UiPath™

UiPath Automation

Walkthrough

Walkthrough - Generate Yearly Report for Vendor

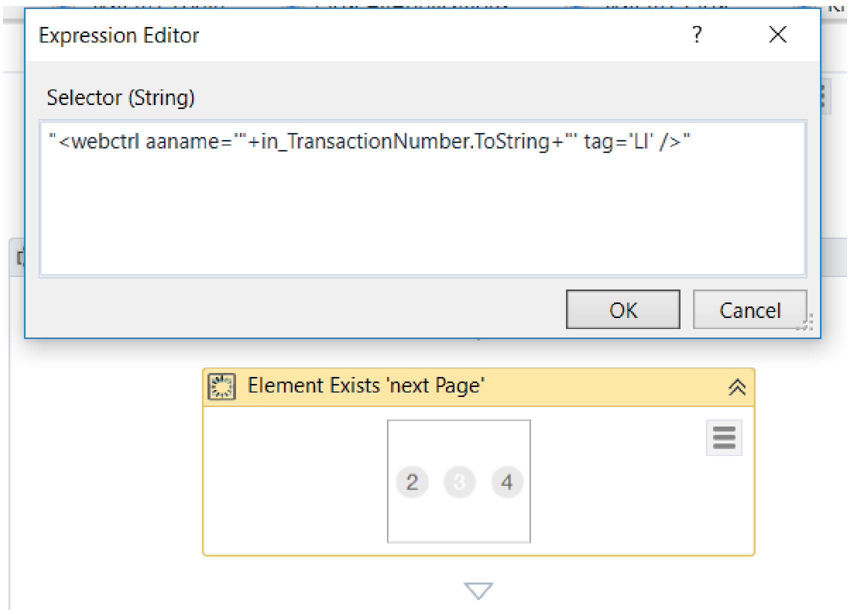
Walkthrough – Generate Yearly Report for Vendor

This time, we are using Orchestrator Queues for the processing of the work items, to understand more about the capabilities of this feature. We will go over an example of how to use multiple robots to process data, how to prevent them from processing the entire Queue Item list from the beginning and instead resume the work in case a system error occurs, etc. We will also split the process by using 2 different processes. One creates the queue of items and is called **Dispatcher**. The other one processes the previously created queue items and is called **Performer**. With this approach, we can load the transactions using the Dispatcher only once, and then use multiple Performer robots to process the queue items created by the Dispatcher.

The Dispatcher Process

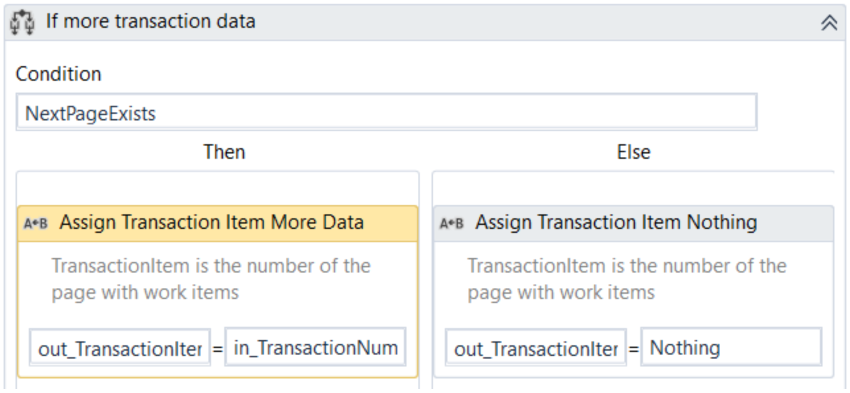
- Start with the REFramework template.
    - The Dispatcher is responsible for uploading the work items to the queue. We should upload the **WIID** to the queue to uniquely identify each transaction item.
    - Let's imagine that the next page arrow is not available for WI 4, so we cannot scrape the data in the table to extract all the work items. Moreover, in case something happens to the System 1 application while navigating through pages, the Dispatcher will recover from the error and resume the work. It will also retry failed transactions. We consider one page in the WorkItems list to be a transaction and the page number to be the transactionItem.
    - The transaction item is a string representing the number of the page which is currently being processed.
- Edit the Config file for the current process.
    - In the **Settings** sheet, add the **InHouse_Process4** value in the **QueueName** parameter. The queue will be defined in Orchestrator using the same name.
    - In the **Settings** sheet, add settings for the **System1 URL** and **System1 Credential parameters**.
    - In the **Constants** sheet, set the value of **MaxRetryNumber** to 2.
- Make the following framework changes:
    - The TransactionItem variable in the Main file should be of the System.String type. You should also make sure that the argument types in the **GetTransactionData**, **Process**, and **SetTransactionStatus** files match the type of TransactionItem.

- We will use only one application in this exercise - ACME System1. Create a folder named **System1** in the solution root folder.
    - The following components in Process 5 can be reused.
        - Copy the **System1_Login.xaml**, **System1_Close.xaml**, and **System1_NavigateTo_WorkItems.xaml** files to the **System1** folder.
        - Copy the **SendEmail.xaml** file to the **Common** folder.
- Open the **InitAllApplications** file.
    - Invoke the **System1\System1_Login.xaml** file.
    - Invoke the **System1\System1_NavigateTo_WorkItems.xaml** file.
- Open the **CloseAllApplications** file.
    - Invoke the **System1\System1_Close.xaml** file.
- Open the **KillAllProcesses.xaml** file in the Framework folder.
    - Add a **Kill Process** activity and rename it "Kill process IE".
- Open the **GetTransactionData.xaml** project in the Framework folder. It can be found in the **Get Transaction Data** state.
    - Delete the **Get Transaction Item** activity as it is not needed, because the Dispatcher process is used to upload data to the queue.
    - Before the **Write Transaction info in Logging Fields** sequence, add an **Attach Browser** activity and attach the WorkItems page.
    - Add an **Element Exists** activity to check if the next page is available. Indicate a page number and modify the selector to use an attribute related to the page number (reminder: the **in_TransactionNumber** argument is the page number for the Dispatcher).
    - This is what the **Element Exists** activity should look like.



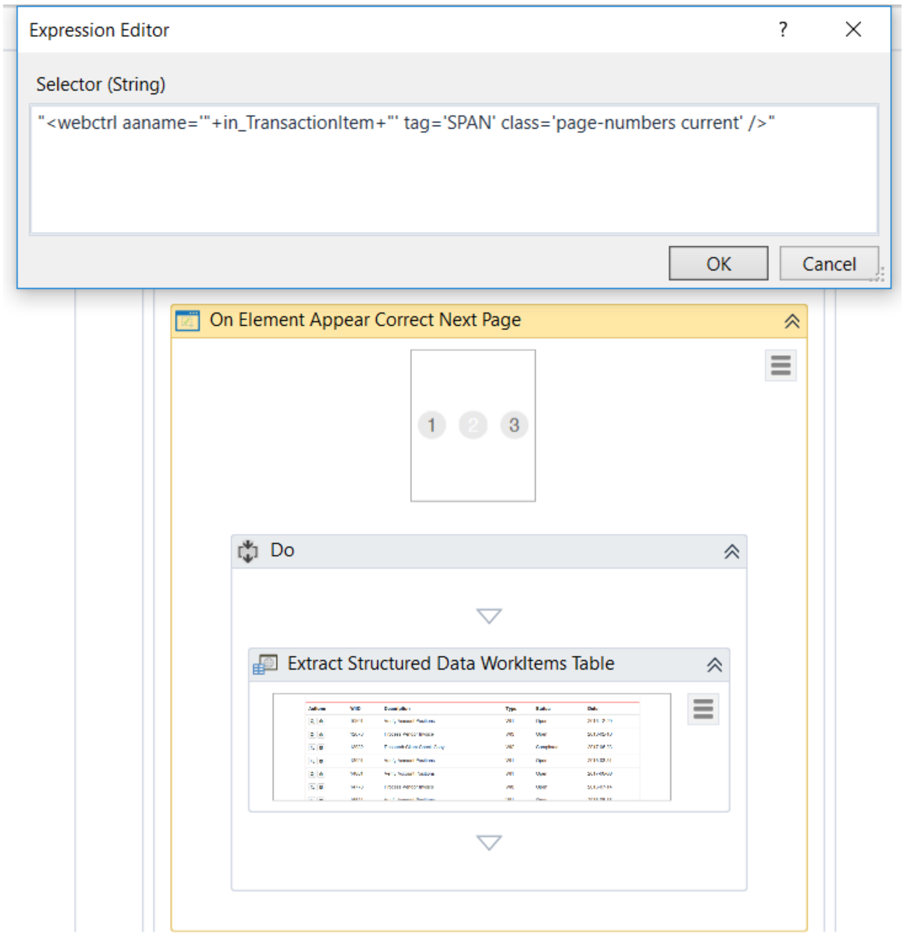
```
"<webctrl aaname='"+in_TransactionNumber.ToString+"' tag='LI' />"
```

