

# Hyland RPA

Version: 23.1

Written by: Documentation Team, R&D  
Date: Wednesday, February 4, 2026

# Documentation Notice

Information in this document is subject to change without notice. The software described in this document is furnished only under a separate license agreement and may only be used or copied according to the terms of such agreement. It is against the law to copy the software except as specifically allowed in the license agreement. This document or accompanying materials may contain certain information which is confidential information of Hyland Software, Inc. and its affiliates, and which may be subject to the confidentiality provisions agreed to by you.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright law, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Hyland Software, Inc. or one of its affiliates.

Hyland, HXP, OnBase, Alfresco, Nuxeo, and product names are registered and/or unregistered trademarks of Hyland Software, Inc. and its affiliates in the United States and other countries. All other trademarks, service marks, trade names and products of other companies are the property of their respective owners.

© 2026 Hyland Software, Inc. and its affiliates.

The information in this document may contain technology as defined by the Export Administration Regulations (EAR) and could be subject to the Export Control Laws of the U.S. Government including for the EAR and trade and economic sanctions maintained by the Office of Foreign Assets Control as well as the export controls laws of your entity's local jurisdiction. Transfer of such technology by any means to a foreign person, whether in the United States or abroad, could require export licensing or other approval from the U.S. Government and the export authority of your entity's jurisdiction. You are responsible for ensuring that you have any required approvals prior to export.

## Table of Contents

<b>Documentation Notice</b> .....	<b>2</b>
<i>About Hyland RPA</i> .....	5
<i>About Process Identification</i> .....	5
<i>About Evaluation</i> .....	5
<i>About Analysis</i> .....	6
<i>About Implementation</i> .....	6
<i>About Testing and Rollout</i> .....	6
<i>About Getting Well</i> .....	7
<i>About Going Live</i> .....	7
<i>About Maintenance</i> .....	7
Architectural Overview .....	8
<i>Accessing the Account Details</i> .....	8
<i>Resetting a Password</i> .....	8
<i>Changing a Password</i> .....	9
<i>About Hyland RPA Launcher</i> .....	9
Hyland RPA Launcher FAQ .....	9
<i>Installing an Application</i> .....	21
<i>Activating a License Online</i> .....	24
<i>Activating a License Offline</i> .....	27
<i>Deactivating a License</i> .....	31
<i>About Automation</i> .....	33
<i>Fundamentals</i> .....	34
MainScriptRunner .....	34
Golden Rule .....	34
Decision or If Activity .....	35
Flowchart or Sequence .....	35
Logging .....	35
<i>About Microbots</i> .....	36
Process-Specific .....	36
Generic .....	36
Naming Convention .....	36

Arguments in Microbots .....	36
Repository .....	37
<i>About Activities</i> .....	37
<i>About Variables</i> .....	37
<i>About InfoValues</i> .....	38
<i>About Custom Activities</i> .....	38
<i>Fundamentals</i> .....	38
Activity (ViewModel) .....	38
Design (View) .....	38
Test .....	39
Application (Model) .....	39
<i>Requirements for the Custom Activity</i> .....	39
Awesome Calculator - Example .....	39
<i>About Setting up the Solution</i> .....	39
Get the Skeletons .....	39
Configuration and Renaming .....	39
Activity preparation .....	40
<i>Unit Test</i> .....	40
<i>About the Application Project</i> .....	41
<i>About the Activity</i> .....	42
Properties .....	42
Enums .....	43
Attributes .....	43
Metadata .....	43
Synchronous Activities .....	44
Async Activites .....	44
<i>About Error Handling and Logging</i> .....	45
Error Handling .....	45
Write Log .....	45
<i>Design, Deployment and Use of Designer</i> .....	45
Design .....	45
Deployment .....	46
Use of Designer .....	46

## About Hyland RPA

Hyland RPA (Robotic Process Automation) is a software suite specifically designed to support every automation phase from process documentation to maintenance. Hyland RPA creates synergies across people and departments.

Automation only comes to life when all its actors are in perfect sync - RPA Analyst who hands over to RPA Designer, RPA Conductor who sets the timing.

Normally, Robotic Process Automation and its software products are limited to a small part of the automation process - the development and operational phases. Hyland RPA supports all phases of the automation process.

Hyland RPA is the result of years of automation experience and best practice. For each automation phase, we have developed target-group-specific software that fosters interaction and communication. Hyland RPA makes Robotic Process Automation understandable and feasible for everyone.

## About Process Identification

During the process identification phase, there are several methods to achieve the target of process identification. Hyland provides different support options, and you can review our RPA Product Catalog for more details on these various support options.

The main options to assess your processes are:

1. **Process Mining:** We will analyze and evaluate your internal processes
2. **Company KPIs:** Through customer collaboration, we use your KPIs to identify processes that can contribute significant benefits to your business, specifically in the areas of time utilization, customer satisfaction, and employee satisfaction.
3. **Process Discovery:** Again, by working closely with you we will evaluate your business structure to find suitable processes that can be automated.
4. **Workshops:** We conduct process workshops with your teams to discover suitable automation candidates.

## About Evaluation

During the evaluation phase, we determine which type of RPA automation to implement: RDA, RPA, or a hybrid solution of both. We then construct a business case that includes the predicted economic benefit for your organization.

After determining the value and fit of a process, we move on to the next evaluation step: identifying any prerequisites and requirements needed for successful automation. Examples may include software systems with which we will need to interface, compliance with security protocols, or credentials needed to access the company's systems.

## About Analysis

During the analysis phase we create detailed **Click Diagrams** of the automation task.

Our Business Analysts use two of our tools to produce this documentation:

- Hyland RPA Analyst
- Hyland RPA Designer

Working as efficiently as possible, we use Hyland RPA Analyst to create a detailed click manual and process draft as input for Hyland RPA Designer.

This working draft can be revised at any point to create a functioning and logical click workflow. As part of the design process, we split the tasks to be performed by two additional components of our automation system:

- The DataCollector (DCO)
- The ProcessingRobot (PRO)

We use Hyland RPA Designer to build the automation and all key variants within.

Additionally, we build in all exception handling within this phase of the script's drafting.

Once the final edits are completed, the process flowchart is extracted from RPA Designer as an XML file.

## About Implementation

During the implementation phase, the output of Hyland RPA Analyst is finalized, and any additional programming steps are completed by a developer or analyst.

RPA Designer gives experienced analysts or developers the opportunity to access additional detail and control over programming the script, including debugging as needed, to ensure the script executes properly. Using this environment, we simulate as closely as possible, the actual production environment in which the script will be running.

Again, using RPA Designer, we adjust all selector values and fine-tune the interactions with the automated software.

Finally, we add any necessary credentials and artefact handling. Both are used within RPA Designer and managed by RPA Web Manager, which is used to monitor and maintain your bot farm.

## About Testing and Rollout

Once the process script is completed, it must be tested and implemented.

During this step, we will install all components of Hyland RPA's production environment. This creates a live instance of our deployment.

Once this is complete, we deploy the production version of the bot and, using Hyland RPA Web Manager, we begin to monitor the bot's performance. Should certain artefacts need to be modified or selector values need to be adjusted, we will make any necessary adjustments and perform fine-tuning during this phase to ensure success.

## About Getting Well

During the **Get Well** phase, we increase the ticket volume to validate that the system is running as desired and verify we can reach the desired success rate.

Logs, screenshots, and reports will also be created for review purposes. This is achieved by reworking the process script.

Protocol activities are included in the process script so that you can define whether an activity should be recorded. To generate these acceptance protocols, we edit the production in RPA Web Manager so that the protocol is saved automatically in the desired format.

If the process is running, we want to make sure you have a good remark analysis. For this, we use the reporting system in our RPA Web Manager.

## About Going Live

All testing is now completed and the bot has successfully performed all tests.

It is now ready to go live and is officially handed over to the process owner.

During this phase of the deployment, the process owner will receive access to all process documentation and information, including access to Hyland RPA Web Manager and Watchdog, the production monitoring system.

From here, the process owner can validate the bot and take any desired action should the bot not perform as expected.

## About Maintenance

Should changes occur in any underlying software agents, such as a change in a UI, the entire bot does not need to be rewritten.

This is a unique feature of our architecture and solutions. Instead, we handle these changes by adjusting **Artefacts**. Through **Artefacts**, global changes such as selector values can be adjusted in a single location. This allows users to make simple changes that require little time and effort.

Similarly, if something is changed in the process logic, the entire bot does not need to be rewritten. Instead, the changes are implemented by changing Microbots in RPA Designer. Each process in which this change is required is then automatically updated.

Finally, in the situation where a new screen pop-up is encountered that could interfere with the bot's functionality, we use the **Pop-Up Watcher**. The **Pop-Up Watcher** helps us to build logic around additional, previously unencountered, pop-up events.

## Architectural Overview

Hyland RPA consists of several components used to kickstart your enterprise automation journey. Integral components are:

- Hyland RPA Analyst
- Hyland RPA Designer
- Hyland RPA Conductor Platform
- Hyland RPA Platform

Our architecture is based on an enterprise approach, designed to quickly implement scalable RPA ecosystems. We ensure this through an "easy to maintain" approach that provides reusable objects such as artefacts and microbots to deliver the full advantages of robotic process automation. Through our unique task processing, we are able to stabilize large numbers of software robots and ensure the highest level of bot utilization within the entire RPA Market. This is complemented by our multiskilled software robots that easily and successively perform several processes.

### Advantages of Hyland RPA's architecture

- Scalability
- Stability/Robustness
- Reliability
- Multiskilled BOTs
- Transparency
- Easy maintenance

Hyland operates one of the largest bot farms in the entire world, performing over 15 million minutes of transactions each month. This is equivalent to the work of 1.500 FTEs. Our solutions come with very low required maintenance efforts, less than 1% - 4 FTE run a 1.500 bot farm running over 300 different processes.

**This video is based on the former AM products before the switch and rebranding to Hyland RPA**

<https://youtu.be/p4bAolWBPeo>

## Accessing the Account Details

Sign in to our [Customer Portal](#) with your email address and password and choose the settings menu to access your account details.

## Resetting a Password

To reset a forgotten password or change your existing password:

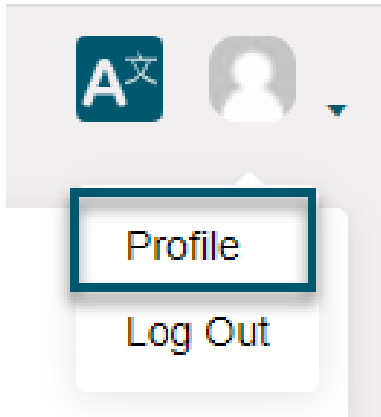
1. Navigate to the [Customer Portal](#).
2. Click **Forgot Password**.
3. In the **Forgot Password** dialog box, enter the email address connected with your customer account, and then click **Reset Password**.
4. Hyland sends an email with a link to reset your password.

5. Click the link provided in the email from Hyland.
6. Enter your new password and then click **Reset Password**.

## Changing a Password

Learn how you can change/update your account password.

1. Navigate to the [Customer Portal](#).
2. Sign in to the portal with your credentials.
3. In upper right corner, click on your profile icon and then select **Profile**.



4. Enter your current and new password and then click **Update your password**.  
**Why do I see a message that I haven't changed my password yet?**

For security reasons we strongly recommend changing your initial account password, therefore we display a message card, to guide you to the change password process.

## About Hyland RPA Launcher

Hyland RPA Launcher helps you to easily install and update our applications and to create Rollout-Kits.

To download **Hyland RPA Launcher**, click [here](#).

**Note:** If you have a proxy server in place, Hyland RPA Launcher will not be able to access your license server and therefore cannot be executed. In this case, navigate to the [Customer Portal](#) and download the applications separately, as described in [Installing an Application](#).

### See also

[Hyland RPA Launcher FAQ](#)

## Hyland RPA Launcher FAQ

**What is the purpose of Hyland RPA Launcher?**

Hyland RPA Launcher is an easy to use desktop application that manages the application installation & lifecycle management for Hyland RPA. With the Hyland RPA Launcher you can:

- Install and update your licensed applications.
- Create Rollout-Kits for unattended robot clients.

