OGSUiPathFrameworkManual

Prepared By: Architecture Team
Revision 1.1

Table of Content

List of Tables

About The Framework and Its Purpose

The framework is meant to be a template upon which you can build and run unattended business processes, irrespective of process data types and process linearity. At a barebones minimum it provides the developer an easy way to store, read and modify the project configuration data. The framework is composed of small individual workflows which are easy to design and test.

The framework offers a centralized level exception handling and application recovery system which helps all the stakeholders and support team in debugging on the occurrence of any fault.

As the logs generated by any process are the vital components of its report generation, the framework logs messages at each relevant step toward solving a business process and sends those logs to the Orchestrator server.

The framework uses a standardized common logging mechanism to log the messages. Each log message has a standard definition which maintains the consistency among the different processes within the organization. Also, having standard log codes helps the support team while an automation fault occurs.

Using framework, we could define a business process component as the sum of actions by which the data needed for a set of transactions is obtained, processed, and is input into or out of an IT resource.

Introduction

About state machines

As you know, UiPath Studio has 3 types of data flow representations: sequence, flowchart and state machine.

While the framework does contain all 3 data flow representations, we chose the state machine for the main body of the program because it provided a cleaner solution to representing our desired dataflow.

This is how Wikipedia defines a finite state machine:

"A finite-state machine (FSM) or finite-state automaton (FSA, plural: automata), finite automaton, or simply a state machine, is a mathematical model of computation. It is an abstract machine that can be in exactly one of a finite number of states at any given time. The FSM can change from one state to another in response to some external inputs; the change from one state to another is called a transition. An FSM is defined by a list of its states, its initial state, and the conditions for each transition."

Basic rules when using a state machine:

- Since the system can be in only one state at a time, at least one transition condition from a given state to another must become true either by generating a condition in the code running inside the state, an external condition, or a combination of both.
- The transition conditions from each state must be exclusive (two transitions cannot be true at the same time, thus allowing two possible paths of exit from a state).
- Another rule that is agreed upon is that no heavy processing must be done in the Transition actions. All processing should be done inside the state.

Going back to the first chapter, the problems we needed to solve with this template were:

1. Store and read project configuration data
2. Separate *IT resource* start, usage and end
    a. For all retried *transactions* , restart the *IT resource*
3. Implement a robust exception handling and transaction retry scheme
    a. Capture exceptions by type and log each and every exception using the OGS Common Logging module.
    b. Use exception type to retry transactions that failed with an application exception
4. Capture and transmit logging for all exceptions and relevant transaction information.

Framework Component Functions

Table 1 shows the calling structure of the framework. That is, which workflows are called, the order in which they are called, and the State of the main state machine where you can find the workflow invoke.

Table 1 Component Call Tree Structure

Component File Names and Locations	State where File is Called
Init Config\InitAllSettings.xaml	Init Config
Resuable Components\SetSecureKey.xaml	Init Config
Init Application\InitAllApplications.xaml	Init Application
Set Data\SetData.xaml	Set Data
Resuable Components\EncryptData.xaml	Set Data
Get Data\GetData.xaml	Get Data
Resuable Components\DecryptData.xaml	Get Data
Process Data\ProcessData.xaml	Process Data
Process Data\SetTransactionStatus.xaml	Process Data
Close Application\CloseAllApplications.xaml	Close Application
Kill Process\KillAllProcesses.xaml	Kill Process
CommonLogg.xaml	All

Global Variables

The global variables are those variables whose scope is the main program, or main workflow. They can be found in the Main.xaml workflow file, by first clicking anywhere inside the main state machine and then clicking the variables pane. Table 2 is a list of the project's global variables.

These are used to store information that will be available throughout the runtime of the process. It is important to understand where each variable is written and where it is read.

