**Technical Design Document**

**VendorReporting Performer**

Table of Contents

# **1 Introduction**

Download monthly reports for vendor, merge reports into single yearly report, and upload yearly report

# **2 Purpose of this document**

This is a Technical Design Document that explains the technical aspects of the robot designed and developed using UiPath in detail. This will give an overview of the design of the bot and can be used by developers or other stakeholders to understand the prerequisites and requirements to execute the bot successfully.

# **3 Scope**

**The scope of this document includes:**

• Environment Specification  
• System Requirements  
• Prerequisites  
• UiPath Enterprise  
• File Folder Structure  
• Robot Design  
• Issues and Risks

**The scope of this document does not include:**

• Availability of Systems  
• System Changes  
• Changes to Input Files or Data Format  
• Process Changes

*Note – this is a Technical Design Document which only covers the technical aspects. Please refer to the Business Requirements Document for any other information about business processes*

# **4 System Requirements**

The Developer System (specifications below) was used to develop this Technical Design Document. It is important to note that the system the robot is migrated to should also have similar specifications to ensure a proper functioning of the robot.  
The system specifications for VendorReporting Performer process:

|  |  |
| --- | --- |
| **Operating System** |  |
| **Processor** |  |
| **RAM** |  |
| **Hard Disk** |  |
| **Components** |  |

# **5 Prerequisites**

The prerequisites for the robot to successfully run are as follows:

a. The following applications are installed in the system:  
 • UiPath 22.12.0.0  
b. The system has a valid studio and back office robot licence for UiPath  
c. Robot has access to all required applications  
d. Files are located in the system following the File Structure indicated below (Section 7).

# **6 UiPath Enterprise**

UiPath version 22.12.0.0 is the software used for developing the Technical Design Document.

# **7 File/Folder Structure**

**• Temporary Folder:**

Path at which all the temporary files are stored.

**• Input Folder:**

Path at which all the input files are stored.

**• Output Folder:**

Path at which all the output files are stored.

**• Code Repository**

Path at UiPath process workflows are be stored.

**• Config File Path**

Path at config files are be stored.

|  |  |
| --- | --- |
| File/Folder | File/Folder Location |
| Temporary Folder |  |
| Input Folder |  |
| Output Folder |  |
| Code Repository |  |
| Config File Path |  |

# **8 Bot Design**

8.1 Main.xaml

Process Name: VendorReporting\_Performer  
Description: Download monthly reports for vendor, merge reports into single yearly report, and upload yearly report  
Author: Lindsay Kane  
Date Created: 01.10.2023  
1 Get the next transaction to be processed.  
2 Retrieve a new transaction data to be processed.  
The TransactionNumber variable holds the current transaction number and incrementing this variable makes the framework retrieve the next transaction. If the framework is retrying a failed transaction, this variable is not incremented until the maximum number of retry attempts is reached.  
3 Process a single transaction.   
The result of the processing can be: 1) Success, 2) Business Exception, 3) System Exception.   
In the case of a system exception, the transaction can be automatically retried.  
4 Read configuration file and initialize applications used in the process.  
5 An uninitialized Config dictionary indicates that it is the first run of the process.  
6 Overwrite queue name from in configuration file in case the argument in\_OrchestratorQueueName is specified.  
This provides backward compatibility with < 2018.3.  
7 Overwrite folder name from in configuration file in case the argument in\_OrchestratorQueueFolder is specified.  
This provides backward compatibility with < 2018.3.  
8 Kills all Windows processes representing applications used in this business process to assure that the execution starts in a clean state.  
Since the applications are assumed to be already closed, CloseAllApplications is skipped and just KillAllProcess is invoked.  
9 Add the process name to the logs generated after this point.   
This log field can be used to create reports and visualizations about the process.  
10 If MaxConsecutiveSystemExceptions number was reached, throw Exception at initialization and go to End Process state, thus finalizing the execution.   
If MaxConsecutiveSystemExceptions is 0, then any number of consecutive System Exceptions is allowed.  
11 Failures during the initialization are considered system exceptions and lead to the End Process state, thus finalizing the execution.  
12 End process and close all applications used.  
13 Business Rule Exception  
14 There is no need for any action in case of successful transaction.  
The process should simply go to next transaction.  
15 This is a simple mecanism to stop the process.  
In reallife scenario you can stop the process when there is no more data to process or on a schedule.  
16 Transaction item to be processed. The type of this variable can be changed to match the transaction type in the process. For example, when processing data from a spreadsheet that is read into a DataTable, this type can be changed to DataRow.   
17 Used during transitions between states to represent exceptions other than business exceptions.  
18 Used during transitions between states and represents a situation that does not conform to the rules of the process being automated.  
19 Sequential counter of transaction items.  
20 Dictionary structure to store configuration data of the process (settings, constants and assets).  
21 Used to control the number of attempts of retrying the transaction processing in case of system exceptions.  
22 Optionally used to include additional information about the transaction item.  
23 Optionally used to include additional information about the transaction item.  
24 Transaction ID used for information and logging purposes. Ideally, the ID should be unique for each transaction.   
25 Used in case transactions are stored in a DataTable, for example, after being retrieved from a spreadsheet.  
26 Used to control the number of consecutive system exceptions.