**Which packages contain which products?**

- **Designer:** Designer and Activities
- **RPA Conductor:** RPA Conductor and Activities
- **Analyst:** Analyst
- **RPA Platform:** Web Manager

**Note:** The database installer is not yet included in the Installation/Update by Hyland RPA Launcher. If a database update is required, please visit our Customer Portal to download the respective database installer version. The installer can be found in the product section Hyland RPA Manager.

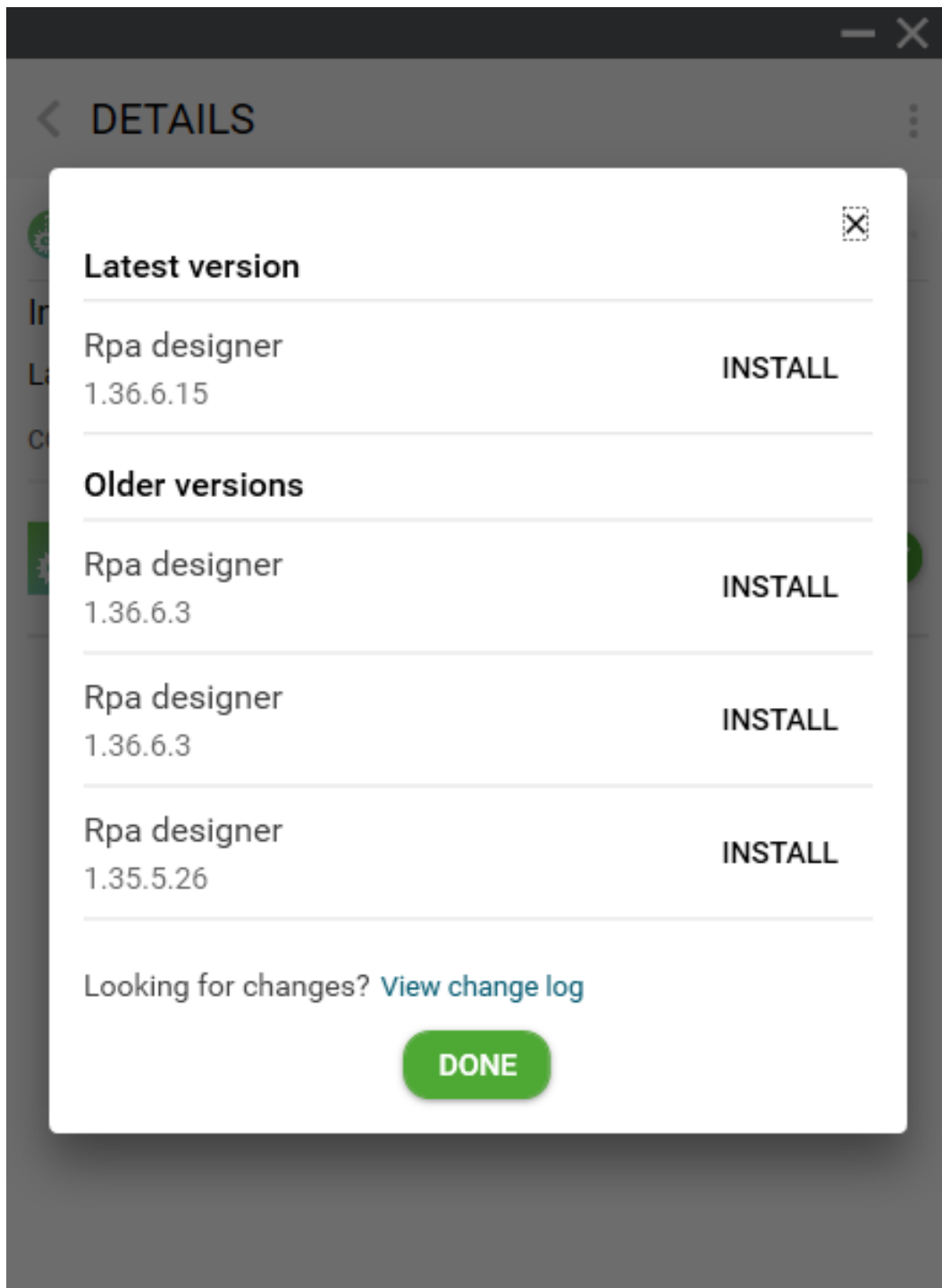
For information on how to update your database, see the **Hyland RPA Installation & Update Help**.

**Can I access an older product version with the Hyland RPA Launcher?**

Yes, click on the product (for example, Designer) select a different product version and click install. Hyland RPA Launcher will then install the selected product version.

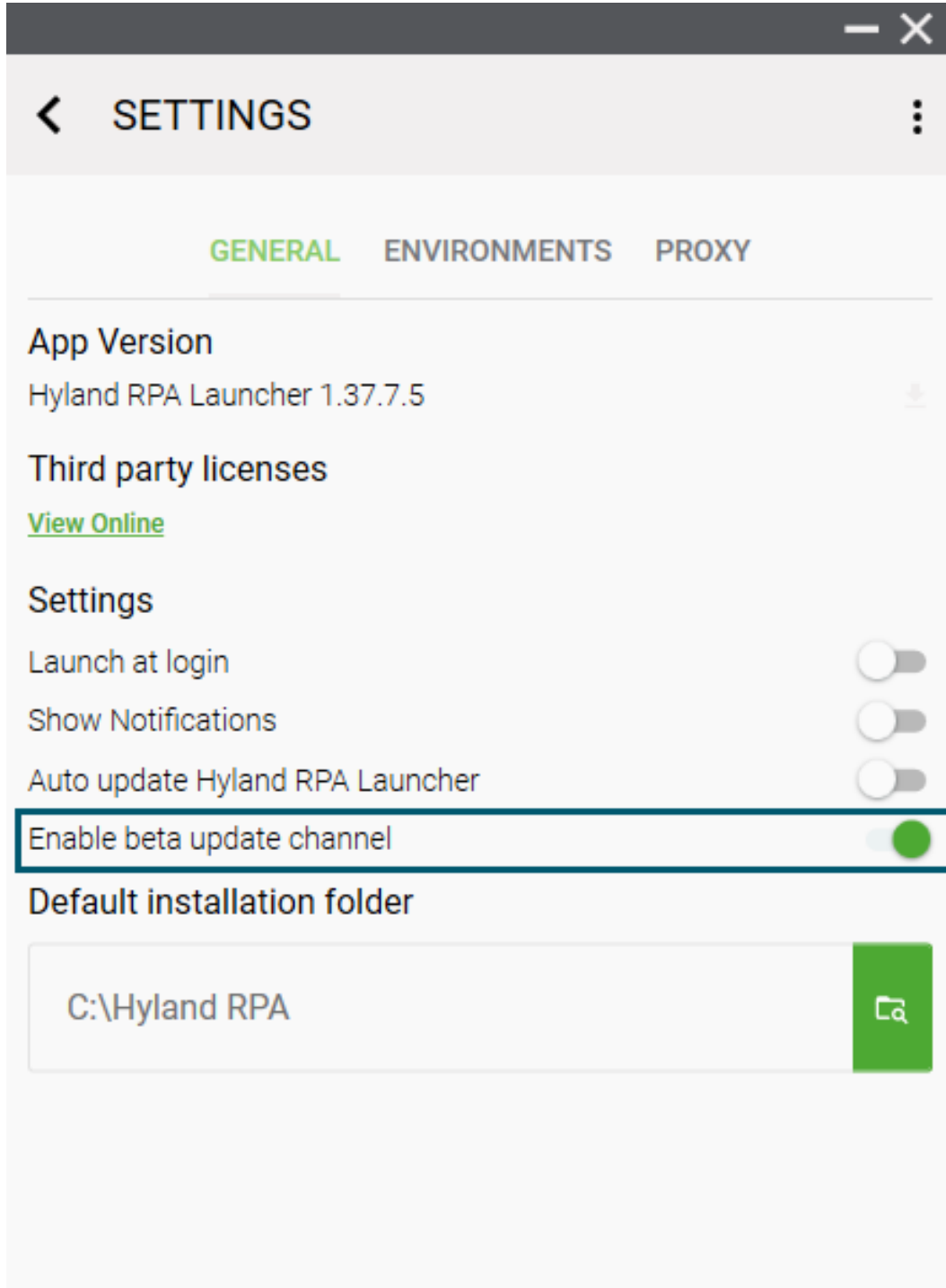
**How can I switch to an older product version?**

To switch to an older product version, simply uninstall the latest version and select the version of your choice within the version-dropdown.



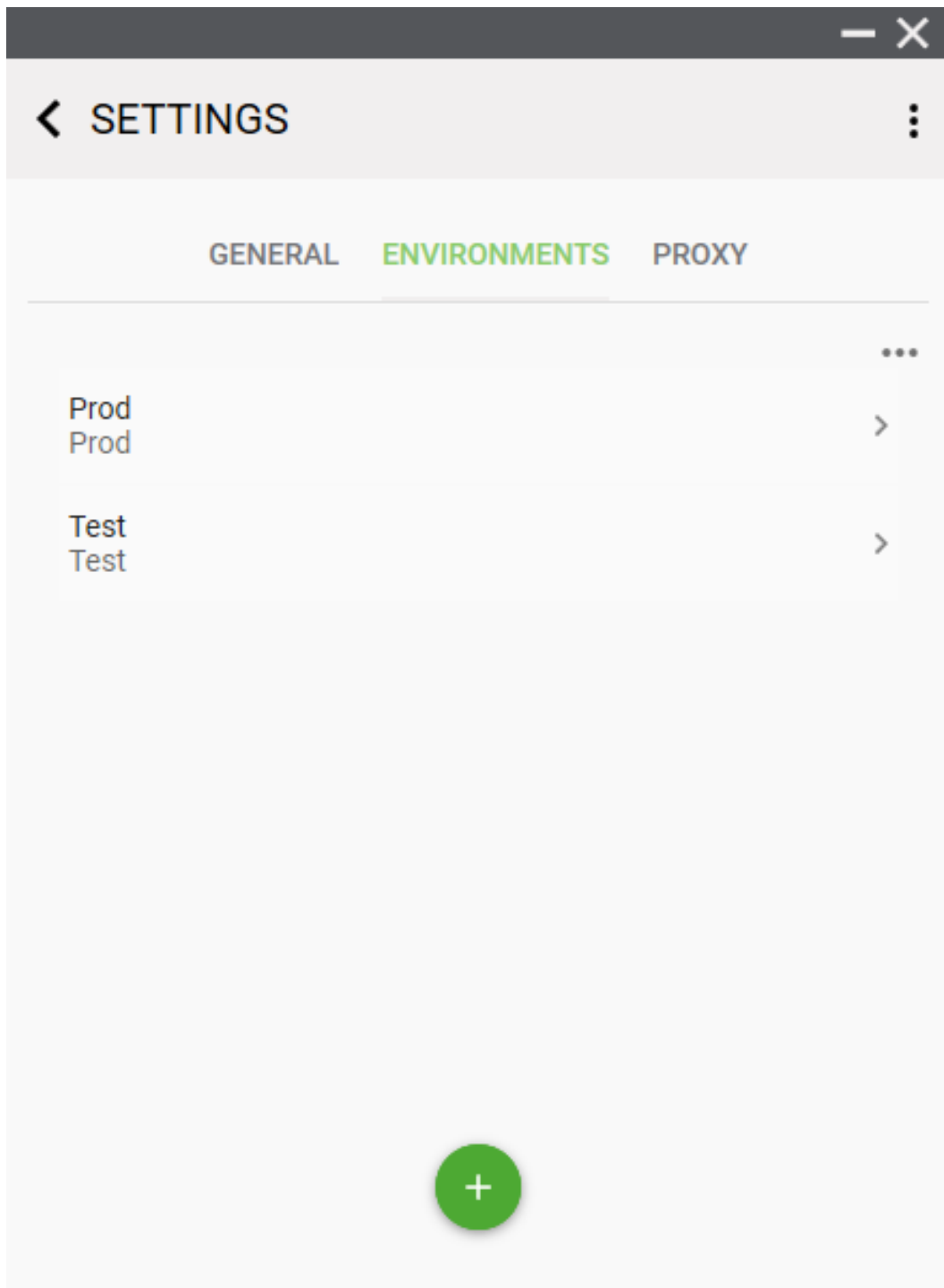
Can I install Beta-Releases with Hyland RPA Launcher?

