# Document Understanding Process

Studio Template

## Revision History

| Date | Changes |
|---|---|
| June-2021 | 1.0 |
| Oct-2021 | 2021.10 |
| Feb-2022 | 22.2-preview |
| May-2022 | 22.4 |
| June-2022 | 22.6-preview |
| Oct-2022 | 22.10 |
| November-2023 | 23.10 |
| April-2024 | 24.4-preview |
| July-2024 | 24.10-preview |
| Nov-2024 | 24.10 |

## Release Notes:

### 24.10:

- Introducing the cross-platform template.
- Update flowcharts in all templates.
- Updates to the AutocorrectOcrMistakes workflow.

### 24.10-preview:

- Enabled by default the use of modern projects ML models.

### 24.4-preview:

- Fixed bug with AutoCorrectOcrMistakes Test

### 23.10:

New features:

- AutocorrectOcrMistakes (See dedicated chapter):
    - Added AutocorrectOcrMistakes under the reusable components folder.
    - Added AutocorrectOcrMistakes sheet in the Config file.
- Orchestrator Configuration Quick Start (See dedicated chapter) – workflow for quickly setting up Storage Buckets, Queues, Assets and/or Task Catalogs.

### 22.10:

New features:

- Testing:
    - Added 19 tests under the test folder that test basic DU and framework functionality.
    - Added BatchTestingUtilities under the test folder. These help the tests to run faster, but also during the development to save data to cache, and not run the whole process again.
    - Added a Cache folder based on the sample file we have.
- Main Logic:
    - Added a flow decision that would make the workflow end successfully if no more transaction items are present.
    - Automatic detection of decimal points in Invoice Post Processing
    - Added Receipt Post Processing
- Other:
    - Validated new Out of the Box Insight dashboards.

Changes:

- Cleaned up the Config File and added the Receipt Post Processing part.
- Hidden the option of C# for now.

Bugfixes:

- "For each" activity in "Parallel for each" issue should be solved.

## 22.6-preview:

New features:

- Added C# support.
- Introduced Asset naming convention for better visibility of Document Understanding Assets in Orchestrator
- Added new overriding Assets for Orchestrator Configuration.

Bugfixes:

- Fixed error in reading of the Config file in **Main-Attended.xaml.**
- Fixed the Log Message from the **Classify** workflow in C#. It now displays the correct classification results.

## 22.4:

New features:

- Added .NET 5.0 support.

Changes:

- Removed *Windows – Legacy* support.
- Updated all dependencies to the latest stable versions.
- Rescoped the *extractionResults* variable in all **Main** workflows to prevent misuse

Bugfixes:

- Corrected small typos.

## 22.2-preview:

Changes:

- **Framework\ReusableWorkflows\SetTransactionProgress.xaml** - Retry Scope within was updated with ContinueOnError = True
- **Framework\40_TrainClassifiers.xaml** – Invoke UnlockFile activity within was updated with ContinueOnError = True

Bugfixes:

- Fixed the page range logging on document exceptions in **Main-Attended.xaml**

## 21.10:

New features:

- New workflows for advanced Business Rule validation for Classification and Extraction
- New workflow for post processing Invoices based on the OOTB (out of the box) model.
- Validation Station/Action now only displays the relevant pages instead of the whole file. Removed the PDF splitting functionality.
- Support for ML Extractor Trainer to use the auto-retrain feature in AI Center
- New assets / settings in the config file for skipping classification/extraction training.
- New assets / settings in the config file to force documents through manual validation (useful in development/UAT especially)
- New exception type: **DocumentRejectedByUserException**. This exception is thrown when a user explicitly rejects a document during validation. When setting the transaction status, a DocumentRejectedByUserException is treated the same way as a BusinessRuleException

Changes:

- Exception handling workflows now receive the Exception as an argument instead of an exception message.
- Improved logic for loading Orchestrator assets
- Improved exception messages
- Improved comments and annotations
- Improved logging

## Overview

As the number of Document Understanding engagements keeps increasing in complexity, the need for a common implementation/RPA approach becomes evident. Document Understanding processes have a particular logical flow and requirements that are better suited for a dedicated approach of their own, instead of relying on the existing RE-Framework.

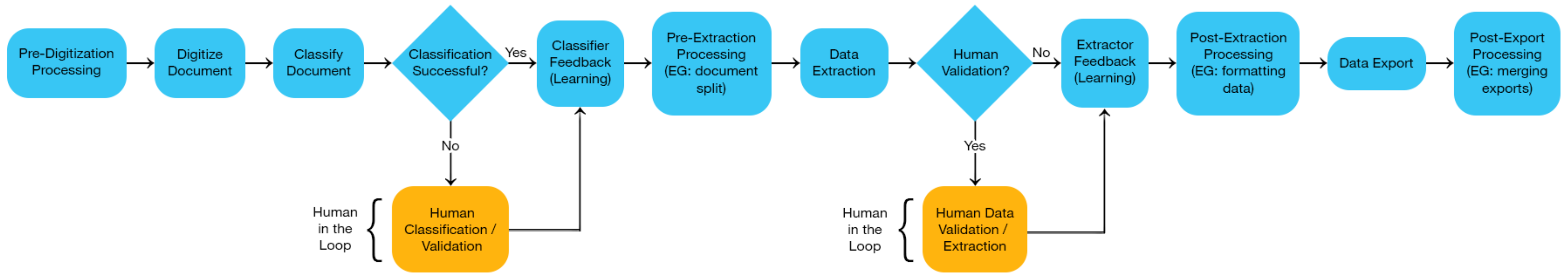
### Key Features

- Seamless getting started process with new Document Understanding projects.
- Suitable for all use cases: from quick demos to scalable implementation projects
- Production-ready. Implements built-in logging, exception handling, and retry mechanisms.
- Common architecture for both Attended and Unattended (plus Action Center) implementations. Simple to switch between solutions.
- 2021.10: Ability to post-process Invoices to minimize the number of hits to Action Center
- Designed to make development, testing, deployment, debugging, and scaling easy.
- Follows the best practices pertaining to RPA, Document Understanding, Orchestration Processes, and Long-running workflows.

## Generic Document Processing Flow

### Classic/Windows:

Processing flow for each valid document

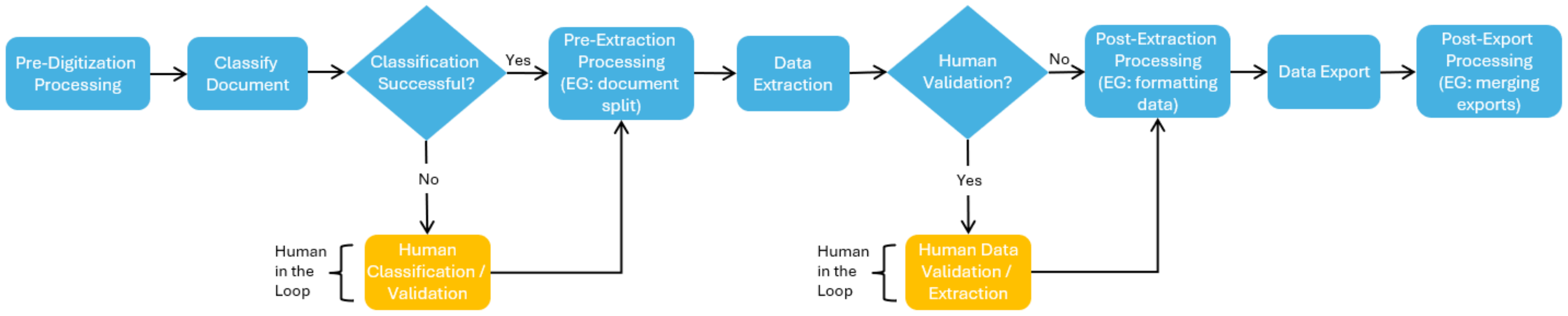


Pre-Digitization Processing
Digitize Document
Classify Document
Classification Successful?
Yes
No
Classifier Feedback (Learning)
Pre-Extraction Processing (EG: document split)
Data Extraction
Human Validation?
No
Yes
Extractor Feedback (Learning)
Post-Extraction Processing (EG: formatting data)
Data Export
Post-Export Processing (EG: merging exports)
Human in the Loop
Human Classification / Validation
Human in the Loop
Human Data Validation / Extraction

Keep in mind that the diagram above shows the most detailed logical flow split into the smallest possible modules. In practice, it is to be expected that some parts could be merged or might be completely excluded, as they are not required in a particular implementation.

