LECTURE 07. ROBOT ENTERPRISE FRAMEWORK

Robotic Process Automation
[14 November 2023]

Elective Course, 2023-2024, Fall Semester

Camelia Chisăliţă-Creţu, Lecturer PhD
Babeş-Bolyai University

Acknowledgements

This course is presented to our Faculty with the support of UiPath Romania.

UiPath

Robotic Process Automation

Contents

- State Machine
  - Details. Example
  - Demo 1. State Machine
- REFramework
  - Details
  - Types of Processes
  - Demo 2. REF
- REFramework Architecture
  - States
  - Predefined workflows
  - Transitions
  - Shared variables
- REFramework Features
  - Config File
  - Exception Handling
  - Logging
- REFramework Implementation
  - Without Orchestrator
  - With Orchestrator
- Best Practices
- Next Lecture…
- References

State Machine. Details

- **State Machine** is
  - an abstract machine consisting of a **finite number of pre-defined states**;
- at any point, based on the external inputs and conditions verified, it can be in only one of the states;
- E.g.: the vending machine, the elevator, or the traffic lights.
- there are two activities that are specific to state machines:

State

- it serves as an initial state;
- it contains three areas:
  - **Entry**
  - **Exit**
  - **Transitions**

Final State

- it ends a **State Machine** cycle;
- it contains one area:
  - **Entry**

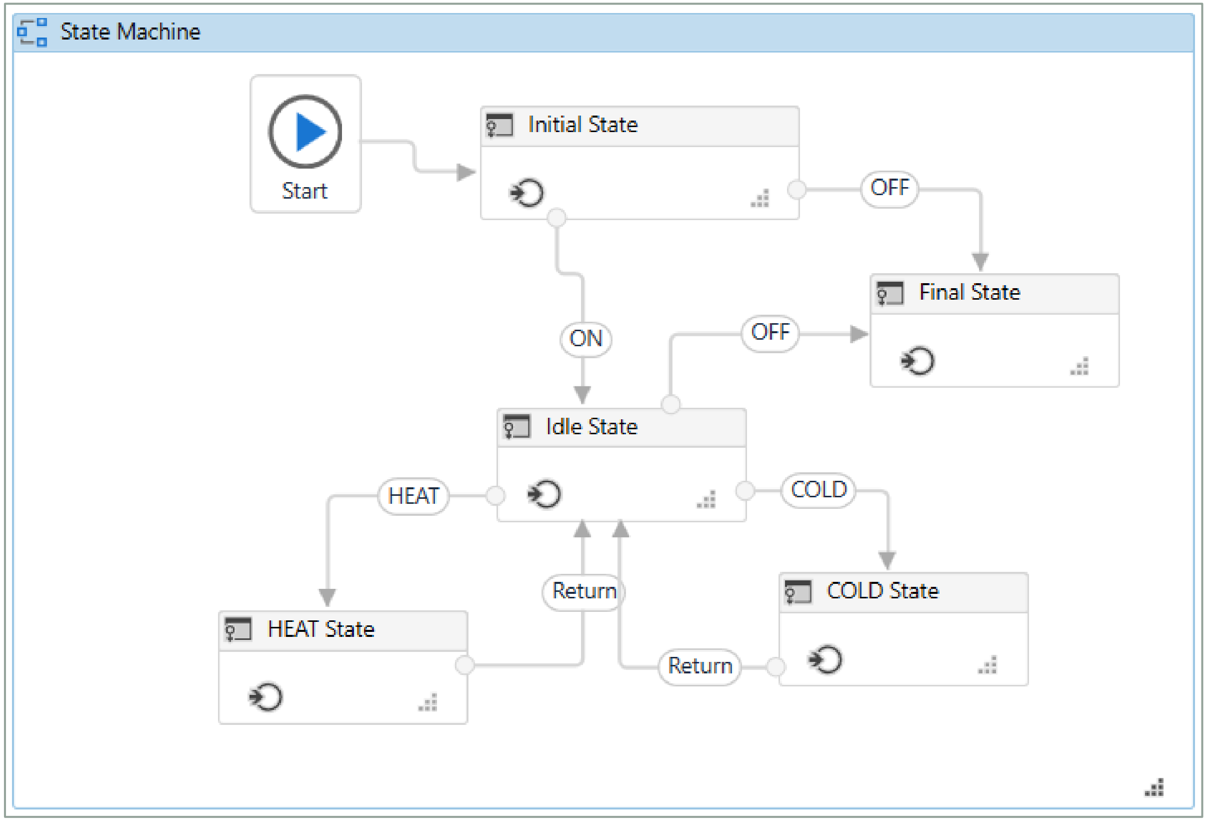
State Machine. Advantages. Disadvantages

- **Advantages:**
  - Can be used for continuous workflows that are more complex;
  - Transitions between states can be easily defined and offer flexibility;
  - Can be used for processes that are more complex and cannot be captured by simple *loops* and *if* activities;
  - It is easier to cover all the possible cases/transitions with state machines.
- **Disadvantages:**
  - Longer development time due to their complexity (splitting the process into logical "states“, transitions identification, etc.);
- **state machines should not be overused - they are appropriate to define only the skeleton of the project.**

- there are templates built upon **State Machines** especially designed to build large enterprise automations. The most commonly used is the Robotic Enterprise Framework (REF).