Yes, you can. Simply go to the Settings and enable the beta update channel. Now, if there is a new beta of a product available, you will get an update notification. Please be aware of possible problems using a beta version. To apply this change, you need to exit the Launcher in the tray on the bottom right and then restart it.



**Why can I maintain environments in Hyland RPA Launcher settings?**

The environment inside the Hyland RPA Launcher settings is used to configure certain products for your specific environment. Per default, we will use your default environment. As of today, we only configure the RPA Conductor during the installation process. Additional products will follow with future updates.

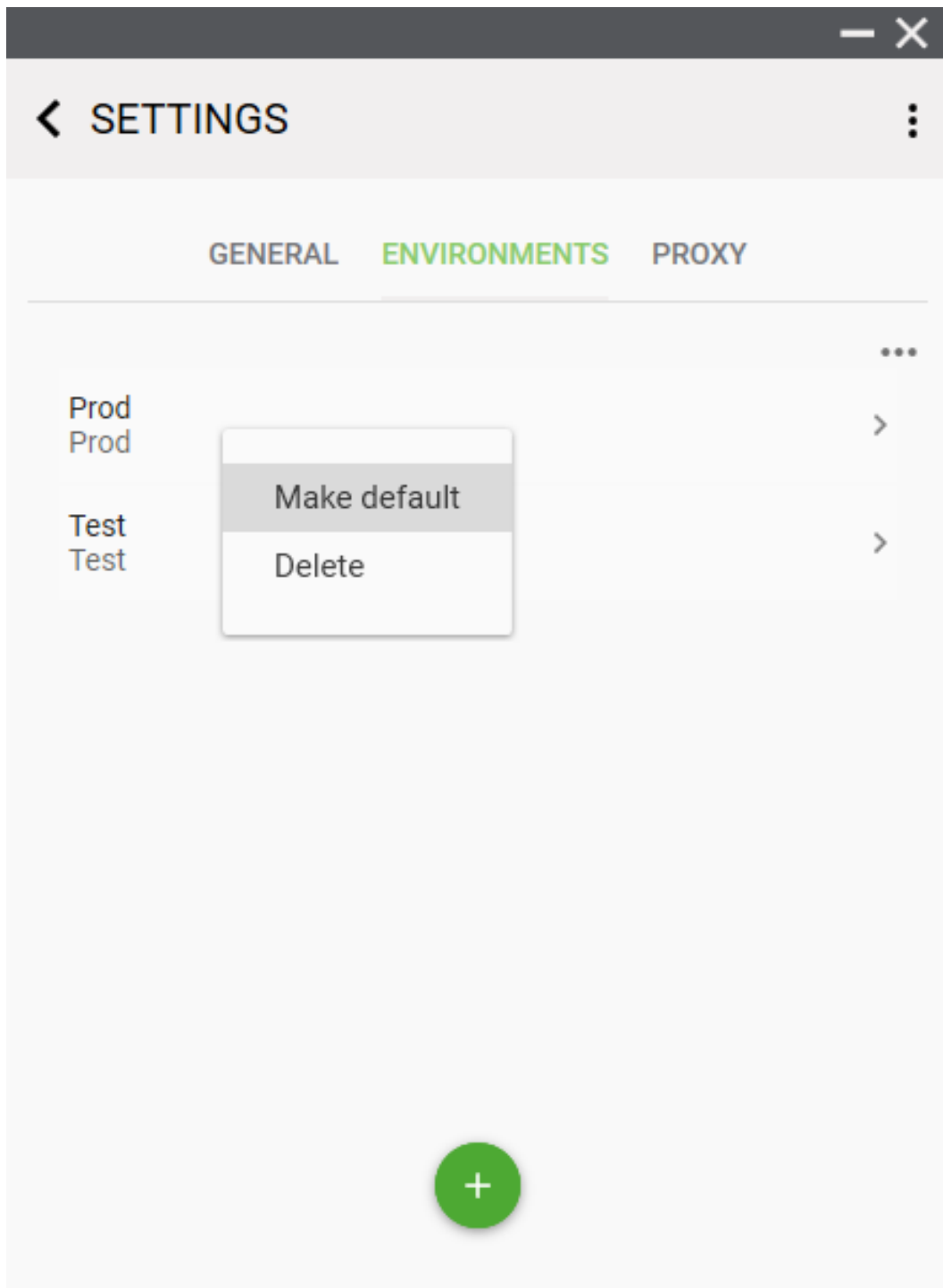


**Which environment is used when I install the RPA Conductor?**

When you install the RPA Conductor, we will use your default environment to configure the Hyland RPA Conductor for you. So, with one click on Install the RPA Conductor & the latest Activities are installed on your system, under the same folder path logic. Your RPA Conductor is ready to go after the installation is completed incl. the encrypted connection string. The Workflowrobot is also configured with the encrypted connection and the activity folder path, based on your default installation path.

**How can I set a default environment?**

Navigate to Settings - Environments and right-click on the environment. Choose to Make default in the context menu. You can also edit the environment and change the default environment there.

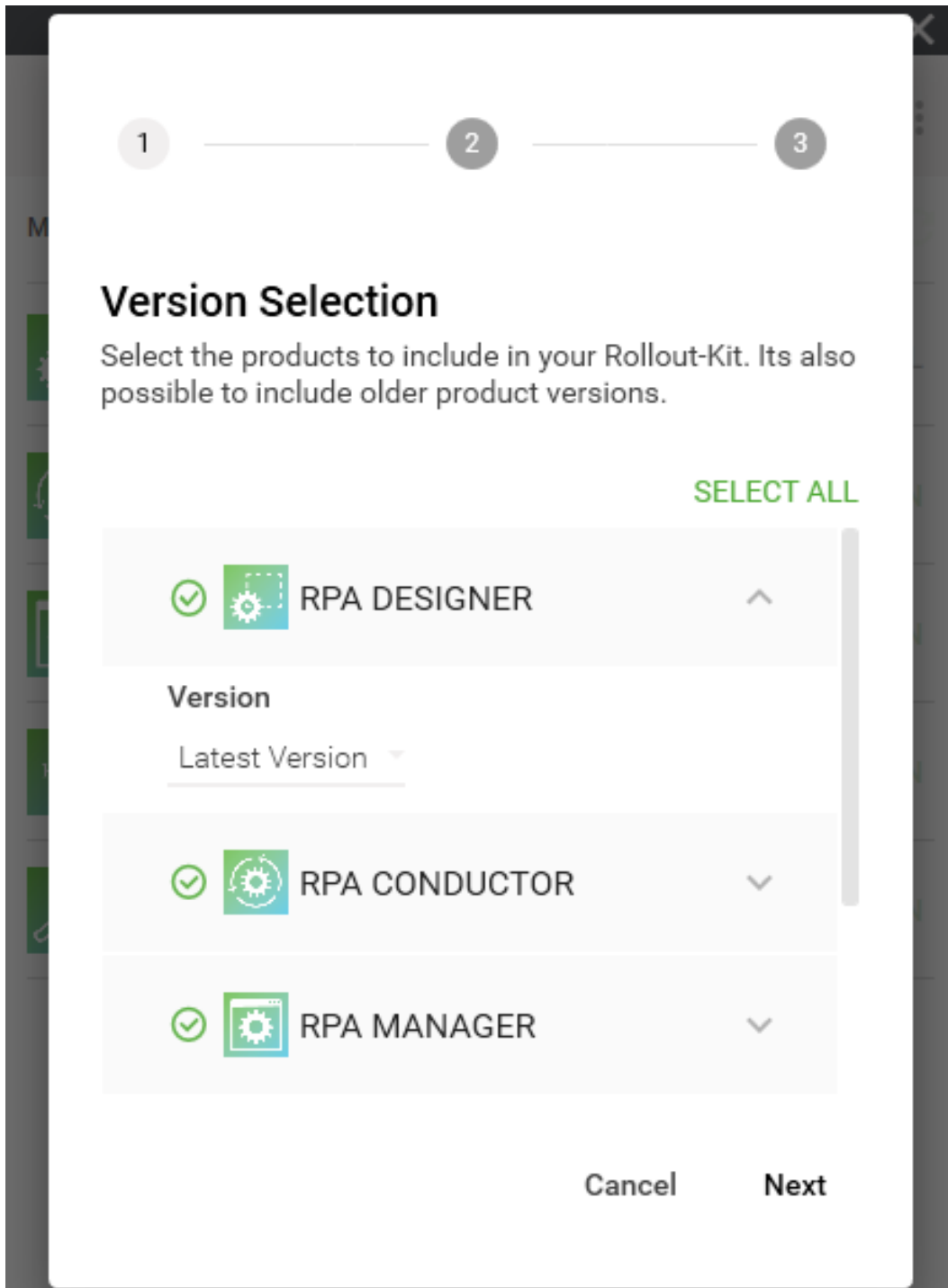


**Does the Hyland RPA Launcher activate my products?**

Yes, with an active internet connection we will try to activate your product against our license services. If the activation failed, you need to activate the application during the first launch of the app.

**When can I use a Rollout-Kit?**

You can use a Rollout-Kit whenever you want to install products on clients without access to the internet or without access to the Hyland RPA Launcher. General use-cases for Rollout-Kits are the distribution of Hyland RPA updates. Rollout-Kits configure our products up to a certain point so that they're ready to work after distribution.

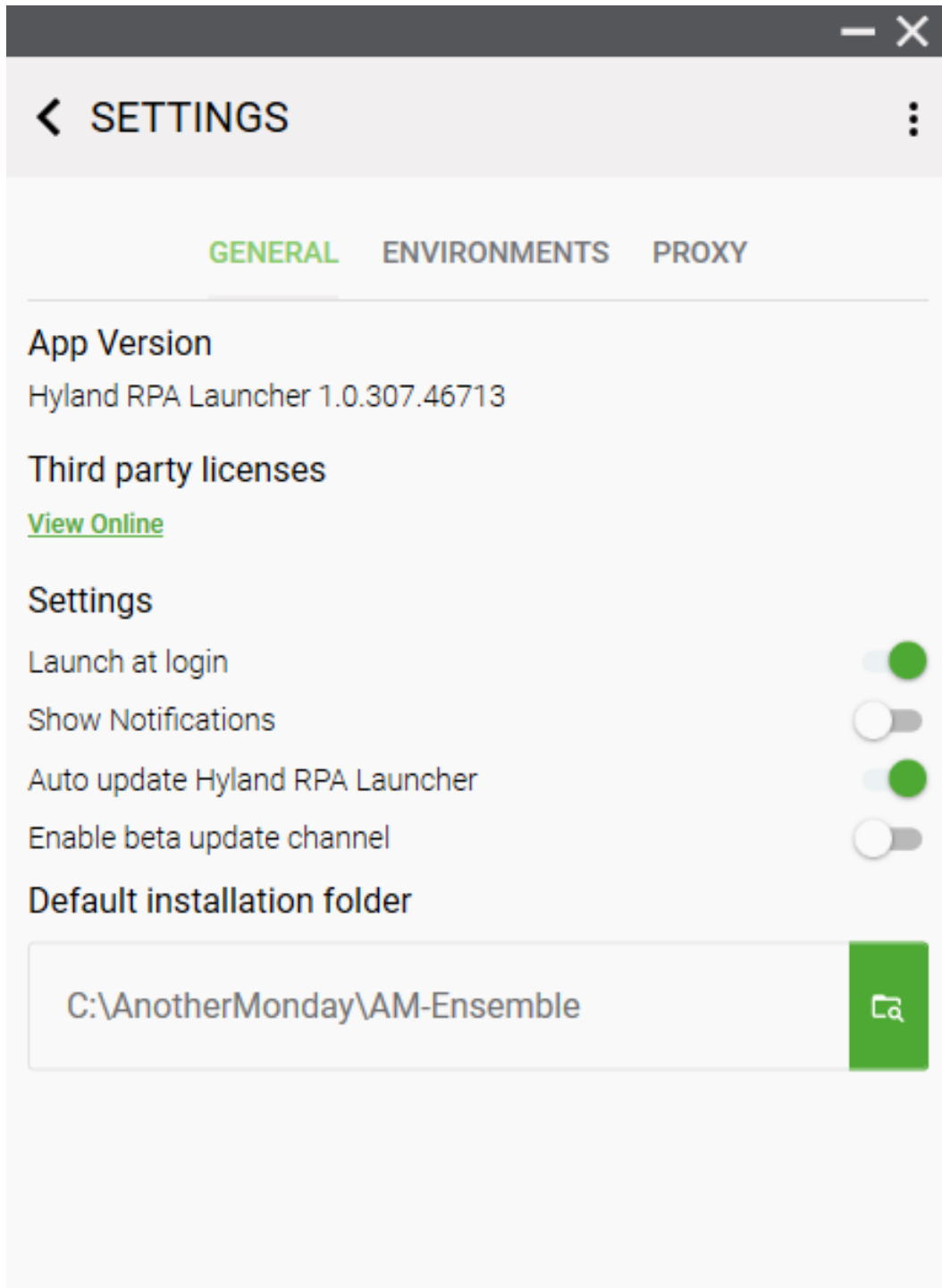


**Can I enable Auto-Updates for my products?**

Auto-Update for your products is not yet possible but might be added going forward.