**Cross-platform:**

Processing flow for each valid document


Pre-Digitization Processing
Classify Document
Classification Successful?
Yes
No
Human in the Loop
Human Classification / Validation
Pre-Extraction Processing (EG: document split)
Data Extraction
Human Validation?
No
Yes
Human in the Loop
Human Data Validation / Extraction
Post-Extraction Processing (EG: formatting data)
Data Export
Post-Export Processing (EG: merging exports)

Same as for Classic/Windows, keep in mind that the diagram above shows the most detailed logical flow split into the smallest possible modules. In practice, it is to be expected that some parts could be merged or might be completely excluded, as they are not required in a particular implementation.

### Pre-Digitization Processing

Any processing that needs to take place before digitizing the document. E.g., applying grayscale or a skew correction.

### Digitize Document

**Classic/Windows:**

A digital version of the document is obtained. In case the document is scanned, OCR (Optical Character Recognition) is required.

**Cross-platform:**

Digitize is not a separate step. It is included and done together with classification or extraction.

### Classify Document

**Classic/Windows:**

The document is classified.

**Cross-platform:**

The document is classified. Digitization is done at the same time.

### Classification Successful?

The logic required to determine whether the classification was successful or not. Based on this decision, classification validation might be needed.

**Note**: Deciding whether human validation is needed or not is a business decision.

### Human Classification / Validation

If the automatic classification was unsuccessful (low confidence, business rules are not met, etc.), send the document to a human operator for manual classification/validation or rejection.

### Classifier Feedback (Learning)

**Classic/Windows:**

Integration of the learning mechanisms, if any, for classifier(s). It is recommended to only train with human-validated data, but not mandatory.

**Cross-platform:**

Learning mechanisms are not available.

### Pre-Extraction Processing

Any processing that needs to take place before the actual data extraction begins.

### Data Extraction

**Classic/Windows:**

Extract the data using the appropriate extractor(s).

**Cross-platform:**

Extract the data using the appropriate extractor(s). Digitization is done at the same time if not already done by the classifier.

### Human Validation?

The logic required to determine whether the human validation of the extracted data (validating the extracted data against pre-defined business rules, checking confidence levels, OCR confidence, missing data, etc.) is needed or not.

**Note**: Deciding whether human validation is needed or not is a business decision.

### Human Data Validation / Extraction

Human-in-the-loop step where the human operator can correct automatically extracted data or manually extract the missing data.

### Extractor Feedback (Learning)

**Classic/Windows:**

Integration of the learning mechanisms, if any, of the used extractor(s). It is recommended to only train with human-validated data, but it is not mandatory.

**Cross-platform:**

Learning mechanisms are not available.

### Post-Extraction Processing

Any processing that should be done to the extracted data before exporting it. E.g., formatting the data in a specific manner.

### Data Export

Exporting the data to a persistent location so that it can be used by other processes or by business users. Common examples: exporting to the Data Service, updating a Database, or writing the results to an Excel, CSV, or JSON file.

### Post-Export Processing

Any processing that should be done after the data is exported. E.g., merging data extracted from multiple documents into a single result or sending a notification email.

## Solution Architecture

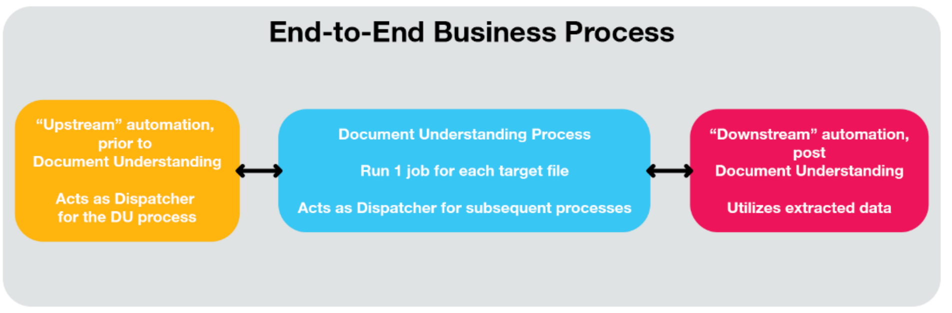
### Document Understanding, Part of End-to-End Business Processes

Document Understanding processes are not standalone. They are part of bigger business processes to be automated.

Benefits:

- Prevents external issues from impacting Document Understanding processes and from causing unneeded consumption of license pages through re-execution.
- Better overview of workload and robot utilization.
- Easier to scale.

The architecture for an end-to-end Business Process involving Document Understanding consists of:


End-to-End Business Process
"Upstream" automation, prior to Document Understanding
Acts as Dispatcher for the DU process
Document Understanding Process
Run 1 job for each target file
Acts as Dispatcher for subsequent processes
"Downstream" automation, post Document Understanding
Utilizes extracted data

1. **Upstream automation** which handles all the business logic that should be executed prior. It consists of one or several RPA processes and acts as a Dispatcher for Document Understanding jobs.
2. **Document Understanding process** - acts as a Performer for "Upstream" and as a Dispatcher for "Downstream" jobs.
3. **Downstream automation** which handles all the business logic that makes use of the extracted data. It consists of one or several RPA processes and acts as a Performer for Document Understanding jobs.

**The One-Job-Per-File Approach**

**Classic/Windows**:

Document Understanding processes will not run as batch jobs. Instead, an individual job starts for each file to be processed. This approach is used for both Attended and Unattended implementations.

**Cross-platform:**

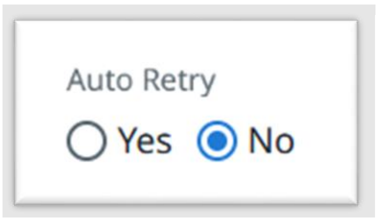
Similar to Classic/Windows, the Document Understanding processes will start an individual job for each file to be processed. Since splitting is not supported yet, each file must contain one type of document only. This approach is using only Unattended implementations. Attended implementations are not supported.

## Document Understanding and Queues

### Orchestrator v20.10.8 or Newer

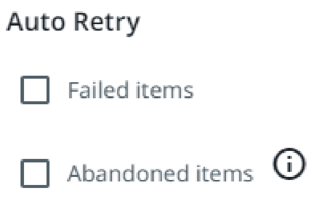
Orchestrator version 20.10.8 introduced queue support for Persistence activities enabling queues to be used.

**Note**: To avoid Document Understanding license consumption through re-executions, the **Auto Retry** functionality of the queue should be **disabled**.



Auto Retry

Yes [✓ No]



Auto Retry

☐ Failed items

☐ Abandoned items

### Orchestrator Versions Prior to 20.10.8

Classic/Windows:

Queues should not be used for Document Understanding processes, as they are not supported. Items waiting for Human Validation for more than 24h will be marked as **Abandoned**.

