = {
56     method: "post",
57     url: process.env.BASE_URL + '/cms/instances/' + category + '/' + type + req.originalUrl.replace(/^[^?]*/, ""),
58     headers: {
59       'Authorization': authorization
60     },
61     json: req.body
62   };
63
64   return new Promise((resolve, reject) => {
65     request(postRequest, (error, response) => {
66       if (error) throw new Error("Error in creating cms instance: " + error);
67       if (!String(response.statusCode).startsWith('2')) {
68         console.log('Request failed: ', response.statusCode, response.body);
69         let errorDescription;
70         if (response.body) {
71           if (response.body.details) {
72             errorDescription = response.body.details;
73           }
74           if (response.body.fault) {
75             errorDescription = response.body.faultstring;
76           }
77         }
78         return reject({
79           status: response.statusCode,
80           description: errorDescription
81         });
82       }
83       resolve(response.body);
84     });
85   });
86 }

```

The key part of the above excerpt is what you can find in the **postRequest** object:

```

let postRequest = {
  method: "post",
  url: process.env.BASE_URL + '/cms/instances/' + category + '/' + type + req.originalUrl.replace(/^[^?]*/, ""),
  headers: {
    'Authorization': authorization
  },
  json: req.body
};

```

This should look very familiar, as it corresponds to what we've already been covering in the [Working with the IMaaS APIs](#) exercise. More specifically, if you look at the [CMS API reference](#) and Postman collection respectively, the **postRequest** object represents exactly the same as the following two screen shots:

The left screenshot shows the Postman interface with the 'create new instance' endpoint selected. The right screenshot shows the 'Payload' tab for the same endpoint, displaying a JSON schema for the request body.

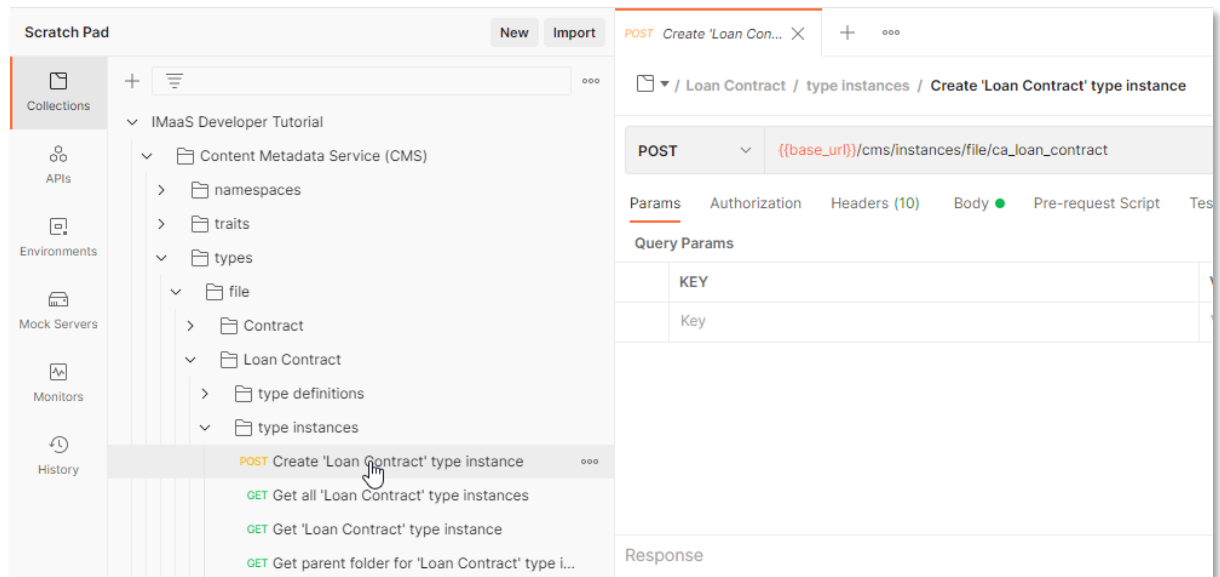
API Endpoint: POST /instances/{category}/{type}

Request Body Schema: application/json

```

{
  "name": "string",
  "description": "string",
  "tags": [
    "string"
  ],
  "traits": {
  },
  "properties": {
  },
  "ancestor_ids": [
    "string"
  ],
  "policies": [
    "string"
  ],
  "renditions": [
    + { ... }
  ]
}

```



This actually takes us to the end of this exercise, as we have covered the front (`server.js`) to back (`CMS.js`) example of creating a new CMS type instance in the Contract Approval application backend.

Feel free to further explore **`server.js`** or the different files under the **`services`** folder if you want more code examples on how to talk to the different IMaaS service endpoints and how to expose their features to the Contract Approval application frontend.

We can now move on to the next exercise, where we will look at the code for the Contract Approval application frontend (i.e.: the UI).

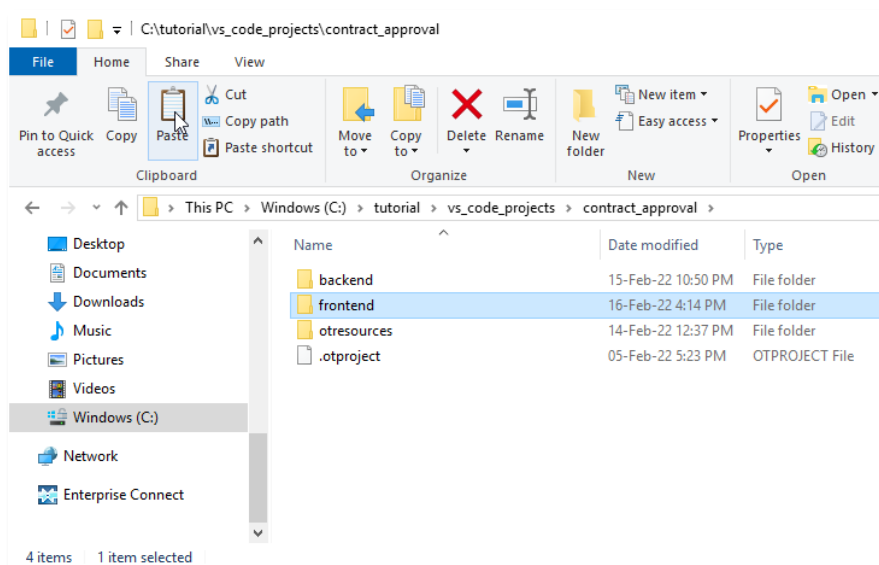
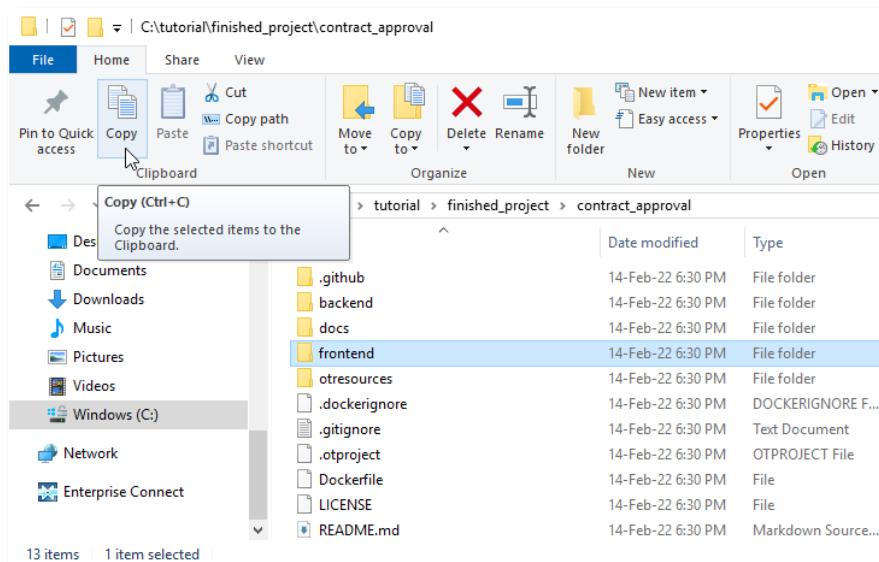
3.13[15'] Building the application frontend

During this exercise we will be going through the frontend code (the UI) of the Contract Approval application. Like with the previous exercise, we will import the code (written in React) and go over its structure, logic and how it calls the Contract Approval application backend. It is again not the intent to go over every single detail of how the code was written, and which file does what, but we will touch upon key aspects of how the frontend has been developed.

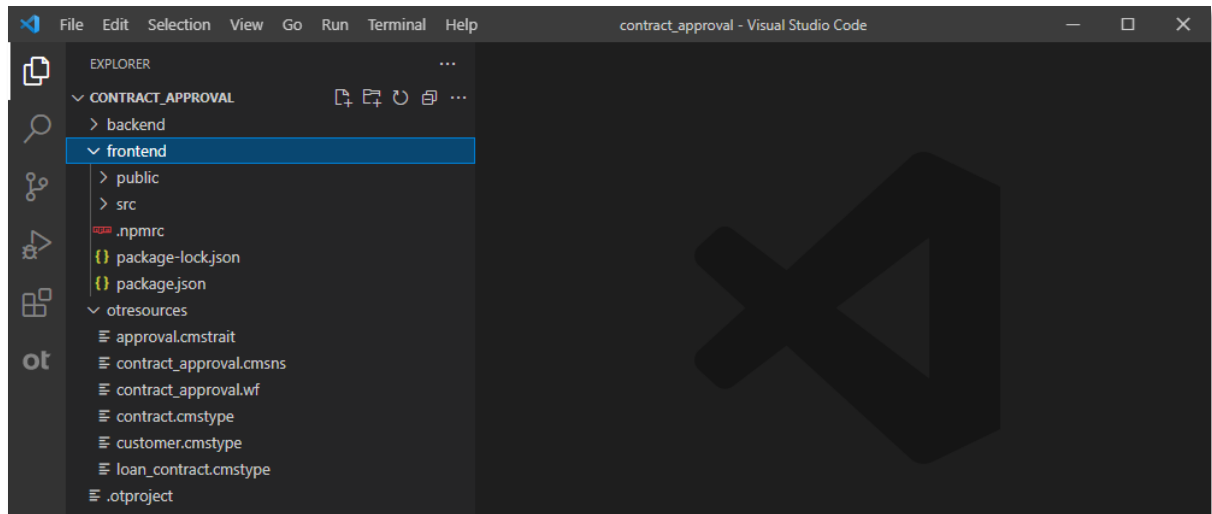
Once you are done with this section, you will have understood how the frontend code of the Contract Approval application has been written, and how it calls the backend services from the previous exercise. This is also the last step in building your Contract Approval application, so in the next exercise you will be running the application and testing the functionality you developed.

To import and go over the frontend code of the Contract Approval application, proceed as follows:

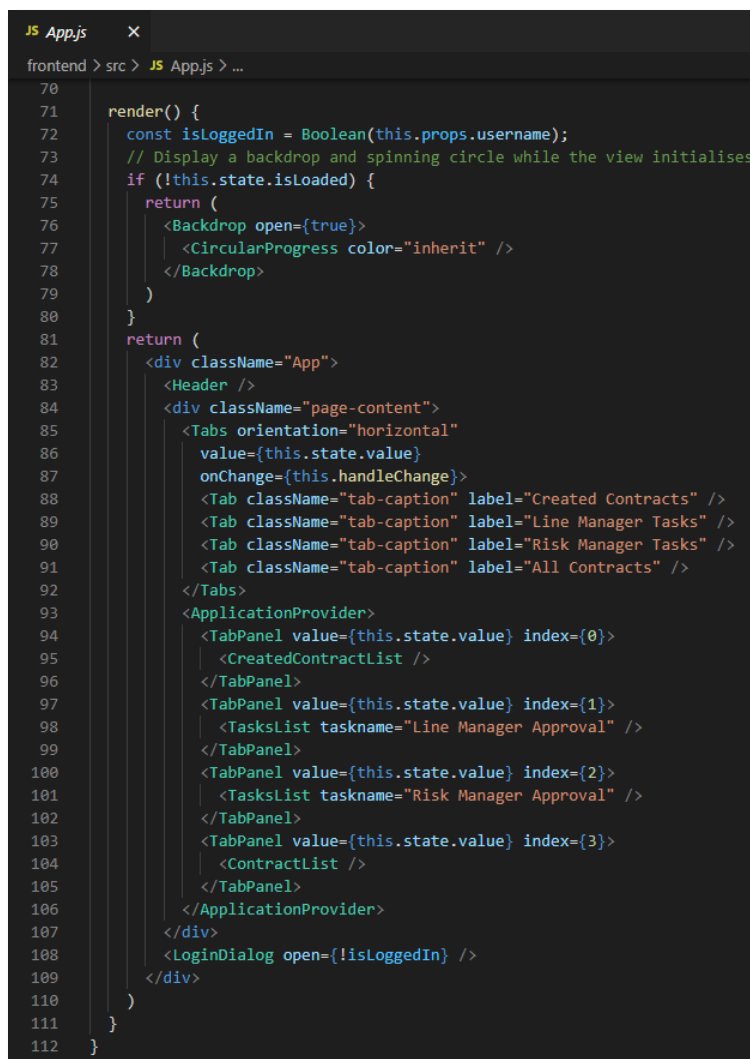
- The first step is to import the frontend code into your VS Code project. To do this, copy the **frontend** folder from the previously extracted finished version of the Contract Approval application project into the root of your Contract Approval project.



You can now open VS Code. The frontend folder should be visible in the **Explorer** view.



- The main entry point for the frontend, representing the application UI as a whole is the **App.js** file, directly available from the **/frontend/src** folder. When you open it, you can directly go to line **71** as the main React code is located in the **render** method.



If we focus on the contents of the **Tabs** and **ApplicationProvider** container components, we can distinguish a four (horizontal) tabs UI layout (with the **Tab** components representing the tabs and the **TabPanel** components representing the corresponding views when clicking the tab). In short, this code will generate a UI with four horizontally stacked tabs:

- The **Created Contracts** tab (with the **CreatedContractList** component providing the “created contracts list” view to show all newly created contracts, i.e.: 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)
- 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)
- The **All Contracts** tab (with the **ContractList** component providing the “all contracts” view to show all contracts in the application, independently of their status)
- Let’s have a closer look at the first of the four views. The “created contracts list” view (i.e.: the **CreatedContractList** React component) does not only show the newly created contracts (status = ‘CREATED’), but it also provides the button to add new contracts to the system. You can open the corresponding **CreatedContractList.js** file directly from the same **/frontend/src** folder you already opened the **App.js** file from.