Table 2 Global variables table

Name	Data Type	Is Written in Workflows	Is Read in Workflows
TransactionItem	QueueItem	Main.xaml GetData.xaml DecryptData.xaml	Main.xaml ProcessData.xaml SetTransactionStatus.xaml DecryptData.xaml
SystemError	System.Exception	Main.xaml InitAllSettings.xaml SetData.xaml	Main.xaml SetTransactionStatus.xaml
BusinessRuleException	UiPath.Core.BusinessRule Exception	Main.xaml InitAllSettings.xaml InitAllApplications.xaml	Main.xaml SetTransactionStatus.xaml
Config	Dictionary(String,Object)	Main.xaml InitAllSettings.xaml	Main.xaml InitAlLApplications.xaml SetData.xaml GetData.xaml ProcessData.xaml SetTransactionStatus.xaml CloseApplications.xaml KillAllProcesses.xaml
OutputJSON	String	Main.xaml GetData.xaml	SetTransactionStatus.xaml
FirstRun	Boolean	Main.xaml	Main.xaml
IOException	IOException	Main.xaml	Main.xaml
AppInstance	List(of Objects)	Main.xaml InitAllApplications.xaml CloseApplications.xaml	CloseApplications.xaml

Name	Data Type	Is Written in Workflows	Is Read in Workflows
TransactionNumber	Int32	Main.xaml SetTransactionStatus.xaml	GetData.xaml
TransactionID	String	Main.xaml GetData.xaml	SetTransactionStatus.xaml
RetryNumber	Int32	Main.xaml SetTransactionStatus.xaml	Main.xaml SetTransactionStatus.xaml

Init Config

Reads project configuration file. Initialize all the global variables. Check whether required files and queues exist. Reads and sets Securekey if *"SecurityEnabled"* is set to *"True"* in *Config.json*

Precondition: N/A

Post condition: Config file is loaded and project dependencies are checked.

InitAllSettings.xaml workflow

This workflow outputs a settings Dictionary with key/value pairs to be used in the project. Settings are read from local config file and if needed are fetched from Orchestrator assets.

Table 3 InitAllSettings.xaml Arguments and Values

Name	Data Type	Argument Type	Values
in_ConfigFile	String	In	"Data\Config.json"
Out_Config	Dictionary(String,Object)	Out	Config
out_SystemError	System.Exception	Out	SystemError
out_BusinessRuleException	UiPath.Core.BusinessRuleException	Out	BusinessRuleException

Init Config Transitions

At the end of the *"Init Config"* we should have read the configuration file and checked the project dependencies.

Table 4 Init Config Transitions

Name	Condition	Transition To State	Description
Success	SystemError is Nothing and BusinessRuleException is Nothing and IOException is Nothing	Init Applications	If during initialization we have no error than open Applications
Error	SystemError isNot Nothing or BusinessRuleException IsNot Nothing or IOException IsNot Nothing	Exception Handling	If any type of exception occur during *"Init Config"* we move on to exception handling, log errors and end the process.

Init Applications

Initialize the applications. Validates applications are open and running. Login’s into the target application if required.

Precondition: Config file must be loaded.

Post condition: Target Applications are up and running.

InitAllApplications.xaml workflow

This workflow will try to open all the required applications and throw exception if any error occur during opening applications.

Table 5 InitAllApplications.xaml Arguments and Values

Name	Data Type	Argument Type	Values
in_Config	Dictionary(String,Object)	In	Config
io_AppInstance	List(of Object)	In/Out	AppInstance

Init Application Transitions

At the end of the *“Init Application”* the bot should have opened the desired applications and should have checked if all applications are up and running.

Table 6 Init Application Transitions

Name	Condition	Transition To State	Description
FirstRun	SystemError is Nothing and BusinessRuleException is Nothing and FirstRun = True	Set Data	If all applications opened successfully and it was the first time bot was running.
ReRun	FirstRun = False and SystemError is Nothing and BusinessRuleException is Nothing	Get Data	If all applications opened successfully and data has been already set once.
Error	SystemError IsNot Nothing or BusinessRuleException IsNot Nothing	Exception Handling	If any type of error occurs during *“Init Application”* we move on to exception handling, log errors and end the process.

Set Data

Adds data items to the queue. If *SecurityEnabled* is set to *True* will encrypt the specified columns of the inventory and then add them to the queue.