***Location: \Main.xaml***

***Parameters:***

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| in\_OrchestratorQueueName | InArgument(x:String) | Allows the Orchestrator queue name to be passed as an argument, instead of only being defined in the configuration file. |
| in\_OrchestratorQueueFolder | InArgument(x:String) | Allows the Orchestrator folder name where the queue is created to be passed as an argument, instead of only being defined in the configuration file. |

8.2 ACME\_GetMonthlyReports.xaml

For each month in a year, given the year, download reports for given tax id. Read downloaded report into datatable, and merge each report's data with datatable to store all month's report data.

***Location: \ACME\ACME\_GetMonthlyReports.xaml***

***Parameters:***

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| in\_ACMEMonthlyReportsUrl | InArgument(x:String) |  |
| in\_DownloadsFolder | InArgument(x:String) |  |
| in\_TaxId | InArgument(x:String) |  |
| in\_ReportYear | InArgument(x:String) |  |
| out\_AllMonthsDT | OutArgument(sd:DataTable) |  |

8.3 ACMEGetTaxId.xaml

***Location: \ACME\ACME\_GetTaxId.xaml***

***Parameters:***

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| WIId | InArgument(x:String) |  |
| WIUrl | InArgument(x:String) |  |
| out\_TaxId | OutArgument(x:String) |  |

8.4 ACME\_UpdateWorkItem.xaml

***Location: \ACME\ACME\_UpdateWorkItem.xaml***

***Parameters:***

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| in\_WIId | InArgument(x:String) |  |
| in\_WIUrl | InArgument(x:String) |  |
| in\_ConfirmationId | InArgument(x:String) |  |

8.5 ACME\_UploadYearlyReport.xaml

Upload csv report containing data for all downloaded monthly invoices for given tax id and year.  
1 Write datatable containing data from reports for all months for given year and tax id into csv file to be uploaded to yearly report.

***Location: \ACME\ACME\_UploadYearlyReport.xaml***

***Parameters:***

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| in\_ACMEYearlyReportsUrl | InArgument(x:String) |  |
| in\_TaxId | InArgument(x:String) |  |
| in\_ReportYear | InArgument(x:String) |  |
| in\_FinalReportFilePath | InArgument(x:String) |  |
| out\_ConfirmationId | OutArgument(x:String) |  |
| in\_AllMonthsDT | InArgument(sd:DataTable) |  |

8.6 CloseAllApplications.xaml

Do the necessary procedures for ending the process (e.g., logout) and close the used applications.

***Location: \Framework\CloseAllApplications.xaml***

8.7 GetTransactionData.xaml

Get a transaction item from a specified source (e.g., Orchestrator queues, spreadsheets, databases, mailboxes or web APIs).   
  
If there are no transaction items remaining, out\_TransactionItem is set to Nothing, which leads to the End Process state.   
  
For cases in which there is only a single transaction (i.e., a linear process), use an If activity to check whether the argument in\_TransactionNumber has the value 1 (meaning it is the first and only transaction) and assign the transaction item to out\_TransactionItem. For any other value of in\_TransactionNumber, out\_TransactionItem should be set to Nothing.  
  
If there are multiple transactions, use the argument in\_TransactionNumber as an index to retrieve the correct transaction to be processed. If there are no more transactions left, it is necessary to set out\_TransactionItem to Nothing, thus ending the process.  
1 Get a transaction item from the specified Orchestrator queue.  
If queues are not used in this process, replace this activity with the appropriated logic to retrieve transaction items.  
For example, if transactions are rows from a DataTable, the row corresponding to the current transaction is retrieved at this point.  
2 This optional step can be used to include more information about a transaction item and it is used mainly for logging and visualization purposes.  
For example, if transaction items are invoices, then out\_TransactionID can be the invoice number, out\_TransactionField1 can be the invoice date and out\_TransactionField2 can be the invoice amount.  
3 Write data to the added log fields that identify the transaction.