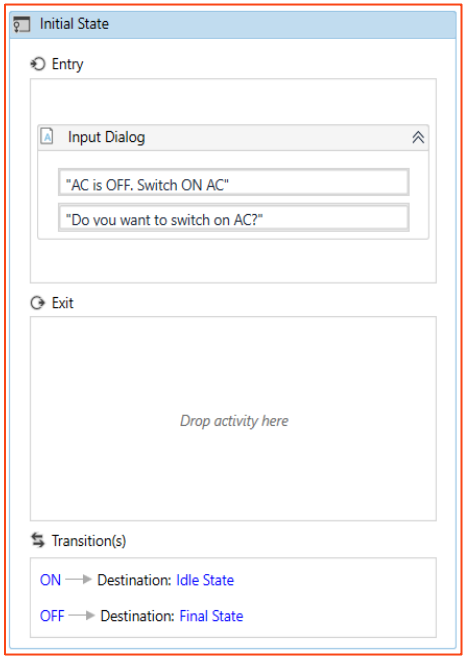
State Machine. Example

- an ***air conditioner*** function can be represented as a **state machine;**
- Key components:
  - **INITIAL state;**
  - Intermediate states:
    - **IDLE**
    - **HEAT**
    - **COLD**
  - **Transitions;**
  - **FINAL state;**
- the order of the **Transitions** shown in each state is very important, as it is the order in which they are assessed.



State Machine. Example (cont.)

Initial State

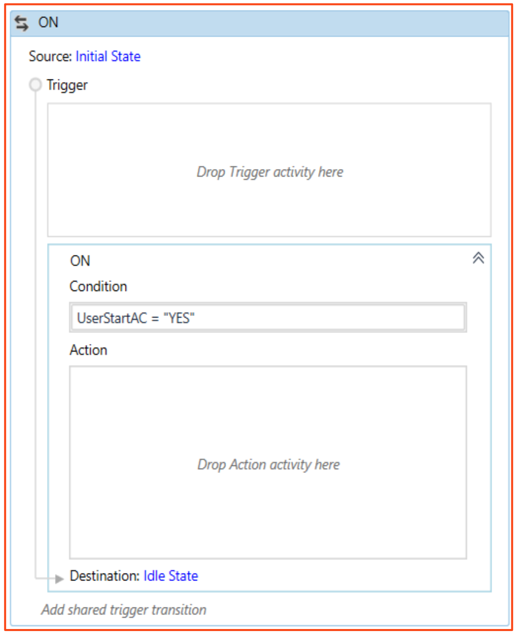


"AC is OFF. Switch ON AC"

"Do you want to switch on AC?"

The Initial state activity has three sections: **Entry**, **Exit**, and **Transition(s)**

Transition(s)

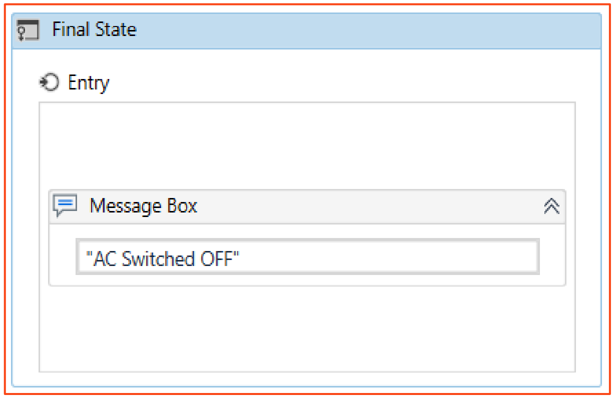

ON
Source: Initial State
Trigger
Drop Trigger activity here
ON
Condition

UserStartAC = "YES"

Action
Drop Action activity here
Destination: Idle State
Add shared trigger transition

There are three sections in Transition(s): **Trigger**, **Condition**, and **Action**

Final State



"AC Switched OFF"

OFF is the Final State of the State Machine

Demo 1. State Machine (Guess Random Number)

- *This demo is similar to **Demo 7** from **Lecture 02**, with the difference that* **State Machine***s are used;*
- **Create a process that performs the following actions:**
  - **1. *generate* an integer number from 1 to 7;**
  - **2. *read* a number to guess the generated number;**
  - **3. *compare* the generated value**
    - **3.1. print the message “Enter a smaller number!” or**
    - **3.2. print the message “Enter a bigger number!”;**
  - **4. *repeat* steps 2 and 3 until you succeed to find the number;**
  - **5. *show* the message “Congratulations!!!”**

see **Demo1 – StateMachine**

Robotic Enterprise Framework. Overview

- **REFramework details**
- **Types of processes**
- **REFramework Architecture**
  - **States**
  - **Predefined workflows**
  - **Transitions**
  - **Shared variables**
- **REFramework with/out Orchestrator**