Precondition: Project Dependency should have been met.

Post Condition: Data is added to the Orchestrator Queue.

SetData.xaml workflow

This workflow will add the data items to the Orchestrator Queue.

Table 7 SetData.xaml Arguments and Values

Name	Data Type	Argument Type	Values
in_Config	Dictionary(String,Object)	In	Config
out_SystemError	System.Exception	Out	SystemError

If *SecurityEnabled* is set to *True “EncryptData.xaml”* workflow is invoked, which encrypts the specified columns of the inventory and then adds the data items to the queue. The details of the workflow can be found here.

Set Data Transitions

At the end of the *“Set Data”* the bot should have added all the data items to the Orchestrator Queue.

Table 8 Set Data Transitions

Name	Condition	Transition To State	Description
Success	SystemError is Nothing and BusinessRuleException is Nothing	Get Data	If all the data items were added successfully move onto Get Data state.
Error	SystemError IsNot Nothing or BusinessRuleException isNot Nothing	Exception Handling	If any type of exception occur during *“Set Data”* we move on to exception handling, log errors and end the process.

Get Data

Get next data item from the Queue.Set the TransactionItem.

Precondition: DataItems should have been added to the queue.

Post Condition: Gets TransactionItem to be processed.

GetData.xaml workflow

This workflow will get the data items from the Orchestrator Queue.

Table 9 GetData.xaml Arguments and Values

Name	Data Type	Argument Type	Values
in_TransactionNumber	Int32	In	TransactionNumber
in_Config	Dictionary(String,Object)	In	Config
out_TransactionItem	QueueItem	Out	TransactionItem
out_TransactionField1	String	Out	TransactionField1
out_TransactionID	String	Out	TransactionID
io_TransactionData	DataTable	In/Out	TransactionData
io_outputJson	String	In/Out	OutputJSON

Get Data Transitions

From the *"Get Data"* state we have two possible outcomes. The first is that we have obtained new transaction data in TransactionItem variable and so we move on to the *"Process Data"* state. The other outcome is that we have exhausted our data collection, and as a consequence of this, we have set the TransactionItem variable to Nothing in which case we cannot get Data.

Table 10 Get Data Transitions

Name	Condition	Transition To State	Description
New Data Item	TransactionItem IsNot Nothing	Process Data	If TransactionItem contains data, process it.
No Data	TransactionItem Is Nothing	Close Application	If TransactionItem is Nothing, goto *"Close Application"*

Process Data

Process the current Data Item according to the Business Needs. The process may have succeeded or failed. The process if failed can be due to Business Rule Exception or System Error. If System Error occurs the data item is retried. Additionally performs decryption of the data item if *"SecurityEnabled"* is set to *True.*

Precondition: TransactionItem should be set.

Post Condition: Transaction may succeed or failed and the status of the particular data item is either set to Successful or Failed.

ProcessData.xaml workflow

In this file all other process specific files will be invoked. If an application exception occurs, the current transaction can be retried. If a BusinessRuleException is thrown, the transaction will be skipped.

Table 11 ProcessData.xaml Arguments and Values

Name	Data Type	Argument Type	Values
in_Config	Dictionary(String,Object)	In	Config
in_TransactionItem	QueueItem	In	TransactionItem

If *SecurityEnabled* is set to *True "DecryptData.xaml"* workflow is invoked, which decrypts the *TransactionItem* which is fetched from the queue. The details of the workflow can be found here.

SetTransactionStatus.xaml workflow

This workflow sets the TransactionStatus and Logs that status and details in output field of the queue item.

The flowchart branches out into the three possible Transaction Statuses: Success, Business Exception and Application Exception.

Each branch analyzes the type of content of TransactionItem. If it's not empty and is a QueueItem, then it means we are using a Orchestrator queue, so we must call the "Set Transaction Status" activity to inform Orchestrator about the outcome of our transaction. If TransactionItem is not a QueueItem, we can skip passing it and the "Set Transaction Status" activity will not be triggered.