***Location: \Framework\GetTransactionData.xaml***

***Parameters:***

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| in\_TransactionNumber | InArgument(x:Int32) | Sequential counter of transaction items. |
| in\_Config | InArgument(scg:Dictionary(x:String, x:Object)) | Dictionary structure to store configuration data of the process (settings, constants and assets). |
| out\_TransactionItem | OutArgument(ui:QueueItem) | Transaction item to be processed. |
| out\_TransactionField1 | OutArgument(x:String) | Allow the optional addition of information about the transaction item. |
| out\_TransactionField2 | OutArgument(x:String) | Allow the optional addition of information about the transaction item. |
| out\_TransactionID | OutArgument(x:String) | Transaction ID used for information and logging purposes. Ideally, the ID should be unique for each transaction. |
| io\_dt\_TransactionData | InOutArgument(sd:DataTable) | This variable can be used in case transactions are stored in a DataTable (for example, after being retrieved from a spreadsheet). |

8.8 Initialize\_Applications.xaml

Open applications used in the process and do necessary initialization procedures (e.g., login).

***Location: \Framework\InitAllApplications.xaml***

***Parameters:***

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| in\_Config | InArgument(scg:Dictionary(x:String, x:Object)) | Dictionary structure to store configuration data of the process (settings, constants and assets). |

8.9 InitiAllSettings.xaml

Initialize, populate and output a configuration Dictionary to be used throughout the project.   
Settings and constants are read from the local configuration file, and assets are fetched from Orchestrator.   
Asset values overwrite settings and constant values if they are defined with the same name.  
1 Read settings and constants from the configuration file and add them to the Config dictionary.  
2 Read non-empty rows in the sheet.  
3 Get from Orchestrator the values of assets listed in the Assets sheet.  
4 Logs a message and throw exception in case the asset is specified in the Config file, but it could not be loaded from Orchestrator.

***Location: \Framework\InitAllSettings.xaml***

***Parameters:***

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| in\_ConfigFile | InArgument(x:String) | Path to the configuration file that defines settings, constants and assets. |
| in\_ConfigSheets | InArgument(s:String[]) | Names of the sheets corresponding to settings and constants in the configuration file. |
| out\_Config | OutArgument(scg:Dictionary(x:String, x:Object)) | Dictionary structure to store configuration data of the process (settings, constants and assets). |

8.10 KillAllProcesses.xaml

Use the Kill Process activity to force the termination of the Windows processes representing applications used in the business process being automated.  
Note that killing processes might have undesirable outcomes, such as losing unsaved changes to files.

***Location: \Framework\KillAllProcesses.xaml***

8.11 Process.xaml

Create file path with file nam containing tax id.

***Location: \Framework\Process.xaml***

***Parameters:***

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| in\_TransactionItem | InArgument(ui:QueueItem) | Transaction item to be processed. |
| in\_Config | InArgument(scg:Dictionary(x:String, x:Object)) | Dictionary structure to store configuration data of the process (settings, constants and assets). |
| out\_ConfirmationId | OutArgument(x:String) |  |
| out\_TaxId | OutArgument(x:String) |  |

8.12 RetryCurrentTransaction.xaml

Manage the retrying mechanism for the framework and it is invoked in SetTransactionStatus.xaml when a system exception occurs.   
The retrying method is based on the configurations defined in Config.xlsx.

***Location: \Framework\RetryCurrentTransaction.xaml***

***Parameters:***

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| in\_Config | InArgument(scg:Dictionary(x:String, x:Object)) | Dictionary structure to store configuration data of the process (settings, constants and assets). |
| io\_RetryNumber | InOutArgument(x:Int32) | Used to control the number of attempts of retrying the transaction processing in case of system exceptions. |
| io\_TransactionNumber | InOutArgument(x:Int32) | Sequential counter of transaction items. |
| in\_SystemException | InArgument(s:Exception) | Used during transitions between states to represent exceptions other than business exceptions. |
| in\_QueueRetry | InArgument(x:Boolean) | Used to indicate whether the retry procedure is managed by an Orchestrator queue. |

8.13 SetTransactionStatus.xaml

Set and log the transaction's status along with extra log fields.   
There can be three possible statuses: Success, Business Exception and System Exception.  
  
Business Rule Exception characterizes an irregular situation according to the process's rules and prevents the transaction to be processed. The transaction is not retried in this case, since the result will be the same until the problem that causes the exception is solved.  
For example, it can be considered a BusinessRuleException if a process expects to read an email's attachment, but the sender didn't attach any file. In this case, immediate retries of the transaction will not yield a different result.  
  
On the other hand, system exceptions are characterized by exceptions whose types are different than BusinessRuleException. When this kind of exception happens, the transaction item can be retried after closing and reopening the applications involved in the process. The rationale behind this is that the exception was caused by a problem in the applications, which might be solved by restarting them.  
  