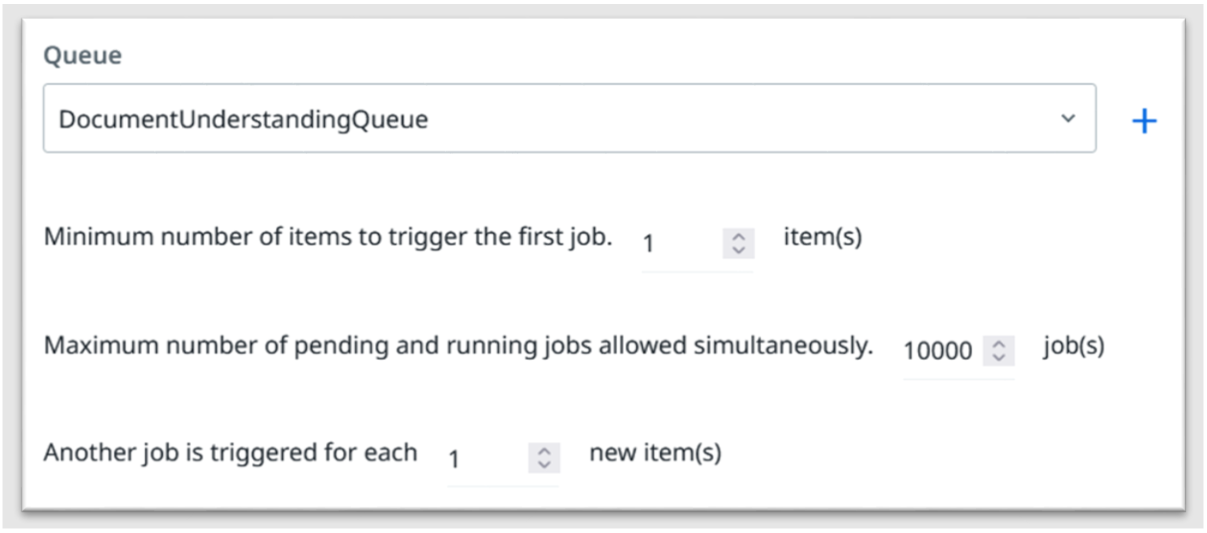
Cross-platform:

Not supported.

### Starting Document Understanding Jobs

A dedicated Dispatcher mechanism is required to start Document Understanding jobs in Unattended scenarios. There are two recommended approaches:

1. A Queue Trigger, tasked to start a new job for every new Queue item, using the settings stated in the image below



Queue

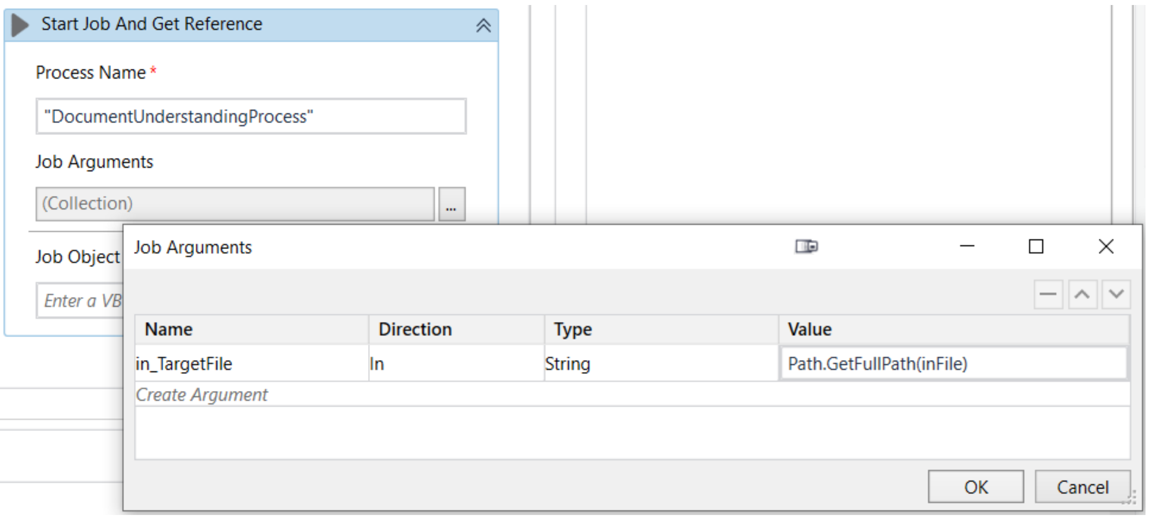
DocumentUnderstandingQueue

Minimum number of items to trigger the first job. 1 item(s)

Maximum number of pending and running jobs allowed simultaneously. 10000 job(s)

Another job is triggered for each 1 new item(s)

2. Having the Dispatcher process use a Start Job activity



| Name | Direction | Type | Value |
|---|---|---|---|
| in_TargetFile | In | String | Path.GetFullPath(inFile) |

**Note**: Currently, only the **Start Job And Get Reference** activity has support for passing job arguments. This activity is part of the **UiPath.Persistence.Activities** package.

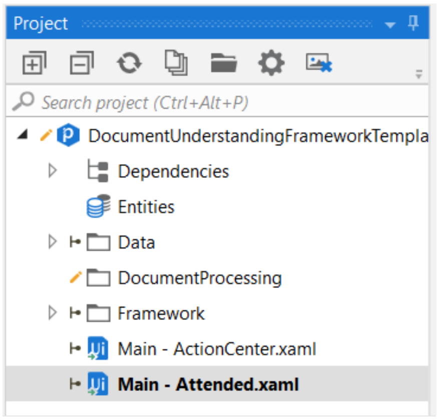
## Studio Project Overview

The Document Understanding Process is available as a UiPath Studio project template and the projects created using it automatically include all files.

### Settings for Attended Processes

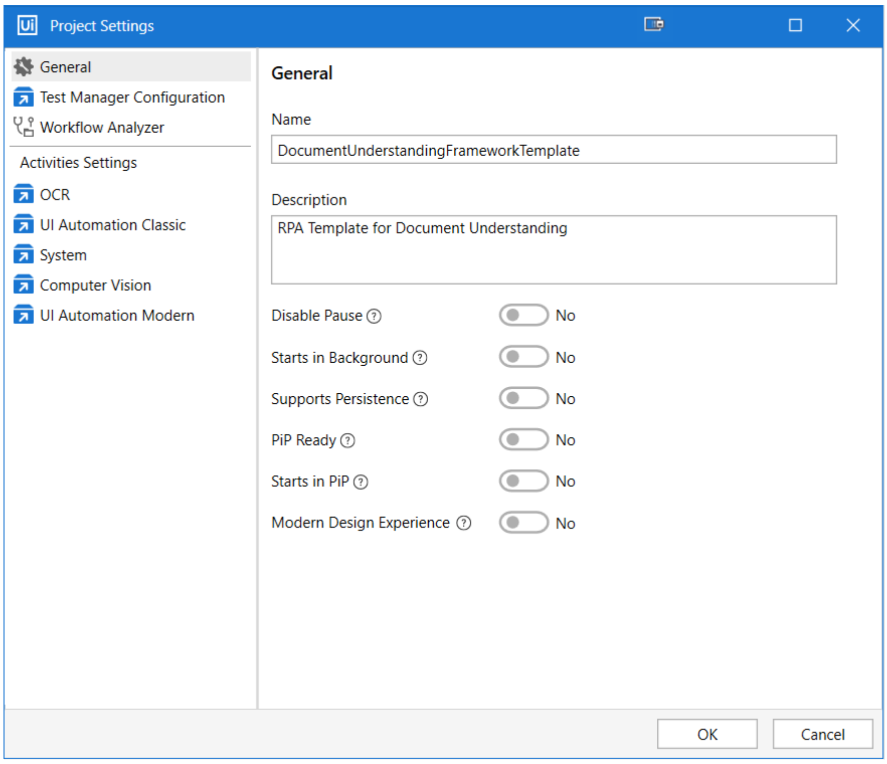
#### Main workflow for Attended Processes

Right-click on **Main-Attended.xaml** and set it as the **Main** workflow for the project.



#### Disable Background Running & Persistence Support

Open the Project Settings and make sure that **Starts in Background** and **Supports Persistence** are set to **No**.