- Use an **If** activity to check if there is more transaction data left.
- If the next page exists, set the output argument **out_TransactionItem** to the value of the current page, namely **in_TransactionNumber**.
- If the next page doesn't exist, set the **out_TransactionItem** to Nothing.
- This is how the **If** activity should look:



If more transaction data
Condition
NextPageExists
Then
Else
Assign Transaction Item More Data
TransactionItem is the number of the page with work items
out_TransactionIter = in_TransactionNum
Assign Transaction Item Nothing
TransactionItem is the number of the page with work items
out_TransactionIter = Nothing

- Open the **Process.xaml** file in the Framework folder. It can be found in the **Process Transaction** state.
  - Use a **Click** activity to select the processing page number using the same dynamic selector to identify the current page (**in_TransactionNumber** argument).
  - Next, add an **On Element Appear** activity to check if the processing page was opened. You can use UiExplorer. The class attribute can be used to identify active or inactive pages, so it can also be used when creating the dynamic selector. Inside the **On Element Appear** activity, scrape the table containing work items. Create a variable called **dt_WorkItems** in the **Output** property to store them.
  - This is how the **On Element Appear** activity should look:

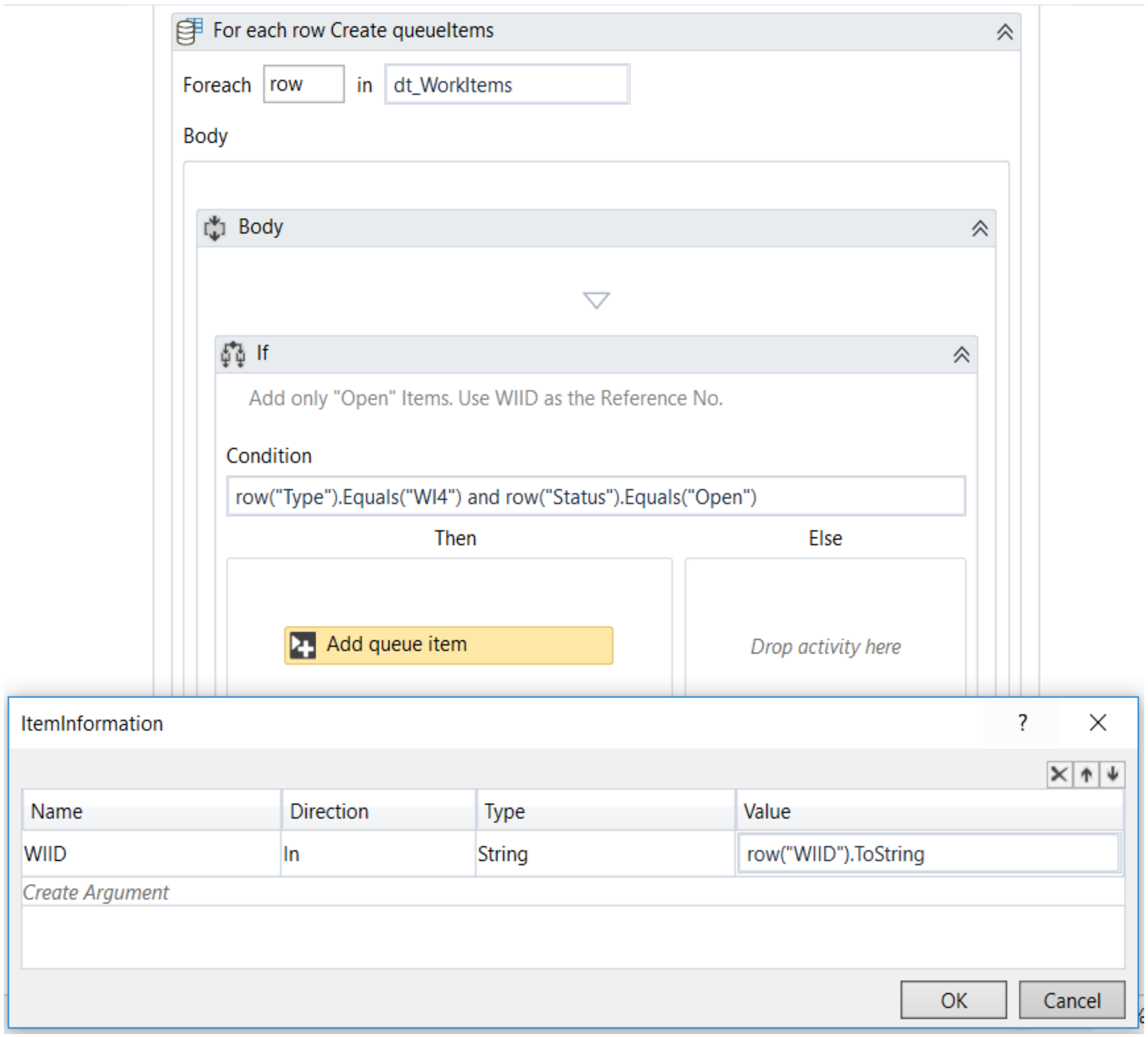

Expression Editor

Selector (String)

```
"<webctrl aaname='"+in_TransactionItem+"' tag='SPAN' class='page-numbers current' />"
```

OK
Cancel
On Element Appear Correct Next Page
1 2 3
Do
Extract Structured Data WorkItems Table

- The next step is to upload the **WIID** for all the rows inside the scraped data tablethat have the WI4 type and the Open status to the queue. To upload the value to the queue, use an **Add Queue Item** activity. In the **Properties** panel, use the value in the **in_Config** dictionary to fill the **QueueName** field. In the **ItemInformation** field, create an argument called **WIID**. Set the corresponding value for the argument.
- This is what the **Add Queue Item** activity should look like:


For each row Create queueItems

Foreach row in dt_WorkItems

Body

If

Add only "Open" Items. Use WIID as the Reference No.

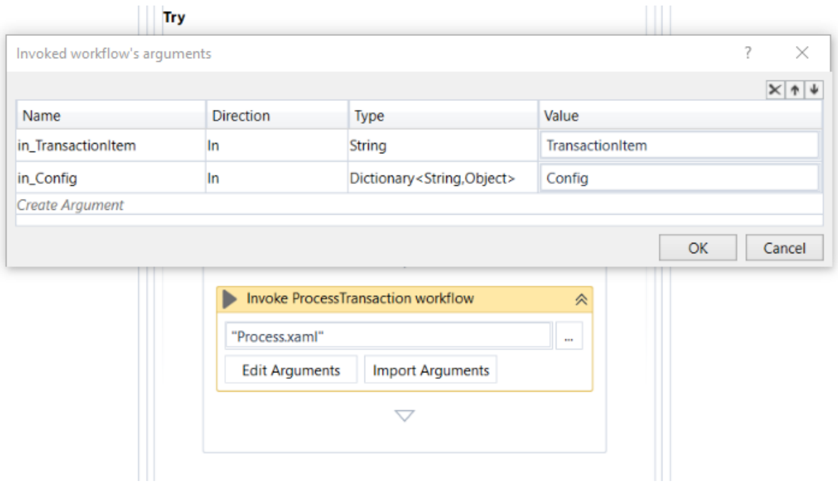
Condition

```
row("Type").Equals("WI4") and row("Status").Equals("Open")
```

ItemInformation

Name	Direction	Type	Value
WIID	In	String	row("WIID").ToString

- Open the **Main.xaml** file.
    - Make sure that the arguments for the **Process.xaml** invoked project are set correctly.
        - This is what the arguments should look like:


Invoked workflow's arguments

Name	Direction	Type	Value
in_TransactionItem	In	String	TransactionItem
in_Config	In	Dictionary<String,Object>	Config

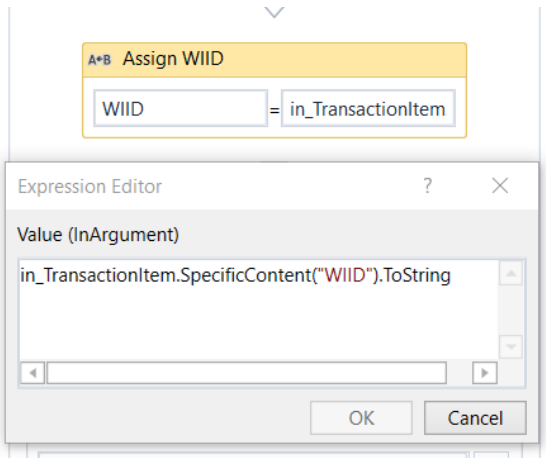
- We are done with the process implementation. Next, we need to test the process and check that the values are uploaded to the queue correctly.