**Can I enable Auto-Updates for the Hyland RPA Launcher?**

Yes, you can, just set the checkbox Enable Auto-Update Hyland RPA Launcher. The Hyland RPA Launcher will then keep itself up to date during startup.



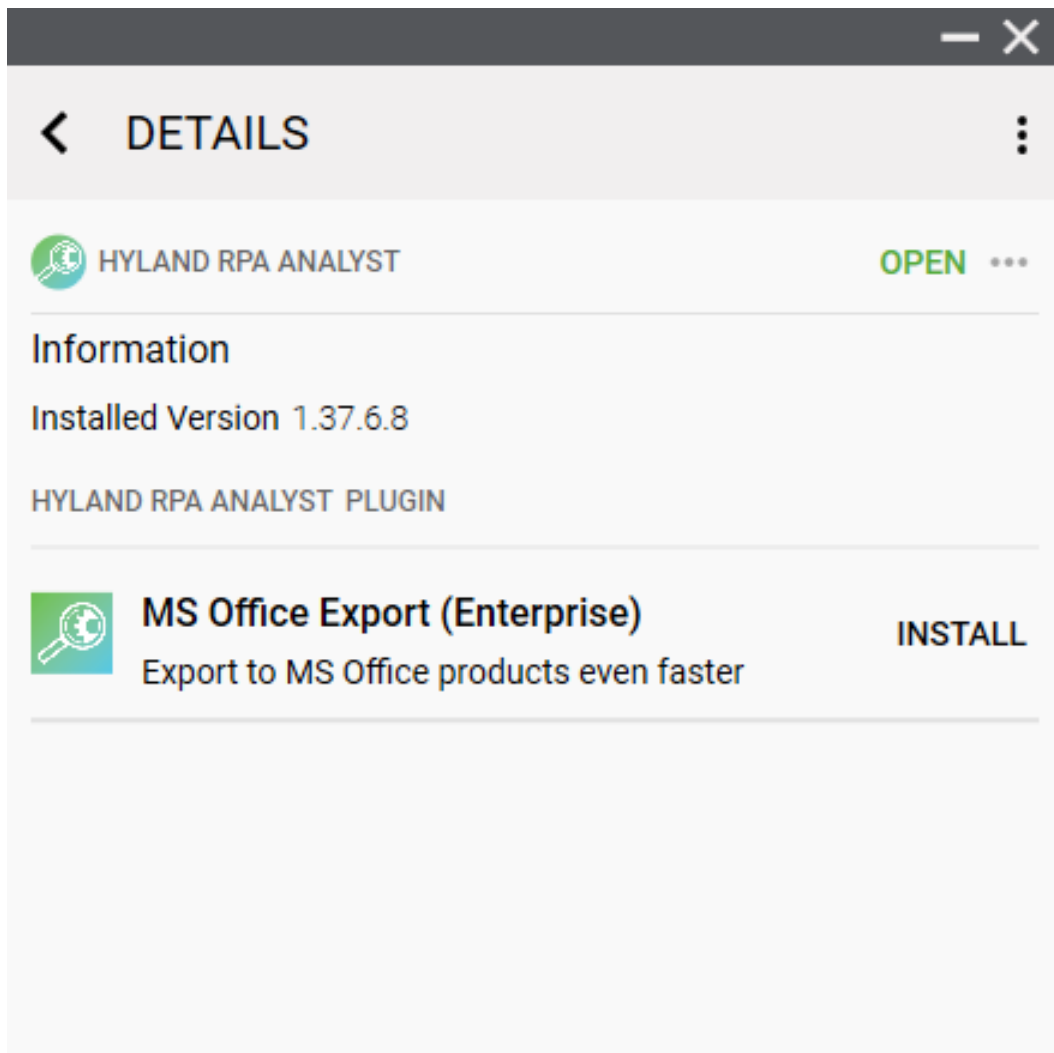
How can I set the installation path for my products?

To change the installation path of your products, go to **Settings** and change the **default installation folder**. Your next product installation will be installed in the selected directory. Please note that we do not move your existing installations.

### How can I manage Enterprise Plug-ins using the Hyland RPA Launcher?

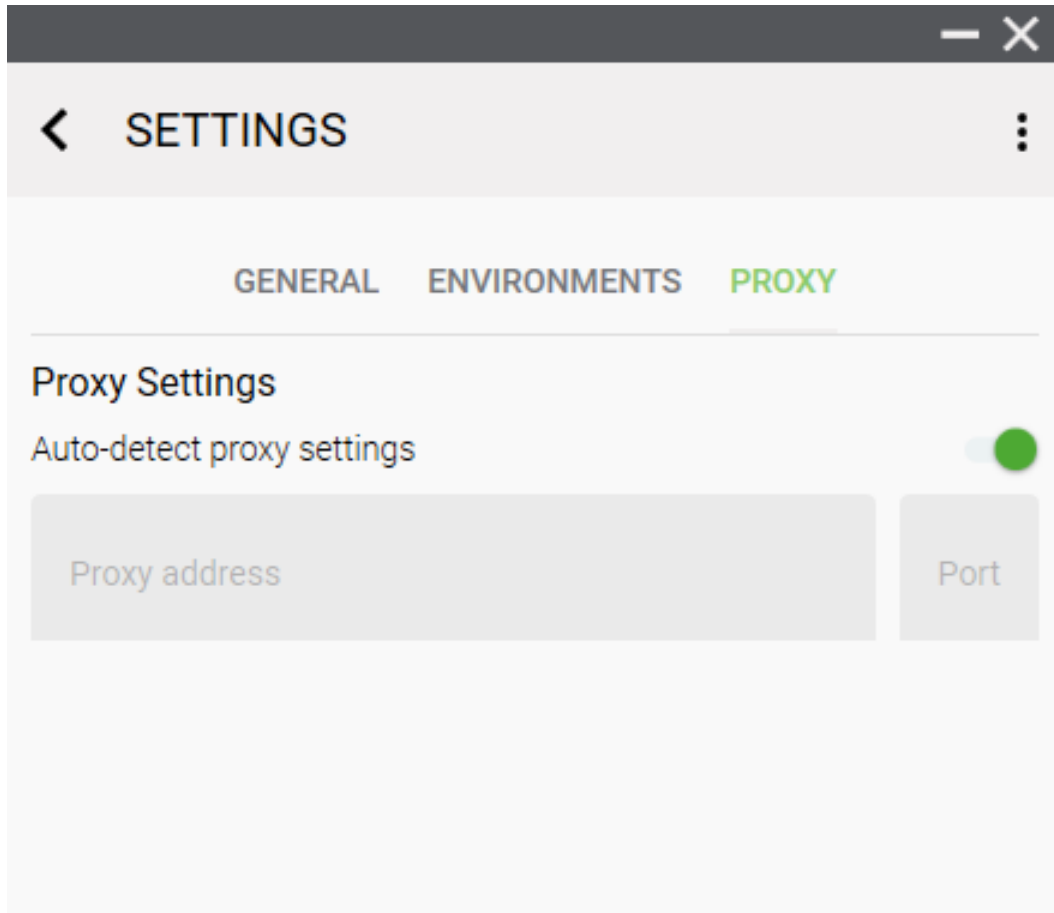
Enterprise plug-ins can easily be handled on the product page of the Hyland RPA Launcher. If you have purchased a license for an enterprise plug-in, it will be shown on the detail page. Here you can, install and activate your plugin as well as disable and enable it for the product.

**Note:** To make the changes applicable in the product, the product is not allowed to be running.



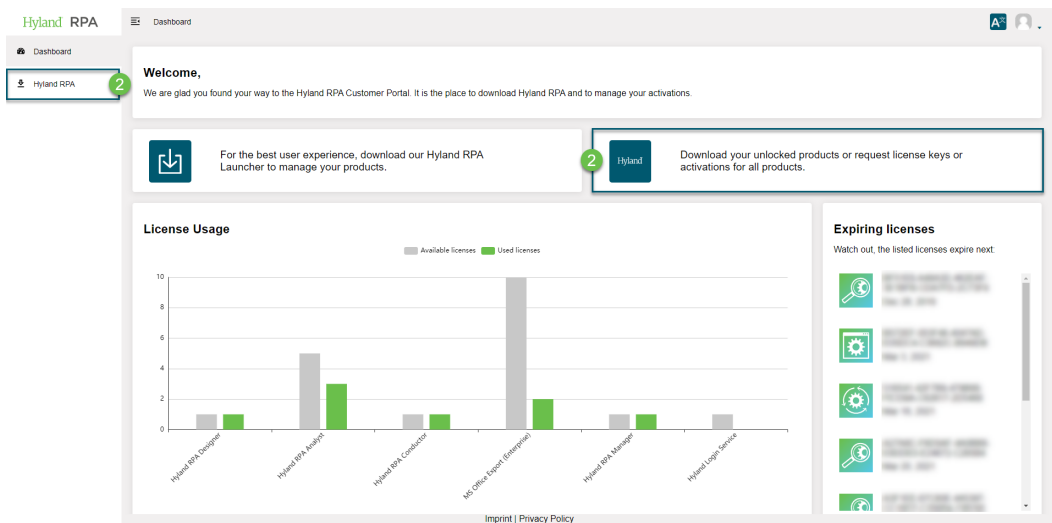
**Is there a way to use the Hyland RPA Launcher when my company is using a proxy server?**

Yes, it is. Go to settings and choose the tab proxy. Here, you have the possibility to either automatically detect the proxy-settings (default) or manually set them. In any case, afterward you will be asked for further authentication.



## Installing an Application

1. [Sign in](#) to your account with your email address and password.
2. Go to the Product overview by either clicking on Hyland RPA in the Menu or the link on the dashboard.



