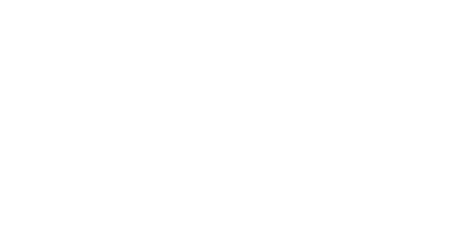
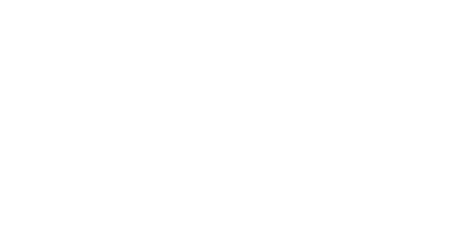
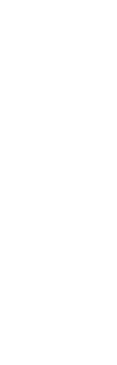
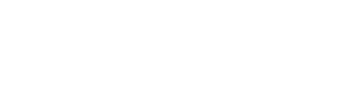
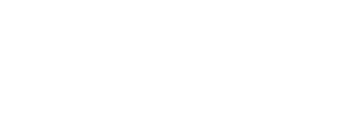
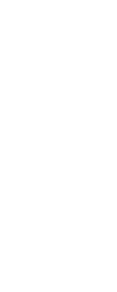
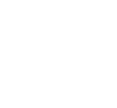
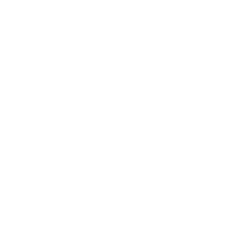
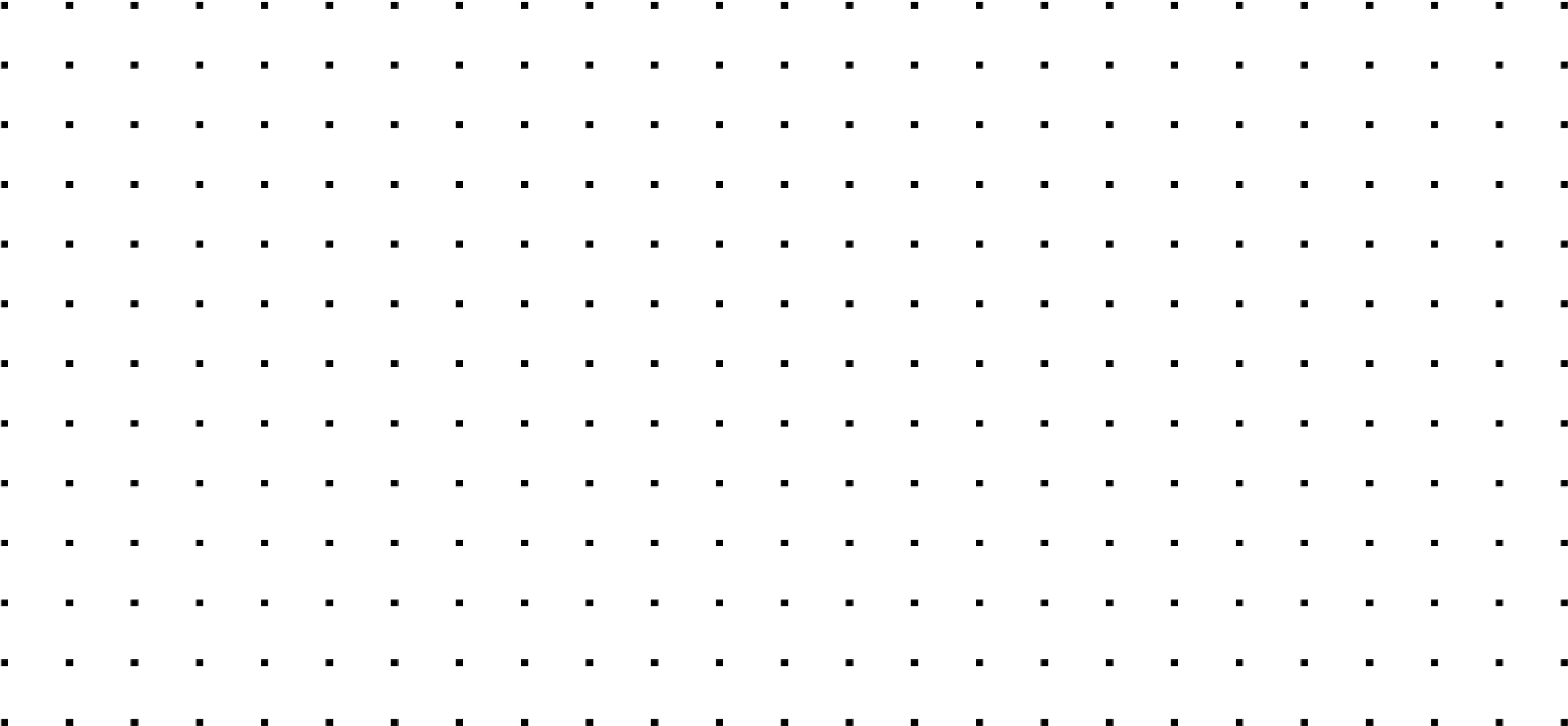