The Performer Process

The Performer processes all the transactions loaded by the Dispatcher to the queue. For that reason, the TransactionItem type needs to be **QueueItem**.

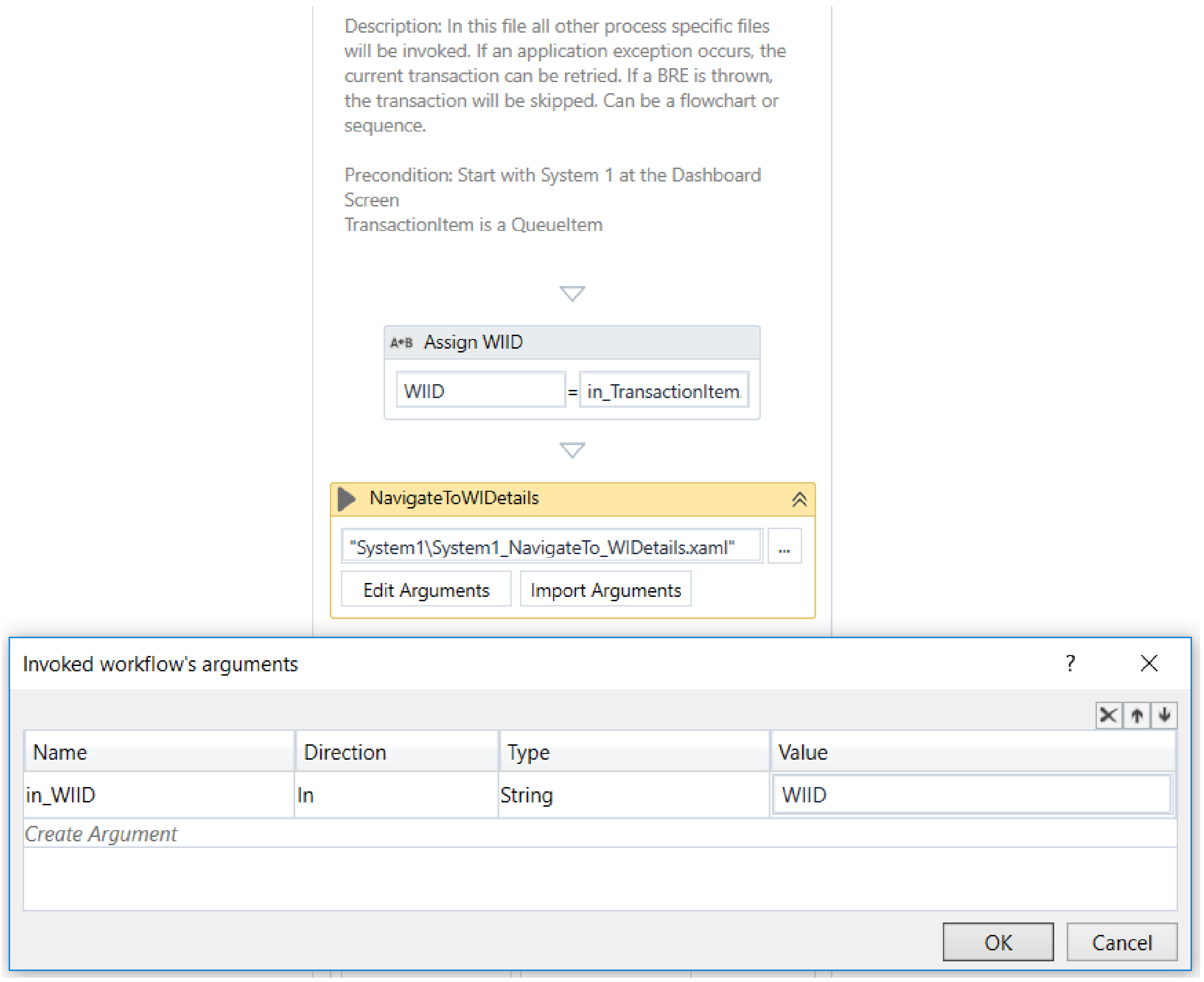
- Start with the REFramework template.
  - The **TransactionItem** argument should be of the **QueueItem** type. This is the default type of TransactionItem in the REFramework.
- Edit the **Config** file for the current process as follows:
  - In the **Settings** sheet, add the "InHouse_Process4" value in the **QueueName** parameter. The queue will be defined in Orchestrator using the same name.
  - In the **Settings** sheet, add settings for the **System1 URL** and **System1 Credential** parameters.
  - In the **Constants** sheet, keep the value of **MaxRetryNumber** at 0, because we are using queue items in the Performer process and the retry mechanism is handled in Orchestrator.
- We will use only one application in this exercise: ACME System1. Create a folder named **System1** in the solution root folder.
  - The following components in the **Performer** process can be reused.
    - Copy the **System1_Login.xaml**, **System1_Close.xaml**, **System1_NavigateTo_WorkItems.xaml**, **System1_NavigateTo_WIDetails.xaml** and **System1_UpdateWorkItem.xaml** files to the **System1** folder.
    - Copy the **SendEmail.xaml** file to the **Common** folder.
- Open the **InitAllApplications** file.
  - Invoke the **System1\System1_Login.xaml** file.

- Open the **CloseAllApplications** file.
  - Invoke the **System1\System1_Close.xaml** file.
- Open the **KillAllProcesses.xaml** project in the Framework folder. Add a **Kill Process** activity and rename it **Kill process IE**.
  - Set the **ProcessName** property to **iexplore**.
- Open the **Process.xaml** project in the Framework folder. It can be found in the **Process Transaction** state.
  - Create a String variable to set the current work item ID (WIID).
  - Add an **Assign** activity and set the variable created above to the value in the queue. To do that, we can use the SpecificContent method - in_TransactionItem.SpecificContent("WIID").ToString
  - This is what the **Assign** activity should look like:



in_TransactionItem.SpecificContent("WIID").ToString

  - Open the **Process.xaml** workflow in the Framework folder. Invoke the **System1\System1_NavigateTo_WIDetails.xaml**. Import and bind the argument that should be taken from the queue using the **SpecificContent** method, as described earlier.
  - This is what the **Invoke** activity and the WIID argument should look like:


Description: In this file all other process specific files will be invoked. If an application exception occurs, the current transaction can be retried. If a BRE is thrown, the transaction will be skipped. Can be a flowchart or sequence.

Precondition: Start with System 1 at the Dashboard Screen
TransactionItem is a QueueItem

Assign WIID: WIID = in_TransactionItem.