3. Choose the product you want to install and click on Download to download the latest release. (If you need further releases, go to the Product page by clicking Details and choose the desired release under Product Downloads)

## Hyland RPA

Download your unlocked products or request license keys or activations for all

**RPA Analyst**  
Quickly and intuitively analyze processes

**RPA Designer**  
Leverage low-code, drag-and-drop tool

Download >> Details

Download 3 >> Details

4. Download the products you want to install either as .msi or .zip files.

Latest Release
✕

**MSI** ↓ Download all

Name	Action
AM.Activities_v1.36.6.23.msi	↓ Download
AM.Composer_v1.36.6.15.msi	↓ Download

**ZIP** ↓ Download all

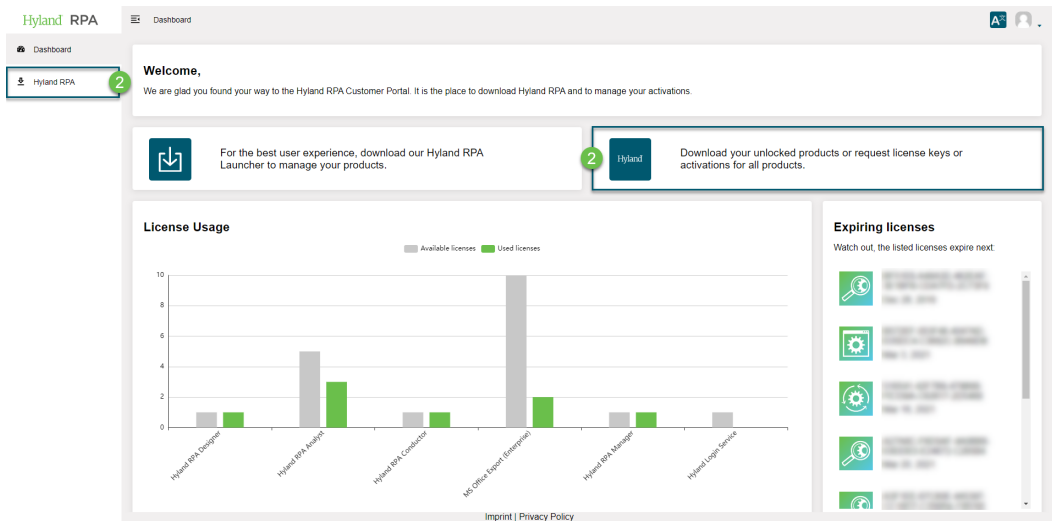
Name	Action
AM.Activities_v1.36.6.23.zip	↓ Download
AM.Composer_v1.36.6.15.zip	↓ Download

5. Check your download folder when the download has been completed.
6. Run the Installer and set up the app following the on-screen instructions or unzip the zip (preferably with 7zip).

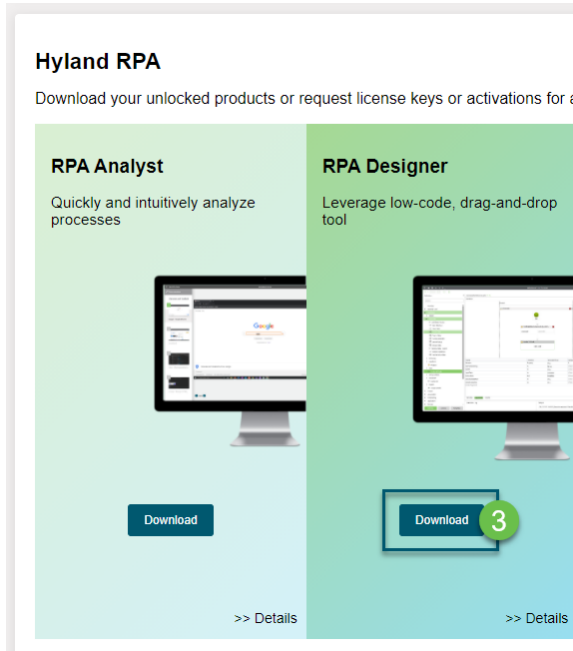
## Activating a License Online

Hyland RPA supports either online or offline license activation. Learn how you can activate your license online.

1. [Sign in](#) to your account with your email address and password.
2. Go to the Product overview by either clicking on Hyland RPA in the Menu or the link on the dashboard.



3. Go to the product page you want to activate, by either clicking on Details or the text/logo.



4. Copy a valid license key from the tab **Licenses & Activations** into your clipboard

**Hyland RPA Designer**  
Leverage low-code, drag-and-drop tools to quickly and easily build bots and create brand-new automations.

**Licenses & Activations**

Product Downloads

- Product -

Product	Expiration Date	Status	Activations	Deactivations
Hyland RPA Designer	Aug 7, 2022	✓	1/1	6/99

4

- Open the application & follow the on-screen instructions to activate your license.

**Hyland RPA Designer**

**Welcome.** Please enter your license key.

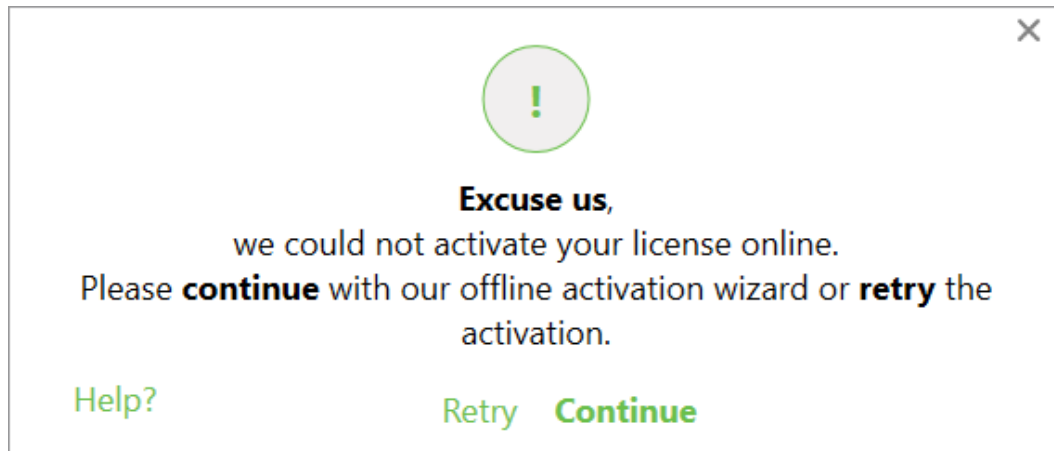
Request Trial      Customer Portal

**Activate**

- Enter your 32-characters license key here.
- Click on **Activate** to register your device.
- To request a trial license key, click on the [request trial link](#) to insert your data to access the portal.
- If you want to manage or access your licenses, click on the Customer Portal link to log in and view all products.

After clicking Activate, the tool tries to activate your device online first - if this is successful you are redirected to the Start window to start capturing processes.

If online registration is not possible, you will be notified by the following screen

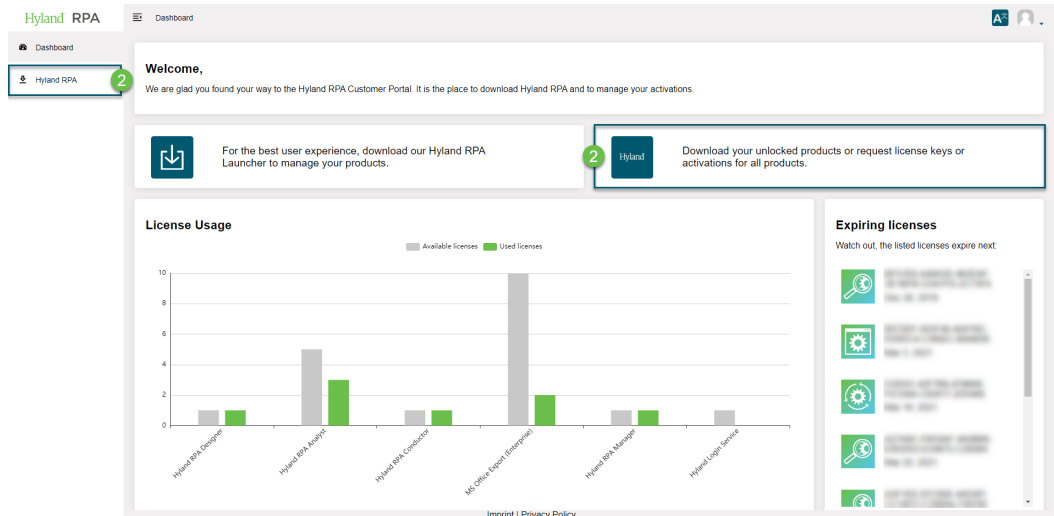


1. If you want to retry registration and go to the first screen, click **Retry**.
2. If you want to continue offline registration, click **Continue**.

## Activating a License Offline

Learn how you can activate your license offline, for clients without an active internet connection or firewall restrictions

1. [Sign in](#) to your account with your email address and password.
2. Go to the Product overview by either clicking on Hyland RPA in the Menu or the link on the dashboard.