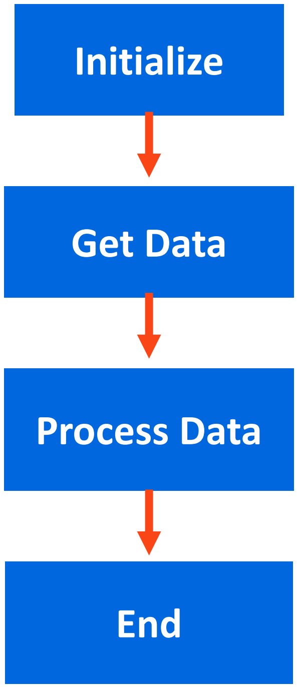
Robotic Enterprise Framework. Details

- a **framework** is
  - a template that helps the user to design automation processes;
- **REFramework** stands for **Robotic Enterprise Framework**;
  - it is a project template which is based on a **State Machine**;
  - it contains several pre-made state containers for ***initializing applications***, ***retrieving input data***, ***processing it***, and ***ending the transaction***;
  - the states are connected through multiple transitions;
  - there are also multiple invoked workflows, each handling particular aspects of the project;
  - it offers a way to store, read, and easily modify ***project configuration data***, a robust ***exception handling scheme***, ***event logging for all exceptions*** and relevant ***transaction information***.
  - it can be upgraded or extended independently of the business code, by editing only one file, i.e., `Main.xaml`.

REF. Types of Processes

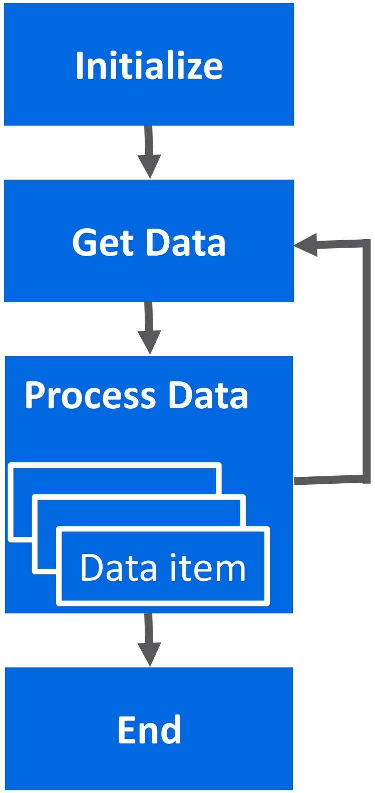
Linear

- The steps of the process are performed only once



Repetitive

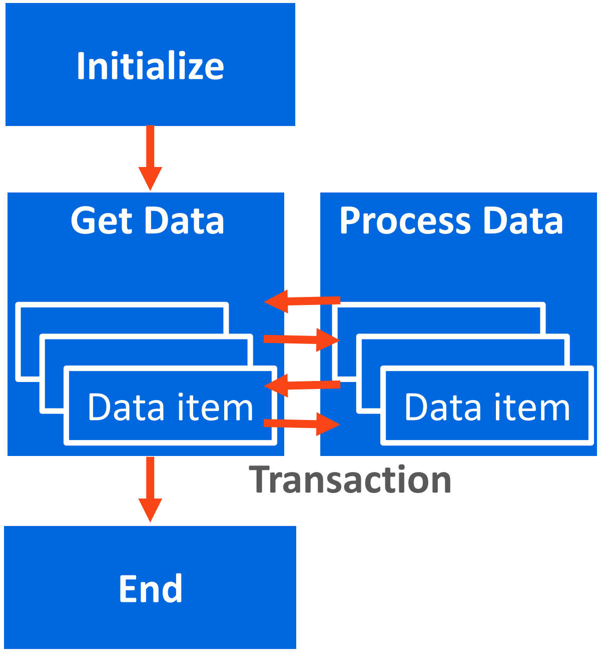
- The steps of the process are performed multiple times on different data items



Transactional

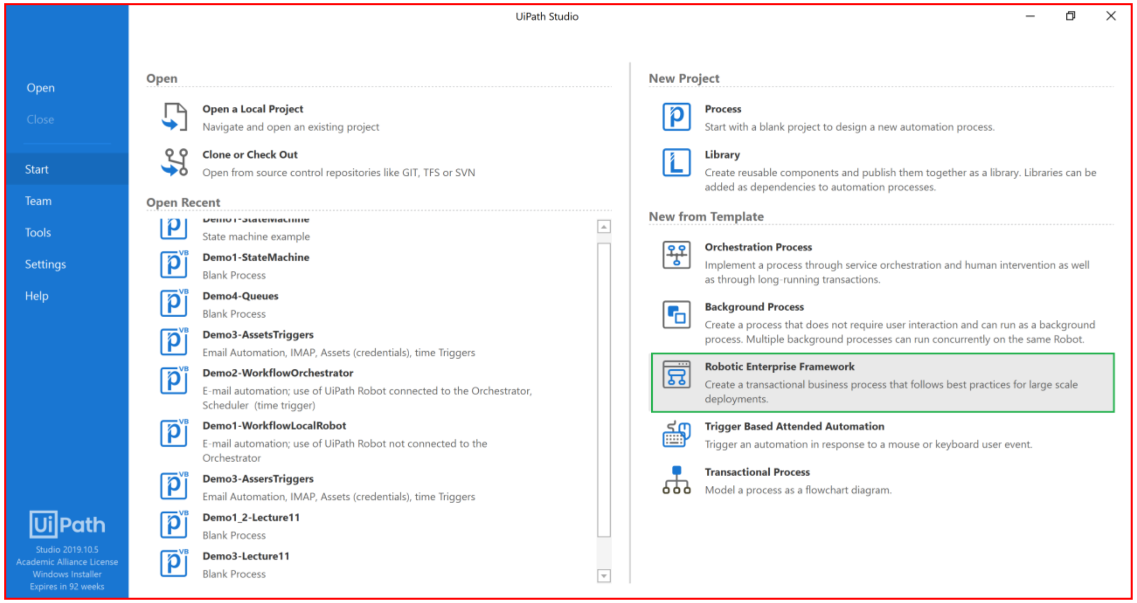
•A transaction represents the minimum amount of data and the steps required to process the data to complete a section of a business process.