**Human Assisted Resolution Process**



Studio Template

[Revision History 4](#_Toc1857122105)

[Release Notes: 5](#_Toc2056179926)

[Overview 6](#_Toc1446144689)

[Key Features 6](#_Toc1632782576)

[Generic Human In The Loop Flow 7](#_Toc214789939)

[Solution Architecture 9](#_Toc1850171908)

[HART, Part of End-to-End Business Processes 9](#_Toc388564452)

[Studio Project Overview 10](#_Toc1802620236)

[Settings for Unattended Processes 10](#_Toc1745146954)

[Example Implementation 10](#_Toc158513543)

[Project Files 11](#_Toc2131747913)

[📄 Data\Config.xlsx 11](#_Toc1283392848)

[📄 Main.xaml 11](#_Toc1703232656)

[📄 InitializeOrchestrator.xaml 11](#_Toc1703232656)

[📄 Framework\00\_ReadConfigFile.xaml 11](#_Toc1481765943)

[📄 Framework\10\_InitializeProcess.xaml 11](#_Toc493074592)

[📄 Framework\20\_GetTransactionItem.xaml 11](#_Toc9987047)

[📄 Framework\30\_SelectTemplate.xaml 11](#_Toc696201077)

[📄 Framework\40\_DataPostProcessing.xaml 11](#_Toc1102430081)

[📄 Framework\50\_EndProcess.xaml 11](#_Toc1533318372)

[📄 Framework\ERR\_AbortProcess.xaml 11](#_Toc763314117)

[📄 Framework\ReusableWorkflows\GenerateId.xaml 11](#_Toc1717872582)

[📄 Framework\ReusableWorkflows\GenerateKey.xaml 11](#_Toc1185130109)

[📄 Framework\ReusableWorkflows\OrchestratorRequest\_GET.xaml 11](#_Toc847423109)

[📄 Framework\ReusableWorkflows\OrchestratorRequest\_POST.xaml 12](#_Toc47346288)

[📄 Framework\ReusableWorkflows\SetTransactionStatus.xaml 12](#_Toc1658176543)

[Quick Start Guide 13](#_Toc1984736008)

[🔧Orchestrator Configuration 13](#_Toc1197827997)

[🔧 Unattended Automation 13](#_Toc993644251)

# Revision History

|  |  |
| --- | --- |
| **Date** | **Changes** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Release Notes:

# Overview

When a workflow step stops running, regardless of the reason, it should be recovered from its error condition. Some projects may have as requirement the manual error handling or any human intervention, therefore, Action Center tasks, corresponding to each type of error, need to be created.  To have an optimized and scalable version of this, it would be more suitable to have a separate error handling/human intervention framework.

## **Key Features**

* Everything can be generated automatically on the back end (including assets, storage buckets and task catalogs)
* Simple process, usable in all cases, from small processes to complex solutions
* Easy to integrate into larger automation flows
* Production-ready: has built-in logging, exception handling and retry mechanisms
* Follows the best practices pertaining to RPA, Document Understanding, Orchestration Processes, and Long-running workflows.
* Based on the Document Understanding Framework - Meant to make development, deployment and debugging much easier
* HART processes will not run as batch jobs. Instead, an individual job should be started for each file to be processed. For this reason, it is much easier to search in Orchestrator for an individual job or to perform debugging.