NavigateToWIDetails: "System1\System1_NavigateTo_WIDetails.xaml"

Invoked workflow's arguments

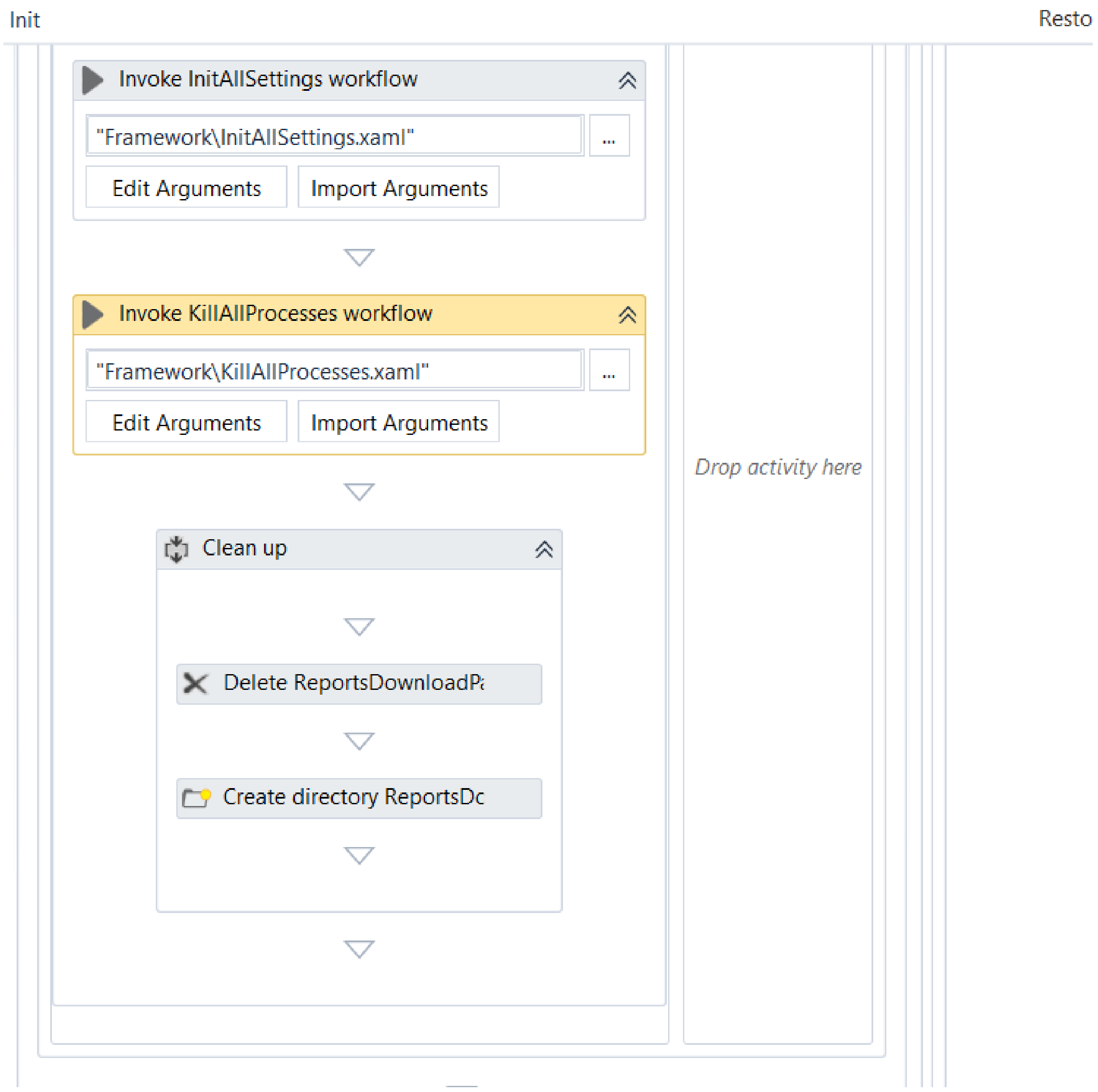
Name	Direction	Type	Value
in_WIID	In	String	WIID

- ○ Next, create a blank sequence workflow in the System1 folder. We'll use it to retrieve the TaxID value from the Work Item Details page. We can name this workflow System1_ExtractVendorInformation.xaml. The output should be the TaxID argument.

- ○ Open the **Process.xaml** and create a String variable named TaxID to store the value of the out argument from the previous created workflow file. Invoke **System1 \System1_ExtractVendorInformation.xaml** and bind the argument.

- ○ Next, create a new workflow to navigate to the Dashboard page. The workflow should include a **Click** activity to select the Dashboard page. We can name this workflow **System1_NavigateTo_Dashboard.xaml**.
- ○ Invoke the workflow in the **Process.xaml** file.

- Now that the TaxID value has been retrieved, we need to navigate to the **Download Monthly Report** page. To do that, let us create a new blank sequencenamed **System1_NavigateTo_MonthlyReport**.
- Downloadingexistinginvoices might cause some issues. However,to prevent any exceptions,we should make sure that the environmentis clean everytime the robot starts. To do that, we can delete the **Download Reports** folder mentionedin the **Config** file, and then recreate it from scratch. Open the Init Statein the Main.xamlfile. Add a Sequence named **Clean Up** afterthe invoked KillAllProcesses.xaml file. Let us use two activities: **Delete** and **Create Directory**.
- In the **Properties** panel of the **Delete** activity,set the **Path** field to the value in the **Config** file. This way, you can always have a new empty Data\Temp directorywhen the robot runs. This is what the Clean Up sequence should look like:


Init
Resto
Invoke InitAllSettings workflow
"Framework\InitAllSettings.xaml"
...
Edit Arguments
Import Arguments
Invoke KillAllProcesses workflow
"Framework\KillAllProcesses.xaml"
...
Edit Arguments
Import Arguments
Drop activity here
Clean up
Delete ReportsDownloadP
Create directory ReportsDc

- Next, going back to the **Process.xaml** file, invoke the **System1\System1_NavigateTo_MonthlyReport.xaml** file created above.
- Create a new variable called **ReportYear.** Use an **Assign** activity to set its value to the previous year.
- The next step is to create the yearly report. To do that, we need to create a new blank sequence file named **System1_CreateYearlyReport.xaml**.
    - Start with a relevant annotation. The precondition is that the Monthly Report Page be open in the ACME System1 application.
    - Create three In arguments, as follows:

- **in_TaxID** – to be provided from the main file. It stores the TaxID value.
- **in_Year** – to be provided from the main file. It stores the year for which the report will be created
- **in_ReportsDownloadPath** – the folder where the monthly reports will be downloaded.

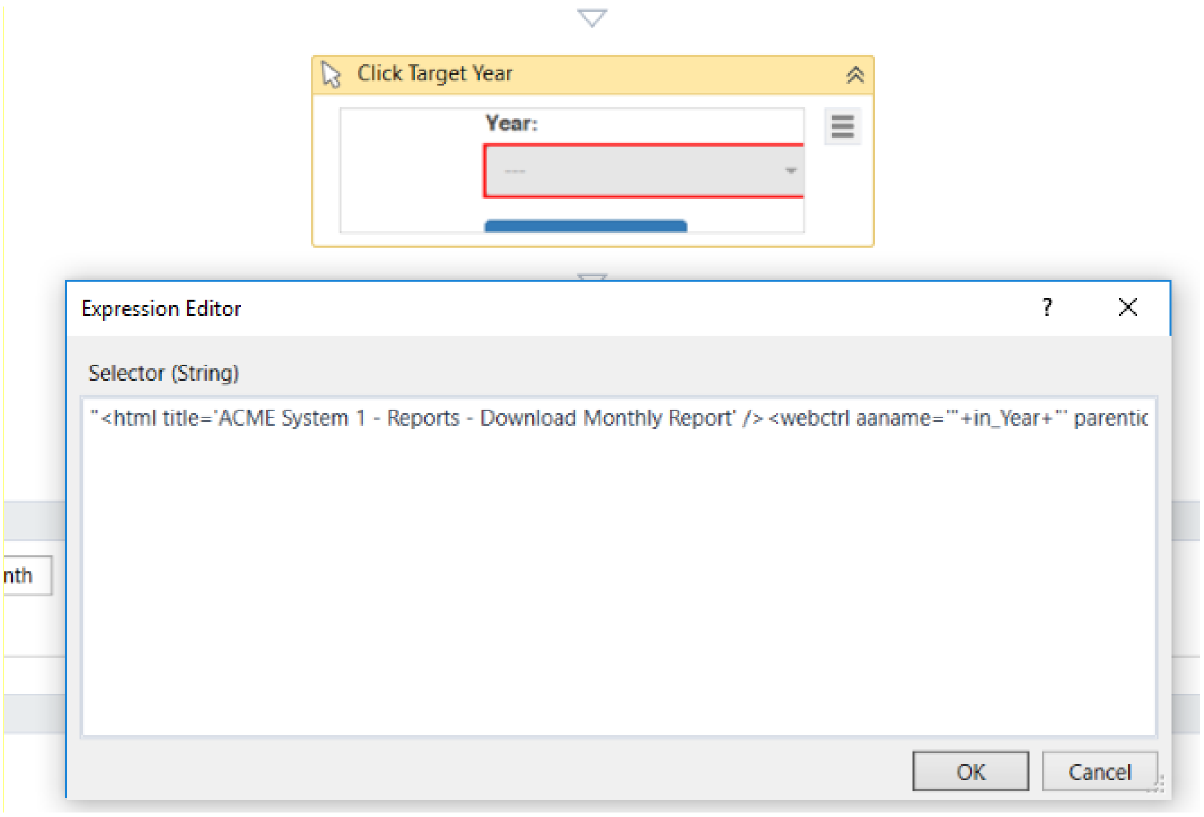
▪ Create one Out argument called **out_YearlyReportPath** to store the path to the yearly report file created after merging all the monthly reports.