```
JS CreatedContractList.js X
frontend > src > JS CreatedContractList.js > ...
1  import React from 'react';
2  import axios from 'axios';
3  import {connect} from 'react-redux';
4  import {
5      Backdrop,
6      Button,
7      CircularProgress,
8      IconButton,
9      Paper,
10     Snackbar,
11     Table,
12     TableBody,
13     TableCell,
14     TableContainer,
15     TableHead,
16     TableRow
17 } from '@material-ui/core';
18 import ArrowForwardIosIcon from '@material-ui/icons/ArrowForwardIos';
19 import AddIcon from '@material-ui/icons/Add';
20 import CloseIcon from '@material-ui/icons/Close';
21
22 import ContractDetails from './ContractDetails';
23 import AddContract from './AddContract';
24 import Pagination from './Pagination';
25 import DocumentDialogView from './DocumentDialogView';
26 import MuiAlert from '@material-ui/lab/Alert';
27 import RiskClassification from './RiskClassification';
28
29 function Alert(props) {
30     return <MuiAlert elevation={6} variant="filled" {...props} />;
31 }
32
33
34 /**
35  * This view displays the list of created contracts. From here the user can request approval for any of them.
36  */
37 class CreatedContractList extends React.Component {
38     constructor(props) {
39         super(props);
40
41         this.state = {
```

Again, let's start by looking at the **render** method of this React component.

```

232   render() {
233     return (
234       <div>
235         <Button variant="contained" color="primary" disabled={!this.props.username} startIcon={AddIcon} /> <AddIcon /> onClick={() => this.openCont
236         <div className='content-header'>All created contracts</div>
237         <TableContainer component={Paper}>
238           <Table size="small" aria-label="a dense table">
239             <TableHead>
240               <TableRow>
241                 <TableCell>Contract name</TableCell>
242                 <TableCell align="left">Creation date</TableCell>
243                 <TableCell align="left">Value</TableCell>
244                 <TableCell align="left">Risk classification</TableCell>
245                 <TableCell align="left">View document</TableCell>
246                 <TableCell align="left">Action</TableCell>
247                 <TableCell align="left" />
248               </TableRow>
249             </TableHead>
250             <TableBody>
251               {this.state.contracts.map((row) => (
252                 <TableRow key={row.id}>
253                   <TableCell component="th" scope="row">
254                     {row.name}
255                   </TableCell>
256                   <TableCell align="left">{this.getDateValue(row.create_time)}</TableCell>
257                   <TableCell align="left">{row.properties.value}</TableCell>
258                   <TableCell align="left">{RiskClassification row=row} /></TableCell>
259                   <TableCell align="left">
260                     <Button size="small" variant="outlined" color="primary" onClick={() => { this.openDocumentDialogView(row
261                   </TableCell>
262                   <TableCell align="left">
263                     <Button size="small" variant="outlined" color="primary" onClick={() => { this.startContractForApproval(r
264                   </TableCell>
265                   <TableCell align="left">
266                     <IconButton size="small" variant="outlined" color="primary" title="Show details" onClick={() => { this.s
267                       <ArrowForwardIosIcon />
268                     </IconButton>
269                   </TableCell>
270                 </TableRow>
271               )})
272             </TableBody>
273           </Table>
274         </TableContainer>
275         <Pagination pageNumber={this.state.pageNumber} count={this.state.count} handlePageNumber={this.handlePageNumber} />
276         <ContractDetails open={this.state.openContractDetails} selectedContract={this.state.selectedContract} raiseError={this.raiseError
277         <AddContract open={this.state.openAddContract} onAddContract={this.handleContractAdded} onClose={this.handleCloseAddContract} />
278         <DocumentDialogView open={this.state.openDocumentDialogView} downloadHref={this.state.downloadHref} onClose={this.handleCloseDoc

```

The **TableContainer** component will provide the table layout to display the list of the different contracts that have the 'CREATED' status. As you can see from the **TableHead** and **TableBody** components, each table row will show the following information for the displayed contract:

- **Contract name**
- **Creation date**
- **Value**
- **Risk classification**
- **View document** (this is not a property value but a button to open the actual document in a viewer)
- **Action** (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)

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

If you look at the **getContracts** method on line **113**, you can easily recognize the (axios) GET request to the **/api/cms/instances** Contract Approval application backend endpoint, which in turn is responsible for calling the **/instances** endpoint of the **CMS** IMaaS API and providing the result back to the frontend.

```

113   getContracts() {
114       if (this.props.username) {
115           this.setState({ showBackdrop: true });
116           axios({
117               method: 'get',
118               url: '/api/cms/instances/file/ca_contract/?include-total=true&sortby=create_time desc&filter=status eq "CREATED"&page='
119                   + (this.state.pageNumber + 1),
120           }).then(res => {
121               this.setState({
122                   contracts: res.data && res.data._embedded ? res.data._embedded.collection : [],
123                   count: res.data.total
124               });
125           }).catch(error => {
126               let errorMessage = 'Could not get contracts: ';
127               if (error.response != null && error.response.data != null) {
128                   errorMessage += error.response.data.exception;
129               } else {
130                   errorMessage += error.message;
131               }
132               this.raiseError(errorMessage);
133           }).finally(() => {
134               this.setState({ showBackdrop: false });
135           })
136       } else {
137           this.setState({ contracts: [], count: -1 });
138       }
139   }

```

Going back to the **render** method, let's also take a look at the **AddContract** component (line **277**), as using this feature will trigger a call to the type instance creation endpoint from the Contract Approval backend services we looked at during the previous exercise.

```

277   <AddContract open={this.state.openAddContract} onAddContract={this.handleContractAdded} onClose={this.handleCloseAddContract} />

```

More specifically, it will open the contract creation dialog box to add a contract.

- How the contract creation works is part of the dialog box code, so as the last step in this exercise, let's open the **AddContract.js** React component (again from the **/frontend/src** folder) and look at line **221** where the actual action of calling the IMaaS APIs to create the contract happens.