•The steps of the process are performed multiple times on different data items independently



REF. In UiPath Studio

- automation projects can be built with REF template through the **Start** tab in UiPath Studio.



UiPath Studio

Studio 2019.10.5
Academic Alliance License
Windows Installer
Expires in 92 weeks

Open

Open a Local Project
Navigate and open an existing project

Clone or Check Out
Open from source control repositories like GIT, TFS or SVN

Open Recent

Demo1-StateMachine
State machine example

Demo1-StateMachine
Blank Process

Demo4-Queues
Blank Process

Demo3-AssetsTriggers
Email Automation, IMAP, Assets (credentials), time Triggers

Demo2-WorkflowOrchestrator
E-mail automation; use of UiPath Robot connected to the Orchestrator, Scheduler (time trigger)

Demo1-WorkflowLocalRobot
E-mail automation; use of UiPath Robot not connected to the Orchestrator

Demo3-AssersTriggers
Email Automation, IMAP, Assets (credentials), time Triggers

Demo1_2-Lecture11
Blank Process

Demo3-Lecture11
Blank Process

New Project

Process
Start with a blank project to design a new automation process.

Library
Create reusable components and publish them together as a library. Libraries can be added as dependencies to automation processes.

New from Template

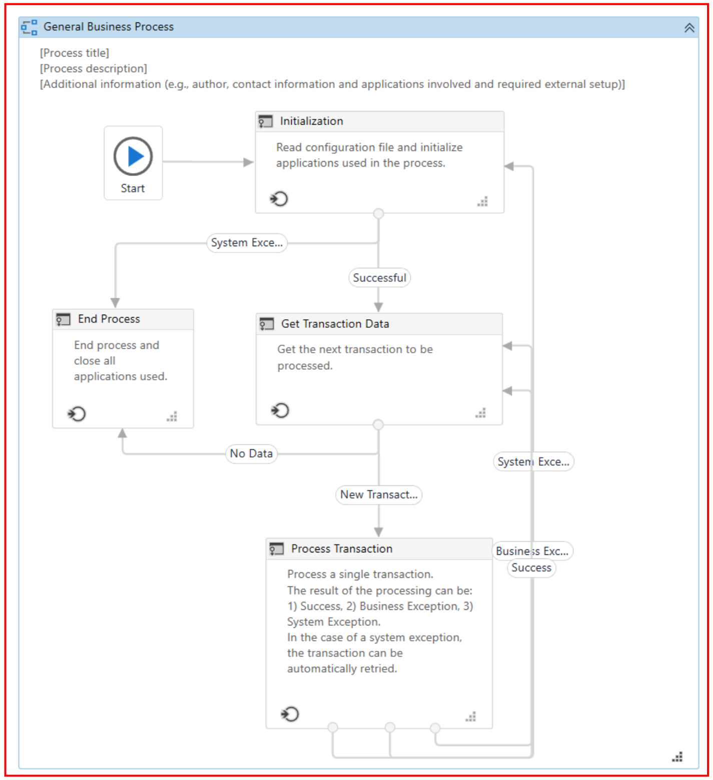
Orchestration Process
Implement a process through service orchestration and human intervention as well as through long-running transactions.

Background Process
Create a process that does not require user interaction and can run as a background process. Multiple background processes can run concurrently on the same Robot.

Robotic Enterprise Framework
Create a transactional business process that follows best practices for large scale deployments.

Trigger Based Attended Automation
Trigger an automation in response to a mouse or keyboard user event.

Transactional Process
Model a process as a flowchart diagram.



General Business Process

[Process title]
[Process description]
[Additional information (e.g., author, contact information and applications involved and required external setup)]

Initialization
Read configuration file and initialize applications used in the process.

End Process
End process and close all applications used.

Get Transaction Data
Get the next transaction to be processed.

Process Transaction
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.