Project Settings

- General
- Test Manager Configuration
- Workflow Analyzer
- Activities Settings
- OCR
- UI Automation Classic
- System
- Computer Vision
- UI Automation Modern

General

Name: DocumentUnderstandingFrameworkTemplate

Description: RPA Template for Document Understanding

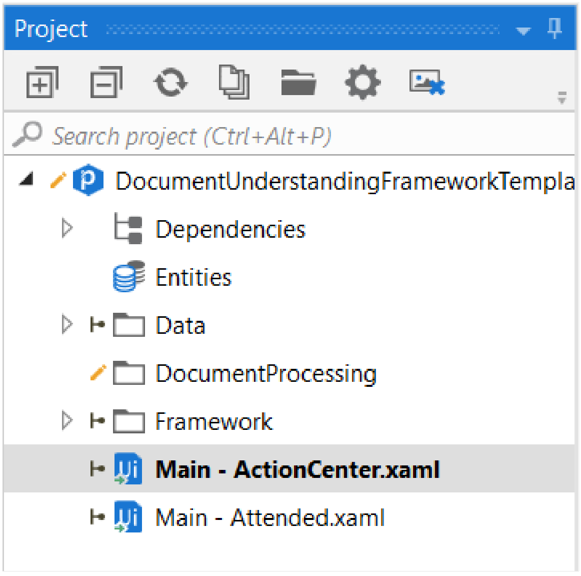
| | |
|---|---|
| Disable Pause | No |
| Starts in Background | No |
| Supports Persistence | No |
| PiP Ready | No |
| Starts in PiP | No |
| Modern Design Experience | No |

OK Cancel

## Settings for Unattended Processes

### Main workflow for Unattended Processes

Right-click on **Main-ActionCenter.xaml** and set it as the **Main** workflow for the project.



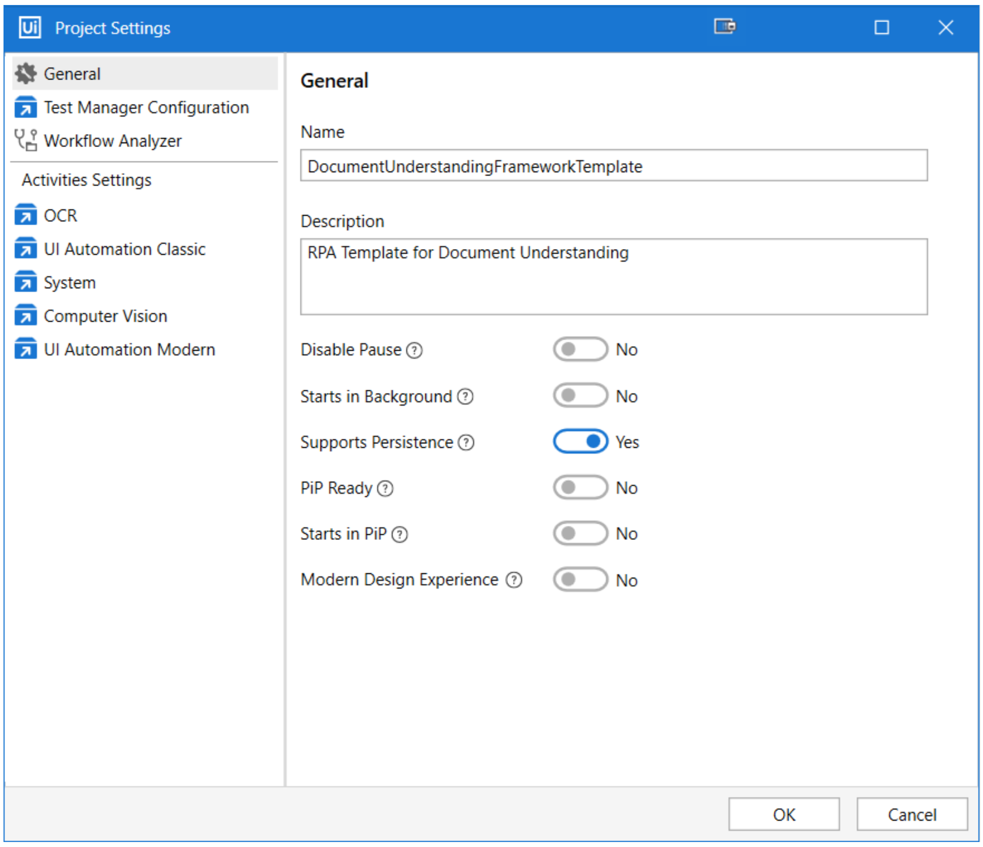
Project

Search project (Ctrl+Alt+P)

- DocumentUnderstandingFrameworkTempla
  - Dependencies
  - Entities
  - Data
  - DocumentProcessing
  - Framework
  - **Main - ActionCenter.xaml**
  - Main - Attended.xaml

### Enable Persistence Support

Open the Project Settings and make sure that **Supports Persistence** is set to **Yes**.



General

Name: DocumentUnderstandingFrameworkTemplate

Description: RPA Template for Document Understanding

Disable Pause: No

Starts in Background: No

Supports Persistence: Yes

PiP Ready: No

Starts in PiP: No

Modern Design Experience: No

**Example Implementation**

Classic/Windows:

To facilitate understanding the Document Understanding Process template, it comes with a pre-implemented example. It showcases the processing of 4 types of documents.

Workflow-specific details can be found in the next chapter.

Cross-platform:

To facilitate understanding the Document Understanding Process template, it comes with a pre-implemented example. It showcases the processing of 3 types of documents.

## Project Files

### Classic/Windows:

### Data\Config.xlsx

Configuration file for project settings. Minimum configuration required:

- Configure your Document Understanding Api Key under an **Orchestrator Asset** named **DocumentUnderstandingApiKey**.

- If using Action Center, configure the **StorageBucketName** under the **Settings** sheet.
- Configure the **DocumentUnderstandingQueueName** if using queues.

### 🗎 Main-ActionCenter.xaml

Workflow to be set and used as Main for attended Document Understanding processes that use Action Center for Human-in-the-Loop.

Arguments:

- in_TargetFile (default **Nothing**): specifies what file should be processed.
- in_UseQueue (default **True**): specifies whether Orchestrator queues are used. If set to True, the value of the in_TargetFile argument is ignored and the file to be processed is fetched from the Transaction Item.

### 🗎 Main-Attended.xaml

Workflow to be set and used as Main for attended Document Understanding processes.

Arguments:

- in_TargetFile (default **Nothing**): specifies what file should be processed.
- in_UseQueue (default **False**): specifies whether Orchestrator queues are used. If set to True, the value of the in_TargetFile argument is ignored and the file to be processed is fetched from the Transaction Item.

### 🗎 Framework\00_ReadConfigFile.xaml

Reads the contents of the Config file into a Config dictionary at runtime.

**Note**: Does not load Orchestrator assets!

No custom code was added here for the purpose of creating the example implementation.

### 🗎 Framework\10_InitializeProcess.xaml

Workflow that loads the Taxonomy and Orchestrator assets. Any process-specific initialization code belongs here.

No custom code was added here for the purpose of creating the example implementation.

### 🗎 Framework\15_GetTransactionItem.xaml

Gets the next Transaction Item when using Orchestrator queues. The target file to be processed is expected to be found under the **TargetFile** key of the Transaction Item's **SpecificContent**.

Also loads all the Transaction Item's **SpecificContent** into the Config dictionary for ease of use.

No custom code was added here for the purpose of creating the example implementation.

### 🗎 Framework\20_Digitize.xaml

Workflow for Pre-Digitization and Digitization logic.

The example implementation uses the **UiPath Document OCR** engine.

### 🗎 Framework\30_Classify.xaml

Workflow for Classification.

The example implementation uses the **Intelligent Keyword Classifier** for all document types. Please know that using the **LearningData** string is also possible.

### 🗎 Framework\35_ClassificationBussinessRuleValidation.xaml

2021.10: Workflow for checking Classification Success. Custom logic is required to determine classification success based on process-specific business rules. A flag can be set to force all documents through manual validation.