```

220 // Adding Contract
221 axios.post(
222   '/api/css/uploadcontent?avs-scan=false',
223   formData, {
224     headers: {
225       'Content-Type': 'multipart/form-data'
226     },
227   }
228 ).then(res => {
229   let cmsType;
230   if (this.isLoanContract()) {
231     cmsType = 'ca_loan_contract'
232   } else {
233     cmsType = 'ca_contract'
234   }
235
236   // Setting metadata
237   return axios({
238     method: 'post',
239     url: `/api/cms/instances/file/${cmsType}`,
240     data: {
241       "name": this.state.newContractName,
242       "parent_folder_id": parentFolderId,
243       "renditions": [
244         {
245           "name": res.data.entries[0].fileName,
246           "rendition_type": "primary",
247           "blob_id": res.data.entries[0].id
248         }
249       ],
250       "properties": {
251         "value": parseInt(this.state.newContractValue, 10),
252         "status": "CREATED",
253         "requester_email": customerEmail,
254         "risk_classification": this.state.contractRisk,
255         "extracted_terms": this.state.extractedTerms,
256         ...this.isLoanContract() && {
257           "monthly_installments": parseInt(this.state.newContractMonthlyInstallments),
258           "yearly_income": parseInt(this.state.newContractYearlyIncome)
259         }
260       },
261       "traits": {
262         "ca_approval": {
263           "Automatic Approval": ☐
264           "is_required": true,
265           "has_been_granted": false,
266           "approver": "",
267           "approver_role": "",

```

Note that there are two subsequent (axios) POST requests. One at line **221** to upload the file to the **Content Storage Service (CSS)** and one at line **237** to create the contract metadata in the **Content Metadata Service (CMS)** with the contract's properties, rendition (linked to the previously uploaded CSS file), and traits as payload.

The second post request (with url: ``/api/cms/instance/file/${cmsType}``) is the one that will call the **server.js** (and underlying **CMS.js**) code we discussed in [Building the application backend](#).

CONGRATULATIONS!

You have now completely finished building your Contract Approval application. In the next exercise we will test it and run through the different contract approval scenarios.

3.14[50'] Testing your application

During this exercise we will be testing the Contract Approval application you have just built. We 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.

Once you are done with this section, you will have tested the different application flows and will have effectively completed the main part of the tutorial. Still, there will be one more remaining exercise that we recommend you run through. During this bonus exercise, you will learn how to use the OT2 Command Line Interface (CLI) to perform IMaaS platform related operations (such as deploying a project). This is certainly valuable, for example, in context of build automation and CI/CD use cases.

READ THIS IF YOU SKIPPED THE PREVIOUS EXERCISES:

If you decided to skip ahead and test the application without actually going through the building exercises, you will need to do a few additional steps before you can start.

To ensure that you can correctly test the Contract Approval application, perform the following activities:

- Be certain that you have checked and fulfilled the [Prerequisites](#).
- Set up your IDE as described in [Setting up the IMaaS Developer IDE](#).
- Connect to your developer organization as described in [Adding an organization and testing the connection](#).
- Download the finished version of the Contract Approval application (ZIP file) through this [link](#) and extract it.
- Once downloaded and extracted, make sure to open the **contract_approval** folder in VS Code, as this is the root of your project.
- Finally, deploy the application into your developer organization as described in [Deploying the application to the IMaaS services](#).

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

REMARK:

Although you did not go through the steps to build the application yourself, you might still want to understand how it was built. To that end, you can just browse the previous exercises, and/or look at the VS Code project you just downloaded.

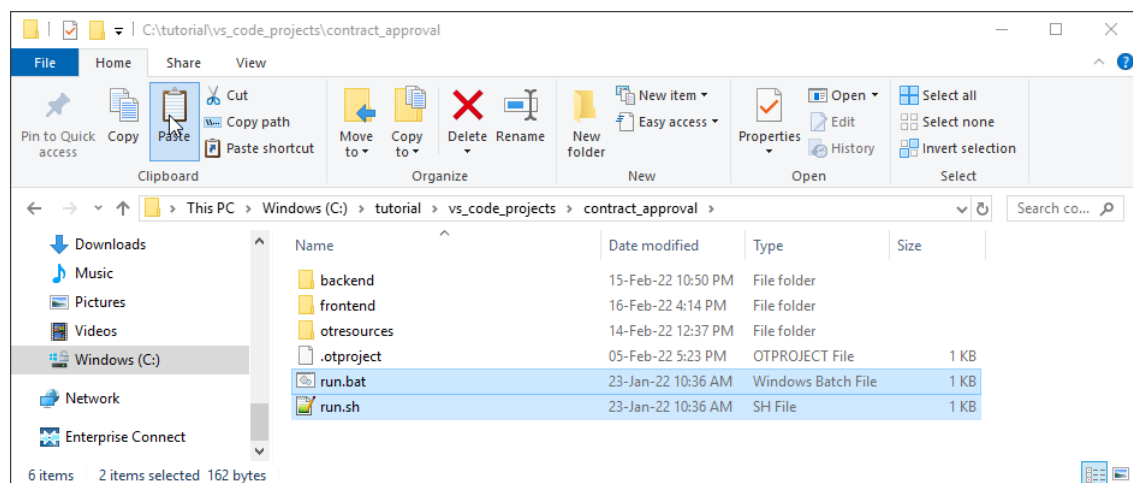
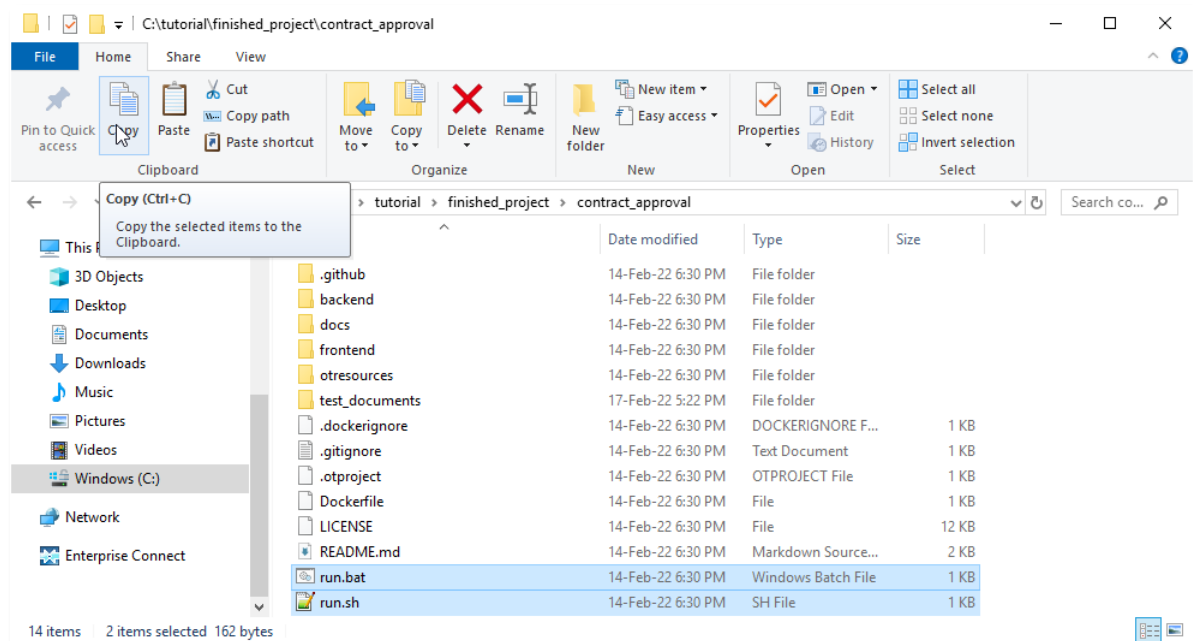
If you want to dive into the VS Code project, this is a very short explanation of its main project folders:

- **backend**: contains the node.js code that delivers the REST API backend services of the application (this is where the communication with the IMaaS APIs happens)
- **frontend**: contains the React code that delivers the User Interface (UI) or frontend of the application
- **otresources**: contains the different IMaaS models (built with the OpenText IMaaS Tools for VS Code) that will be deployed to the IMaaS services of the OT2 platform

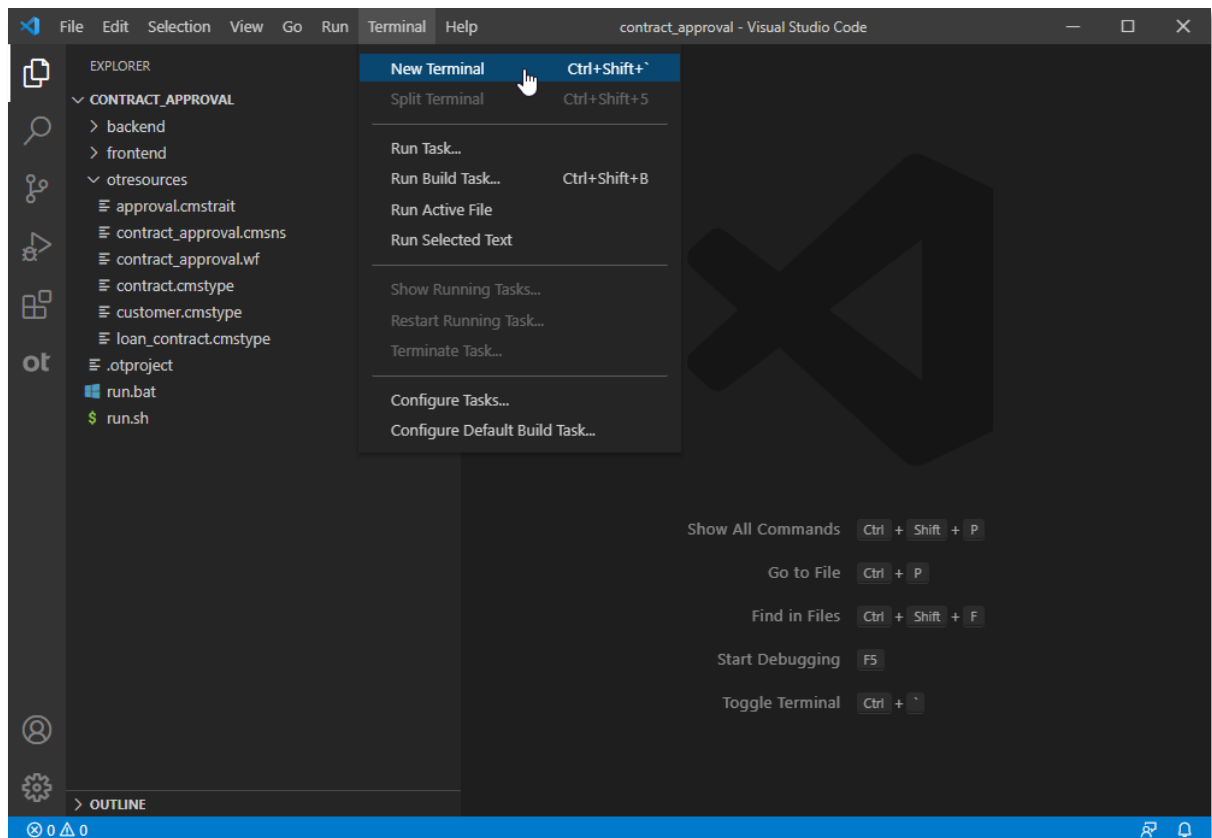
To test the Contract Approval application, proceed as follows:

- To simplify running the Contract Approval application, which implies starting the frontend and backend servers (on different ports), we have prepared a (shell) script for Linux/macOS and a bat file for Windows.

So, before you actually start running and testing the application, please copy the **run.sh** and **run.bat** files from the previously extracted finished version of the Contract Approval application project into the root of your Contract Approval project.



- We can now start the application. To do this, open your Contract Approval application project in VS Code and open a new **Terminal** window.



In the **Terminal** window execute **run.bat** (for Windows) or **run.sh** (for non-Windows). This should start all the application components and result in a new browser window opening on **localhost:3000**.

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\tutorial_qa\vs_code_projects\contract_approval> .\run.bat

> core-js@2.6.12 postinstall C:\tutorial_qa\vs_code_projects\contract_approval\backend\node_modules\core-js
> node -e "try{require('./postinstall')}catch(e){}"

Thank you for using core-js ( https://github.com/zloirock/core-js ) for polyfilling JavaScript standard library!

The project needs your help! Please consider supporting of core-js on Open Collective or Patreon:
> https://opencollective.com/core-js
> https://www.patreon.com/zloirock

Also, the author of core-js ( https://github.com/zloirock ) is looking for a good job -)

added 398 packages from 303 contributors and audited 398 packages in 9.688s

32 packages are looking for funding
  run `npm fund` for details

found 3 vulnerabilities (2 moderate, 1 high)
  run `npm audit fix` to fix them, or `npm audit` for details

> contract-approval@0.1.0 preinstall C:\tutorial_qa\vs_code_projects\contract_approval\frontend
> npx npm-force-resolutions

npx: installed 6 in 3.987s

