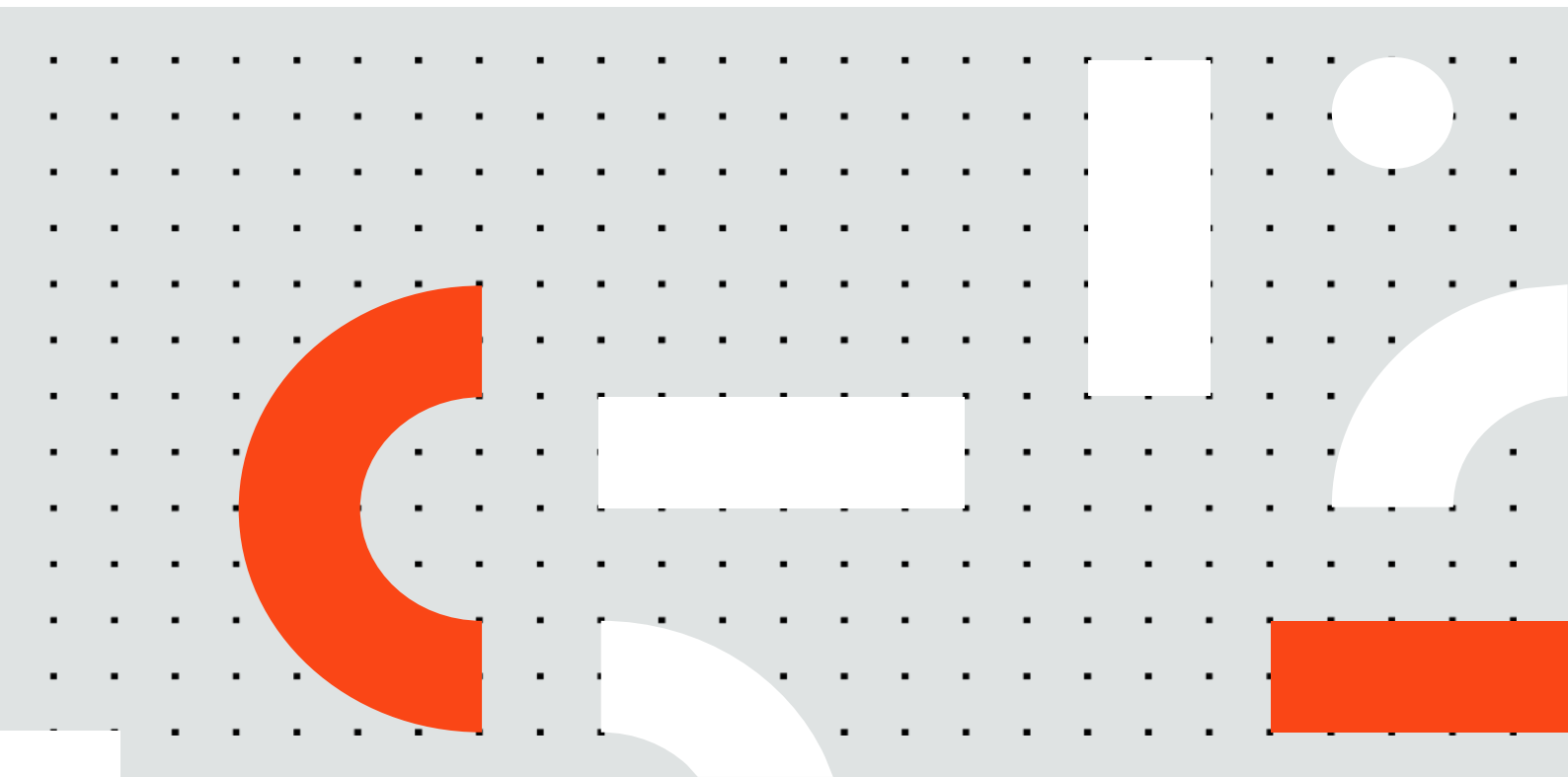



















Document Understanding Process

Studio Template



Contents

Revision History.....	4
Overview.....	5
Key Features	5
Generic Document Processing Flow.....	6
Solution Architecture	8
Document Understanding, Part of End-to-End Business Processes	8
The One-Job-Per-File Approach.....	9
Document Understanding and Queues.....	9
Orchestrator v20.10.8 or Newer	9
Orchestrator Versions Prior to 20.10.8	10
Starting Document Understanding Jobs.....	10
Studio Project Overview.....	12
Settings for Attended Processes.....	12
Settings for Unattended Processes	13
Example Implementation	14
Project Files	14
📄 Data\Config.xlsx	14
📄 Main-ActionCenter.xaml	14
📄 Main-Attended.xaml	14
📄 Framework\00_ReadConfigFile.xlsx	14
📄 Framework\10_InitializeProcess.xaml	15
📄 Framework\15_GetTransactionItem.xaml	15
📄 Framework\20_Digitize.xaml	15
📄 Framework\30_Classify.xaml	15
📄 Framework\40_TrainClassifiers.xaml	15
📄 Framework\50_Extract.xaml	16