3. Go to the product page you want to activate, by either clicking on Details or the text/logo.

## Hyland RPA

Download your unlocked products or request license keys or activations for all

**RPA Analyst**  
Quickly and intuitively analyze processes

**RPA Designer**  
Leverage low-code, drag-and-drop tool

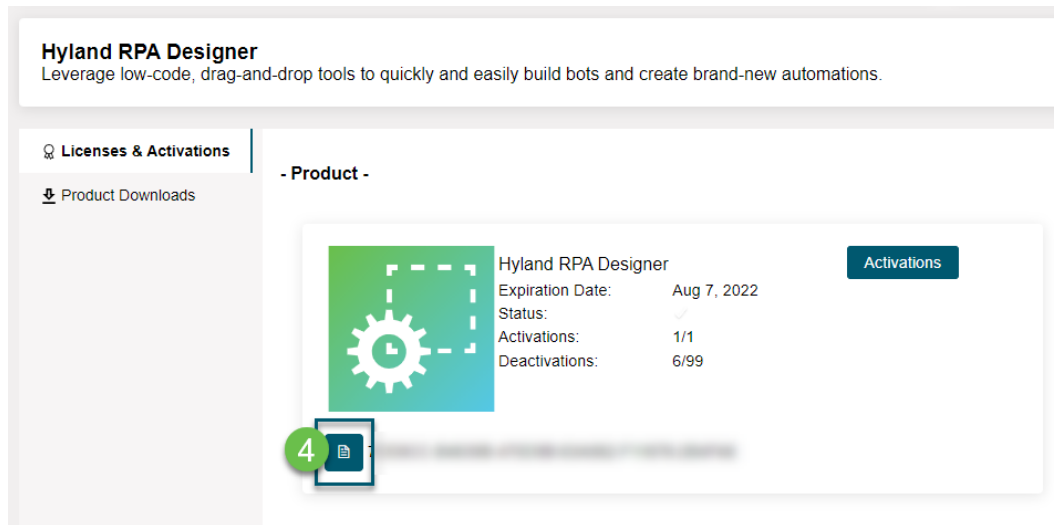
Download

>> Details

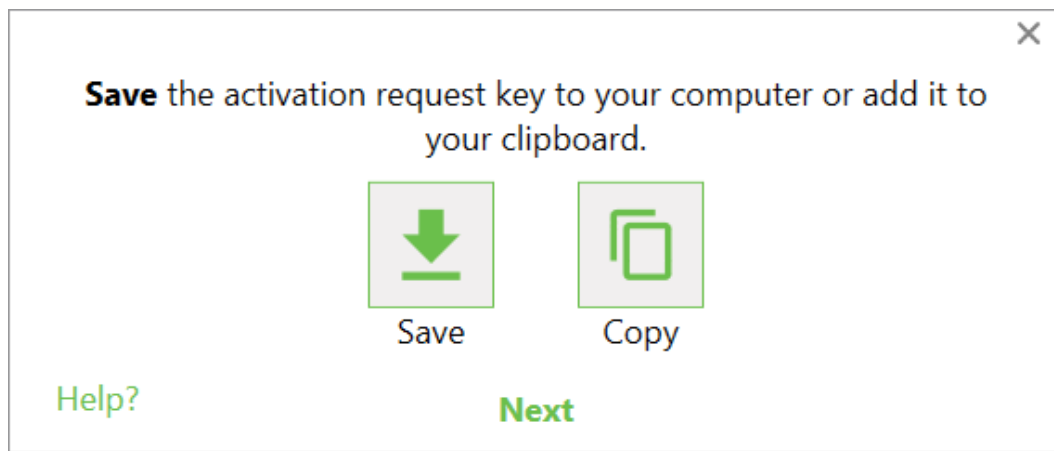
Download 3

>> Details

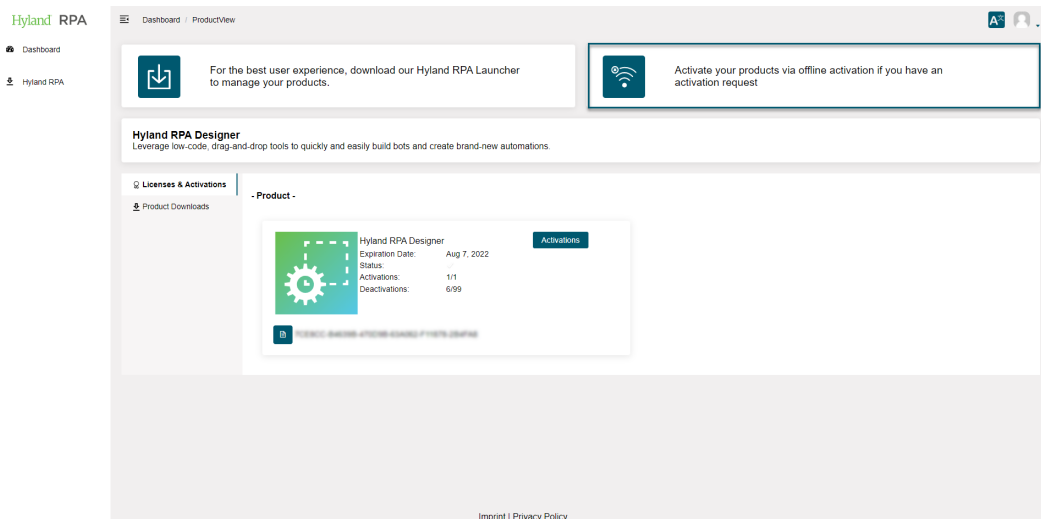
4. Copy a valid license key from the tab **Licenses & Activations** into your clipboard



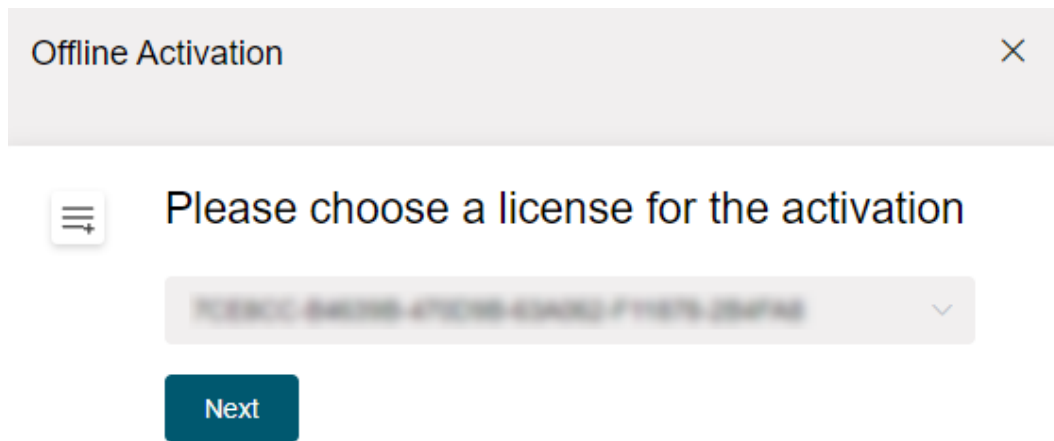
5. Open the application & follow the on-screen instructions to activate your license offline.
6. After clicking Continue with Offline Activation the following window appears:



7. If you want to save your offline registration key locally, click **Save**, choose a file location and save the key.
8. If you want to copy the offline registration key in your clipboard, click **Copy**.
9. After choosing one of the upper options, click **Next**.  
The next screen gives you detailed information on how to proceed with the offline registration key in the Hyland Customer Portal. Follow each of the four steps on a device with internet access.
10. Go to the product page and click on the single activation button in the upper right corner.



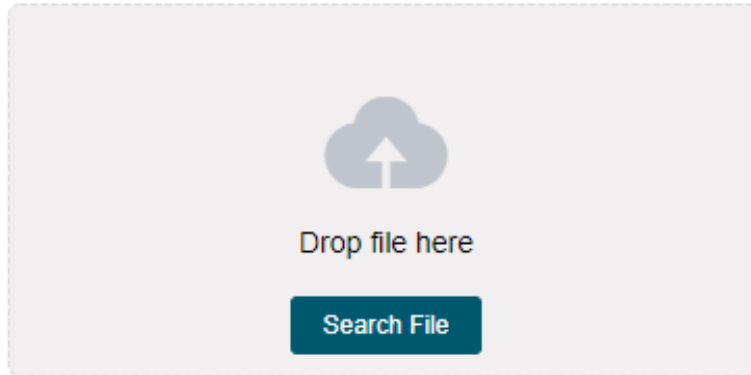
11. Choose a license to activate offline and press **Next**.



12. Upload, drop, paste or get your offline activation request from your clipboard and press **Activate**.



Please upload your activation request



Get Request from clipboard

- 13. Download the response to load into the product activation screen



Please upload your activation request

Offline-Activation-Challenge\_2020\_9\_22\_17\_41\_27.dat

Activate

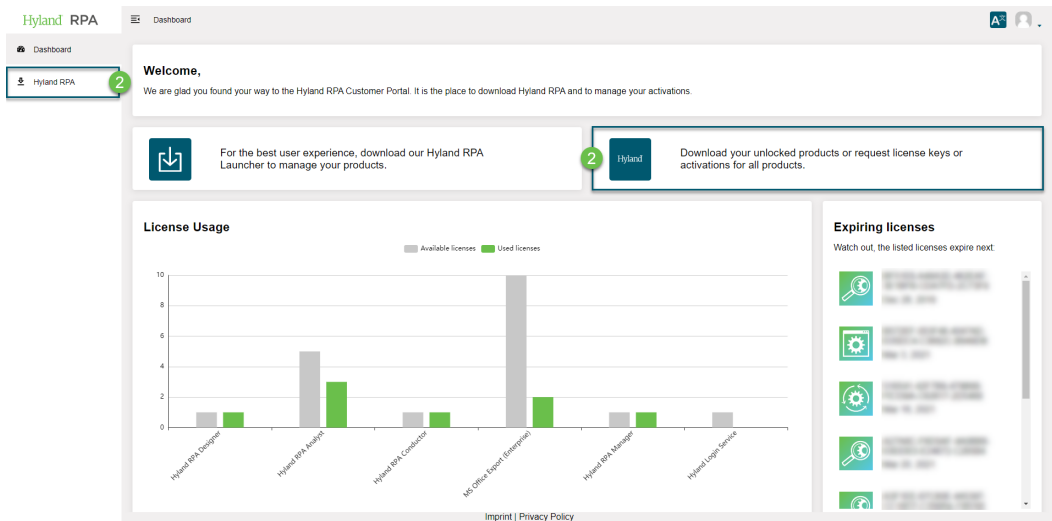
- 14. Go back to the offline client.
- 15. Click **Next** to insert your offline activation response key.
- 16. Enter or load the stored response key and click **Activate**.

## Deactivating a License

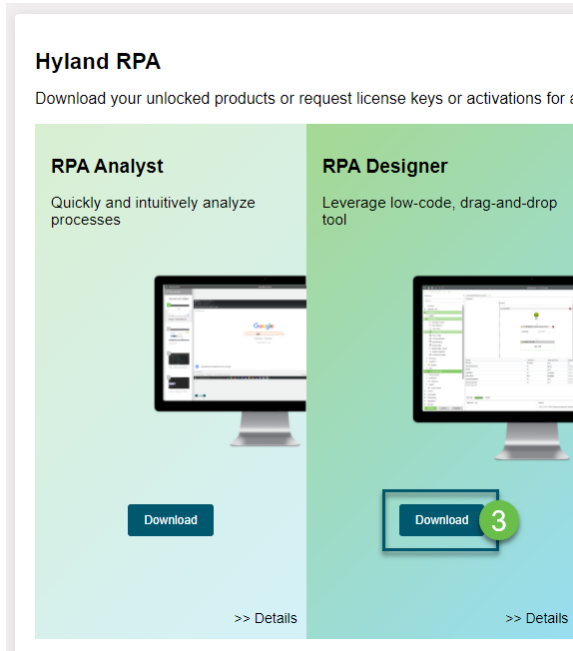
Client activation is limited based on your key and license metadata. Please note that license deactivation is limited as well. If you're having trouble deactivating your license, please get in touch with our customer support. See how you can remove your license activation for a client.

Generally, you can also remove the licenses within the products on the registered client - or you manage them in the Customer Portal:

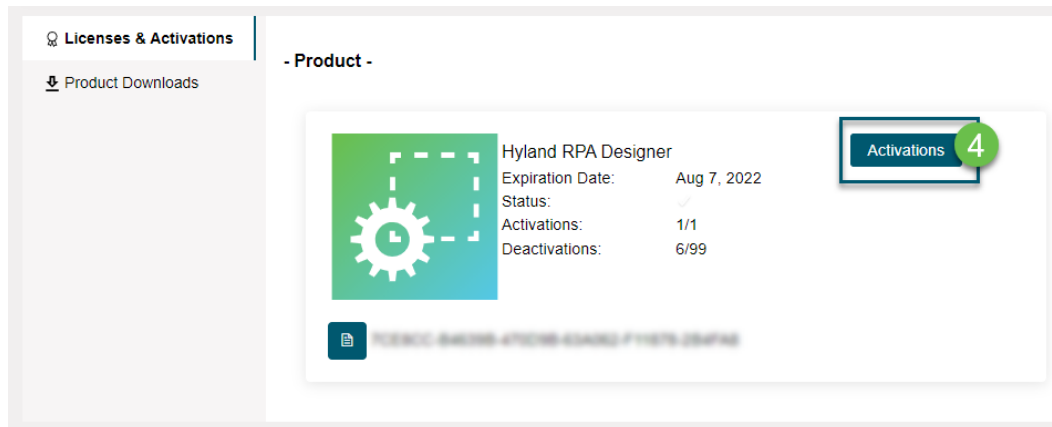
- 1. [Sign in](#) to your account with your email address and password.
- 2. Go to the Product overview by either clicking on Hyland RPA in the Menu or the link on the dashboard.



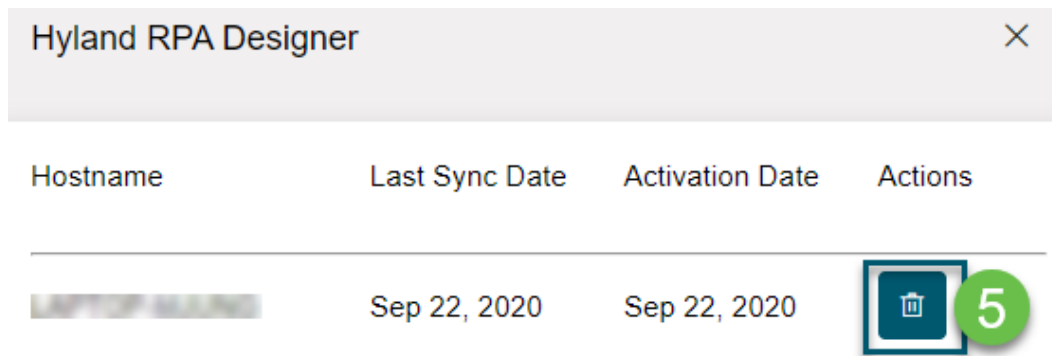
3. Go to the product page you want to activate, by either clicking on Details or the text/logo.



4. Click on Activations for a license to manage to view all registered users.



- In the activation overview dialog click on **Delete activation**.



- Confirm the action.

If you want to activate and manage your Hyland RPA Conductor and Hyland Login Service, see **Hyland RPA Conductor Setup** in the **Hyland RPA Conductor** documentation for more information.