By default, all results are sent to human validation.

### 🗎 Framework\40_TrainClassifiers.xaml

Workflow for Classifier training.

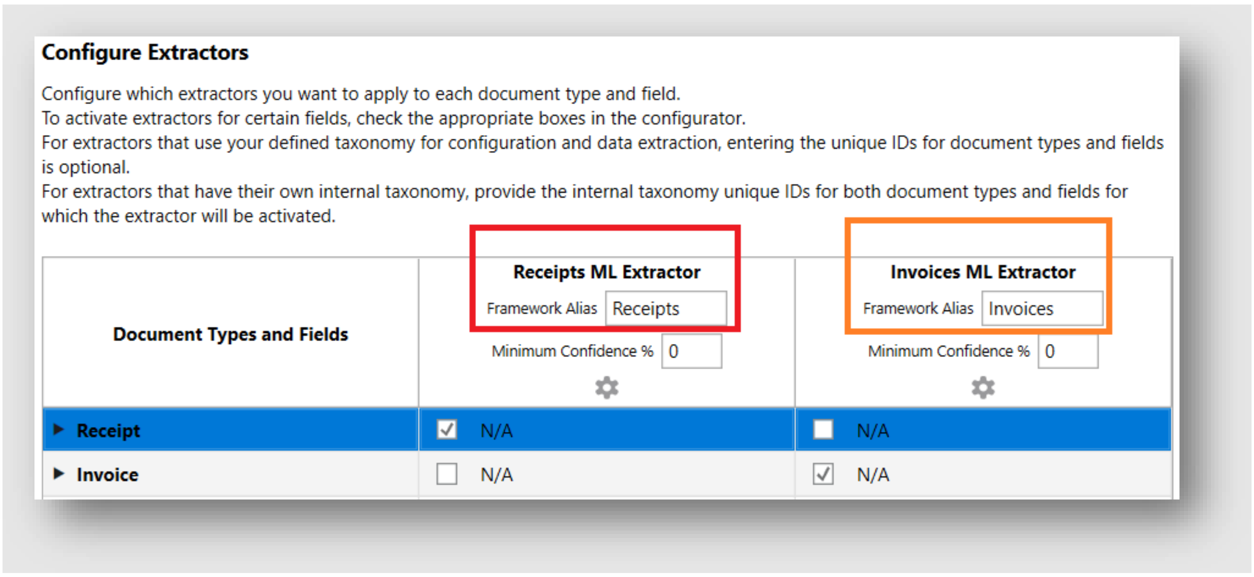
The example implementation uses the **Intelligent Keyword Classifier Trainer** for all document types. It also uses the **LockFile/UnlockFile** reusable workflows in order to regulate access when updating the learning file. Please know that using the **LearningData** string is also possible.

### 🗎 Framework\50_Extract.xaml

Workflow for Pre-Extraction Processing, Data Extraction.

The example implementation uses 4 extractors:

- **Regex Based Extractor** for fixed-form Certificates
- **Intelligent Form Extractor** for fixed-form W9 Forms
- **Machine Learning Extractor** for Receipts – using “Receipts” Framework Alias for ML Extractor training
- **Machine Learning Extractor** for Invoices – using “Invoices” Framework Alias for ML Extractor training


**Configure Extractors**

Configure which extractors you want to apply to each document type and field.
To activate extractors for certain fields, check the appropriate boxes in the configurator.
For extractors that use your defined taxonomy for configuration and data extraction, entering the unique IDs for document types and fields is optional.
For extractors that have their own internal taxonomy, provide the internal taxonomy unique IDs for both document types and fields for which the extractor will be activated.

| Document Types and Fields | Receipts ML Extractor<br>Framework Alias: Receipts<br>Minimum Confidence %: 0 | Invoices ML Extractor<br>Framework Alias: Invoices<br>Minimum Confidence %: 0 |
|---|---|---|
| ▶ Receipt | ✓ N/A | ☐ N/A |
| ▶ Invoice | ☐ N/A | ✓ N/A |

### Framework\55_ExtractionBussinessRuleValidation.xaml

2021.10: Workflow for checking Extraction Success. Custom logic is required to determine extraction success based on process-specific business rules. A flag can be set to force all documents through manual validation.

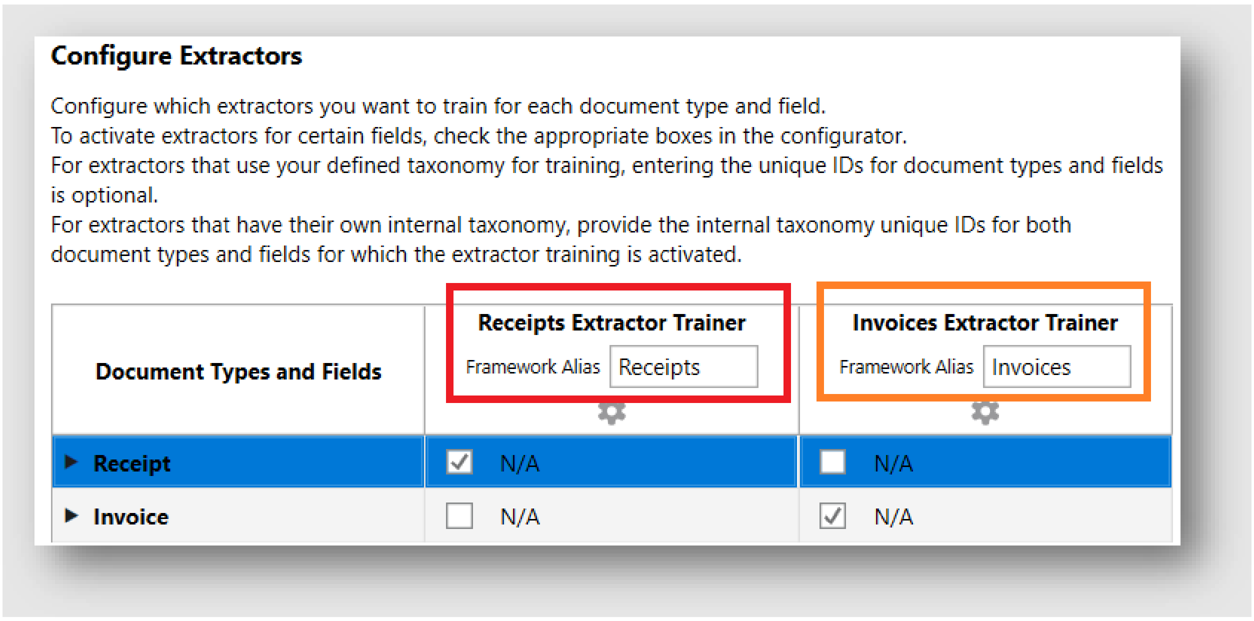
By default, all results are sent to human validation, except for invoices that go through the Invoice Post Processing workflow.

### Framework\60_TrainExtractors.xaml

Workflow for Extractor training.

The example implementation uses 2 extractor trainers:

- **Machine Learning Extractor Trainer** for Receipts – using “Receipts” Framework Alias for ML Extractor training
- **Machine Learning Extractor Trainer** for Invoices – using “Invoices” Framework Alias for ML Extractor training



**Configure Extractors**

Configure which extractors you want to train for each document type and field.
To activate extractors for certain fields, check the appropriate boxes in the configurator.
For extractors that use your defined taxonomy for training, entering the unique IDs for document types and fields is optional.
For extractors that have their own internal taxonomy, provide the internal taxonomy unique IDs for both document types and fields for which the extractor training is activated.

| Document Types and Fields | Receipts Extractor Trainer (Framework Alias: Receipts) | Invoices Extractor Trainer (Framework Alias: Invoices) |
|---|---|---|
| ▶ Receipt | ✓ N/A | ☐ N/A |
| ▶ Invoice | ☐ N/A | ✓ N/A |

## Framework\70_Export.xaml

Workflow for Post-Extraction Processing and Data Export logic.