▪ Add a new item in the Config file, in the Settings sheet, to indicate the path of the folder where the reports are downloaded. Complete the Name field by typing **ReportsDownloadPath**, and the Value field, by filling in **Data\Temp**.

▪ Create a new Data Table variable called dt_YearlyReport. We use it to merge all the monthly reports. Set its value to **new Datatable** using an **Assign** activity.

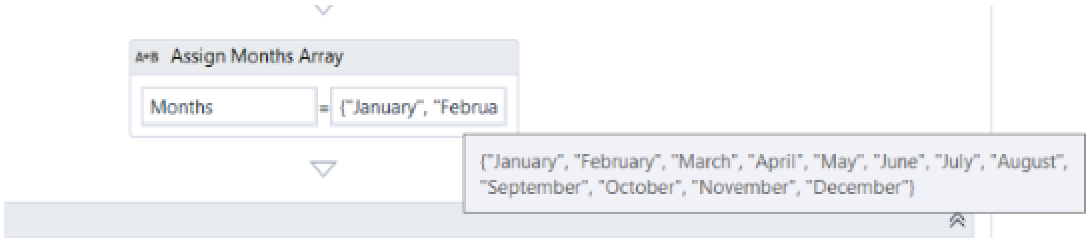
▪ Next, add a **Type Into** activity to type the TaxID value in the ACME System1 application. The MonthlyReport page should already be open at this step.

▪ Add a **Click** activity to select the year. Enable the **Simulate Click** property, and then select the target year. This way, the activity can be executed in the background, even if the drop-down menu is not open and the element is invisible. Change the aaname attribute in the selector to the **in_Year** argument. This is what the **Click** activity and the selector should look like:



```
"<html title='ACME System 1 - Reports - Download Monthly Report' /><webctrl aaname='"+in_Year+"' parentic
```

- Create an Array of Strings variable called **Months**. Use an **Assign** activity to set its values according to the options in the **Month** drop-down list. This is what the **Assign** activity should look like:

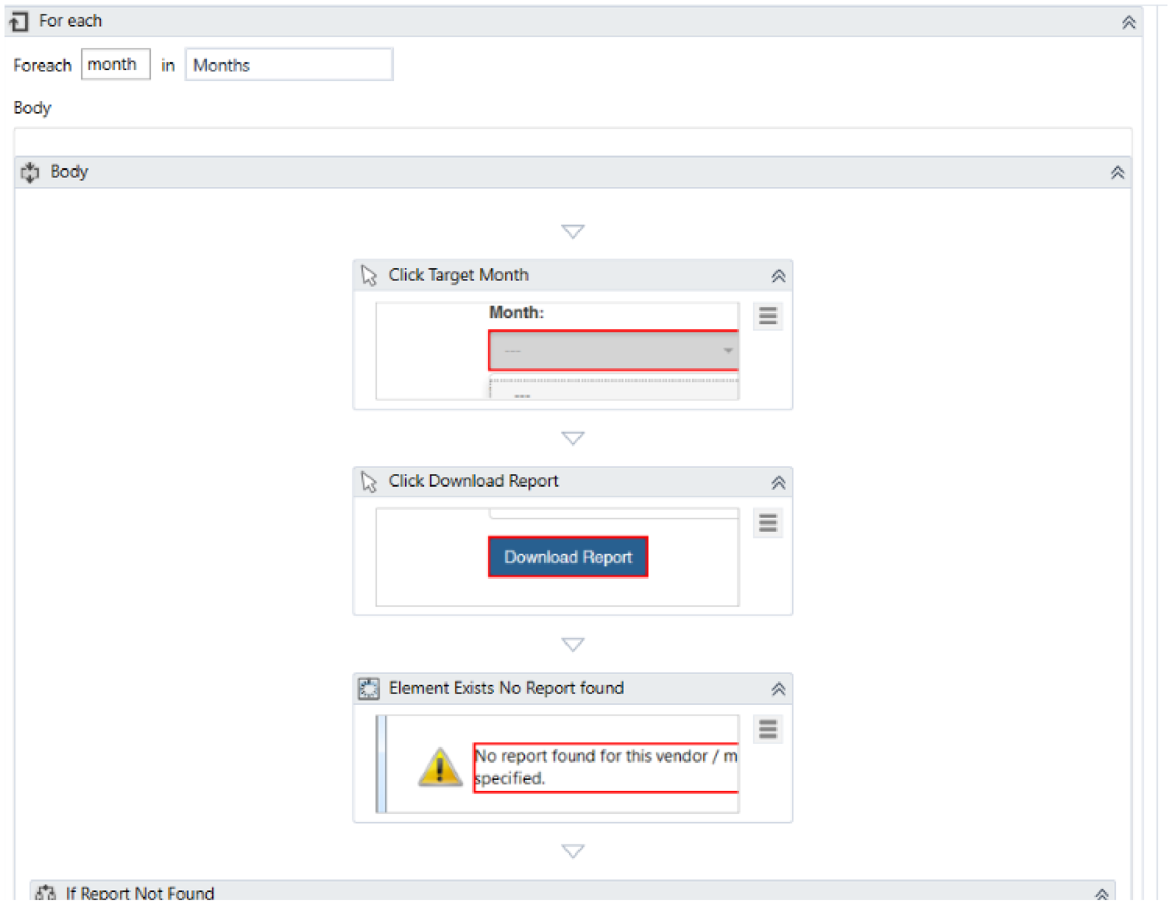


```
{"January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"}
```

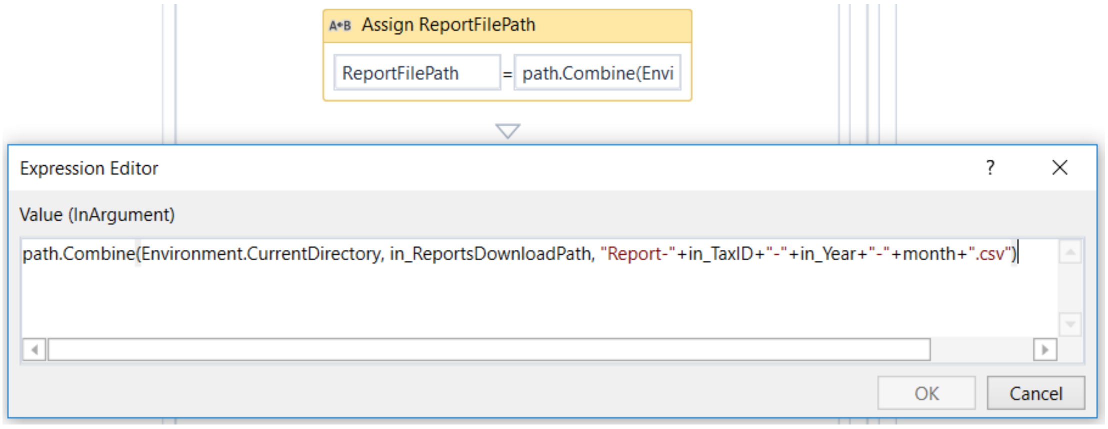
- Next, to download the reports for each month, we need to add a **For Each** activity to iterate through the Months array, select the specified month from the drop-down box, and download the report.
- Inside the **For Each** activity, add a **Click** activity to select the target Month. Edit the selector similarly to how you edited the selector for the Year dropdown, by using the dynamic aaname attribute.
- Next, add a **Click** activity and indicate the Download button. Select the **Simulate Click** property.
- Because some reports don't exist, add an **Element Exists** activity and indicate the label of the pop-up window that is shown in this case.

Create a Boolean variable named ReportNotFound in the O**utput** property.