If Orchestrator queues are the source of transactions, the Set Transaction Status activity is used to update the status. In addition, the retry mechanism is also implemented by Orchestrator.  
  
If Orchestrator queues are not used, the status can be set, for example, by writing to a specific column in a spreadsheet. In such cases, the retry mechanism is covered by the framework and the number of retries is defined in the configuration file.  
  
At the end, io\_TransactionNumber is incremented, which makes the framework get the next transaction to be processed.  
1 If the transaction item is processed without any exception, its status is updated as Successful.  
2 Includes custom log fields to the log message.  
They are removed after logging to prevent duplicated status messages about a single transaction.  
3 Increment the TransactionNumber to get the next transaction to be processed.  
4 Reset the counter of retries to allow the next transaction to be retried the correct amount of times.  
5 Reset the counter of consecutive system exceptions.  
6 If a BussinessRuleException is thrown during the process, the transaction item's status is updated as Failed (Exception Type: Business).  
  
7 Includes custom log fields to the log message.  
They are removed after logging to prevent duplicated status messages about a single transaction.  
8 If a system exception occurs during the process, the transaction item's status is updated as Failed (Exception Type: Application).  
9 Take a screenshot of the current state of the screen to facilitate debugging.  
10 Close all applications before returning to the Initialization state and opening them again.  
If applications cannot be closed, kill their respective processes.

***Location: \Framework\SetTransactionStatus.xaml***

***Parameters:***

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| in\_BusinessException | InArgument(ui:BusinessRuleException) | Exception variable that is used during transitions between states and represents a situation that does not conform to the rules of the process being automated. |
| in\_TransactionField1 | InArgument(x:String) | Optionally used to include additional information about the transaction item. |
| in\_TransactionField2 | InArgument(x:String) | Optionally used to include additional information about the transaction item. |
| in\_TransactionID | InArgument(x:String) | Used for information and logging purposes. Ideally, the ID should be unique for each transaction. |
| in\_SystemException | InArgument(s:Exception) | Used during transitions between states to represent exceptions other than business exceptions. |
| in\_Config | InArgument(scg:Dictionary(x:String, x:Object)) | Dictionary structure to store configuration data of the process (settings, constants and assets). |
| in\_TransactionItem | InArgument(ui:QueueItem) | Transaction item to be processed. |
| io\_RetryNumber | InOutArgument(x:Int32) | Used to control the number of attempts of retrying the transaction processing in case of system exceptions. |
| io\_TransactionNumber | InOutArgument(x:Int32) | Sequential counter of transaction items. |
| io\_ConsecutiveSystemExceptions | InOutArgument(x:Int32) | Used to control the number of consecutive system exceptions. |
| in\_ConfirmationId | InArgument(x:String) |  |
| in\_TaxId | InArgument(x:String) |  |

8.14 TakeScreenshot.xaml

Capture a screenshot, log its name and location and save it with the PNG extension.  
If no specific filepath is passed as argument, it saves the image in the folder specified by in\_Folder.  
1 The whole screen is captured.

***Location: \Framework\TakeScreenshot.xaml***

***Parameters:***

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| in\_Folder | InArgument(x:String) | Path to the folder where the screenshot should be saved. |
| io\_FilePath | InOutArgument(x:String) | Optional argument that specifies the path and the name of the screenshot to be taken. |

8.15 ReadReport.xaml

***Location: \Helpers\ReadReport.xaml***

***Parameters:***

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| in\_DownloadedFile | InArgument(x:String) |  |
| io\_AllMonthsDT | InOutArgument(sd:DataTable) |  |

8.16 GetTransactionDataTestCase.xaml

Given the TransactionNumber, verify if GetTransactionData workflow works as expected.  
Once a Transaction Item has been processed, its status will be In Progress.  
Queue name should be configured in order for the test case to run.  
1 Note to also change the transaction status

***Location: \Tests\GetTransactionDataTestCase.xaml***

8.17 InitAllApplicationsTestCase.xaml

Verify if the InitAllApplications workflow works as expected.  
The verification should check if after opening the applications, the expected state is reached.  
1 Check if the applications are in the expected state

***Location: \Tests\InitAllApplicationsTestCase.xaml***

8.18 InitAllSettingsTestCase.xaml

Verify if the InitAllSettings workflow works as expected.  
The verification should check if initalization of settings was successful: if the Config dictionary was created, if it contains a certain key etc.  
1 Check if the Config contains key-value pairs  
Check if the Config contains certain key information

***Location: \Tests\InitAllSettingsTestCase.xaml***

8.19 MainTestCase.xaml

Verify if the Main workflow works as expected.  
The verification should check whether the status file or report built after the process run is the expected one.  
1 Please add the path to the status file  
2 Please add the path to the expected status file

***Location: \Tests\MainTestCase.xaml***

8.20 ProcessTestCase.xaml

Verify if the Process workflow works as expected.  
The verification should check whether the output of the Process workflow is the expected one.