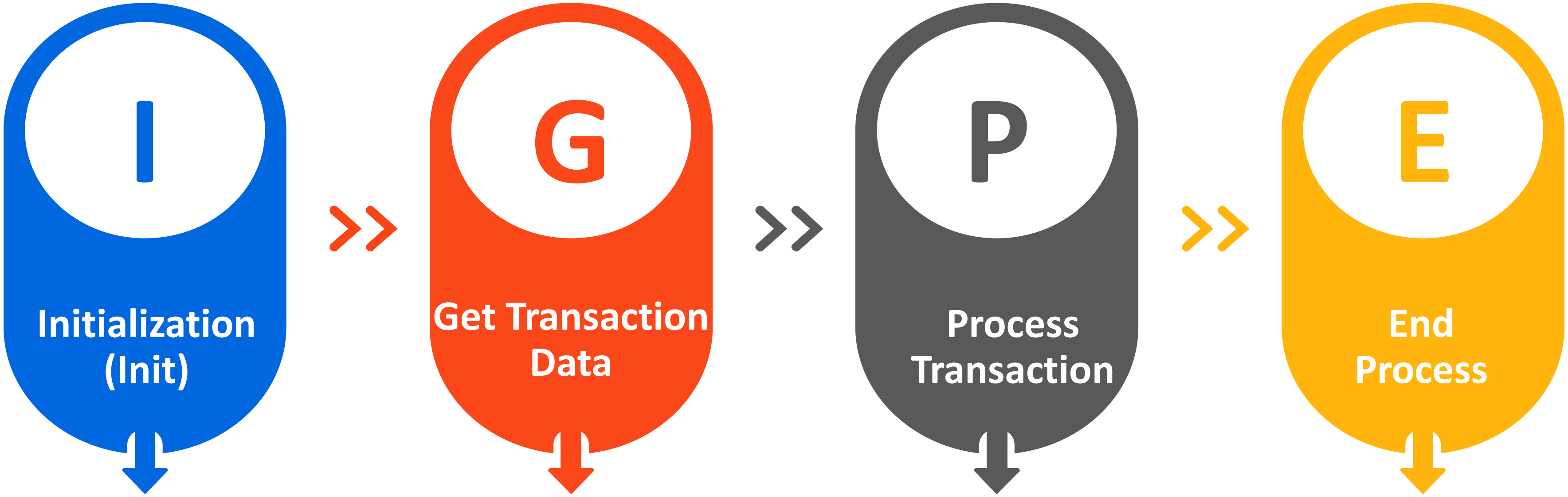
Demo 2. REF

- **Create a process based on REF template:**
  - ***Discuss the structure and components* (states, transitions, workflows, shared variables).**

see **Demo2 – REF**

REF. States

- REF states represent a particular **event** in the execution; depending on certain conditions, the execution can transition from one state to another to represent the **steps of a process**;
- the states of a REF process are:



I

Initialization (Init)

Reads the **configuration file** and **initialize the applications** used in the process