- This is what the **For Each** activity should look like so far:

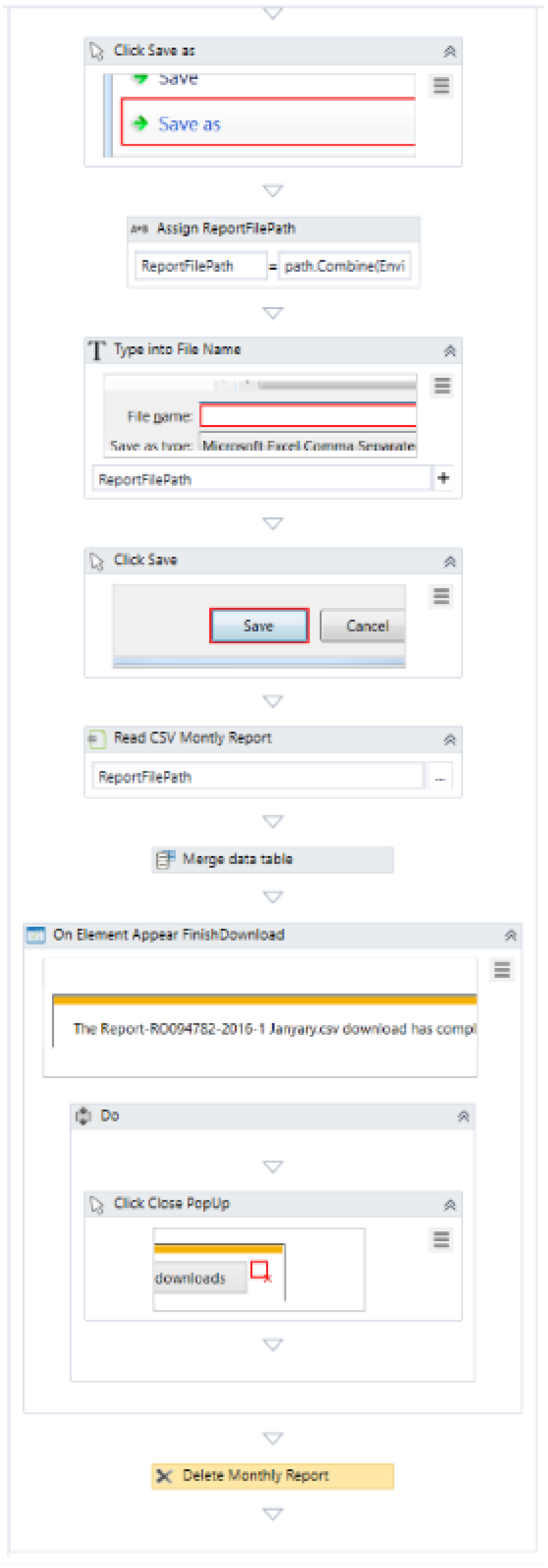


- The next step is to add an **If** activity to check if the report is not found. Use the ReportNotFound variable as the **Condition**. In the **Then** section, add a **Click** activity and set its target to the **OK** button of the pop-up window, to move on to the next month.
- In the Else section, download the Report using the below activities:
  - Add a **Click** activity and direct it towards the **Save as** button. Enable the **Simulate Click** property.
  - Add an **Assign** activity. Create a new variable called ReportFilePath and set its value to the path and the file name of the monthly report that is downloaded in .csv format.



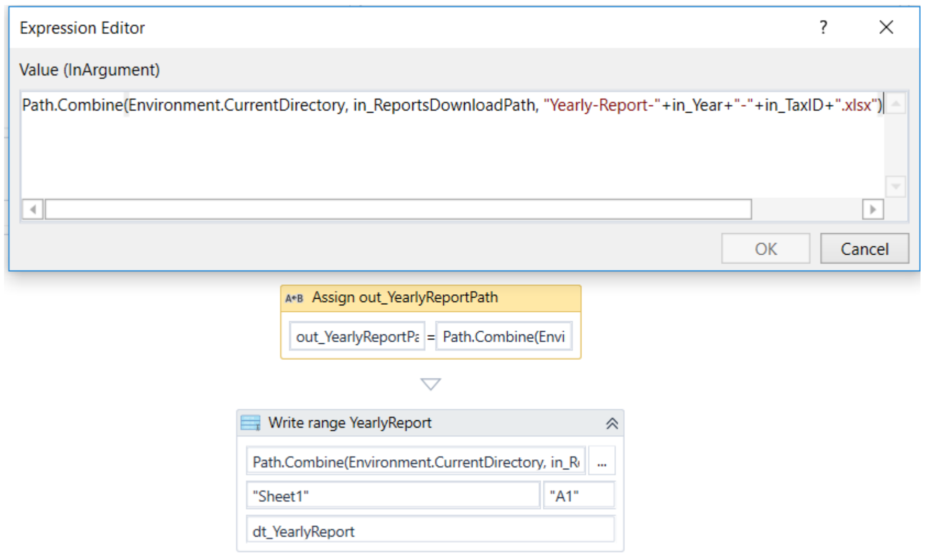
```
path.Combine(Environment.CurrentDirectory, in_ReportsDownloadPath, "Report-"+in_TaxID+"-"+in_Year+"-"+month+".csv")
```

- Using a **Type Into** activity, fill in the value of ReportFilePath in the File Name field of the Save As window. Select the **Simulate** Type property.
- Add a **Click** activity and direct it towards the **Save** button. Enable the **Simulate** Click property.
- Read the csv file just downloaded. In the output property create a data table variable named dt_MonthlyReport.
- Next, using a **Merge Data Table** activity, append the values of **dt_MonthlyReport** to the **dt_YearlyReport** data table.
- Downloading a monthly report takes a variable amount of time. To check that the file was fully downloaded before downloading the next month report, add an **On Element Appear** activity and indicate the download pop-up. Update the selector with wildcard for dynamic attribute values. Enable the **Wait Visible** property.
- In the **Do** section of the **On Element Appear** activity, add a **Click** to close the pop-up. Select the **Simulate Click** property.
- Add a **Delete File** activity to delete the Monthly Report file before downloading the next one.
- This is what the **Else** section should look like:



The Report-RO094782-2016-1 Janyary.csv download has compl

- The next activity in the sequence is **Assign**, which is used to set the valueof the **out_YearlyReportPath** argument.Make sure the name of the the Yearly Report Excel file is in accordance in the model in the PDD file, and the path is the one in the Config file.



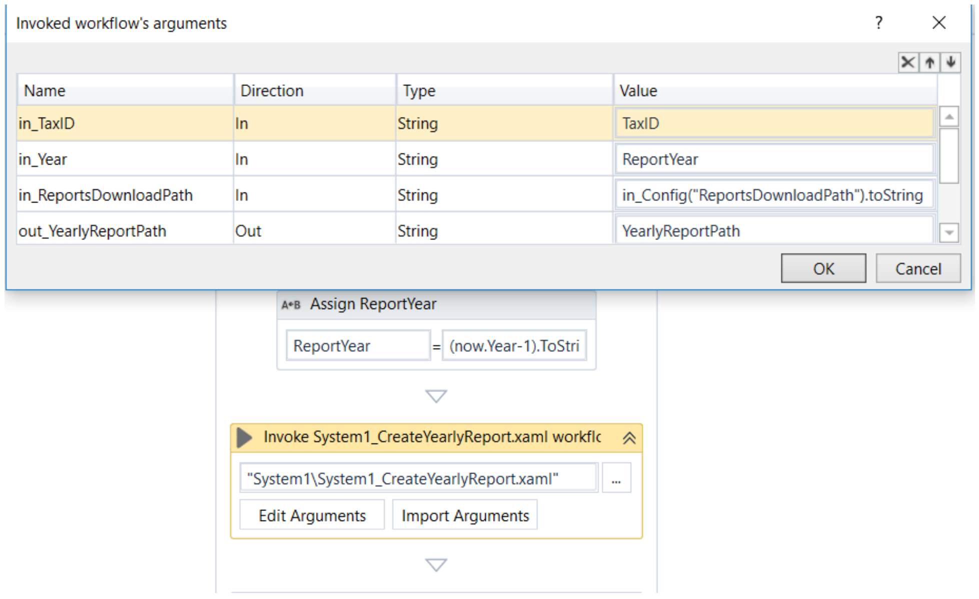
```
Path.Combine(Environment.CurrentDirectory, in_ReportsDownloadPath, "Yearly-Report-"+in_Year+"-"+in_TaxID+".xlsx")
```

```
out_YearlyReportPa = Path.Combine(Envi
```

```
Path.Combine(Environment.CurrentDirectory, in_R
"Sheet1"   "A1"
dt_YearlyReport
```

- Set default values for the In arguments and test the workflow.