## About Automation

RPA Designer offers its users numerous possibilities structuring workflows to automate processes.

Well-engineered processes, however, do aim for more than just automation. They are designed to serve three general purposes: stability, efficiency, and reusability. Because automation begins where the human has no need to intervene anymore, stable process designs represent the foundation for every RPA project. Even though processing speed is consequently not the primary objective of any RPA project, processes shall be developed efficiently. Efficiency on the one hand in terms of processing time, on the other hand in terms of maintenance, since following the development of a process, other people will work with the developed script. Finally, to take the opportunity of substantial synergy effects, the process design should consider the reusability of process sequences from the very beginning.

Consequently, Hyland RPA Designer's fundamental characteristics induce the need to define a standard, guiding the user during the development of processes. This document introduces the reader to the most efficient way of designing processes within Hyland RPA Designer by offering a modulation standard.

It may not be possible to meet certain criteria of the offered standard during development. In such cases, it is necessary to document deviations as precisely as possible, so that others are still able to understand the process in detail.

## Fundamentals

### MainScriptRunner

MainScriptRunner\_DCO.xml + X MainScriptRunner\_PRO.xml\* + X

In the **MainScriptRunner**, no matter if DCO or PRO, only the following activities should be included:

- Scope Activities, such as **Open Browser** or **Open Application**.
- Microbots
- Decisions / Switches

### Golden Rule

To automate processes with RPA Designer, it is required to understand the general structure of displayed workflows. The scope-based modulation principal of the RPA Designer must be considered in every case.

Follow the **Golden Rule** of modulation within RPA Designer. To build your processes, use the following scopes:

#### Scope 0 - The Microbot Scope

*Microbots/Buildingblocks*

Structure your processes so that the upper layer of your Hyland RPA Designer flowchart will only contain the fundamental logic of your process. In best cases, scope 0 will only contain Microbots, Buildingblocks and Decisions.

#### Scope 1 - The Application Scope

*Applications, Decisions, Microbots*

This scope is meant to give an overview about the different applications used for the corresponding Microbot/Buildingblock. In here, try to use necessary Decisions, Loops and Microbots.

#### Scope 2 - The Window Scope

The Window-Scope will illustrate the logic happening within the corresponding superordinated application. Try to use only windows, Decisions, Loops and Microbots

#### Scope 3 - The Activity Scope

The activity scope is meant to be your lowest scope level. In this scope all activities for a certain window/application take place such as Get Cell Value or Click Element. Besides the Activities Loops, Decisions or Microbots can be placed in this scope.

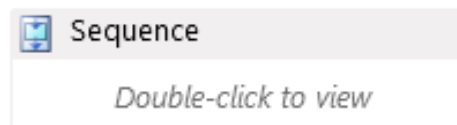
## Decision or If Activity



Within the **Flowchart**, use the **Decision** activity in case you want to split up the flow of activities.

To create a decision within any **Sequence**, use the **If** activity, whose layout is optimized for this use case.

## Flowchart or Sequence



In general, use a flowchart to better represent **Decisions** and different branches. A flowchart should also be prioritized for straightforward processes, so that it can be extended more easily if necessary.

## Logging

To be able to control, monitor and trace a process, use so-called logs (WriteLog activities) after certain events in the process. The description of the log must be self-explanatory and unique.

- After **Decisions**, to understand the decision path
- When reading out data, to check whether the data is correct and has been completely read out.
- When assigning variables to another type, to check if the variable is assigned correctly.

## About Microbots

Microbots have two basic uses - process-specific and generic. While process-specific Microbots represent a sub-process that only occurs in a specific process, the focus of generic Microbots is on reuse in loops, later in the same process or even in other processes.

### Process-Specific

Process-specific Microbots should reflect the building blocks in the process diagram and, if available, include generic Microbots to take advantage of reusability.

### Generic

Generic Microbots have the goal of being reused in another process.

The Microbot should always perform only one specific task, for example, filling out an input mask or creating a ticket. As in classical programming, the size of Microbots should be as small as possible and as large as necessary. The more complex a Microbot becomes, the more error-prone and specific it becomes.

At the beginning of each Microbot, there should be a check of the passing arguments. If there are no mandatory variables, start the check at the beginning. Optional variables should only be used if they are valid and present. In other words, optional arguments can be omitted and necessary ones forced.

It is also essential that the corresponding steps within the Microbot are commented sufficiently to simplify their use. The more structured the Microbot is developed, the less comments are necessary. Generic Microbots should not call any more Microbots to prevent nesting.

#### Short:

- Provide good and detailed comments for other users
- No further scopes (microbots)

## Naming Convention

Basically, the name of a Microbot must reflect its function. The name of the interface used by the Microbot must also be added. Since Microbots are generally to be kept short, their functionality is also limited, so that a useful name can be found.

#### Short:

- [Interface] [Functionality]
- **Example:** SapLogin, MCisCreateRequestTicket, WbciReadJobTable

## Arguments in Microbots

To be able to recognize arguments more quickly, a prefix is added to the variable name. For incoming arguments **In**, for outgoing arguments **Out** and for arguments which work in both directions **InOut**. In this way, you can immediately see in the script whether it is an argument and whether it is an input or output argument.

#### Short:

- Clear naming conventions of arguments
- If-switch-branches for individual usage (if an argument is empty, check it and ignore it if it is empty)
- **In** before input argument - InText
- **Out** before output argument - OutCount
- **InOut** before input/output argument - InOutList

## Repository

RPA Designer provides the functionality to store Microbots in a central file location. This allows different projects to benefit from the same Microbots. In order to keep the repository clear, a folder structure is required. Each interface receives a folder in which the corresponding Microbots are stored. In addition to the interface folders, a helper folder can also be created to store system processes or procedures that do not use an interface.

### Short:

- **For Application Microbots** - ..\ApplicationName\Microbot (\SAP\SapLogin)
- **For Helper Microbots** - ..\Helper\Microbot (Helper\DateTimeHelper

Microbots once stored in the repository may only be modified by the creator. If the Microbot has been transferred from the repository to an RPA Designer project and subsequently adapted, it must be renamed. It is now no longer allowed to commit to prevent a change of the origin Microbot. As long as no technical solution has been implemented, extreme caution must be exercised to prevent overwriting.

## About Activities

Basically, the name of the activity must indicate what this activity does.

### Example

A **Click Element / Image** activity that clicks on a **Save** button must be renamed to **Click Save Button**.

For **Getters** or **Setters**, specify what exactly is read or set or from which field is read. For example, if the personnel number is to be read from a field, the name of the activity must be **Get Personnel Number**. If the variable into which the text or similar is to be read does not have the same name, it must also be specified, for example **Get personnel number in [Variable name]** or **Set personnel number from [Variable name]**. The name of the activity should not be longer than the activity block itself.

## About Variables

Basically, the name of a variable must reflect its function and should be self-explanatory. In general, variables should always be given an English designation, unless the element of the variable to be designated is German, for example, when reading the **Mobilfunknummer** in a German CRM system, the variable would also be called **Mobilfunknummer**, provided that the field designation is also German. We also use the CamelCase notation. This means, if a variable consists of several terms, for example, **LoginnameSap**, this is not separated by an underscore, but highlighted by a capital letter. If necessary, the used application should always be written after the actual name of the variable.

### Short:

- Self-explanatory
- Leading language: Englisch
- CamelCase, for example, LoginnameSAP, FoundImageExcel

## About InfoValues

InfoValues are values that are stored in the database in the Info table. These can be changed via the Manager. This means that the process itself no longer needs to be touched.

Since there is no separation between the processes and all users access the same table, naming conventions must also be observed here.

The table consists of three columns: **InfoId**, **InfoName**, and **InfoValue**.

While the **InfoId** is assigned automatically and the **InfoValue** is defined by the usage, the name of the **InfoName** can be freely chosen. In principle, the names are written in English unless they are proper names. Furthermore, all letters are written around Upper Case. Words are separated by \_ (underscore). Here we adhere to the database naming convention.

**Short:**

- **Example:** Orderprocessing\_SUPPORTEMAIL

## About Custom Activities

In this tutorial, we provide a hands-on introduction to developing a custom activity.

You will see, the examples are simple enough even if you have never developed an activity with Workflow Foundation before, you can learn it in no time.

In the following lessons, you will be introduced to the most essential elements on how to develop an activity, one feature at a time. As a beginner, you should either start with the first lesson or, if you want a very brief introduction, take a look at the skeleton activity project. While the beginner's course will enable you to develop a simple activity, the skeleton activity has example's on some advanced features, which are a bit harder to grasp but can take your activity development to the next step.

## Fundamentals

### Activity (ViewModel)

The activity project is the connection between the application (logic) and the design (view) and can be called a view model. The project also contains information for descriptions and tooltips.

### Design (View)

The design project contains the visible part of the activity (view) and is responsible for how the activity will look in the Designer. The XAML and the code-behind-class are stored here.

## Test

With unit tests, you can validate the functionality of the activity and helps to avoid side effects when changes to the activity are done in the future.

## Application (Model)

The application project contains the business logic of the activity; this includes interfaces to other services, databases or external systems. Therefore it is the backend of the activity.

## Requirements for the Custom Activity

Before we start developing, we have to think about what we desire the activity to do. You should ask yourself two central questions.

1. Input - What kind of data is at my disposal?
2. Output - What kind of data should the activity return

## Awesome Calculator - Example

For this tutorial we will develop the following activities:

The first activity takes a string as input, with a simple arithmetic formula of numbers and the four basic operations (plus, minus, multiply and divided) and will perform a calculation on it.

- Name: CalculateFromStringActivity
- Input: Calculation as String, e. g. 1 + 3
- Output: The result is an integer

The second activity is also intended to perform a calculation, but this time the values are passed directly as input.

- Name: CalculateSimpleActivity
- Input: Number 1 as int, number 2 as int and the calculation type should be selectable via a drop-down list
- Output: The result as int

## About Setting up the Solution

### Get the Skeltons

The Skeleton Activities are available on [GitHub](#).

### Configuration and Renaming

The first step is to rename the solution, project and the corresponding namespaces to the new name. Also, rename the file "AM.Skeleton.Activites\_metadata.xml" in the activities project (AM.Skeleton.Activities). The naming should be structured as follows: "AM. interface/functionpackage.Activities".

## Activity preparation

We can delete the ExampleActivity.cs. And create our first activity "CalculateFromStringActivity". Let the new class inherit from "AbstractCodeActivity", implement the missing methods and make the class "public sealed". Since we have already defined input and output, we add the following variables.

### CalculateFromStringActivity:

- public InArgument<string> CalculationString { get; set; }
- public OutArgument<int> Result { get; set; }

Proceed the same way with the second activity "CalculateSimpleActivity". Since we want to select the basic arithmetic operation, we need to create an Enum. Create in your application project a new file "ElementaryArithmetic.cs" with a public enum "ElementaryArithmetic".

### CalculateSimpleActivity

- public InArgument<int> NumberOne { get; set; }
- public InArgument<int> NumberTwo { get; set; }
- public ElementaryArithmetic CalculationMethod { get; set; }
- public OutArgument<string> Result { get; set; }

We will discuss the use of InArgument and OutArgument later.

## Unit Test

Before we venture further into the application project, we need to consider test scenarios to validate future functions. We already defined the basic framework for our activities.