Table 12 SetTransactionStatus.xaml Arguments and Values

Name	Data Type	Argument Type	Values
io_TransactionNumber	Int32	In/Out	TransactionNumber
io_RetryNumber	Int32	In/Out	RetryNumber
in_Config	Dictionary(String,Object)	In	Config
in_TransactionItem	QueueItem	In	TransactionItem
in_TransactionField1	String	In	TransactionField1
in_TransactionID	String	In	TransactionID
in_OutputJson	String	In	OutputJSON
in_SystemError	System.Exception	In	SystemError
in_BusinessRuleException	UiPath.Core.BusinessRule Exception	In	BusinessRuleException

<u>TakeScreenshot.xaml workflow</u>

Usage: Set in_Folder to the folder Name where you want to save the screenshot. Alternatively, supply the full path including file name in io_FilePath. Description: This workflow captures a screenshot and logs it's name and location. It then saves it. If io_FilePath is empty, it will try to save the picture in in_Folder. It uses .png extension.

Table 13 TakeScreenshot.xaml Arguments and Values

Name	Data Type	Argument Type	Values
in_Folder	String	In	in_Config("ExScreenshotsFolderPath").ToString
io_FilePath	String	In/Out	
In_LogFile	String		in_Config("LogFile").ToString

<u>Process Data Transitions</u>

The Process Data State is where the processing work for all transactions takes place. After the ProcessData.xaml file is executed, we look for an exception having been generated (either Business Rule or Application). In case no exception was caught, it means we were successful. The SetTransactionStatus.xaml workflow manages both the logging of the ProcessData.xaml output, as well as the management of the next transaction or the retrying of the current one. This workflow is where TransactionNumber and RetryNumber are written, allowing for automatic retry in case of an Application Exception.

Table 14 Process Data Transitions

Name	Condition	Transition To State	Description
Success	SystemError Is Nothing And BusinessRuleException is Nothing	Get Data	If no exception occurs, go to *"Get Data"* to get new TransactionItem to process.
Rule Exception	BusinessRuleException IsNot Nothing	Get Data	If any Rule Exception occurs log it and go to "Get Data" state to get new TransactionItem.
Process Error	SystemError IsNot Nothing	Close Application	If any system error occurs, try to close all applications by moving to *"Close Applications"*

Close Application

Close the instances of the open applications.

PreCondition: Each instance should be present in the list of instances.

PostConditon: The opened instances are closed.

<u>CloseAllApplications.xaml workflow</u>

It will close all the applications opened by our business process.

Table 15 CloseAllApplications.xaml Arguments and Values

Name	Data Type	Argument Type	Values
in_Config	Dictionary(String,Object)	In	Config
io_AppInstnace	List(of Object)	In/Out	AppInstance

<u>Close Application Transitions</u>

There could be 3 possible outcomes of this state. First, all applications closed without any error and there is no more data items left to be processed. Second, some applications may not have closed properly due to some error and need to be killed. And third, all applications closed successfully but there still data items left that are to be processed.

Table 16 Close Applications Transitions

Name	Condition	Transition To State	Description
Close Successful and No Data	SystemError is Nothing and TransactionItem is Nothing	Finalize	If all applications closed successfully and no more data item to process move to *"Finalize"* state for finishing up the process.
Close Unsuscessful	SystemError IsNot Nothing	Exception Handling	If any application throws error while closing, move to *"Exception Handling"* state to log the error.
Close Successful and On Retry	SystemError is Nothing and TransactionItem IsNot Nothing	Init Applications	If all applications closed successfully but still there is data items which are to be processed move to *"Init Applications"* to start the applications and process all over again.

Exception Handling

Checks the type of exception which has occured and performs logging of the same accordingly.

PreCondition: N/A

PostCondition: Kill the processes or Finalize.

Exception Handling Transitions

There could be two possible transitions from "*Exception Handling"* state i.e. to move to *"Kill Process"* state or to move to "Finalize" state depending on the type of error which occurs.