G

Get Transaction Data

Gets the next **transaction** to be processed

P

Process Transaction

Processes a single transaction

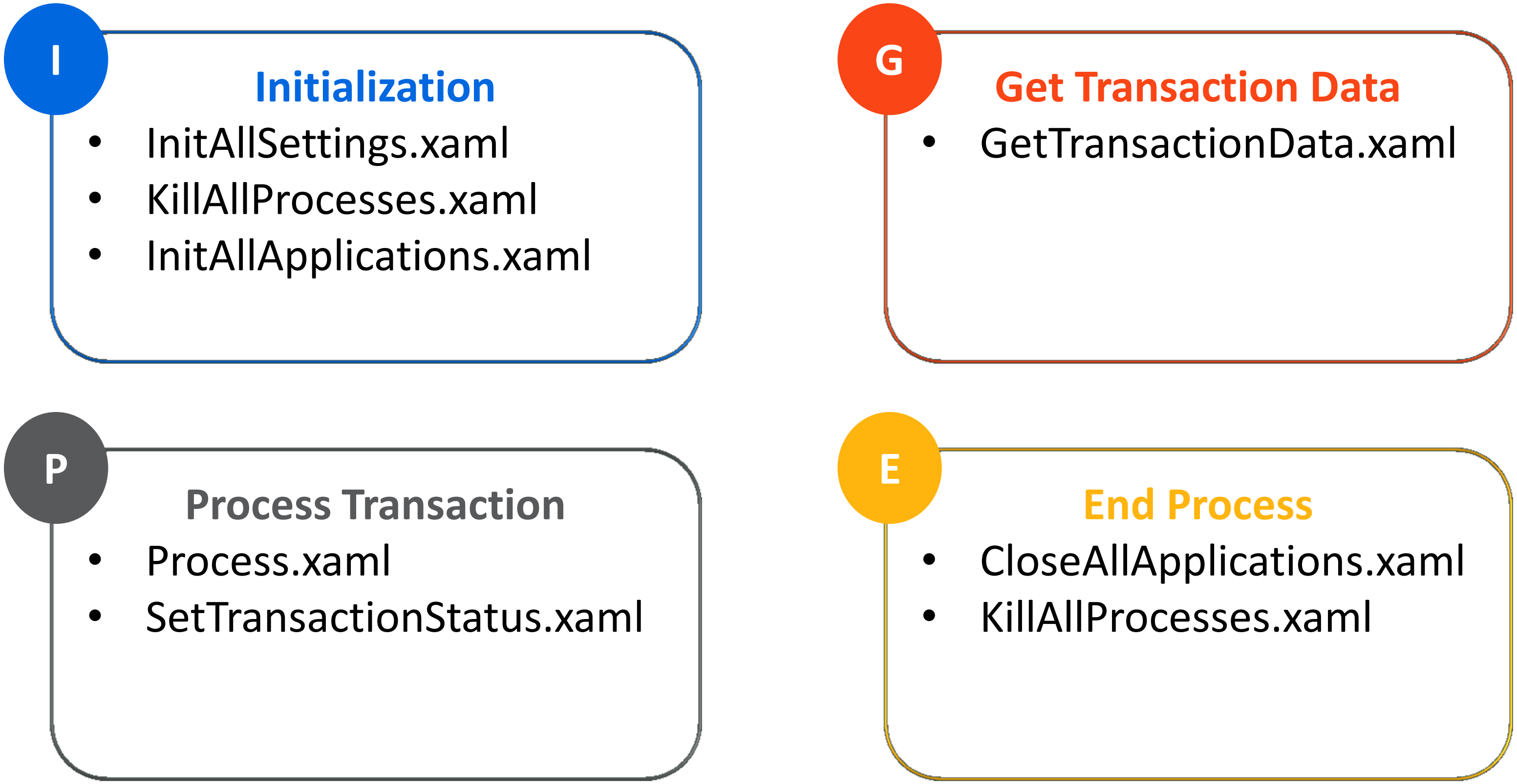
E

End Process

Ends the process

REF. Predefined Workflows

- the workflows invoked in different REF states are:



Initialization

- InitAllSettings.xaml
- KillAllProcesses.xaml
- InitAllApplications.xaml

Get Transaction Data

- GetTransactionData.xaml

Process Transaction

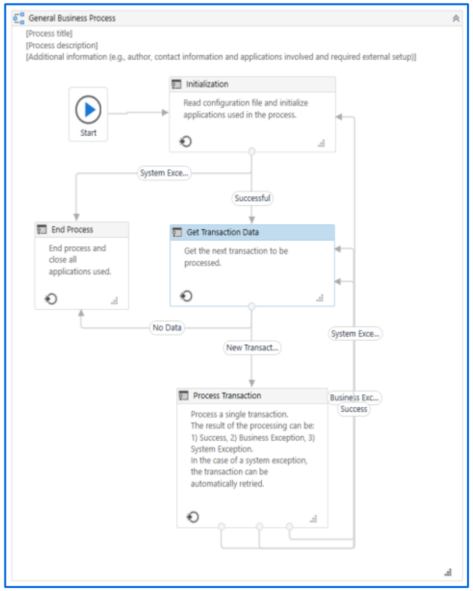
- Process.xaml
- SetTransactionStatus.xaml

End Process

- CloseAllApplications.xaml
- KillAllProcesses.xaml

REF. *Initialization* State Workflows

- the workflows invoked in the **Initialization** state are:



General Business Process

[Process title]
[Process description]
[Additional information (e.g., author, contact information and applications involved and required external setup)]

Initialization: Read configuration file and initialize applications used in the process.

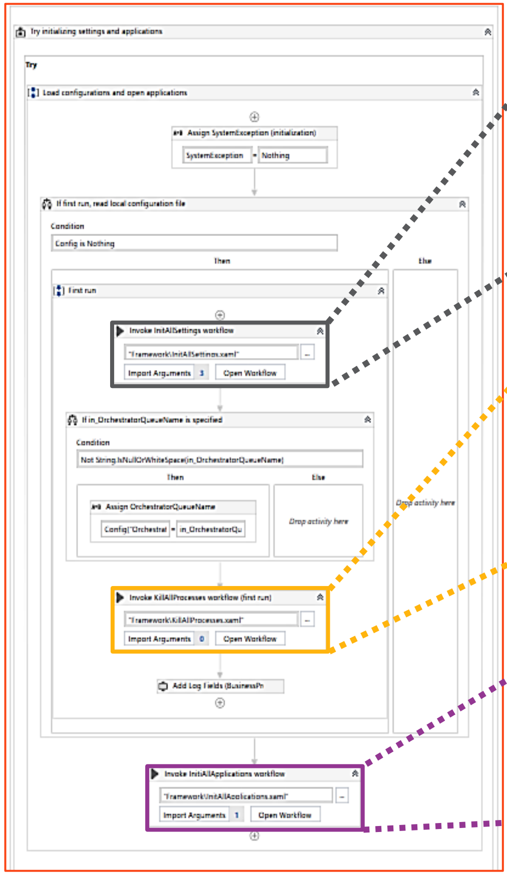
Get Transaction Data: Get the next transaction to be processed.

End Process: End process and close all applications used.

Process Transaction: 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.

REFramework Workflow

Initialization Workflow



Try initializing settings and applications

Try

Load configurations and open applications

Assign SystemException (initialization): SystemException = Nothing

If first run, read local configuration file

Condition: Config is Nothing

First run

Invoke InitAllSettings workflow: "Framework\InitAllSettings.xaml"

If in_OrchestratorQueueName is specified

Condition: Not String.IsNullOrWhiteSpace(in_OrchestratorQueueName)

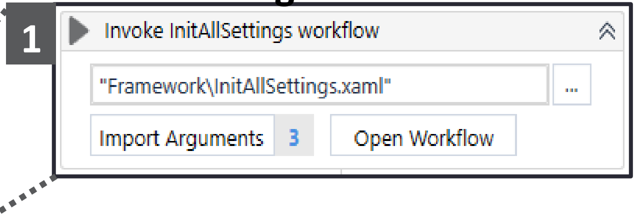
Assign OrchestratorQueueName: Config("Orchestrat = in_OrchestratorQu

Invoke KillAllProcesses workflow (first run): "Framework\KillAllProcesses.xaml"