 Framework\60_TrainExtractors.xaml	16
 Framework\70_Export.xaml.....	17
 Framework\80_EndProcess.xaml.....	17
 Framework\ERR_HandleDocumentError.xaml.....	17
 Framework\ERR_AbortProcess.xaml	17
 Framework\ReusableWorkflows\GetWritePermission.xaml	18
 Framework\ReusableWorkflows\GiveUpWritePermission.xaml.....	18
 Framework\ReusableWorkflows\LockFile.xaml	18
 Framework\ReusableWorkflows\UnlockFile.xaml	18
 Framework\ReusableWorkflows\SetTransactionProgress.xaml.....	18
 Framework\ReusableWorkflows\SetTransactionStatus.xaml.....	18
Quick Start Guide	19
 Orchestrator Configuration.....	19
 Attended Automation	19
 Unattended Automation.....	19
 Dispatcher Mechanisms for Unattended Implementations	19
Known Issues and Limitations	20

Revision History

Date	Changes
Jun-2021	V1.0

Overview

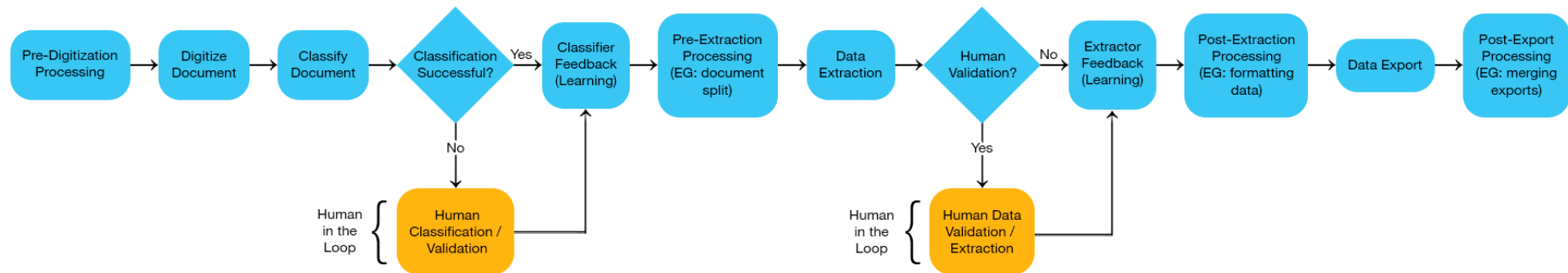
As the number of Document Understanding engagements keeps increasing in complexity, the need for a common implementation/RPA approach becomes evident. Document Understanding processes have a particular logical flow and requirements that are better suited for a dedicated approach of their own, instead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

Processing flow for each valid document



Keep in mind that the diagram above shows that 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

A digital version of the document is obtained. In case the document is scanned, OCR is required.

▶ Classify Document

The document is classified.

▶ 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)

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

▶ Pre-Extraction Processing

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

▶ Data Extraction

Extract the data using the appropriate extractor(s).

▶ 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)

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.

▶ 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

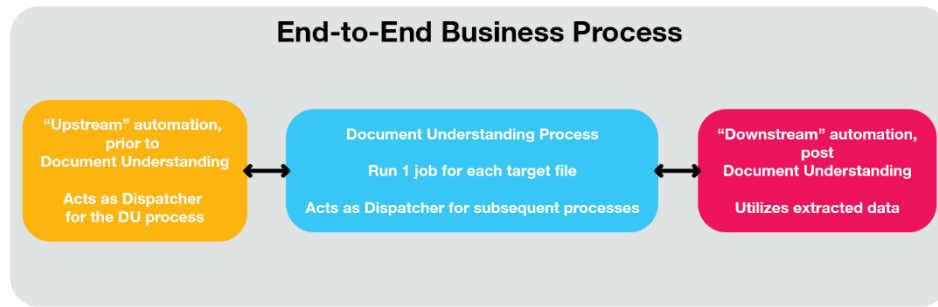
Document Understanding, Part of End-to-End Business Processes

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

Benefits:

- Prevents external issues from impacting DU 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:



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

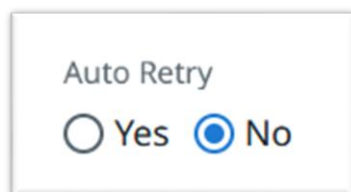
Document Understanding processes will not run as batch jobs. Instead, an individual job starts for each file to be processed. This approach is used for both Attended and Unattended implementations.