***Location: \Tests\ProcessTestCase.xaml***

8.21 TestWorkflowTemplate.xaml

Template workflow used to create tests for workflows in the process.  
Create a new test workflow by copying and renaming this file.  
1 Dictionary structure to store configuration data.  
2 Verification to pe performed

***Location: \Tests\WorkflowTestCaseTemplate.xaml***

# **9 Workflows**

|  |  |  |
| --- | --- | --- |
| Name | Invoked Workflow | Invoked In |
| \Main.xaml | •Framework\GetTransactionData.xaml •Framework\Process.xaml •Framework\SetTransactionStatus.xaml •Framework\InitAllSettings.xaml •Framework\KillAllProcesses.xaml •Framework\InitAllApplications.xaml •Framework\CloseAllApplications.xaml | •Tests\MainTestCase.xaml |
| \ACME\ACME\_GetMonthlyReports.xaml | •Helpers\ReadReport.xaml | •Framework\Process.xaml |
| \ACME\ACME\_GetTaxId.xaml |  | •Framework\Process.xaml |
| \ACME\ACME\_UpdateWorkItem.xaml |  | •Framework\Process.xaml |
| \ACME\ACME\_UploadYearlyReport.xaml |  | •Framework\Process.xaml |
| \Framework\CloseAllApplications.xaml |  | •Main.xaml •Framework\SetTransactionStatus.xaml •Tests\InitAllApplicationsTestCase.xaml •Tests\ProcessTestCase.xaml |
| \Framework\GetTransactionData.xaml |  | •Main.xaml •Tests\ProcessTestCase.xaml |
| \Framework\InitAllApplications.xaml |  | •Main.xaml •Tests\ProcessTestCase.xaml |
| \Framework\InitAllSettings.xaml |  | •Main.xaml •Tests\GetTransactionDataTestCase.xaml •Tests\InitAllApplicationsTestCase.xaml •Tests\ProcessTestCase.xaml •Tests\WorkflowTestCaseTemplate.xaml |
| \Framework\KillAllProcesses.xaml |  | •Main.xaml •Framework\SetTransactionStatus.xaml |
| \Framework\Process.xaml | •ACME\ACME\_GetTaxId.xaml •ACME\ACME\_GetMonthlyReports.xaml •ACME\ACME\_UploadYearlyReport.xaml •ACME\ACME\_UpdateWorkItem.xaml | •Main.xaml •Tests\ProcessTestCase.xaml |
| \Framework\RetryCurrentTransaction.xaml |  |  |
| \Framework\SetTransactionStatus.xaml | •Framework\TakeScreenshot.xaml •Framework/RetryCurrentTransaction.xaml •Framework\CloseAllApplications.xaml •Framework\KillAllProcesses.xaml | •Main.xaml |
| \Framework\TakeScreenshot.xaml |  | •Framework\SetTransactionStatus.xaml |
| \Helpers\ReadReport.xaml |  | •ACME\ACME\_GetMonthlyReports.xaml |
| \Tests\GetTransactionDataTestCase.xaml | •Framework\InitAllSettings.xaml •Framework/GetTransactionData.xaml |  |
| \Tests\InitAllApplicationsTestCase.xaml | •Framework\InitAllSettings.xaml •Framework/InitAllApplications.xaml •Framework\CloseAllApplications.xaml |  |
| \Tests\InitAllSettingsTestCase.xaml | •Framework/InitAllSettings.xaml |  |
| \Tests\MainTestCase.xaml | •Main.xaml |  |
| \Tests\ProcessTestCase.xaml | •Framework\InitAllSettings.xaml •Framework\InitAllApplications.xaml •Framework\GetTransactionData.xaml •Framework\Process.xaml •Framework\CloseAllApplications.xaml |  |
| \Tests\WorkflowTestCaseTemplate.xaml | •Framework\InitAllSettings.xaml |  |

# **10 Dependencies**

• ACME: [1.0.1]  
• UiPath.Excel.Activities: [2.17.0-preview]  
• UiPath.System.Activities: [22.10.4]  
• UiPath.UIAutomation.Activities: [22.10.4]

# **11 Issues and Risks**

Below are the issues and risks identified during development and testing