```

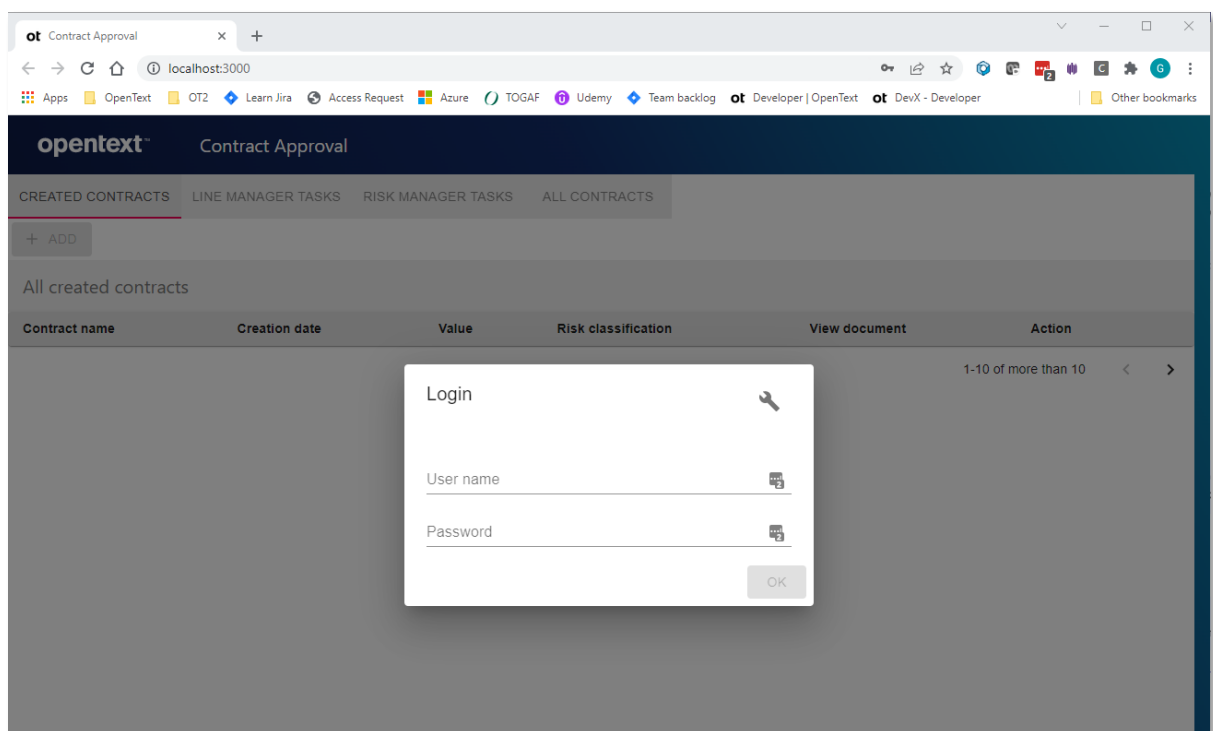



```

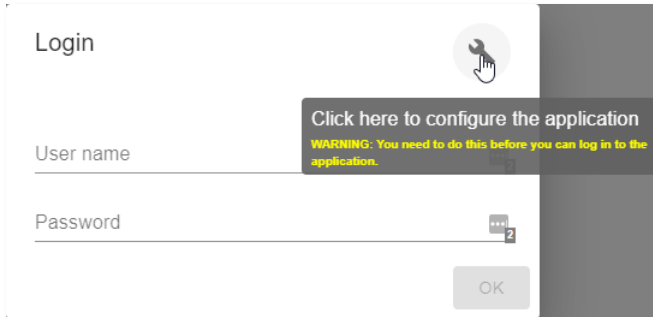
> contract-approval@0.1.0 start C:\tutorial_qa\vs_code_projects\contract_approval\frontend
> concurrently "react-scripts start" "npm start --prefix ../backend"

[1]
[1] > lookup-server@0.1.0 start C:\tutorial_qa\vs_code_projects\contract_approval\backend
[1] > npm run server
[1]
[1]
[1] > lookup-server@0.1.0 server C:\tutorial_qa\vs_code_projects\contract_approval\backend
[1] > node server.js
[1]
[1] Find the server at: http://localhost:3001/
[0] i [wds]: Project is running at http://10.230.188.19/
[0] i [wds]: webpack output is served from
[0] i [wds]: Content not from webpack is served from C:\tutorial_qa\vs_code_projects\contract_approval\frontend\public
[0] i [wds]: 404s will fallback to /
[0] Starting the development server...
[0]
[0] Browserslist: caniuse-lite is outdated. Please run:
[0] npx browserslist@latest --update-db
[0]
[0] Why you should do it regularly:
[0] https://github.com/browserslist/browserslist#browsers-data-updating
[0] Compiled successfully!
[0]
[0] You can now view contract-approval in the browser.
[0]
[0] Local:      http://localhost:3000
[0] On Your Network: http://10.230.188.19:3000
[0]
[0] Note that the development build is not optimized.
[0] To create a production build, use npm run build.
[0]
[1] Request received to /session
[1] Request received to /configuration/url
[1] Request received to /configuration
[1]

```

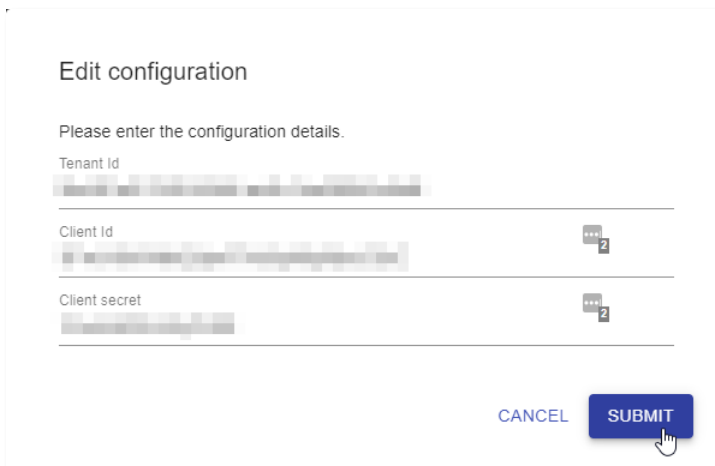


- Before you can log in, you need to configure the application. More specifically, you need to add the application keys and tenant id to the application configuration settings to allow the Contract Approval application backend services to authenticate and interact with the IMaaS APIs. To do this, click the  on the top right of the **Login** screen.



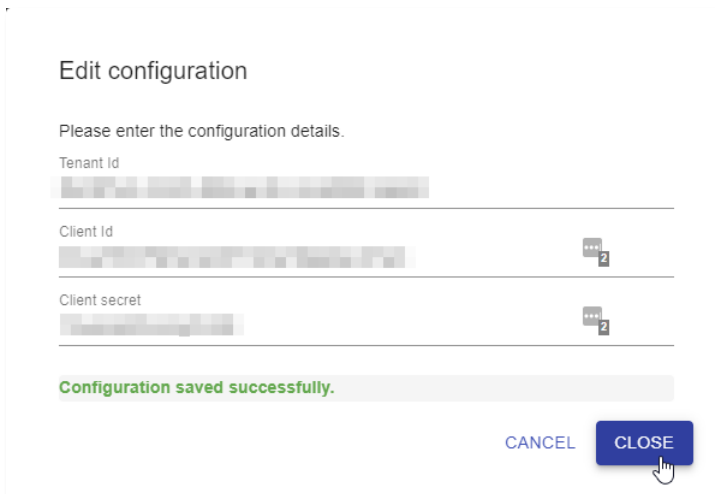
The screenshot shows the 'Login' screen. At the top right, there is a wrench icon. A dark grey tooltip box is overlaid on the screen, containing the text 'Click here to configure the application' and a yellow warning message: 'WARNING: You need to do this before you can log in to the application.' Below the tooltip, there are input fields for 'User name' and 'Password', and an 'OK' button at the bottom right.

Enter the **Tenant Id**, **Client Id** (make sure to fill the confidential client id, not the public one) and **Client secret** for your deployed application, and click **Submit** to store them in your application configuration settings. If you followed the tutorial exactly and used the **contract_approval_app_config.txt** file to save the application configuration information, the values to fill should be available from that file.



The screenshot shows the 'Edit configuration' screen. It has a title 'Edit configuration' and a subtitle 'Please enter the configuration details.' Below this are three input fields: 'Tenant Id', 'Client Id', and 'Client secret'. Each field has a small icon to its right. At the bottom right, there are two buttons: 'CANCEL' and 'SUBMIT'. A mouse cursor is pointing at the 'SUBMIT' button.

After submitting, you should get a **Configuration saved successfully** message. You can click **Close** to close the application configuration editing screen.



The screenshot shows the 'Edit configuration' screen after a successful submission. It has the same title and subtitle as the previous screen. Below the input fields, there is a green message bar that says 'Configuration saved successfully.' At the bottom right, there are two buttons: 'CANCEL' and 'CLOSE'. A mouse cursor is pointing at the 'CLOSE' button.

- You can now log in using your tenant service account email (as username) and password. Again, if you have been following the exact tutorial steps, the password should be available from the **contract_approval_app_config.txt** file.



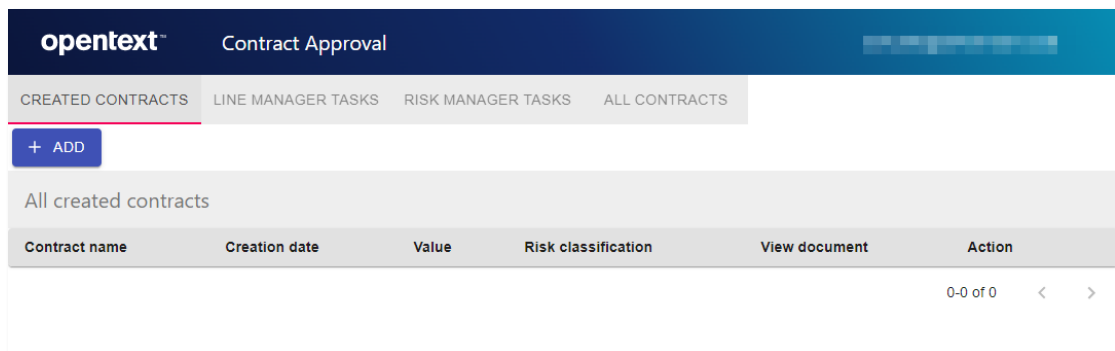
Login

User name

Password

OK

- You are now logged in to your Contract Approval application. Let's have a look at the different tabs (the four tabs we previously described when discussing the frontend code).
 - CREATED CONTRACTS: this tab shows all newly created contracts that have not yet been submitted for approval (i.e.: contracts with the status attribute equal to 'CREATED')



opentext Contract Approval

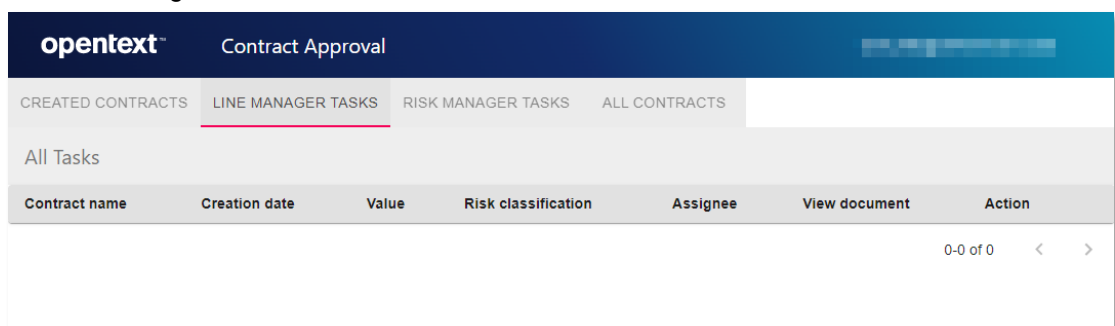
CREATED CONTRACTS LINE MANAGER TASKS RISK MANAGER TASKS ALL CONTRACTS

+ ADD

All created contracts

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

- LINE MANAGER APPROVAL TASKS: this tab shows all approval tasks to be performed by the Line Manager



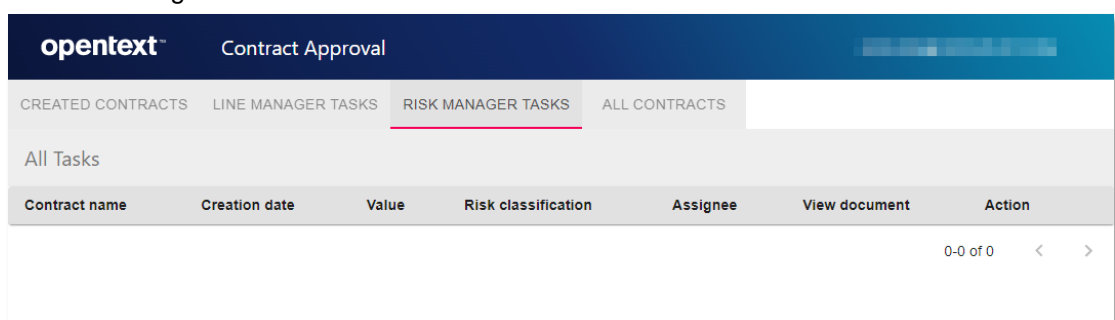
opentext Contract Approval

CREATED CONTRACTS LINE MANAGER TASKS RISK MANAGER TASKS ALL CONTRACTS

All Tasks

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

- RISK MANAGER APPROVAL TASKS: this tab shows all approval tasks to be performed by the Risk Manager