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



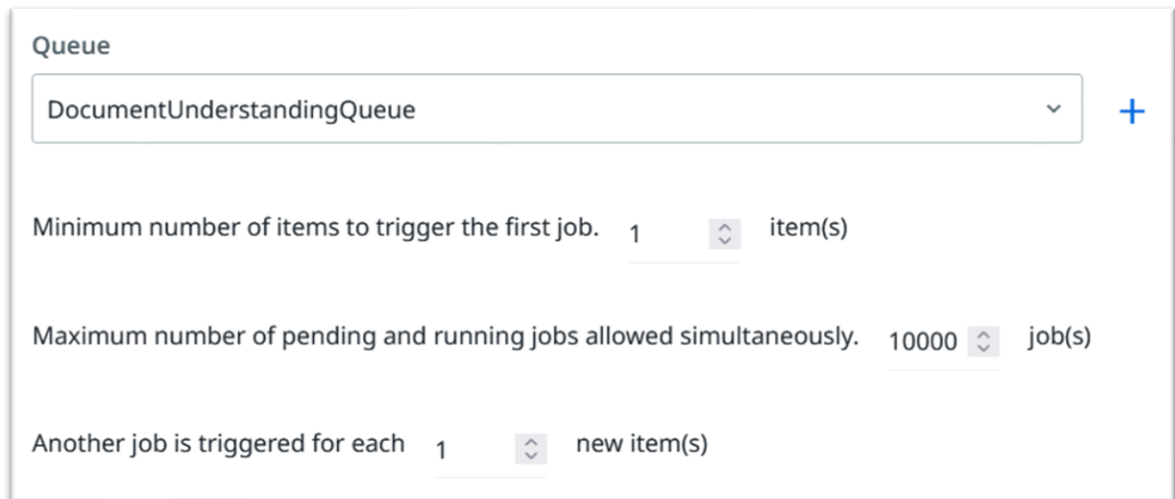
Orchestrator Versions Prior to 20.10.8

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**.

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

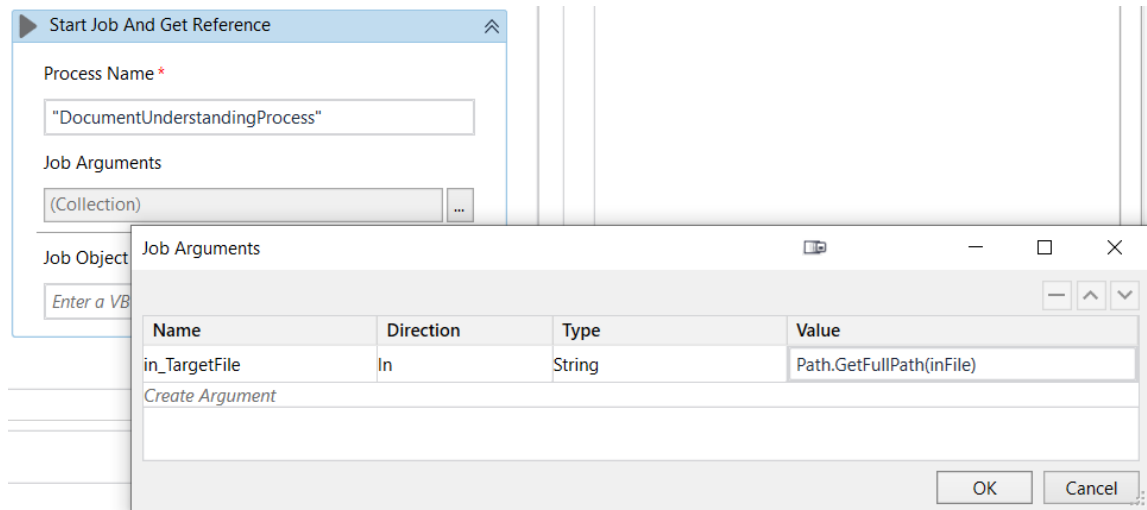


The image shows a configuration window for a Queue Trigger. At the top, there is a section labeled "Queue" with a dropdown menu set to "DocumentUnderstandingQueue" and a blue "+" icon to its right. Below this, there are three settings, each with a label, a numeric input field, a spinner icon, and a unit label:

- "Minimum number of items to trigger the first job." with a value of "1" and unit "item(s)".
- "Maximum number of pending and running jobs allowed simultaneously." with a value of "10000" and unit "job(s)".
- "Another job is triggered for each" with a value of "1" and unit "new item(s)".