The example implementation uses a simple mechanic of writing the extracted data to XLSX files. This is only one of many export possibilities besides using Data Service, a Database, using CSV, JSON format, etc.

## Framework\80_EndProcess.xaml

Workflow for Post-Export Processing and Process Cleanup logic.

No custom code was added here for the purpose of creating the example implementation.

## Framework\ReusableWorkflows\InvoicePostProcessing.xaml

2021.10: This workflow implements invoice post-processing using the default OOTB model. The ruleset to check against is defined in a dedicated sheet of the Config file.

The ruleset and the logic should be updated as required by the business process.

Default post-processing rules & steps:

- Verify that all Mandatory Fields and Columns are extracted
- Verify that all table rows match the rule: *Quantity * Unit Price = Line Amount*
- Verify that the sum of all *Line Amounts* = *Net Amount*
- Sum up Net Amount with all the values defined as Config "SubTotalAdditions". Verify that the sum of *Net Amount* + "SubTotalAdditions" = *Total*
- Verify the extraction confidence of all defined "ConfidenceFields" against their individual confidence thresholds

- Verify the extraction confidence of all the other fields against the "other-Confidence" threshold

This should NOT be used as-is, except for demo purposes. For a real implementation, the post-processing & validation should be tailored to the specifics of the business process.

The default implementation uses EN-US culture information. This means '.' is the decimal separator and "," is the thousand separators. (e.g.: 10,000.00)

### 🗎 Framework\ReusableWorkflows\ReceiptPostProcessing.xaml

2021.10: This workflow implements receipt post-processing using the default OOTB model. The ruleset to check against is defined in a dedicated sheet of the Config file.

The ruleset and the logic should be updated as required by the business process.

Default post-processing rules & steps:

- Verify that all Mandatory Fields and Columns are extracted
- Verify that the sum of all *Line Amounts* = *Net Amount*
- Sum up Net Amount with all the values defined as Config "SubTotalAdditions". Verify that the sum of *Net Amount* + "SubTotalAdditions" = *Total*
- Verify the extraction confidence of all defined "ConfidenceFields" against their individual confidence thresholds
- Verify the extraction confidence of all the other fields against the "other-Confidence" threshold

This should NOT be used as-is, except for demo purposes. For a real implementation, the post-processing & validation should be tailored to the specifics of the business process.

The default implementation uses EN-US culture information. This means '.' is the decimal separator and "," is the thousand separators. (e.g.: 10,000.00)

### 🗎 Framework\ReusableWorkflows\AutocorrectOcrMistakes.xaml

This workflow implements OCR auto correction by saving corrections made in Human in the Loop. This feature is meant to be used for fields where the extracted value is expected to be part of a limited set of options that do not have a high level of similarity between them.

Examples of fields that are suitable for the OCR auto correction – Logo, Company Name, Document Issuer Name.

Examples of fields that are not suitable for the OCR auto correction – Amount, Date, PO Number, First Name, Last Name.

The workflow is not enabled by default and the following settings found in the "AutocorrectOcrMistakes" sheet in the config file need to be customized in order to use it:

- AutocorrectionEnabled – boolean flag that enables/disables the flow
- AutocorrectionFieldName – the name of the field to be corrected

As this feature uses changes done in Human in the Loop, adding blacklisted OCR values or performing maintenance may be required. This can be done by modifying the learning file saved in the DU storage bucket (check StorageBucketName and AutocorrectStorageBucketDirectoryPath in the config file).

### Framework\ERR_HandleDocumentError.xaml

Workflow that is executed when an exception occurs when processing a classified document.

No custom code was added here for the purpose of creating the example implementation.

### Framework\ERR_AbortProcess.xaml

Workflow that is executed if the process is aborted due to a terminating exception. Code for error cleanup or for sending error notifications belongs here.

No custom code was added here for the purpose of creating the example implementation.

### Framework\ReusableWorkflows\GetWritePermission.xaml

Helper workflow using a Queue with a single queue item as a semaphore mechanism to regulate write access to classifier training files.

### Framework\ReusableWorkflows\GiveUpWritePermission.xaml

Helper workflow using a Queue with a single queue item as a semaphore mechanism to regulate write access to classifier training files.

### Framework\ReusableWorkflows\LockFile.xaml

Helper workflow using simple lock/unlock mechanism to regulate write access to classifier training files.

**Framework\ReusableWorkflows\UnlockFile.xaml**

Helper workflow using simple lock/unlock mechanism to regulate write access to classifier training files.

**Framework\ReusableWorkflows\SetTransactionProgress.xaml**

Updates the **TransactionProgress** for the Transaction Item that is being processed (if using queues).

**Framework\ReusableWorkflows\SetTransactionStatus.xaml**

Sets and log the transaction's status. The approach is like the one used by the RE-Framework.

**QuickStart\IntializeOrchestrator\IntializeOrchestrator.xaml**

Workflow that can be used for quickly setting up the necessary Orchestrator resources (Assets, Queues, Storage Buckets, Task Catalogs).

**Cross-platform:**

**Data\Config.json**

Configuration file for project settings. Minimum configuration required:

- Configure your Document Understanding Api Key under an **Orchestrator Asset** named **DocumentUnderstandingApiKey**.
- Create the cross-platform Orchestrator folder and the Storage Bucket.
- Configure the **StorageBucketName** under the **Settings** section.
- Configure the **DocumentUnderstandingInputQueueName** and **DocumentUnderstandingOutputQueueName** under the **Settings** section.

**Main-ActionCenter.xaml**

Workflow to be set and used as Main for attended Document Understanding processes that use Action Center for Human-in-the-Loop.

Arguments:

- in_TargetFile (default **"Data\ExampleDocuments\Invoice.pdf"**): specifies what file should be processed.
- In_TargetFileResource (default **Nothing**): the resources (ILocalResource) of the file to be processed.

- in_UseQueue (default **True**): specifies whether Orchestrator queues are used. If set to True, the value of the in_TargetFile argument is ignored and the file to be processed is fetched from the Transaction Item.

Note: If we run on Linux the in_TargetFile path separator must be changed to **"Data/ExampleDocuments/Invoice.pdf"**

## 🖹 Framework\00_ReadConfigFile.xaml

Reads the contents of the Config file into a Config dictionary at runtime.

**Note**: Does not load Orchestrator assets!

No custom code was added here for the purpose of creating the example implementation.

## 🖹 Framework\10_InitializeProcess.xaml

Workflow that loads the Orchestrator assets. Any process-specific initialization code belongs here.

No custom code was added here for the purpose of creating the example implementation.

## 🖹 Framework\15_GetTransactionItem.xaml

Gets the next Transaction Item from the Orchestrator queue. The target file to be processed is expected to be found under the **TargetFile** key of the Transaction Item's **SpecificContent**.

Also loads all the Transaction Item's **SpecificContent** into the Config dictionary for ease of use.

No custom code was added here for the purpose of creating the example implementation.

## 🖹 Framework\30_Classify.xaml

Workflow for Classification.

The example implementation uses the **Predefined** ML Classification classifier for all document types.

## 🖹 Framework\35_ClassificationBussinessRuleValidation.xaml

Workflow for checking Classification Success. Custom logic is required to determine classification success based on process-specific business rules. A flag can be set to force all documents through manual validation.

By default, all results are sent to human validation.

### Framework\50_Extract.xaml

Workflow for Pre-Extraction Processing, Data Extraction.

The example implementation uses 3 extractors:

- The **Predefined Machine Learning** extractor for Invoices.
- The **Predefined Machine Learning** extractor for Receipts.
- The **Predefined Machine Learning** extractor for Certificate of incorporation.

### Framework\55_ExtractionBussinessRuleValidation.xaml

Workflow for checking Extraction Success. Custom logic is required to determine extraction success based on process-specific business rules. A flag can be set to force all documents through manual validation.