Table 17 Exception Handling Transitions

Name	Condition	Transition To State	Description
Log File Present	IOException is Nothing and SystemError is Nothing	Kill Process	If the error is not due to missing of *CommonLogg.xaml* move to *Kill Process* state.
Log File Not Present	IOException IsNot Nothing or SystemError IsNot Nothing	Finalize	If *CommonLogg.xaml* is not found stop the execution by moving to *Finalize* state.

Kill Process

Kill the existing process if any error occurs or no more data is left for processing.

KillAllProcessData.xaml workflow

It will close all the process related to our business process. It can be due to some error or when no data is left for processing.

Table 18 KillAllProcessData.xaml Arguments and Values

Name	Data Type	Argument Type	Values
in_Config	Dictionary(String,Object)	In	Config

Kill Process Transitions

There could be two possible transitions from "*Kill Process"* state i.e. to move to *"Finalize"* state or to move to *"Init Application"* state.

Table 19 Kill Process Transition

Name	Condition	Transition To State	Description
Retry	TransactionItem IsNot Nothing	Init Applications	If kill process was called due to some error, that means there is more data to processed so go to *"Init Applications"*.
No Data	TransactionItem is Nothing	Finalize	If no data is left to be processed move to *"Finalize"* state to clean up the code and finish the process.

Finalize

In this state code cleanup is done and the process comes to end.

Additional Features

Common Logging

The framework performs logging by a centralized logging mechanism. The purpose of Centralized logging is to standardize logging for all processes. This allows us to be consistent across the various projects we undertake and ideally removes duplication of flows or the developers. By having predefined log codes we can ensure consistency. Should a new code be needed, the responsibility will be on the developer to add it to the appropriate JSON file (outlined below) and not to hard-code log messages.

CommonLogg.xaml workflow

Whenever logging is performed in the framework it is done by invoking the CommonLogg.xaml workflow file. The user needs to change only the logCode and ExtraLog field. JSONPath variable should not be changed.

Table 20 CommonLogg.xaml Arguments and Values

Name	Data Type	Argument Type	Values
logCode	String	In	Log Code to be logged
ExtraLog	String	In	Extra Message to be logged
JSONPath	String	In	Log File Location

Logged Messages

The following is a list of all the message logs within the framework, the places where the corresponding "*CommonLogg.xaml*" workflow is called, the message, the log codes, extra log field, and the level of the log (info, warn, error, fatal).

Table 21 Log Message

Log Codes	Message	Extra Log	State	Workflow	Level
1012	Reading of project configuration file completed.		Init Config	Main	Info
1009	Initialization of project variables started.		Init Config	Main	Info
1010	Initialization of project variables completed.		Init Config	Main	Info

Log Codes	Message	Extra Log	State	Workflow	Level
1013	Project dependency check started.		Init Config	Main	Info
1014	Project dependency check completed.		Init Config	Main	Info
1008	Initialization of project configuration completed.		Init Config	Main	Info
1203	Initialization of project encountered an unexpected exception.		Init Config	Main	Error
1016	Initialization of project applications started.		Init Applications	InitAllApplications	Info
1206	Encountered an unexpected exception while initialising project applications.		Init Applications	Main	Fatal
1017	Initialization of project applications completed.		Init Applications	InitAllApplications	Info
1039	Encryption Started		Set Data	Set Data	Info
1040	Encryption Completed		Set Data	Set Data	Info
1218	Encountered an exception performing encryption.	User Specific Message	Set Data	SetData	Fatal
1200	The process has encountered an unexpected value.	User Specific Message	Set Data	SetData	Error
1020	Get data started.		Set Data	Main	Info
1034	Stop process requested		Get Data	Main	Info
1208	Encountered an unexpected exception while getting next data.	Exception	Get Data	Main	Fatal
1110	Item Processing Started.	User Specific Message	Get Data	GetData	Info
1021	Get data completed.		Get Data	Main	Info
1041	Decryption Started		Process Data	Process Data	Info