Now we create a new test class in the test project, using a public function. The attributes above the method header are test parameters. As test parameters, we input calculations that contain all basic arithmetic operations. As a result, we expect the value 6 in all cases.

```
[TestCase("3 + 3", 6)] [TestCase("7 - 1", 6)] [TestCase("2 * 3", 6)]
[TestCase("36 / 6", 6)] [TestCase("36 / 0", 0)] public void
CalculateFromString(string calculationString, int result) { }
```

Now it's on to the function body. Since our application project should contain the business logic, we have to create new classes and methods to enable the test scenario to execute these. Let's create a new interface with a function CalculateFromString that returns the result of the calculation as int and accepts a string with the arithmetic formula as input.

```
public interface ICalculatorApplication { int CalculateFromString(string
calculationString); }
```

Afterward, we create a new class, that implements the interface, with the method CalculateFromString. Temporarily this method returns the value 0.

```
public class CalculatorApplication : ICalculatorApplication { public int
CalculateFromString(string calculationString) { return 0; } }
```

After this is done we implement the function body of our test. First, create a new variable of the type `CalculatorApplication` and initialize it. Directly after we call the `CalculateFromString` method and the parameters of our test, and assign the return value to a new variable of type `int`.

```
{ ICalculatorApplication calcApp = new CalculatorApplication(); int
  calculationResult = calcApp.CalculateFromString(calculationString); }
```

The test class should now look as follows:

```
[TestFixture] public class CalculateFromStringActivityTest { [Test]
[TestCase("3 + 3", 6)] [TestCase("7 - 1", 6)] [TestCase("2 * 3", 6)]
[TestCase("36 / 6", 6)] [TestCase("36 / 0", 0)] public void
CalculateFromString(string calculationString, int result) {
ICalculatorApplication calcApp = new CalculatorApplication(); int
calculationResult = calcApp.CalculateFromString(calculationString);
Assert.That(calculationResult, Is.EqualTo(result)); } }
```

All tests will fail for now when you run the unit test, since the `CalculateFromString` method returns the value 0 at the moment but the test expects a 6 as result.

## About the Application Project

The application project contains the business logic, related data structures, interfaces to other services or databases.

Because we created already a functioning unit test, we can start with the implementation of the business logic.

Since this tutorial is focused on the development of Activities we will not go into to much detail of the `CalculatorApplication` implementation. See below for a completed Implementation of `CalculatorApplication`.

```
public class CalculatorApplication : ICalculatorApplication { private
const string ArithmeticSignPattern = @"[\+|-|\*|/]"; public int
CalculateFromString(string calculationString) { string
arithmeticSignString = this.GetElementaryArithmeticString
(calculationString); int firstNumber = this.GetFirstNumber
(calculationString, arithmeticSignString); int secondNumber =
this.GetSecondNumber(calculationString, arithmeticSignString); var
arithmeticSign = this.GetElementaryArithmetic(arithmeticSignString);
return this.Calculate(firstNumber, secondNumber, arithmeticSign); } public
int Calculate(int firstNumber, int secondNumber, ElementaryArithmetic
elementaryArithmetic) { switch (elementaryArithmetic) { case
ElementaryArithmetic.Addition: return firstNumber + secondNumber; case
ElementaryArithmetic.Subtraction: return firstNumber - secondNumber; case
ElementaryArithmetic.Division: try { return firstNumber / secondNumber; }
catch (Exception e) { Console.WriteLine(e); throw; } case
ElementaryArithmetic.Multiplication: return firstNumber * secondNumber; }
return 0; } private string GetElementaryArithmeticString(string
calculationString) { Regex regex = new Regex(ArithmeticSignPattern); var
matches = regex.Matches(calculationString); if (matches.Count != 1) {
```

```

Console.WriteLine("More than one calculation sign detected."); } return
matches.Cast<Match>().FirstOrDefault()?.Value; } private
ElementaryArithmetic GetElementaryArithmetic(string arithmeticSign) {
switch (arithmeticSign) { case "+": return ElementaryArithmetic.Addition;
case "-": return ElementaryArithmetic.Subtraction; case "*": return
ElementaryArithmetic.Multiplication; case "/": return
ElementaryArithmetic.Division; default: Console.WriteLine("Elementary
Arithmetic sign not found"); return ElementaryArithmetic.Addition; } }
private int GetFirstNumber(string calculationString, string
arithmeticSign) { string firstNumberString = calculationString.Substring
(0, calculationString.IndexOf(arithmeticSign, StringComparison.Ordinal));
if (!int.TryParse(firstNumberString.Trim(), out int firstNumber)) {
Console.WriteLine("First number not found"); } return firstNumber; }
private int GetSecondNumber(string calculationString, string
arithmeticSign) { int startIndex = calculationString.IndexOf
(arithmeticSign, StringComparison.Ordinal) + 1; string secondNumberString
= calculationString.Substring(startIndex, calculationString.Length -
startIndex); if (!int.TryParse(secondNumberString.Trim(), out int
secondNumber)) { Console.WriteLine("Second number not found"); } return
secondNumber; } }

```

Now that CalculatorApplication is implemented correctly the unit test will now pass successfully.

## About the Activity

Let's start developing an actual activity, which mediates between our business logic and our view.

### Properties

The Hyland RPA Designer will display all declared public properties inside an activity; as options in the property panel.-

These can be used to pass data from the process to the activity. There are two different types of Argument to pass data from the Designer to the activity.

#### Direct Data Type

If, for example, you create a property of type string, it is possible to pass the desired value directly to the activity. Similar is the case with Enums, which will create a drop-down menu with the enum's values. However, it is not possible to pass this data to a variable that is declared inside the Designer and vice versa.

To pass data from the Designer to the activity and the other way around Workflow Foundation provides for this case the data types In- and OutArgument.

```
public string CalculationStringNoArgument { get; set;
```

#### InArguments

The goal is to pass a variable with previously collected data from the running process to the activity. For this, we use the generic type InArgument, which takes the type via the angle brackets.

```
public InArgument<string> CalculationString { get; set; }
```

Now a variable can be passed to the activity. To access the data that is being passed in, we use the context parameter of the execute method (CodeActivityContext context), and with the context.GetValue method we get the desired data.

```
string calculationString = context.GetValue(this.CalculationString);
```

Since we use the generic InArgument, the method GetValue returns the correct data type.

### OutArguments

OutArgument functions the same as InArgument but as the name implies it is the other way around. Using the context parameter, we can pass variables back to the Designer. With the method SetValue, we pass the variable from the activity to the Designer as the correct data type.

```
context.SetValue(this.Result, calculationResult);
```

### Enums

Enum's can be used to show drop-down inside the Designer. Enum's must be declared directly and not as In- or OutArgument.

### Attributes

To control how Arguments are displayed inside the Designer, attributes can be used before each Argument.

Here is a short list of frequently used attributes:

- Category -> Shows the Argument inside a specific Category on the Properties Panel
- DisplayName -> The Name the argument it should have on the Properties Panel
- RequiredArgument -> Indicates that the Argument needs to be set
- VariableSelectionOutputPopup/VariableSelectionInputPopup ->a popup will be displayed when dragging the activity into your workflow. This will prompt the user to specify a variable or enter a value directly. Highly recommended for required arguments
- Description -> A tooltip is shown when hovering over the Argument. Additionally, the description will be displayed inside the input box.

### Metadata

Instead of writing Attributes above an Argument to change how it's displayed inside the Designer you can use the AM.Skeleton.Activities\_metadata.xml for this purpose.

See the example below:

```
<ProjectMetadata> <Metadata> <ActivityMetadata ActivityType=
"AM.Skeleton.Activities.CalculateFromStringActivity">
<DisplayName>Calculate from String</DisplayName> <Description>Calculates a
math formula from a string</Description> <Category>Math</Category>
<SearchTags>calculate, math, from string</SearchTags> <PropertiesMetadata>
```

```

<PropertyMetadata PropertyName= "Result" DisplayName= "Result"
RequiredArgument= "true" HintText= "Integer" Category="Output"
VariableSelectionInputTextPopup= "false"
VariableSelectionInputVariablePopup= "false" VariableSelectionOutputPopup=
"true" Order="0" /> <PropertyMetadata PropertyName= "CalculationString "
DisplayName= "Calculation String" Description= "String with the math
formula" RequiredArgument= "false" HintText= "Boolean" Category="Options"
VariableSelectionInputTextPopup= "false"
VariableSelectionInputVariablePopup= "false" VariableSelectionOutputPopup=
"true" Order="1" /> </PropertiesMetadata> </ActivityMetadata> </Metadata>
</ProjectMetadata

```

## Synchronous Activities

Most common Activities are Synchronous. Your type should inherit from `AbstractCodeActivity` or from a type that does. `AbstractCodeActivity` contains an abstract method that you need to overwrite. To pass Arguments into your business logic, you have to use `context.GetValue()`. After processing the value, you can then return it to the Designer via `context.SetValue()`.

See example below:

```

protected override void Execute(CodeActivityContext context) { string
calculationString = context.GetValue(this.CalculationString);
Console.WriteLine(calculationString); ICalculatorApplication calcApp = new
CalculatorApplication(); int calculationResult =
calcApp.CalculateFromString(calculationString); context.SetValue
(this.Result, calculationResult); }

```

You should add error handling to the activity to show for example when arguments are left empty.

## Async Activities

If you have a long-running task its best to run it on a separate thread, after the task is complete, you can use the type `AbstractTaskAsyncCodeActivity`.

If your task needs to return a value, you need to use the type `AbstractTaskAsyncCodeActivity<T>`

See async without result Example below:

```

public class AsyncActivity : AbstractTaskAsyncCodeActivity { public
InArgument<string> FilePath { get; set; } private static async void
HandleFileAsync(string file) { /// Long running task } protected override
Task ExecuteAsync(AsyncCodeActivityContext context, CancellationToken
cancellationToken) { string file = FilePath.Get(context); return
Task.Factory.StartNew(() => HandleFileAsync(file), cancellationToken); } }

```

For an Async with result example take a look inside the Skeleton Activity.

## About Error Handling and Logging

### Error Handling

To add error handling to the activity, the method `CacheMetadata( CodeActivityMetadata metadata )` must be overridden to add a custom error messages or warnings inside the Composer.

Error message example below:

```
protected override void CacheMetadata(CodeActivityMetadata metadata) {
    base.CacheMetadata(metadata);if (CalculationString == null)
    metadata.AddValidationError("Argument CalculationString has not been
    set."); }
```

Warning message example below:

```
protected override void CacheMetadata(CodeActivityMetadata metadata) {
    base.CacheMetadata(metadata); if (CalculationString == null) {
    ValidationError validationWarning = new ValidationError("Argument
    CalculationString has not been set.", true); metadata.AddValidationError
    (validationWarning); } }
```

### Write Log

To write to logs from your activity, you need to use the class `AM.Logging.LogHelper`

The log levels you can use are the following.

- `LogLevel.Debug`
- `LogLevel.Info`
- `LogLevel.Warn`
- `LogLevel.Error`

See example below:

```
try { // Error here } catch (Exception ex) { LogHelper.Error(ex.ToString
()); throw; }
```

## Design, Deployment and Use of Designer

### Design

Activities can be used without a Design, but will display a default layout inside RPA Designer.

To create a custom Design, add a new **ActivityDesigner** to the `AM.AwesomeCalculator.Activities.Design` named **StringCalculatorDesinger**.

To indicate that the new design is intended for the Analyst view, we need to add the following line to the StringCalculatorDesinger.xaml.cs file, above the class name:

```
[AnalystDesigner (typeof (StringCalculator)) ]
```

To create a Developer view, use the following line:

```
[DeveloperDesigner (typeof (StringCalculator)) ]
```

To use this design as default, add the following resource dictionary inside the StringCalculatorDesinger.xaml:

```
<FrameworkElement.Resources> <ResourceDictionary>
<ResourceDictionary.MergedDictionaries> <ResourceDictionary
Source="pack://application:,,/AM.Common.Activities.Design;component/Stylin
gDictionaries/ActivityStyling.xaml" /> <ResourceDictionary>
<amConverters:PropertyDescriptorConverter
x:Key="PropertyDescriptorConverter" /> <ArgumentToExpressionConverter
x:Key="ArgumentToExpressionConverter" xmlns="clr-
namespace:System.Activities.Presentation.Converters;assembly=System.Activi
ties.Presentation" /> </ResourceDictionary>
</ResourceDictionary.MergedDictionaries> </ResourceDictionary>
</FrameworkElement.Resources>
```

## Deployment

To use your custom activity, compile the solution and copy all files from the **SkeletonActivity** folder into the **Libraries** folder of RPA Designer.

## Use of Designer

Inside RPA Designer, you can use your custom activity from the library tab.