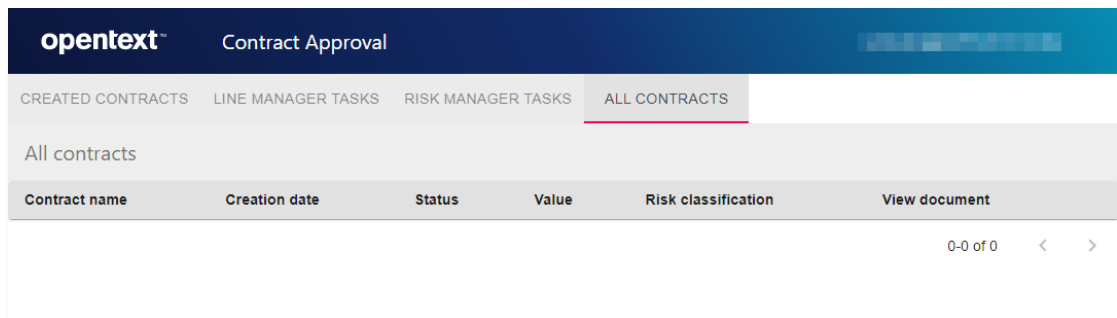
opentext Contract Approval

CREATED CONTRACTS LINE MANAGER TASKS RISK MANAGER TASKS ALL CONTRACTS

All Tasks

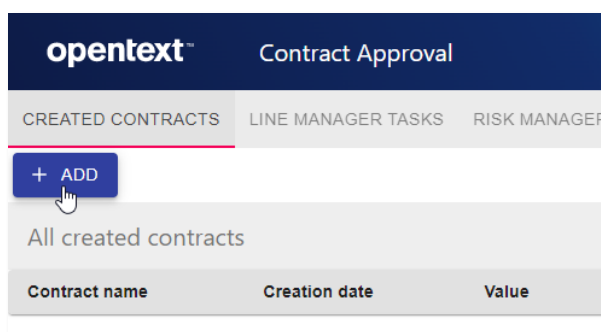
Contract name	Creation date	Value	Risk classification	Assignee	View document	Action
0-0 of 0						

- **ALL CONTRACTS:** this tab shows all contracts, independently of their status (i.e.: newly created, under approval, approved, rejected, expired)



- Now that we have looked at the different tabs, let's create and approve our first contract. Let's start with the simplest approval process, i.e.: 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, i.e.: NONE, LOW or MEDIUM** (doesn't require Risk Manager approval)

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

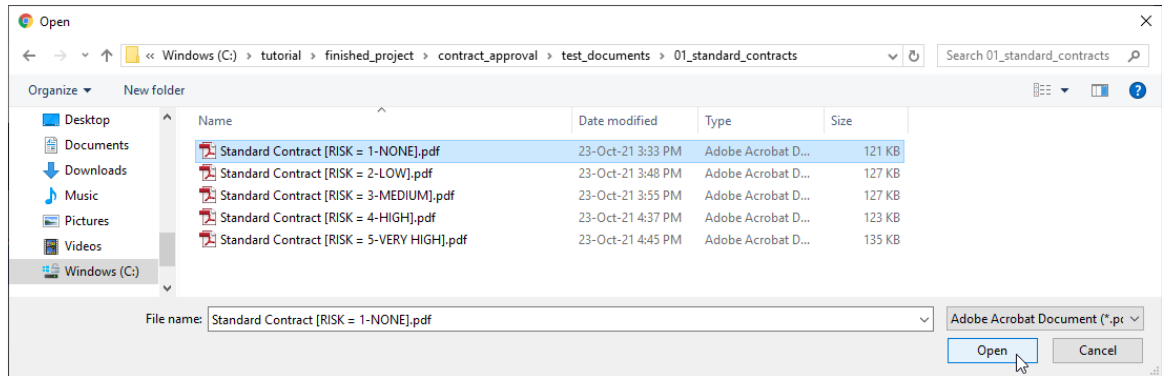


From the **Add Contract** screen, click **SELECT DOCUMENT** to add the contract content file.

 The screenshot shows the 'Add Contract' form. At the top, the title 'Add Contract' is displayed. Below the title, there is a blue button with the text 'SELECT DOCUMENT'. A mouse cursor is pointing at the button. Below the button, there are two radio buttons: 'Standard Contract' (which is selected) and 'Loan Contract'. Below the radio buttons, there are three text input fields: 'Document name', 'Contract value', and 'Contract requester email'. At the bottom right of the form, there are two buttons: 'ADD' and 'CANCEL'.

To help with selecting a file that matches the intended contract properties (certainly the risk classification, as this gets determined by what the Magellan Risk Guard text mining service discovers in the document), we have provided a **test_documents** folder 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.



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

- **Document name:** First standard contract
- **Contract value:** 500
- **Contract requester email:** <your email>

Click **Add** to create the contract.

Add Contract

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

☒ Standard Contract ☐ Loan Contract

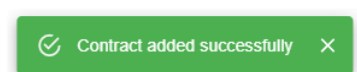
Document name
First standard contract

Contract value
500

Contract requester email
[Redacted]

ADD **CANCEL**

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



Contract name	Creation date	Value	Risk classification	View document	Action
First standard contract	2/17/2022, 6:34:11 PM	500	NONE	ORIGINAL	REQUEST APPROVAL >

1-1 of 1

Your first standard contract is now created. Let's explore the contract list capabilities from this **CREATED CONTRACTS** view.

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

Contract name	Creation date	Value	Risk classification	View document	Action
First standard contract	2/17/2022, 6:34:11 PM	500	NONE	ORIGINAL	REQUEST APPROVAL >

Show extracted personal data

1-1 of 1

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.

Extracted Terms
Social Security Numbers:
Credit Card Numbers:
Bank Accounts:
Person Names:
- Peter M. Townsend
- Tomas U. Britton
Phone Numbers:
Addresses:
- 1200 Irwinton Avenue
- 92 Homer Avenue
Geographic Locations:
- Helena, Georgia
- State of Georgia
Organization Names:
- The Company
CLOSE

Click on the **ORIGINAL** button (in the **View document** column) to view the uploaded document content.

The screenshot shows the OpenText Contract Approval interface. At the top, there's a header with the OpenText logo and 'Contract Approval'. Below it, there are tabs: 'CREATED CONTRACTS' (selected), 'LINE MANAGER TASKS', 'RISK MANAGER TASKS', and 'ALL CONTRACTS'. A '+ ADD' button is on the left. The main area is titled 'All created contracts' and contains a table with the following columns: 'Contract name', 'Creation date', 'Value', 'Risk classification', 'View document', and 'Action'. The table has one row: 'First standard contract', '2/17/2022, 6:34:11 PM', '500', 'NONE' (with an info icon), 'ORIGINAL' (highlighted by a mouse cursor), and 'REQUEST APPROVAL' (with a right arrow icon). At the bottom right, it says '1-1 of 1' with navigation arrows.

The file content of the contract indeed displays to be viewed.
Click **CLOSE** to close the document viewer screen.

The screenshot shows a document viewer for the 'EMPLOYEE REMOTE WORK CONTRACT'. The title is centered at the top. Below it, the document text is displayed. The text includes sections for 'Parties', 'WHEREAS', and 'IN CONSIDERATION'. At the bottom right, there is a blue 'CLOSE' button with a mouse cursor hovering over it.

Before requesting to start the contract approval process (clicking the **REQUEST APPROVAL** button under the **Action** column), click on **>** to view the contract details.

This screenshot is similar to the first one, showing the OpenText Contract Approval interface. The table is the same, but the mouse cursor is now over the right arrow icon in the 'Action' column of the 'First standard contract' row. A tooltip labeled 'Show details' is visible next to the arrow. The 'ORIGINAL' button is still visible in the 'View document' column.

In the **Contract details** screen, you can see two 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).

Click the **APPROVALS** tab to have a look at the different approvals for this contract.

Contract details

PROPERTIES APPROVALS

Name
First standard contract

Status
CREATED

Value
500

Risk classification
1

Creation date
2/17/2022, 6:34:11 PM

Contract requester email
[REDACTED]

CLOSE

The **APPROVALS** tab displays the different approval steps (traits) for the standard contract CMS type (**Automatic Approval**, **Line Manager Approval** and **Risk Manager Approval**). As you can see, the only approval step that has been marked as required is the **Automatic Approval**.

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

Contract details

PROPERTIES APPROVALS

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

CLOSE

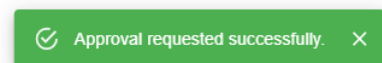
Back into the **CREATED CONTRACTS** view, we can now launch the approval workflow. Click the **REQUEST APPROVAL** button in the **Action** column to do that.

The screenshot shows the 'Contract Approval' interface with the 'CREATED CONTRACTS' tab selected. Below the tabs is a '+ ADD' button and the text 'All created contracts'. A table displays the following data:

Contract name	Creation date	Value	Risk classification	View document	Action
First standard contract	2/17/2022, 6:34:11 PM	500	NONE ⓘ	ORIGINAL	REQUEST APPROVAL

At the bottom right, it shows '1-1 of 1' with navigation arrows.

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



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


The screenshot shows the 'Contract Approval' interface with the 'CREATED CONTRACTS' tab selected. The table is now empty, and the bottom right shows '0-0 of 0'.

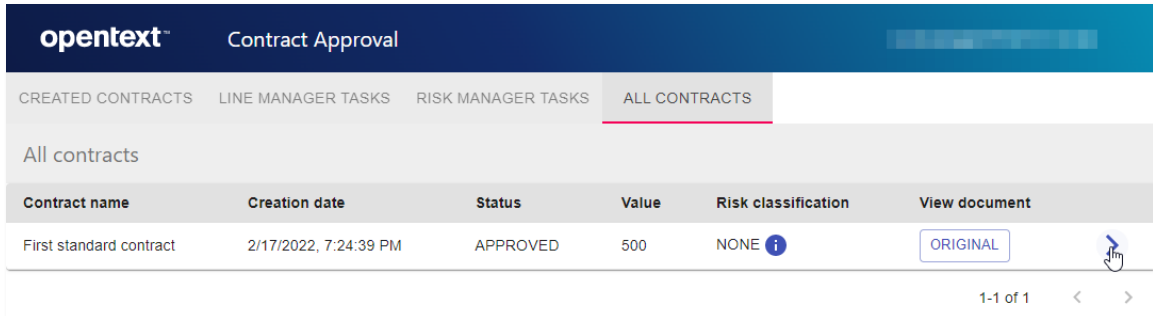
Since the value of the contract is below 1000, there is no approval task waiting in the Line Manager task inbox.


The screenshot shows the 'Contract Approval' interface with the 'LINE MANAGER TASKS' tab selected. The table is empty, and the bottom right shows '0-0 of 0'.

There is also no approval task waiting in the Risk Manager task inbox, as the risk classification is NONE (1) which is certainly below HIGH (4).

The screenshot shows the 'Contract Approval' interface with the 'RISK MANAGER TASKS' tab selected. The table is empty, and the bottom right shows '0-0 of 0'.

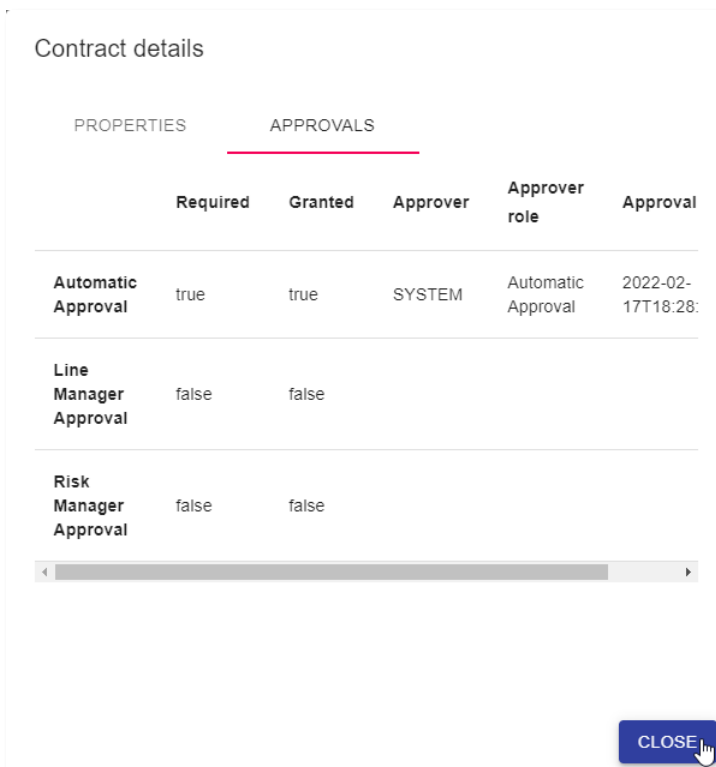
When you open the **ALL CONTRACTS** view, you will see that the contract has been automatically approved (**Status** column shows **APPROVED** status).
Let's now have another look at the contract details as well, and more specifically, the approval steps/traits (click  and select the **APPROVALS** tab).