Note: Due to the current Job Count Strategy for triggers, some queue items experience delays before a processing job is triggered. This behavior is on the roadmap to be addressed.

2. Having the Dispatcher process use a Start Job activity



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.

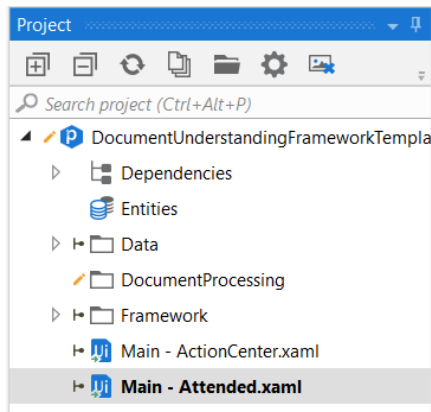
Studio Project Overview

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

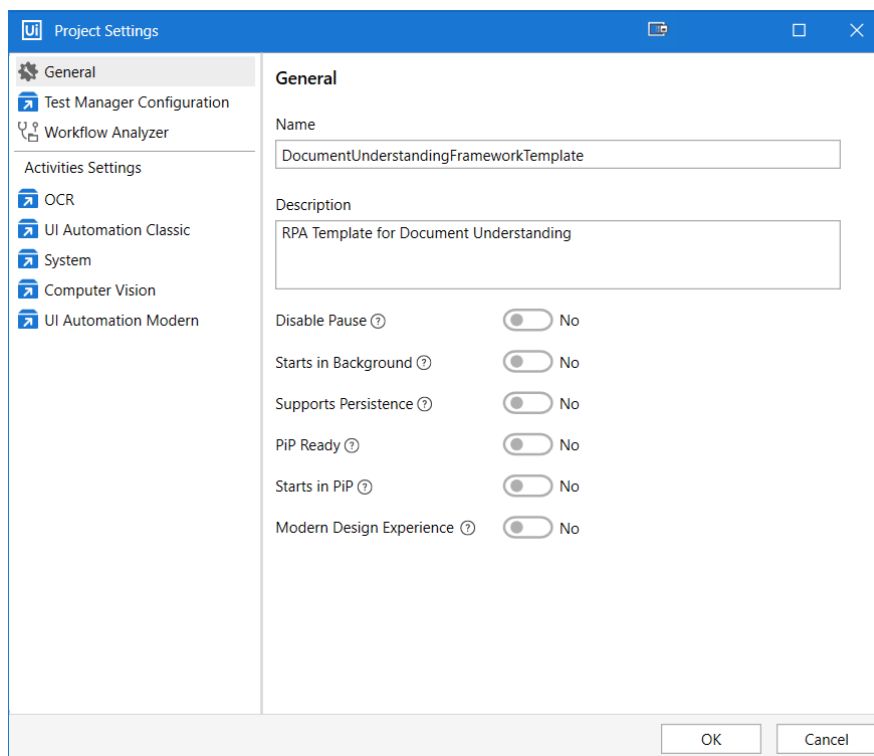
Main workflow for Attended Processes

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



Disable Background Running & Persistence Support

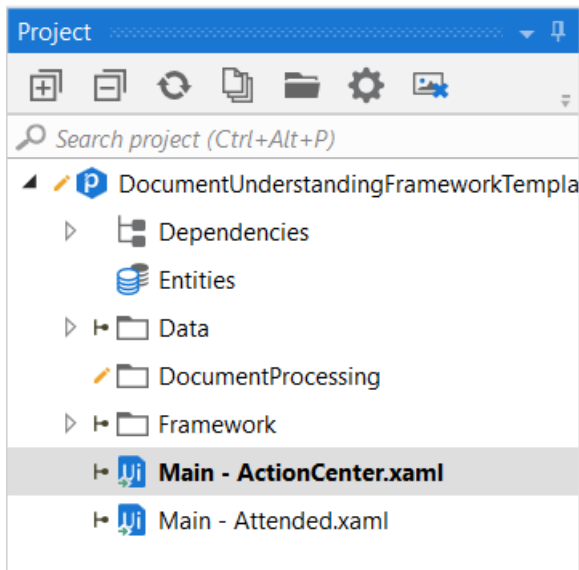
Open the Project Settings and make sure that **Starts in Background** and **Supports Persistence** are set to **No**.