# Generic Human In The Loop Flow

Diagram

Description automatically generated

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.

🚩 Template Selection

All templates are saved in the Data\Templates folder. The selection is based on the exception type and process name.

Text

Description automatically generated with medium confidence

🚩 Task Creation

Creation of the Action Center Task and waiting for it to be resumed. With the latest form task feature, the forms JSON file can be directly loaded at run time.

🚩 Human In the Loop

The human operator completes or rejects the created task.

🚩 Data Post Processing

Processing the Action Center input of the human operator, in case there is any. (E.g. Performing an Orchestrator HTTP request in order to get the assigned task user)

# Solution Architecture

## **Human Assisted Resolution, Part of End-to-End Business Processes**

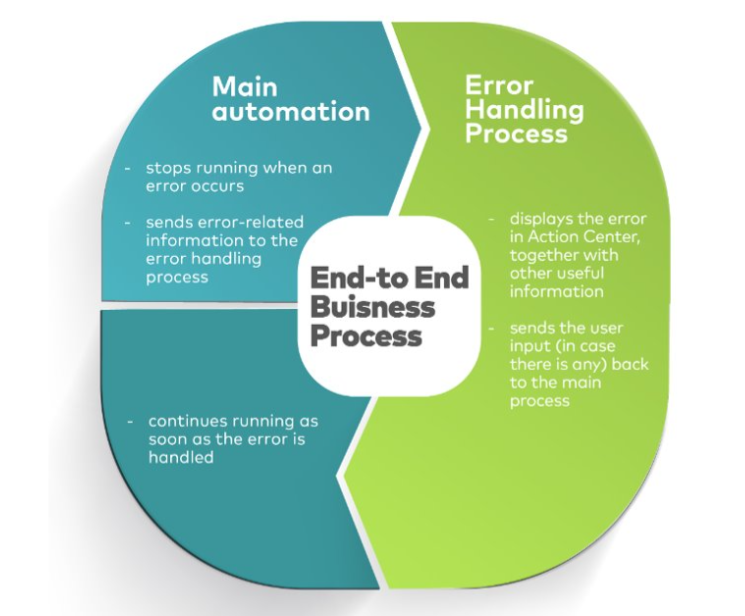
The HART process is part of bigger business processes to be automated.

A single process, HART, handles the creation of Action Center tasks for every process in that specific use case. After the completion of each task, the processes will continue from where they left off, with the corrected data. Like this, both time and cost will be saved.