- Go back to the **Process** workflow.
  - Invoke the **System1\System1_CreateYearlyReport.xaml** file created above. Import and bind the arguments.
  - Create a String variable called YearlyReportPath.It is used to get the value of the **out_ yearlyReportPath** argument in the previous workflow.
  - This is what the Invoke and the Arguments should look like:

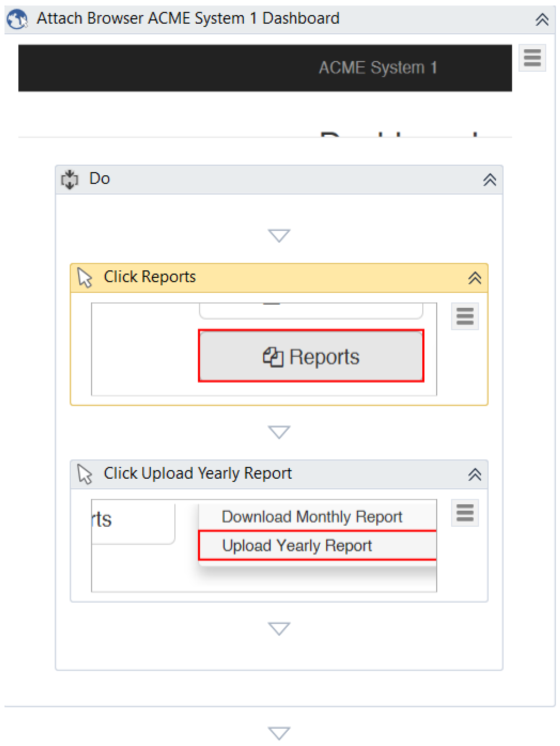


Name	Direction	Type	Value
in_TaxID	In	String	TaxID
in_Year	In	String	ReportYear
in_ReportsDownloadPath	In	String	in_Config("ReportsDownloadPath").toString
out_YearlyReportPath	Out	String	YearlyReportPath

```
ReportYear = (now.Year-1).ToStri
```

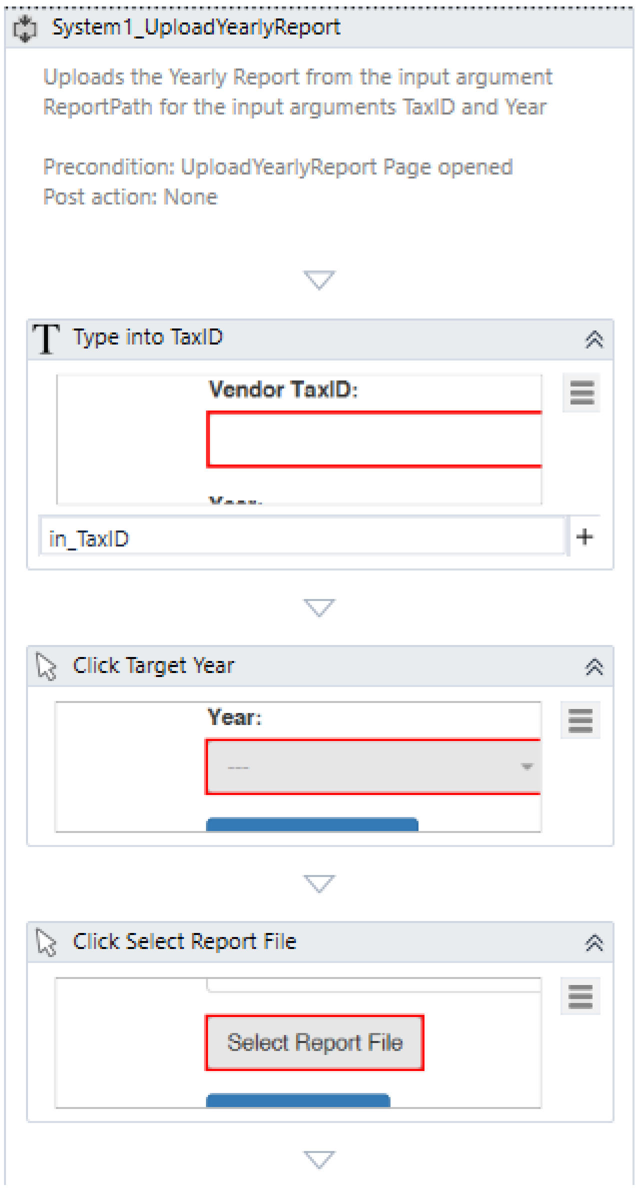
```
Invoke System1_CreateYearlyReport.xaml workfl
"System1\System1_CreateYearlyReport.xaml"
```

- Next, invoke **System1\System1_NavigateTo_Dashboard.xaml** to navigate back to the Dashboard page.
- Now, that we have created the Yearly Report file, we need to navigate to the Reports-Upload Yearly Report page in the ACME System 1 application. To do that we need to create a new blank sequence named **System1_NavigateTo_UploadYearlyReport**.
  - Start the new workflow by adding an annotation. The precondition is that the Dashboard page be open.
  - Add an **Attach Browser** activity and indicate the Dashboard page.
  - Add a **Click** activity to select the **Reports** button. Enable the **SimulateClick** property.
  - Next, add a new **Click** activity to select the **Upload Yearly Report** button. Use **UiExplorer** to click this button, as you did in the **System1_NavigateTo_MonthlyReport.xaml** workflow. Enable the **SimulateClick** property.
  - This is what the workflow should look like:



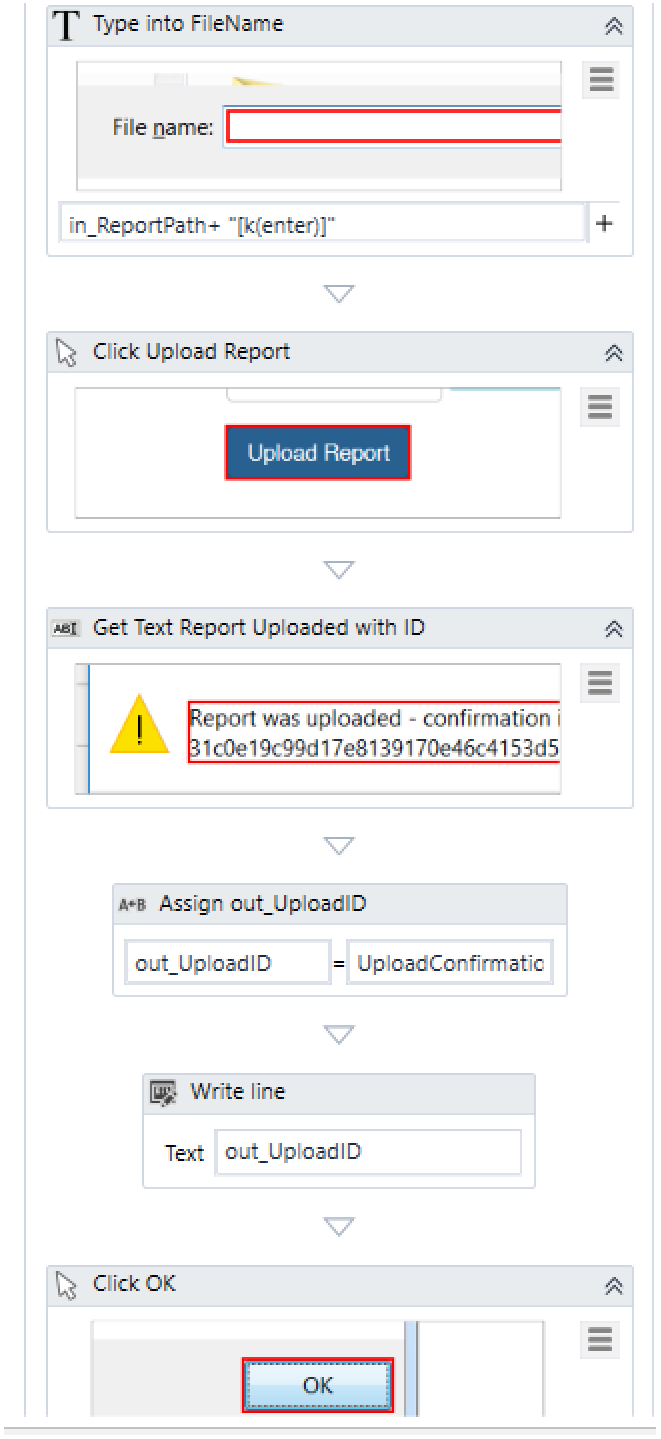
- Go back to the **Process** workflow.
  - Invoke the **System1\System1_NavigateTo_UploadYearlyReport.xaml** file.
  - After navigating to the **Reports - Upload Yearly Report** page, we should upload the yearly report. To do that, let us create a new blank sequence named **System1_UploadYearlyReport**.