Add Log Fields (BusinessPr

Invoke InitiAllApplications workflow: "Framework\InitAllApplications.xaml"

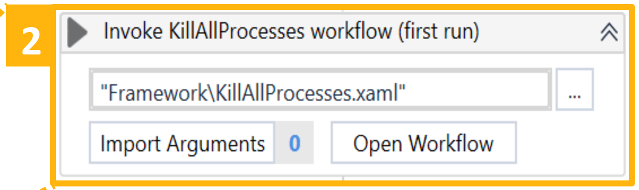
InitAllSettings Workflow



Invoke InitAllSettings workflow

"Framework\InitAllSettings.xaml"

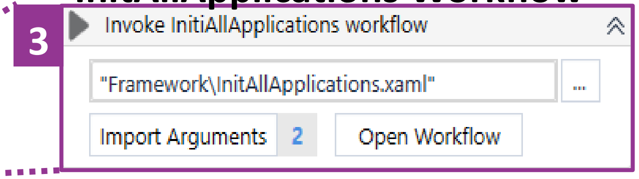
KillAllProcesses Workflow



Invoke KillAllProcesses workflow (first run)

"Framework\KillAllProcesses.xaml"

InitAllApplications Workflow



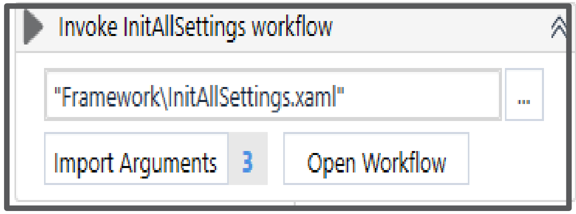
Invoke InitiAllApplications workflow

"Framework\InitAllApplications.xaml"

REF. *Initialization* State Workflows (cont.)

1. InitAllSettings

- Initializes, populates, and outputs a configuration dictionary to be used throughout the project
- Exception in this workflow is caught by the Try Catch activity

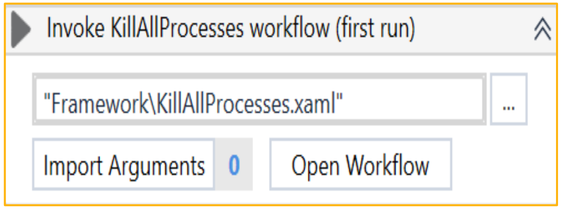


Invoke InitAllSettings workflow

"Framework\InitAllSettings.xaml"

2. KillAllProcesses

- Implements cleanup steps
- Kill Process activity forces the termination of a Windows process representing an application used in the business process

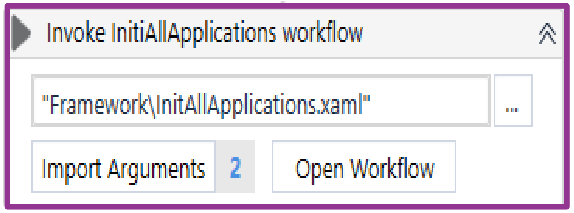


Invoke KillAllProcesses workflow (first run)

"Framework\KillAllProcesses.xaml"

3. InitAllApplications

- Initializes applications operated during the execution of the process
- Contains activities like Open Application activities and Open Browser

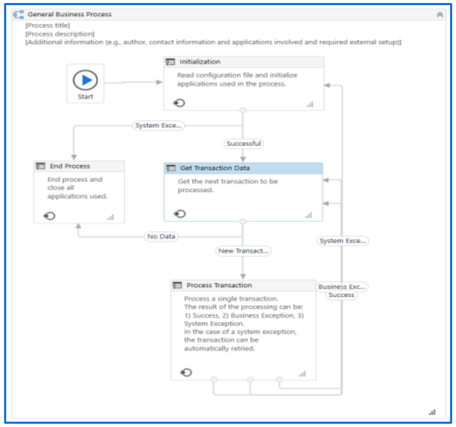


Invoke InitiAllApplications workflow

"Framework\InitAllApplications.xaml"

REF. *Get Transaction Data* State Workflow

- the workflow invoked in the **Get Transaction Data** state is:



[Process title]
[Process description]
[Additional information (e.g., author, contact information and applications involved and required external setup)]

Initialization: Read configuration file and initialize applications used in the process.

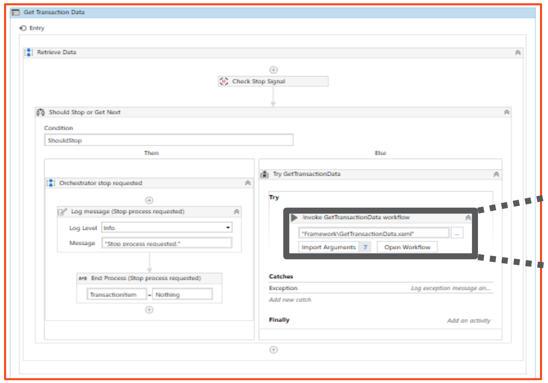
End Process: End process and close all applications used.

Get Transaction Data: Get the next transaction to be processed.

Process Transaction: 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.

REFramework Workflow

Get Transaction Data Workflow



Condition: ShouldStop

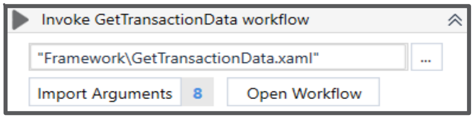
Log Level: Info

Message: "Stop process requested."