Log Codes	Message	Extra Log	State	Workflow	Level
1042	Decryption Completed		Process Data	Process Data	Info
1219	Encountered an exception while performing decryption.	User Specific Message	Process Data	Process Data	Fatal
1111	Outcome ID	User Specific Message	Process Data	SetTransactionStatus	Info
1112	Outcome Status	User Specific Message	Process Data	SetTransactionStatus	Info
1113	Outcome Status Description	User Specific Message	Process Data	SetTransactionStatus	Info
1209	Encountered an unexpected exception while processing current data item.	in_SystemError	Process Data	SetTransactionStatus	Fatal
1215	Automation exiting due to maximum retry count being exceeded.		Process Data	SetTransactionStatus	Fatal
1006	A screenshot has been saved to a shared folder. See extraData field for image location.	User Specific Message	Process Data	TakeScreenshot	Info
1101	The sub-flow failed due to an Error.	User Specific Message	Process Data	Main	Error
1209	Encountered an unexpected exception while processing current data item.		Process Data	Main	Fatal
1028	Close applications started.		Close Application	CloseAllApplications	Info
1029	Close applications completed.		Close Application	CloseAllApplications	Info
1213	Encountered an exception while closing applications.		Close Application	Main	Fatal
1002	The process has terminated	SystemError	Exception	Main	Fatal

	early due to an Error.		Handling		
Log Codes	**Message**	**Extra Log**	**State**	**Workflow**	**Level**
1003	The process has terminated early due to an exception.	BusinessRule Exception	Exception Handling	Main	Fatal
1032	Kill process started.		Kill Processes	KillAllProcesses	Info
1033	Kill process completed.		Kill Processes	KillAllProcesses	Info
1030	Finalize process started.		Finalize	Main	Info

Encryption & Decryption Utility

This features helps in protecting the **Confidential/Private** data by encrypting it and then saving into the queue. It encrypts the specified columns of the inventory and then uploads all the data items into the queue. The developer needs to set the value of *"SecurityEnabled"* to *"True"* in the *Config.Josn* and specify the columns to be encrypted in the Orchestrator.

While retrieving the Transaction Items from queue it decrypts them so that they can be processed.

It uses the below 2 workflow to serve its purpose.

Encrypt.xaml workflow

This workflow encrypts the particular columns of the input data table and provides the user with the new Data Table with encrypted values. It takes in Data table and Config dictionary as inputs. The names of columns which are required to be encrypted and the key to be used is stored as Asset in Orchestrator and read and set in the Config Dictionary Object in the framework. The user should pass these values to the EncryptData.xaml while invoking. Also the user should pass the input data table to the workflow whose columns are to be encrypted.

The values in_SecureColumn and in_Key were passed as input to SetData.xaml.

Table 22 EncryptData.xaml Arguments and Values

Name	Data Type	Argument Type	Values
io_DataTable	DataTable	In/Out	InputTable
in_Config	Dictionary(String,Object)	In	in_Config

Decrypt.xaml workflow

This workflow decrypts the particular fields of the TranscationItem (QueueItem) based on the column names present in the SecureColumn Object of the Config Dictionary. The workflow takes the TransactionItem and Config dictionary as inputs. The encrypted fields are decrypted and the values of the particular fields are updated in the TransactionItem and passed as output.

Table 23 DecryptData.xaml Arguments and Values

Name	Data Type	Argument Type	Values
io_TransactionItem	QueueItem	In/Out	In_TransactionItem
in_Config	Dictionary(String,Object)	In	in_Config

Set Secure Key

This feature sets the Secure Key in Orchestrator if *"SecurityEnabled"* is set *True*. This key is used to encrypt and decrypt the *"SecureColumn"* of the Inventory.

SetSecureKey.xaml workflow

This workflow will load the secure key into the Config dictionary object of the project. It fetches the key from the Orchestrator. If the key is the default key i.e. ***"1234"*** then new random key is generated and stored in the orchestrator or else the key is fetched from the orchestrator and loaded into the Config dictionary. It is invoked if the "*SecurityEnabled*" is set to "*True*" for the project in the Config file of the project.