- Start the new sequence by adding an annotation. The precondition is that the Reports - Upload Yearly Report page be open.
- The information required to upload the yearly report file consists of the taxID, the file path, and the year. When the upload is performed, a confirmation ID is generated. So, in this workflow, we must use 3 String In arguments: **in_TaxID**, **in_ReportPath**, **in_Year**, and one String out argument: **out_UploadID**.
- Now, that everything is set up, add a **Type Into** activity and indicate the **Vendor TaxID** field. Use the **in_TaxID** argument as text.
- Use a **Click** activity on the **Year** drop-down menu. Update the selector using the **in_Year** argument.
- Use the **Click** activity on the **Select Report File** button.
- This is what the sequence should look like so far:


System1_UploadYearlyReport
Uploads the Yearly Report from the input argument ReportPath for the input arguments TaxID and Year
Precondition: UploadYearlyReport Page opened
Post action: None
Type into TaxID
Vendor TaxID:
in_TaxID
Click Target Year
Year:
Click Select Report File
Select Report File

- Now, the **Choose file to Upload** window is displayed, so we need to use a Type Into activity to set the path to the yearly report file(**in_ReportPath**) and hit Enter.
- Select the Upload button using a **Click** activity.
- A pop-up window with the upload confirmationID is displayed, so let's use a **Get Text** activity to retrieve its value. In the **Output** property, create a variable named UploadConfirmation.
- Set the value of the **out_UploadID** argument to the one of the confirmationID using an **Assign** activity. Use the Substring method to retrieve the value, as follows: UploadConfirmation.Substring("Report was uploaded - confirmationid is ".Length)
- Use a **Click** activity on the **OK** button. Now we are done!

- This is what the last part of the workflow should look like:


Type into FileName

File name:

in_ReportPath+ "[k(enter)]"

Click Upload Report

Upload Report

Get Text Report Uploaded with ID

Report was uploaded - confirmation i
31c0e19c99d17e8139170e46c4153d5

Assign out_UploadID

out_UploadID = UploadConfirmatic

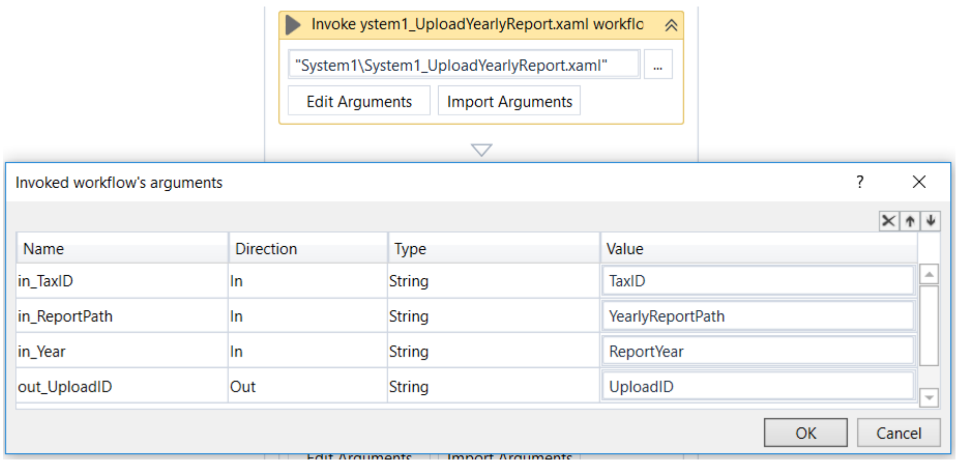
Write line

Text out_UploadID

Click OK

OK

- Go back to the **Process** workflow.
  - Invoke the **System1\System1_UploadYearlyReport.xaml** file and bind the corresponding arguments. Create a variable in the **Process** workflow to store the value of the **out_UploadID** argument. Name the variable UploadID.
  - This is what the invoke should look like:


Invoke ystem1_UploadYearlyReport.xaml workflo

"System1\System1_UploadYearlyReport.xaml"

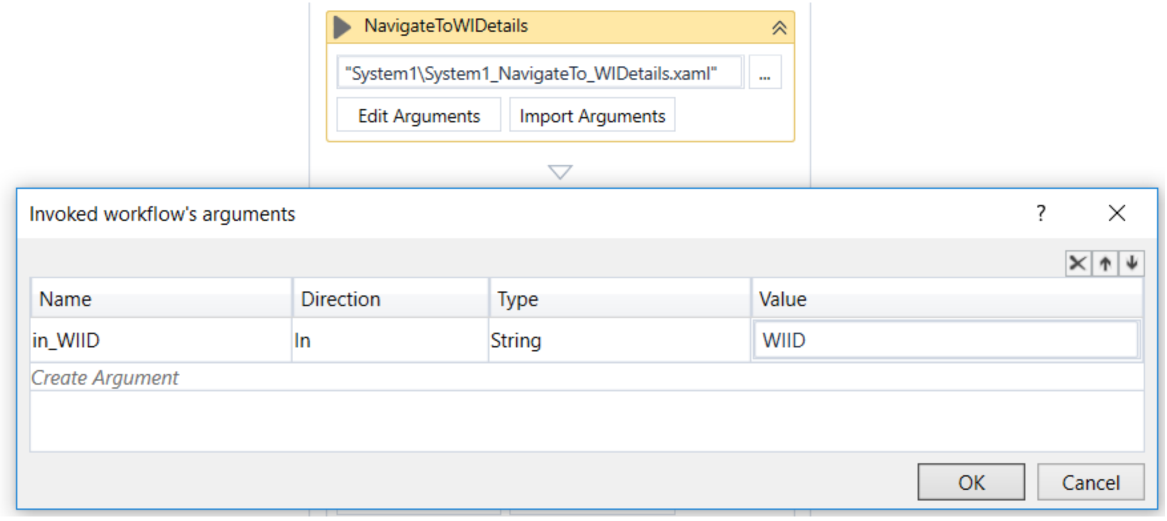
Edit Arguments | Import Arguments

Invoked workflow's arguments

Name	Direction	Type	Value
in_TaxID	In	String	TaxID
in_ReportPath	In	String	YearlyReportPath
in_Year	In	String	ReportYear
out_UploadID	Out	String	UploadID

OK | Cancel

- At this point, we have uploaded the yearly report file, so we should update the status of the Work Items.
- Invoke the **System1\System1_NavigateTo_Dashboard.xaml** file to navigate to Dashboard Page.
- Invoke the **System1\System1_NavigateTo_WIDetails.xaml** file to navigate to the Details page of a specific work item. As you have seen, there is one argument in this workflow - WIID.


NavigateToWIDetails

"System1\System1_NavigateTo_WIDetails.xaml"

Edit Arguments | Import Arguments

Invoked workflow's arguments

Name	Direction	Type	Value
in_WIID	In	String	WIID
Create Argument			

OK | Cancel

- Update the status of the work item to **Complete** by invoking the **System1 \System1_UpdateWorkItem.xaml** file. Be sure to set the correct arguments for the **Comment** and **Status sections**. As mentioned in the PDD file, set the status to **Completed,** and the value of the Comment to **Uploaded with ID uploadID**.
- Finally, we need to leave the application in its initial state, so that we can process next item. To do that, invoke the **System1 \System1_NavigateTo_Dashboard.xaml** file to return to Dashboard page.

- We are done with the process implementation. Next, we need to test the entire process. You should have already tested each individual workflow, right after development, using default values for the arguments.
    - Run the **Main** workflow several times and make sure that it is executed correctly every time. If it isn't, fix the issues and run it again.
    - Use the **Reset test data** option in the User options menu to generate a fresh set of data for testing purposes.
    - Use the **Dispatcher** to upload new items to the Queue if needed.

Process implementation notes.

We started with a **Dispatcher** process that was used for uploading queue items to Orchestrator. Then, we processed each queue item using the **Performer**. Notice that the status of a transaction in the queue changes after it is processed. All the items are independent of one another and can be processed in parallel, using multiple Performer robots.