Contract name	Creation date	Status	Value	Risk classification	View document
First standard contract	2/17/2022, 7:24:39 PM	APPROVED	500	NONE 	ORIGINAL

1-1 of 1

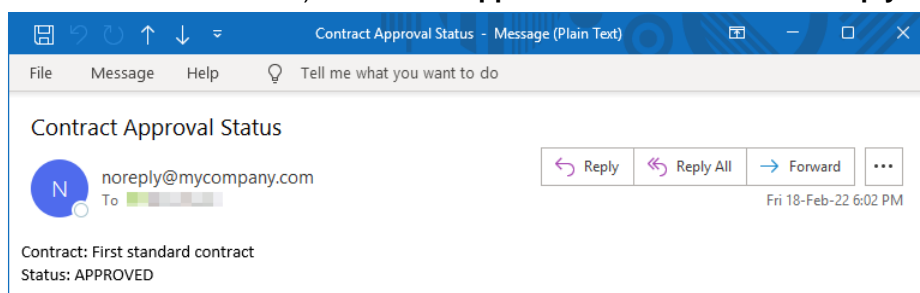
As you can see, the **APPROVALS** tab still shows that only the **Automatic Approval** was required, but with the difference that it has now been granted by **Approver SYSTEM** with the **Approver role** of **Automatic Approval** at a specific **Approval date** and time.
Click **CLOSE** to return to the **ALL CONTRACTS** view.



	Required	Granted	Approver	Approver role	Approval
Automatic Approval	true	true	SYSTEM	Automatic Approval	2022-02-17T18:28:00
Line Manager Approval	false	false			
Risk Manager Approval	false	false			

[CLOSE](#)

Your first standard contract has now been approved, and you should have received (due to the email task in the workflow) a **Contract Approval Status** email from **noreply@mycompany.com**.

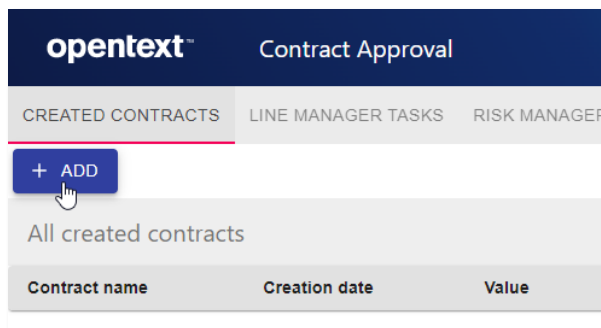


- The first contract we just created and approved has allowed us to go over the different application components and run through the contract approval process. As you know from building the workflow model, there are different possible scenarios that we cater for (standard contract vs. loan contract, solvency check, line manager approval, risk manager approval, automatic approval, reject, and expire). For the rest of this exercise, we will go through these different scenarios by creating and approving (or rejecting, and even expiring) additional contracts.

The second contract we will be creating will follow the most extensive process flow, i.e.: we will create a contract with the following characteristics:

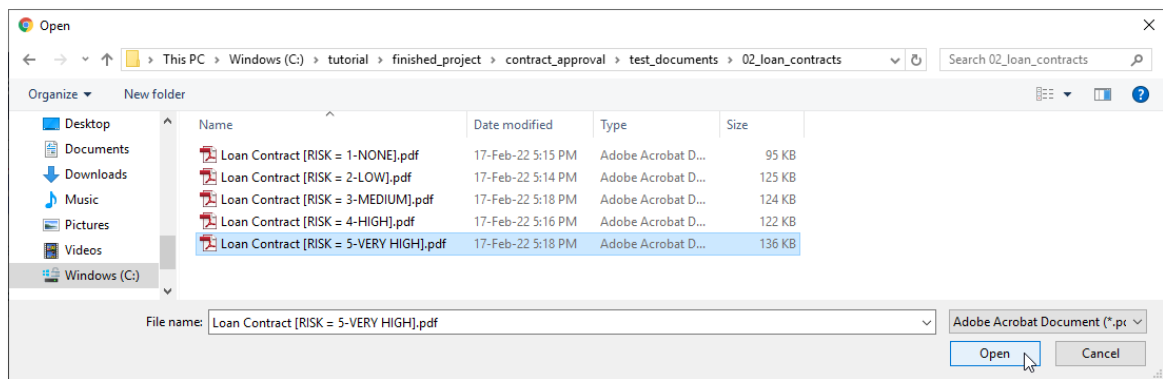
- **Type: loan contract** (requires solvency check)
- **Monthly loan cost is below or equal to 25% of monthly income** (requester is solvent, so automatic solvency check should not reject the contract approval request)
- **Value: above 1000** (requires Line Manager approval)
- **Risk classification: above 3, i.e.: 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.

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



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

- **Document name:** First loan contract
- **Contract value:** 12000
- **Monthly installments:** 12
- **Yearly income:** 100000
- **Contract requester email:** <your email>

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

opentext™ Contract Approval						
<div> CREATED CONTRACTS LINE MANAGER TASKS RISK MANAGER TASKS ALL CONTRACTS </div> <div> + ADD </div>						
All created contracts						
Contract name	Creation date	Value	Risk classification	View document	Action	
First loan contract	2/18/2022, 3:57:36 PM	12000	VERY HIGH i	ORIGINAL	REQUEST APPROVAL	>
1-1 of 1 < >						

Click on the  icon right next to the **VERY HIGH** risk classification value to see which terms the call to the Magellan Risk Guard API has identified and extracted.

opentext™ Contract Approval						
CREATED CONTRACTS LINE MANAGER TASKS RISK MANAGER TASKS ALL CONTRACTS						
+ ADD						
All created contracts						
Contract name	Creation date	Value	Risk classification	View document	Action	
First loan contract	2/18/2022, 3:57:36 PM	12000	VERY HIGH 	ORIGINAL	REQUEST APPROVAL	>
1-1 of 1 < >						

Contrary to the previous contract we 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. 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:	- Emmerson Andrew-James
	- Sandra Kauffmann
	- Andy McIntosh
	- Emily Jackson
	- Pete Peterson
	- Janette Jamison-Johnson
	- Michael Colbert-Johnson
	- James B. Davis
	- Brenda Ann Oliver-Johnson
	- Frank Joe Parker
	- John W. Snow
	- James Bond
	- Anna C. Quebral
	- Anthony Hopkins
Phone Numbers:	
Addresses:	- 4447 Elk City Road
	- 3404 Davisson Street
Geographic Locations:	
Organization Names:	- Innovate Bank
CLOSE	

Before requesting to start the contract approval process, click on ➤ to view the contract details for your first loan contract.

opentext™

Contract Approval

CREATED CONTRACTS

LINE MANAGER TASKS

RISK MANAGER TASKS

ALL CONTRACTS

+ ADD

All created contracts

Contract name	Creation date	Value	Risk classification	View document	Action
First loan contract	2/18/2022, 3:57:36 PM	12000	VERY HIGH ⓘ	<div>ORIGINAL</div>	<div>REQUEST APPROVAL ⓘ</div>

1-1 of 1

<

>

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).

Click the **APPROVALS** tab to have a look at the different approvals for this contract.

Contract details

PROPERTIES

APPROVALS ⓘ

Name

First loan contract

Status

CREATED

Value

12000

Monthly installments

12

Yearly income

100000

Risk classification

5

Creation date

2/18/2022, 3:57:36 PM

Contract requester email

CLOSE

The **APPROVALS** tab displays the different approval steps (traits) for the loan contract CMS type (**Automatic Approval**, **Line Manager Approval**, **Risk Manager Approval**, and the additional **Solvency Check**). As you can see, there are now more approval steps that have been marked as required: the **Automatic Approval** and the **Solvency Check**.

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

Contract details

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

[CLOSE](#)

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

opentext™ Contract Approval					
CREATED CONTRACTS LINE MANAGER TASKS RISK MANAGER TASKS ALL CONTRACTS					
+ ADD					
All created contracts					
Contract name	Creation date	Value	Risk classification	View document	Action
First loan contract	2/18/2022, 3:57:36 PM	12000	VERY HIGH i	ORIGINAL	REQUEST APPROVAL
1-1 of 1 < >					

The new contract has now disappeared from the **CREATED CONTRACTS** view, as it is no longer in 'CREATED' status. Let's have a look at the **ALL CONTRACTS** tab to see its current status. The status is now 'LINE MANAGER APPROVAL' (since the value of the contract is above 1000 a Line Manager approval is required).

Click on > to view the contract details again.

opentext™ Contract Approval						
CREATED CONTRACTS LINE MANAGER TASKS RISK MANAGER TASKS ALL CONTRACTS						
All contracts						
Contract name	Creation date	Status	Value	Risk classification	View document	
First loan contract	2/18/2022, 3:57:36 PM	LINE MANAGER APPROVAL	12000	VERY HIGH <i>i</i>	ORIGINAL	>
First standard contract	2/18/2022, 9:58:13 AM	APPROVED	500	NONE <i>i</i>	ORIGINAL	>
1-2 of 2 < >						

You can now see that the **Solvency Check** approval has been granted (since the requester is indeed solvent) and that the **Line Manager Approval** is required (and the approver has been assigned).

Contract details				
PROPERTIES		APPROVALS		
	Required	Granted	Approver	Approver role
Automatic Approval	true	false		
Line Manager Approval	true	false		Line Manager
Risk Manager Approval	false	false		
Solvency Check	true	true	SYSTEM	Solvency Check
CLOSE				

Go to the **LINE MANAGER TASKS** tab. As you can see, an approval task is waiting for the Line Manager to approve.

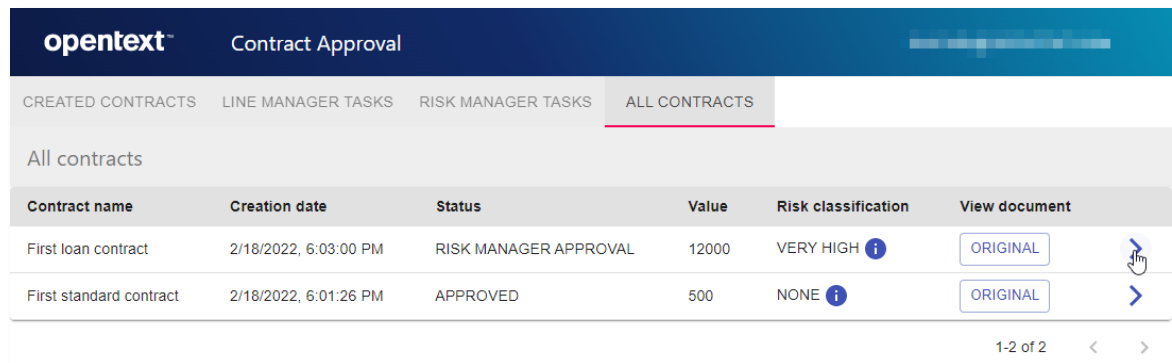
Click **APPROVE** to approve as the Line Manager.

opentext™ Contract Approval						
CREATED CONTRACTS LINE MANAGER TASKS RISK MANAGER TASKS ALL CONTRACTS						
All Tasks						
Contract name	Creation date	Value	Risk classification	Assignee	View document	Action
First loan contract	2/18/2022, 6:03:06 PM	12000	VERY HIGH <i>i</i>		ORIGINAL	APPROVE REJECT >
1-1 of 1 < >						

The contract has now disappeared from the **LINE MANAGER TASKS** view, as it is no longer in 'LINE MANAGER APPROVAL' status. Let's have a look at the **ALL CONTRACTS** tab to see its current status.