Table 24 SetSecureKey.xaml Arguments and Values

Name	Data Type	Argument Type	Values
in_Password	SecureString	In	Password
in_CredentialName	String	In	item.Item("Name").ToString
Out_Key	String	Out	out_Config(item.Item("Name").ToString)

Steps to Use

1. Create a JSON Object in Config file as follows :
   {
   "Name": "ProjectKey",
   "Type": "Credential" // This credential must be present if SecurityEnabled is set to True.
   }
   The name of the JSON Object should always end with the word "Key".
2. Create a Credential asset in the Orchestrator with the same name as of the JSON Object i.e. here *ProjectKey*. The username in the Credential asset should always be *"Key"* and the value by default should be ***"1234".***
3. Set the default value of "*SecurityEnabled" to True* in Config.json.

File Copy

This feature copies file/files from one folder to another based on the filename or a regex pattern. As it is a reusable component it can be utilized anywhere inside the framework.

FileCopy.xaml workflow

This workflow copies file or files from source to destination as specified in the JSON. User can provide either the filename or a regex pattern to copy files from the source to destination. Also the user can set "CreateDestination" in JSON to True to create destination if not present. And also the user has option to delete files from source. The JSON should be kept in asset as text.

Table 25 FileCopy.xaml Arguments and Values

Name	Data Type	Argument Type	Values
in_Config	Dictionary(String,Object)	In	in_Config

Steps to Use

1. Create a JSON file as follows :

```
{
   // Example if files is to be copied from NAS Drive and filename is specified.
   "step1": [
      {
         "Source":
"\\\\nas00782pn\\data\\TEG_RPA_SHARE_DRIVE\\Tech_maturity\\CommonLog",
         "Destination":
"\\\\nas00912pn\\Data\\RPA_DEV_SHARE_DRIVE\\Technical_Maturity\\CommomLogging"
,
         "Regex": "",
         "FileName": "challenge.xlsx",
         "DeleteSource": false,
         "CreateDestination": true
      }
   ],
}
```

   Same structure should be followed while creating the json file.

2. Create a JSON Object in Config file as follows :

```
{
      "Name": "CopyJSON",
      "Type": "Asset"
```

}

3. Create an asset in the Orchestrator with the same name as of the JSON Object i.e. here *CopyJSON*.
4. Upload the JSON file into the above created asset.
5. Invoke the workflow where ever required by passing the required arguments.

Directory Creation

This feature creates directories as per user needs. The user can specify where to create directories in the asset. And the directory structure as JSON in the asset and directories will be created accordingly.

DirectoryCreation.xaml workflow

This workflow creates directories according to the structure defined in the JSON File. The user should provide the base destination where the directories are to be created in the asset. The JSON should be kept in asset as text. It is capable of creating directories upto N Level as per user needs. It takes in an input argument "in_ProcessName" which is the name of the process for which directory structure is to be created.

Table 26 DirectoryCreation.xaml Arguments and Values

Name	Data Type	Argument Type	Values
in_Config	Dictionary(String,Object)	In	in_Config
in_ProcessName	String	In	User Specific

Steps to Use

1. Create a JSON file as follows :

```
{
   "SubProcess1": {
      "Part A": "Data",
      "Part B": [
         {
            "Q1": [
               "January",
               "February",
               "March"
            ],
            "Q2": [
               "April",
               "May",
               "June"
            ],
            "Q3": [
               "July",
               "August",
          "September"
        ],
        "Q4": [
          "October",
          "November",
          "December"
        ]
      }
    ]
  }
}
```

The base object should always be JSON Object followed by any number of JSON Object and JSON Array.

2. Create JSON Object’s in Config file as follows :

```
{
    "Name": "DirectoryDestination",
    "Type": "Asset"
}
```

and

```
{
    "Name": "DirectoryStructure",
    "Type": "Asset"
}
```

3. Create asset’s in the Orchestrator with the same name as of the JSON Object i.e. here *DirectoryDestination* and *DirectoryStructure*.
4. Upload the JSON file into the *DirectoryStructure* asset.
5. Invoke the workflow where ever required by passing the required arguments.