Diagram

Description automatically generated

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



# 

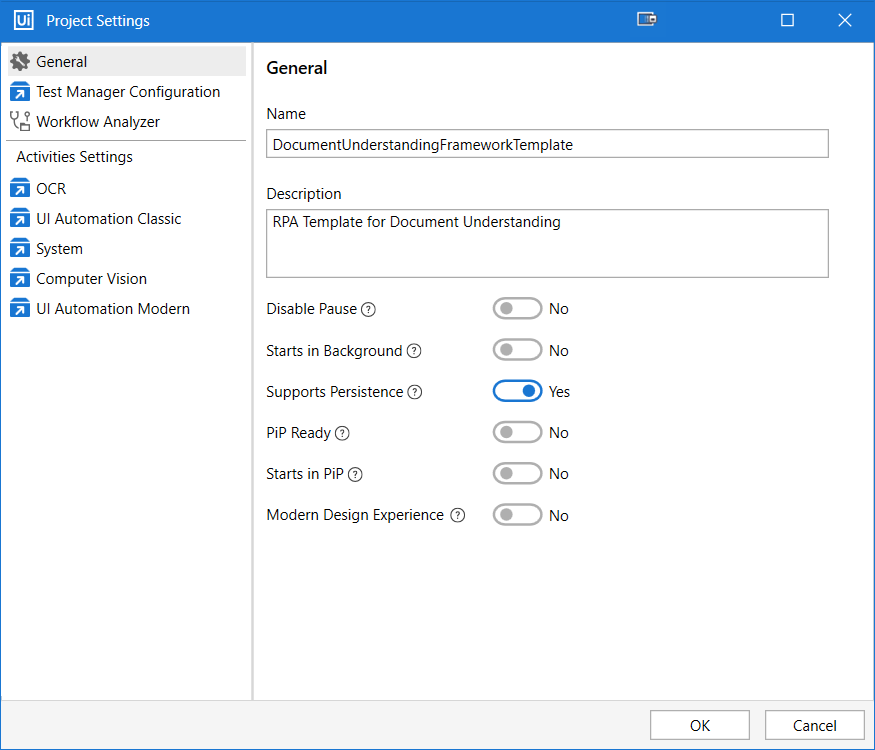
# Studio Project Overview

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

## **Settings for Unattended Processes**

**Enable Persistence Support**

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



## **Example Implementation**

## **Project Files**

### 📄 **Data\Config.xlsx**

Configuration file for project settings. Minimum configuration required:

* Configure the **OrchestratorFolder**
* Configure the **StorageBucketName** under the **Settings** sheet.
* Configure the **TaskTitle**, **Title**, **Description**, **SolutionDescription** and **FileName** under the **Settings**/**Assets** sheet. The configuration of the **TaskCatalog** is optional.
* Configure the **OrchestratorQueueName** if using queues.

### 📄 **Data\Files\HART.pdf**

File, used as a component in one of the Form JSON files. The **Files** folder can contain any pdf/image/documents to be displayed in Action Center.

### 📄 **Data\Templates\BusinessRuleException\_ProcessName.json**

An example of form JSON file for the business rule exception case.

### 📄 **Data\Templates\SystemException\_ProcessName.json**

An example of form JSON file for the system exception case.

### 📄 **Data\Templates\Exception\_AssignedUsersProcess.json**

An example of form JSON file for the case where the user that has been assigned to a task should be retrieved.

### 📄 **Data\Templates\TEO.json**

An example of form JSON file for the case where the user that has been assigned to a task should be retrieved.

### 📄 **Data\HttpRequest\Asset.json**

Payload for asset generation in Orchestrator (the payload is used as parameter in the Http Request activity)

### 📄 **Data\ HttpRequest \StorageBucket.json**

Payload for storage bucket generation in Orchestrator (the payload is used as parameter in the Http Request activity)

### 📄 **Data\ HttpRequest \TaskCatalog.json**

Payload for task catalog generation in Orchestrator (the payload is used as parameter in the Http Request activity)

### 📄 **Main.xaml**

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

Arguments:

* in\_Template (default **Nothing**): specifies what form JSON file should be used during the task creation step.
* 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.

### 📄 **InitializeOrchestrator.xaml**

Workflow should be executed before Main.xaml.

Checks if the storage bucket and the assets from the settings sheet have been manually created in Orchestrator. If not, they are being created through Orchestrator HTTP requests. The creation of the task catalog is optional.

Arguments:

* in\_ConfigFile (default **Data\Config.xlsx**): specifies the path of the configuration file.
* in\_ConfigSheets (default **Settings**): specifies from what sheet the contents (storage bucket, assets, task catalog) should be extracted.

### 📄 **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\20\_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\30\_SelectTemplate.xaml**

Workflow for selecting the correct path of the form JSON file, which is used by the **Create Form Task** activity.

### 📄 **Framework\40\_DataPostProcessing.xaml**

Workflow for processing the task data.

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

### 📄 **Framework\50\_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\_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\GenerateId.xaml**

Generates an ID, which is used in the payload creation of the HTTP requests.

### 📄 **Framework\ReusableWorkflows\GenerateKey.xaml**

Generates a key, which is used in the payload creation of the HTTP requests.

### 📄 **Framework\ReusableWorkflows\OrchestratorRequest\_GET.xaml**

Executes an Orchestrator HTTP request to retrieve data from Orchestrator.

### 📄 **Framework\ReusableWorkflows\OrchestratorRequest\_POST.xaml**

Executes an Orchestrator HTTP request to create different entities, like assets/storage buckets/task catalogs into Orchestrator.

### 📄 **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**

* If using Action Center, create a **Storage Bucket** for your process.
* If needed, create a **Queue** for your process.

## **🔧 Unattended Automation**

* If using Action Center, configure the **StorageBucketName** under the **Settings** sheet,
* If using queues, configure the **OrchestratorQueueName**.
* Configure the **Settings for Unattended Processes**.