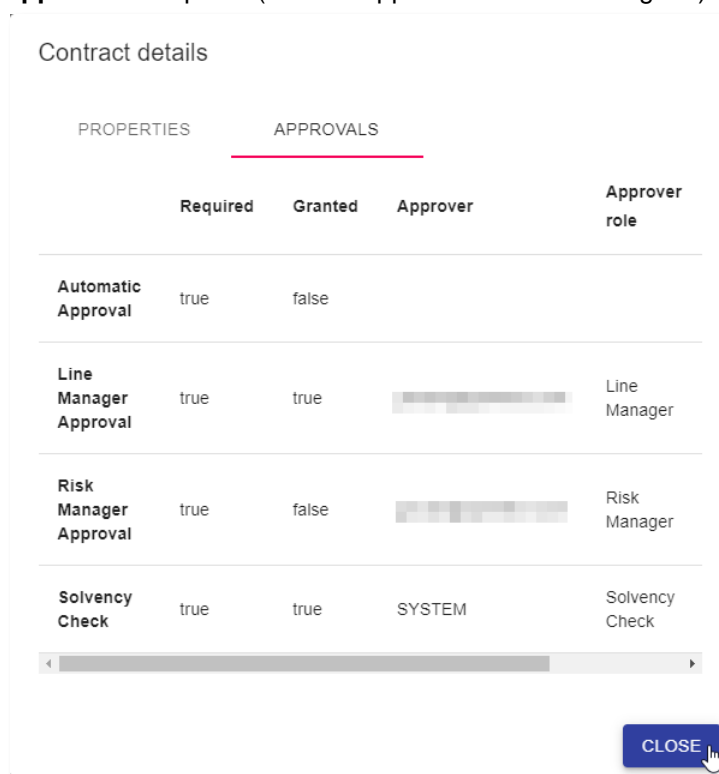
The status is now 'RISK MANAGER APPROVAL' (since the risk classification of the contract is VERY HIGH).

Click on > to view the contract details again.



opentext Contract Approval						
CREATED CONTRACTS	LINE MANAGER TASKS	RISK MANAGER TASKS	ALL CONTRACTS			
All contracts						
Contract name	Creation date	Status	Value	Risk classification	View document	
First loan contract	2/18/2022, 6:03:00 PM	RISK MANAGER APPROVAL	12000	VERY HIGH <i>i</i>	ORIGINAL	<i>hand cursor</i>
First standard contract	2/18/2022, 6:01:26 PM	APPROVED	500	NONE <i>i</i>	ORIGINAL	>
						1-2 of 2 < >

You can now see that the **Line Manager Approval** has been granted and that the **Risk Manager Approval** is required (and the approver has been assigned).



Contract details				
	APPROVALS			
	Required	Granted	Approver	Approver role
Automatic Approval	true	false		
Line Manager Approval	true	true	<i>blurred name</i>	Line Manager
Risk Manager Approval	true	false	<i>blurred name</i>	Risk Manager
Solvency Check	true	true	SYSTEM	Solvency Check

CLOSE *hand cursor*

Go to the **RISK MANAGER TASKS** tab. As you can see, an approval task is waiting for the Risk Manager to approve.

Click **APPROVE** to approve as the Risk Manager.

Contract name	Creation date	Value	Risk classification	Assignee	View document	Action
First loan contract	2/18/2022, 6:07:37 PM	12000	VERY HIGH <i>i</i>	[Avatar]	ORIGINAL	APPROVE REJECT >

1-1 of 1 < >

The contract has now disappeared from the **RISK MANAGER TASKS** view, as it is no longer in 'RISK MANAGER APPROVAL' status. When you open the **ALL CONTRACTS** view, you will see that the contract has been automatically approved (**Status** column shows **APPROVED** status). Click on > to view the contract details to check the different approvals (traits).

Contract name	Creation date	Status	Value	Risk classification	View document
First loan contract	2/18/2022, 6:03:00 PM	APPROVED	12000	VERY HIGH <i>i</i>	ORIGINAL >
First standard contract	2/18/2022, 6:01:26 PM	APPROVED	500	NONE <i>i</i>	ORIGINAL >

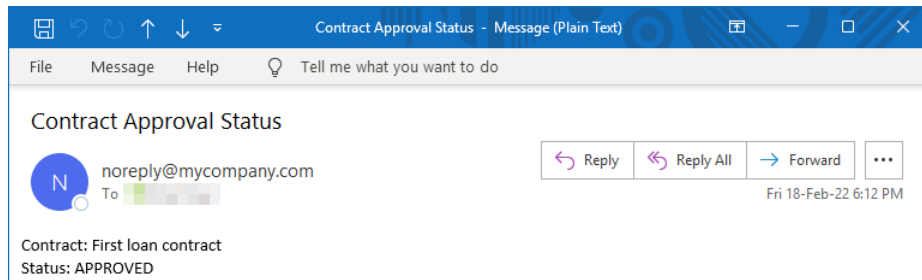
1-2 of 2 < >

You can now see that both the **Risk Manager Approval** and the **Automatic Approval** have been granted. I.e.: all four approvals were required for this loan contract, and all four approvals have been granted.

	Required	Granted	Approver	Approver role
Automatic Approval	true	true	SYSTEM	Automatic Approval
Line Manager Approval	true	true	[Avatar]	Line Manager
Risk Manager Approval	true	true	[Avatar]	Risk Manager
Solvency Check	true	true	SYSTEM	Solvency Check

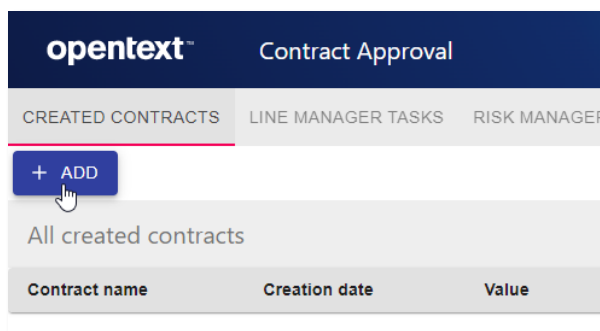
CLOSE

Your first loan contract has now been approved, and you should have received the corresponding **Contract Approval Status** email from **noreply@mycompany.com**.



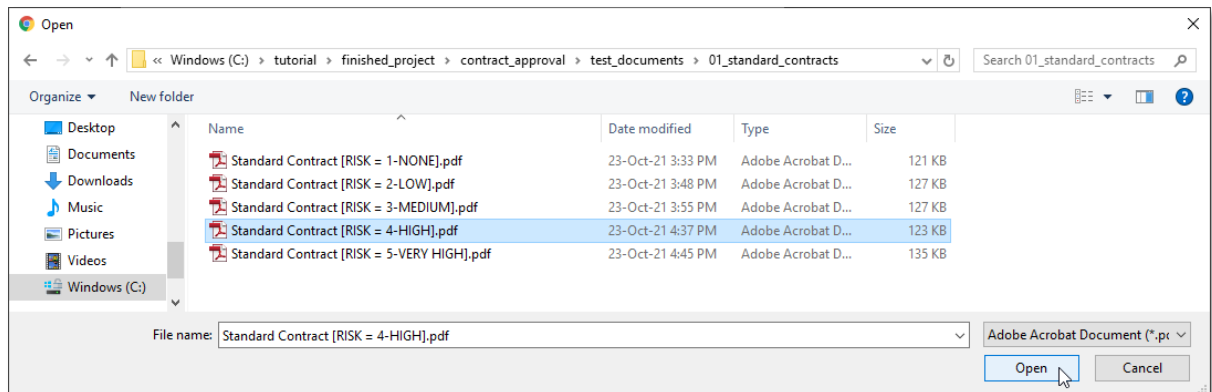
- We have now successfully approved two contracts with two completely different approval flows. Let's continue with the scenario where an approver does not approve (i.e.: rejects) the contract. We will create a contract with the following characteristics:

- **Type: standard contract**
 - **Value: above 1000** (requires Line Manager approval)
 - **Risk classification: above 3, i.e.: 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.

From the **test_documents** folder, open the **01_standard_contracts** subfolder and select the **Standard Contract [RISK = 4-HIGH].pdf** file.

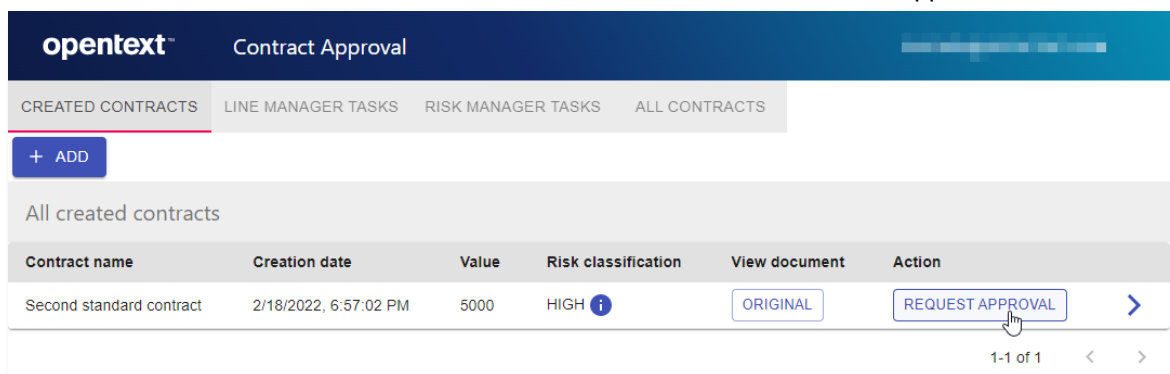


Make sure the **Standard Contract** option is selected and fill the contract properties as follows:

- **Document name:** Second standard contract
- **Contract value:** 5000
- **Contract requester email:** <your email>

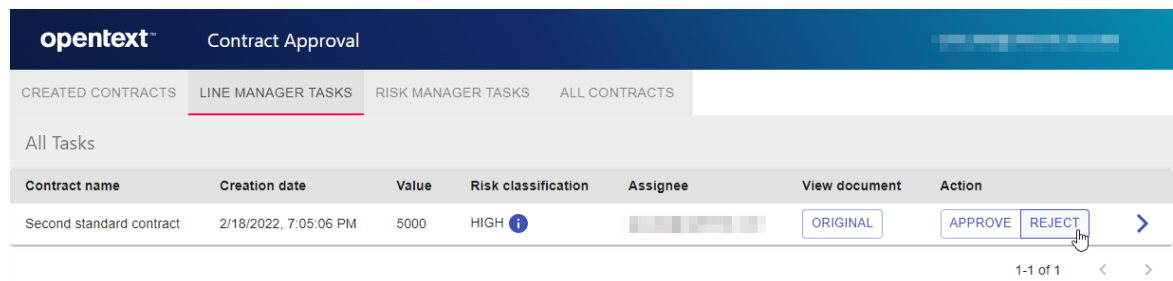
Click **Add** to create the contract.

Click the **REQUEST APPROVAL** button in the **Action** column to launch the approval workflow.



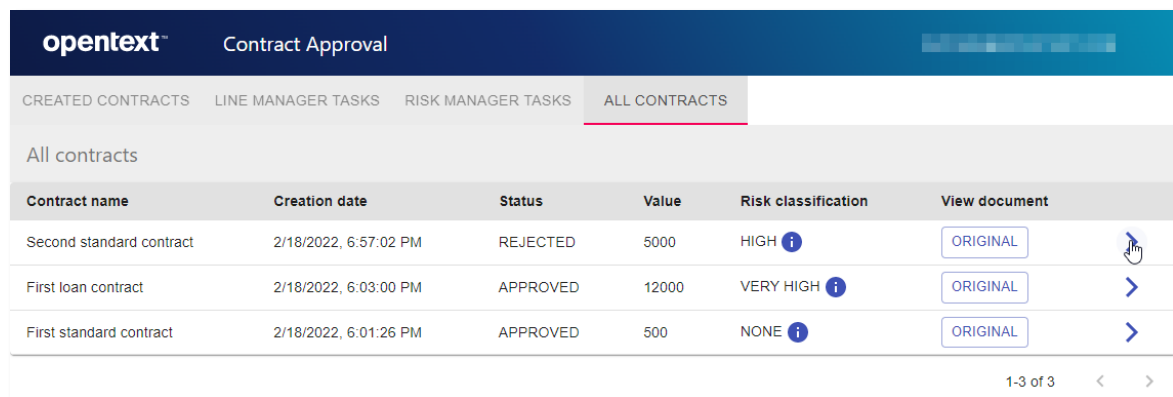
The new contract has now disappeared from the **CREATED CONTRACTS** view, as it is no longer in 'CREATED' status. Go to the **LINE MANAGER TASKS** tab. You should see the new Line Manager approval task.