By default, all results are sent to human validation, except for invoices that go through the Invoice Post Processing workflow.

### Framework\70_Export.xaml

Workflow for Post-Extraction Processing and Data Export logic.

The example implementation uses a simple mechanic of serializing the extracted data and writing it to a JSON file. This is only one of many export possibilities besides using Data Service, a Database, using CSV, XLSX format, etc.

### Framework\80_EndProcess.xaml

Workflow for Post-Export Processing and Process Cleanup logic.

No custom code was added here for the purpose of creating the example implementation.

### Framework\ReusableWorkflows\InvoicePostProcessing.xaml

This workflow implements invoice post-processing using the Predefined model. The ruleset to check against is defined in a dedicated section of the Config file.

The ruleset and the logic should be updated as required by the business process.

Default post-processing rules & steps:

- Verify that all Mandatory Fields and Columns are extracted
- Verify that all table rows match the rule: *Quantity * Unit Price = Line Amount*
- Verify that the sum of all *Line Amounts* = *Net Amount*

- Sum up Net Amount with all the values defined as Config "SubTotalAdditions". Verify that the sum of *Net Amount* + "SubTotalAdditions" = *Total*
- Verify the extraction confidence of all defined "ConfidenceFields" against their individual confidence thresholds
- Verify the extraction confidence of all the other fields against the "other-Confidence" threshold

This should NOT be used as-is, except for demo purposes. For a real implementation, the post-processing & validation should be tailored to the specifics of the business process.

The default implementation uses EN-US culture information. This means '.' is the decimal separator and "," is the thousand separator. (e.g.: 10,000.00)

### Framework\ReusableWorkflows\ReceiptPostProcessing.xaml

This workflow implements receipt post-processing using the Predefined model. The ruleset to check against is defined in a dedicated section of the Config file.

The ruleset and the logic should be updated as required by the business process.

Default post-processing rules & steps:

- Verify that all Mandatory Fields and Columns are extracted
- Verify that the sum of all *Line Amounts* = *Net Amount*
- Sum up Net Amount with all the values defined as Config "SubTotalAdditions". Verify that the sum of *Net Amount* + "SubTotalAdditions" = *Total*
- Verify the extraction confidence of all defined "ConfidenceFields" against their individual confidence thresholds
- Verify the extraction confidence of all the other fields against the "other-Confidence" threshold

This should NOT be used as-is, except for demo purposes. For a real implementation, the post-processing & validation should be tailored to the specifics of the business process.

The default implementation uses EN-US culture information. This means '.' is the decimal separator and "," is the thousand separator. (e.g.: 10,000.00)

### Framework\ReusableWorkflows\AutocorrectOcrMistakes.xaml

This workflow implements OCR auto correction by saving repetitive corrections made in Human in the Loop for certain types of fields. This feature is meant to be used to improve automation when handling documents that have fields where the OCR is expected to have trouble giving clear and consistent results due to the nature of the documents or their quality.

It is meant to only be used for fields where the extracted value is expected to be part of a limited set of options that do not have a high level of similarity between them.

Examples of fields suitable for the OCR auto correction: Logo, Company Name, Document Issuer Name.

Examples of fields that are not suitable for the OCR auto correction: Amount, Date, PO Number, First Name, Last Name.

The feature is not enabled by default and the following settings found in the “AutocorrectOcrMistakes” sheet in the config file need to be customized in order to use it:

- AutocorrectionEnabled – boolean flag that enables/disables the flow
- AutocorrectionFieldIdList – the taxonomy ids of the fields to be corrected
- AutocorrectStorageBucketDirectoryPath – path inside the storage bucket where the learning file is uploaded

Besides that, it is necessary to upload the AutocorrectionLearningFile.json file found under “Data\ExampleDocuments\” to your DU storage bucket (StorageBucketName in the settings tab of the config).

As this feature uses changes done in Human in the Loop, adding blacklisted OCR values or performing maintenance may be required. This can be done by modifying the learning file saved in the DU storage bucket.

### Framework\ERR_HandleDocumentError.xaml

Workflow that is executed when an exception occurs when processing a classified document.

No custom code was added here for the purpose of creating the example implementation.

### Framework\ERR_AbortProcess.xaml

Workflow that is executed if the process is aborted due to a terminating exception. Code for error cleanup or for sending error notifications belongs here.

No custom code was added here for the purpose of creating the example implementation.

### Framework\ReusableWorkflows\SetTransactionProgress.xaml

Updates the **TransactionProgress** for the Transaction Item that is being processed.

**Framework\ReusableWorkflows\SetTransactionStatus.xaml**

Sets and log the transaction's status. The approach is like the one used by the RE-Framework.

## Quick Start Guide

### Classic/Windows:

Please see *Document Understanding and Queues* for the required Orchestrator version of using queues.

#### Orchestrator Configuration

- Configure your Document Understanding Api Key under an **Orchestrator Asset** named **DocumentUnderstandingApiKey**.
- If using Action Center, create a **Storage Bucket** for your process.
- If needed, create a **Queue** for your process.

Fast batch creation of items in Orchestrator can be done by running the workflow **IntializeOrchestrator.xml** placed in **QuickStart\IntializeOrchestrator**.

The process is using the **IntializeOrchestratorConfig.xslx** part by loading an excel document in the assigned section. This can be found in **QuickStart\IntializeOrchestrator\Data**.

The Process can create the following 4 item types in orchestrator:

1. Storage Bucket
2. Queue
3. Asset
4. Task Catalog

The parameters to be filled in are the following:

1. Name: Displayed name on Orchestrator
2. Value: Value that the specific item will have, if needed
3. Type: One of four listed above
4. Folder: Folder location where the item must be create in Orchestrator

Description: item description

#### Attended Automation

- Configure your Document Understanding Api Key under an **Orchestrator Asset** named **DocumentUnderstandingApiKey**.
- If using queues, configure the **DocumentUnderstandingQueueName**.
- Configure the **Settings for Attended Processes**.

**Note**: We recommend becoming familiar with the solution and the project before removing the example implementation and the taxonomy.

### Unattended Automation

- Configure your Document Understanding Api Key under an **Orchestrator Asset** named **DocumentUnderstandingApiKey**.
- If using Action Center, configure the **StorageBucketName** under the **Settings** sheet,
- If using queues, configure the **DocumentUnderstandingQueueName**.
- Configure the **Settings for Unattended Processes**.

**Note**: When developing, it is quite simple to test your implementation in Attended mode. Simply run **Main – Attended.xaml** in order to use the local validation activities instead of going to Action Center (it is **not** needed to change process settings while testing this way).

**Note**: We recommend becoming familiar with the solution and the project before removing the example implementation and the taxonomy.

### Dispatcher Mechanisms for Unattended Implementations

Please see **Starting Document Understanding Jobs**.