Settings for Unattended Processes

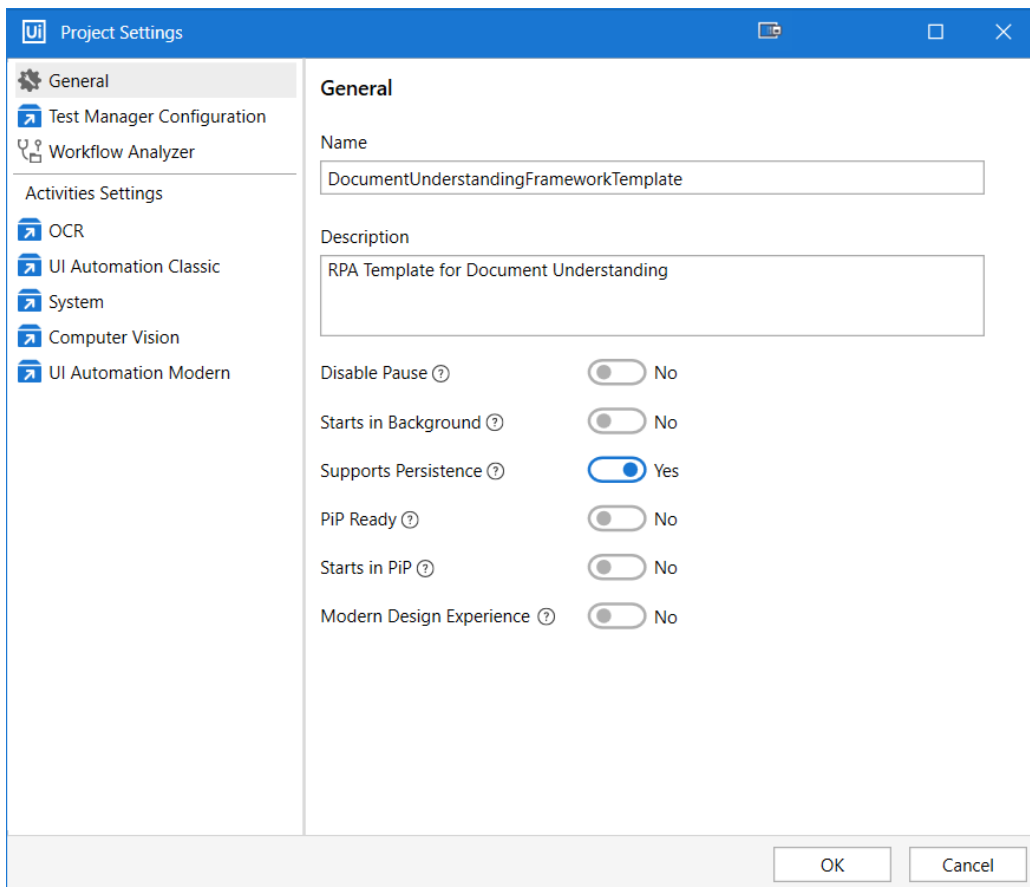
Main workflow for Unattended Processes

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



Enable Persistence Support

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



Example Implementation

In order 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.

Project Files

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 **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.

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 **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.

Framework\00_ReadConfigFile.xlsx

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 and for checking Classification Success. Custom logic is required to determine classification success.

By default, all results are sent to human validation.

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

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 and for checking if Data Validation is needed. Custom logic is required to determine extraction success.

By default, all results are sent to human validation.

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 Framework Alias: Receipts Minimum Confidence %: 0	Invoices ML Extractor Framework Alias: Invoices Minimum Confidence %: 0
► Receipt	<input checked="" type="checkbox"/> N/A	<input type="checkbox"/> N/A
► Invoice	<input type="checkbox"/> N/A	<input checked="" type="checkbox"/> N/A

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	Invoices Extractor Trainer
	Framework Alias <input type="text" value="Receipts"/>	Framework Alias <input type="text" value="Invoices"/>
► Receipt	<input checked="" type="checkbox"/> N/A	<input type="checkbox"/> N/A
► Invoice	<input type="checkbox"/> N/A	<input checked="" type="checkbox"/> 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\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.

Quick Start Guide

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.

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 very 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](#).

Known Issues and Limitations

- Due to the current Job Count Strategy for triggers, some queue items experience delays before a processing job is triggered. This behavior is on the roadmap to be addressed.
- 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.
- The Document Understanding Process is currently **not** performing any cleanup of its Temp folder (where it stores split or downloaded files).
- The file splitting mechanism only works for PDF files. When processing multi-paged TIFF files, file splitting must be turned off.
- There are some errors and warnings issued by the Project Analyzer tool. We are working on having this resolved.
- The job ends successfully even if the processing of all classified documents ends with exceptions.