Click **REJECT** to reject the contract approval.

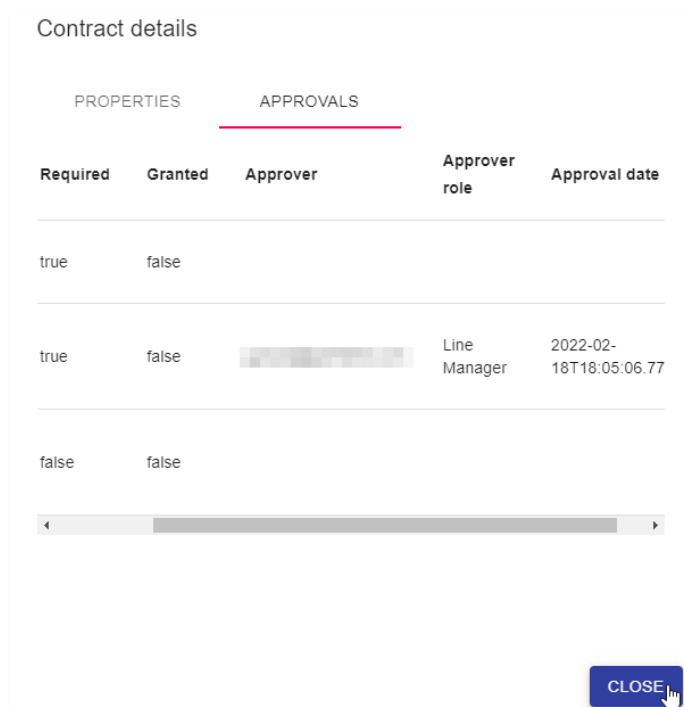


The contract has now disappeared from the **LINE MANAGER TASKS** view, as it is no longer in 'LINE MANAGER APPROVAL' status. When you open the **ALL CONTRACTS** view, you will see that the contract has indeed been rejected (**Status** column shows **REJECTED** status).

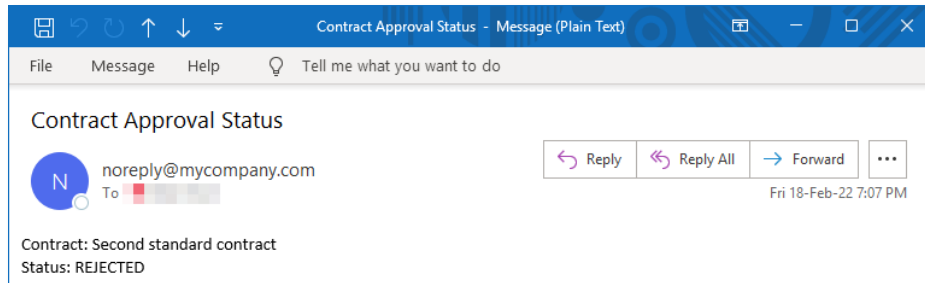
Click on > to view the contract details to check the different approvals (traits).



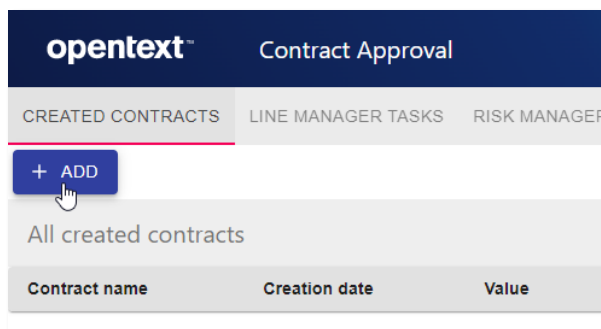
You can see the contract is rejected, since the **Line Manager Approval** has NOT been granted while it has a date at which the approval task was completed.



Your second standard contract has NOT been approved (i.e.: it has been rejected), and you should have received the corresponding **Contract Approval Status** email from **noreply@mycompany.com**.

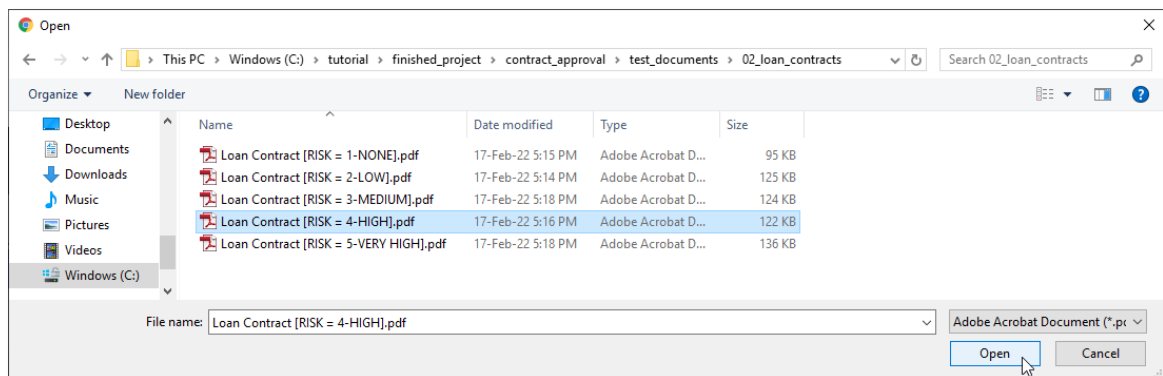


- The last scenario we want to run through is to let an approval task expire (happens after 5 minutes).
We will create a contract with the following characteristics:
 - **Type: loan contract**
 - **Value: below 1000** (doesn't require Line Manager approval)
 - **Risk classification: above 3, i.e.: HIGH 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.

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

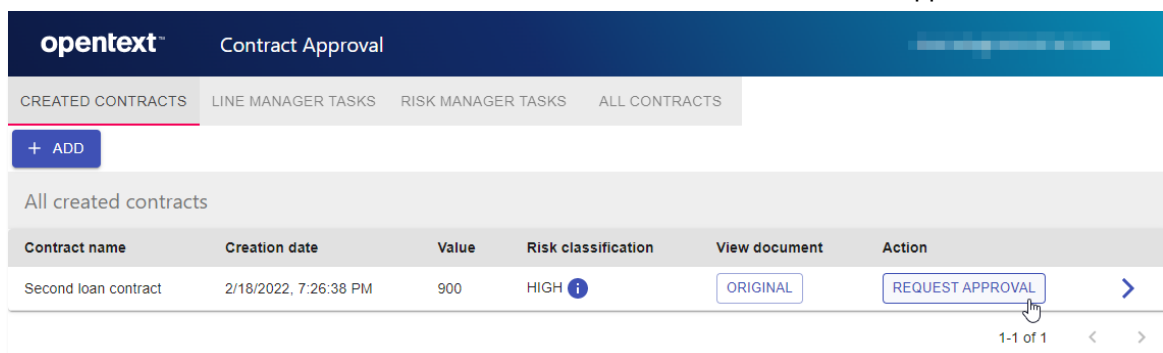


Select the **Loan Contract** option and fill the contract properties as follows:

- **Document name:** Second loan contract
- **Contract value:** 900
- **Monthly installments:** 12
- **Yearly income:** 40000
- **Contract requester email:** <your email>

Click **Add** to create the contract.

Click the **REQUEST APPROVAL** button in the **Action** column to launch the approval workflow.



The new contract has now disappeared from the **CREATED CONTRACTS** view, as it is no longer in 'CREATED' status. Go to the **RISK MANAGER TASKS** tab. You should see the new Risk Manager approval task.

opentext Contract Approval						
CREATED CONTRACTS LINE MANAGER TASKS RISK MANAGER TASKS ALL CONTRACTS						
All Tasks						
Contract name	Creation date	Value	Risk classification	Assignee	View document	Action
Second loan contract	2/18/2022, 7:27:44 PM	900	HIGH <i>i</i>		ORIGINAL	APPROVE REJECT
1-1 of 1 < >						

Since we want to test whether or not the approval task will expire, you need to wait. You have configured the timeout wait time to be 5 minutes, so wait a little longer than that, let's say 10 minutes, and refresh your browser (you might have to log in again as well).

After refreshing your application screen, go back to the **RISK MANAGER TASKS** view.

The contract has now disappeared from the **RISK MANAGER TASKS** view, as it is no longer in 'RISK MANAGER APPROVAL' status.

opentext Contract Approval						
CREATED CONTRACTS LINE MANAGER TASKS RISK MANAGER TASKS ALL CONTRACTS						
All Tasks						
Contract name	Creation date	Value	Risk classification	Assignee	View document	Action
0-0 of 0 < >						


When you open the **ALL CONTRACTS** view, you will see that the contract approval has indeed expired (**Status** column shows **EXPIRED** status).

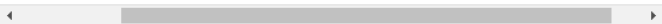
Click on [>](#) to view the contract details to check the different approvals (traits).

opentext Contract Approval						
CREATED CONTRACTS LINE MANAGER TASKS RISK MANAGER TASKS ALL CONTRACTS						
All contracts						
Contract name	Creation date	Status	Value	Risk classification	View document	
Second loan contract	2/18/2022, 8:34:17 PM	EXPIRED	900	HIGH <i>i</i>	ORIGINAL	>
Second standard contract	2/18/2022, 6:57:02 PM	REJECTED	5000	HIGH <i>i</i>	ORIGINAL	>
First loan contract	2/18/2022, 6:03:00 PM	APPROVED	12000	VERY HIGH <i>i</i>	ORIGINAL	>
First standard contract	2/18/2022, 6:01:26 PM	APPROVED	500	NONE <i>i</i>	ORIGINAL	>
1-4 of 4 < >						

You can see 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.

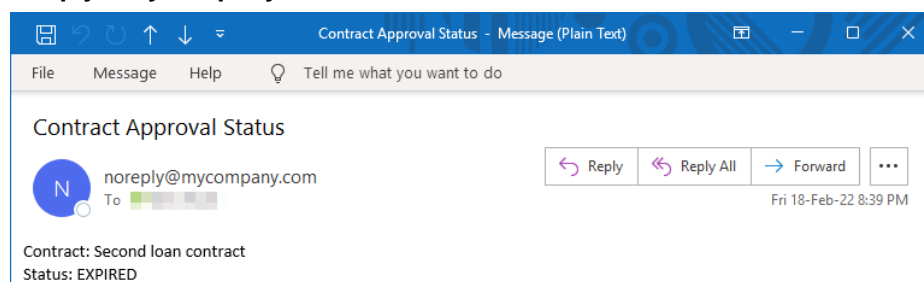
Contract details

PROPERTIES		APPROVALS		
Required	Granted	Approver	Approver role	Approval date
true	false			
false	false			
true	false		Risk Manager	
true	true	SYSTEM	Solvency Check	2022-02-18T19:34:20.24



[CLOSE](#)

Your second loan contract has not been approved since the Risk Manager Approval step expired, and you should have received the corresponding **Contract Approval Status** email from **noreply@mycompany.com**.



This calls for a second **CONGRATULATIONS!**

You have now completely finished building and testing your 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 OT2 CLI. If you are interested in build automation and CI/CD for your IMaaS applications, we recommend you certainly do that exercise as well.

3.15[00'] Bonus exercise: Using the OT2 Command Line Interface

COMING SOON

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](#)