## Cross-platform:

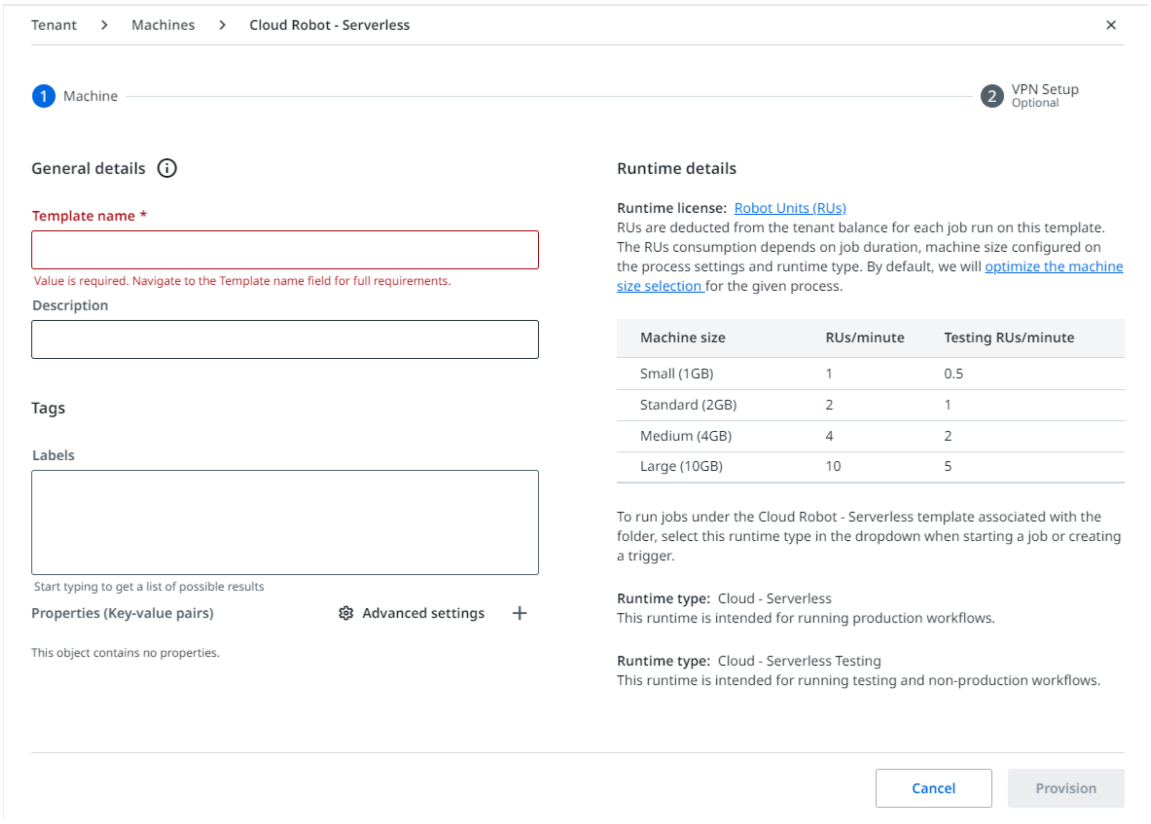
Please see *Document Understanding and Queues* for the required Orchestrator version of using queues.

### Orchestrator Configuration

- Create the cross-platform Orchestrator folder and the Storage Bucket. Default name for both is *Cross-platform*
- Create the **StorageBucketName**.
- Create the **DocumentUnderstandingInputQueueName** and **DocumentUnderstandingOutputQueueName**.

### Create ServerLess Machine and Run Automation

- Go to Tenant -> Machines -> Add machine and choose **Cloud Robot – Serverless**
- Give a name to the machine and click **Provision**


Tenant > Machines > Cloud Robot - Serverless

1 Machine — 2 VPN Setup Optional

General details

Template name *

Value is required. Navigate to the Template name field for full requirements.

Description

Tags

Labels

Start typing to get a list of possible results

Properties (Key-value pairs) Advanced settings +

This object contains no properties.

Runtime details

Runtime license: Robot Units (RUs)
RUs are deducted from the tenant balance for each job run on this template. The RUs consumption depends on job duration, machine size configured on the process settings and runtime type. By default, we will optimize the machine size selection for the given process.

| Machine size | RUs/minute | Testing RUs/minute |
|---|---|---|
| Small (1GB) | 1 | 0.5 |
| Standard (2GB) | 2 | 1 |
| Medium (4GB) | 4 | 2 |
| Large (10GB) | 10 | 5 |

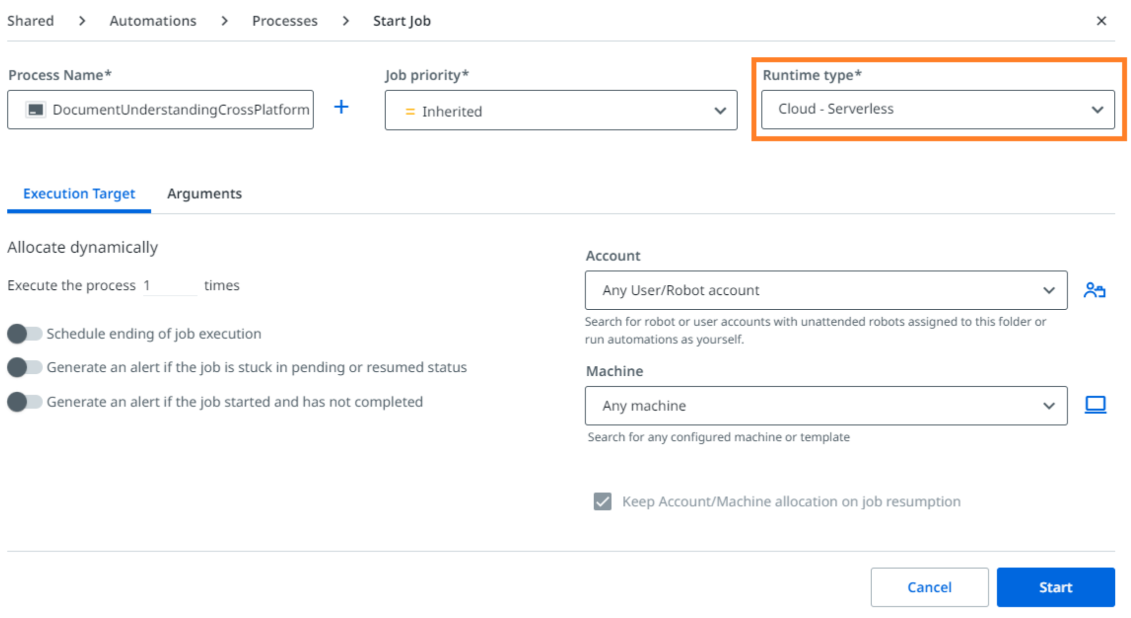
To run jobs under the Cloud Robot - Serverless template associated with the folder, select this runtime type in the dropdown when starting a job or creating a trigger.

Runtime type: Cloud - Serverless
This runtime is intended for running production workflows.

Runtime type: Cloud - Serverless Testing
This runtime is intended for running testing and non-production workflows.

Cancel Provision

- When running the automation choose **Cloud-Serverless** as **Runtime type**


Shared > Automations > Processes > Start Job

Process Name*: DocumentUnderstandingCrossPlatform
Job priority*: Inherited
Runtime type*: Cloud - Serverless

Execution Target | Arguments

Allocate dynamically

Execute the process 1 times

Schedule ending of job execution
Generate an alert if the job is stuck in pending or resumed status
Generate an alert if the job started and has not completed

Account: Any User/Robot account
Search for robot or user accounts with unattended robots assigned to this folder or run automations as yourself.

Machine: Any machine
Search for any configured machine or template

Keep Account/Machine allocation on job resumption

Cancel Start

## Unattended Automation

- Configure your Document Understanding Api Key under an **Orchestrator Asset** named **DocumentUnderstandingApiKey**.
- Configure the **StorageBucketName** under the **Settings** section.

- Configure the **DocumentUnderstandingInputQueueName** and **DocumentUnderstandingOutputQueueName** under the **Settings** section.
- Configure the **Settings for Unattended Processes**.

### Dispatcher Mechanisms for Unattended Implementations

Please see **Starting Document Understanding Jobs**.

## Known Issues and Limitations

- When using the default **LockFile**/**UnlockFile** implementation, there is a small chance that some training data will occasionally be lost – when multiple bots update the training file at the exact same time, the last one will overwrite others' training data. In practice, this does not pose any problems.
- The Document Understanding Process is currently **not** performing any cleanup of its Temp folder (where it stores split or downloaded files).
- By design, the job ends successfully even if the processing of all classified documents ends with exceptions.
- If you are updating to **IntelligentOCR 6.5.0** from an older version, you will have to add to the workflow SetTransactionStatus a new namespace manually: "UiPath.IntelligentOCR.Exceptions". 6.5.0 Introduced this breaking change but creating a new template of 22.10 or later would have this solved.