TransactionItem = Nothing

"Framework\GetTransactionData.xaml"

GetTransactionData Workflow

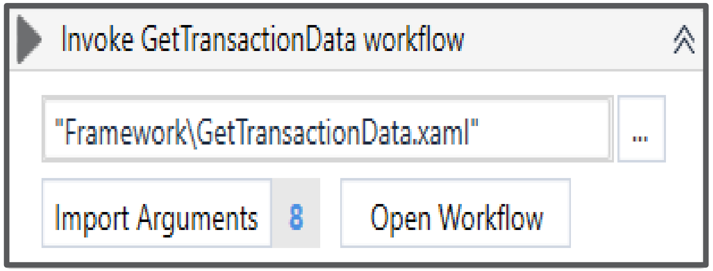


"Framework\GetTransactionData.xaml"

REF. *Get Transaction Data* State Workflow (cont.)

GetTransactionData

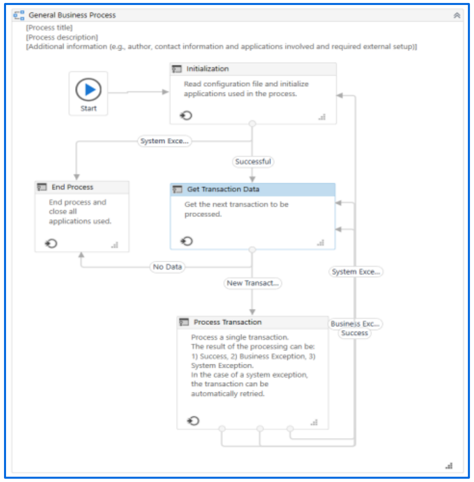
- Retrieves a transaction item from a specified source (e.g., Orchestrator queues (assigned in the Initialization state), spreadsheets, databases, mailboxes or web APIs)
- The first activity tries to retrieve a new transaction item from an Orchestrator queue



"Framework\GetTransactionData.xaml"

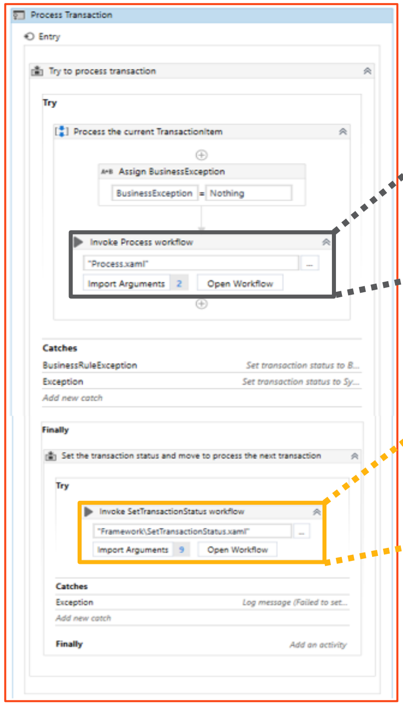
REF. *Process Transaction* State Workflow

- the workflows invoked in the **Process Transaction** state are:


General Business Process
[Process title]
[Process description]
[Additional information (e.g., author, contact information and applications involved and required external setup)]
Start
Initialization
Read configuration file and initialize applications used in the process.
System Exce...
Successful
End Process
End process and close all applications used.
Get Transaction Data
Get the next transaction to be processed.
No Data
New Transact...
System Exce...
Process Transaction
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.
Business Exc...
Success

REFramework Workflow

Process Transaction Workflow


Process Transaction

Entry

Try to process transaction

Try

Process the current TransactionItem

Assign BusinessException

BusinessException = Nothing

Invoke Process workflow

"Process.xaml"

Import Arguments 2 | Open Workflow

Catches

BusinessRuleException — *Set transaction status to B...*

Exception — *Set transaction status to Sy...*

Add new catch

Finally

Set the transaction status and move to process the next transaction

Try

Invoke SetTransactionStatus workflow

"Framework\SetTransactionStatus.xaml"

Import Arguments 9 | Open Workflow

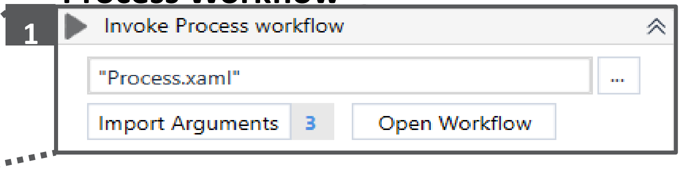
Catches

Exception — *Log message (Failed to set...*

Add new catch

Finally — *Add an activity*

Process Workflow

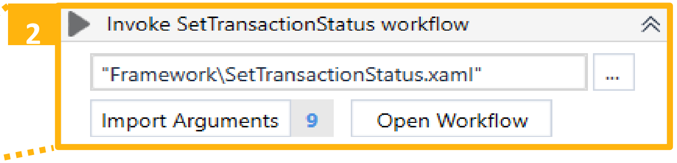

1

Invoke Process workflow

"Process.xaml"

Import Arguments 3 | Open Workflow

SetTransactionStatus Workflow


2

Invoke SetTransactionStatus workflow

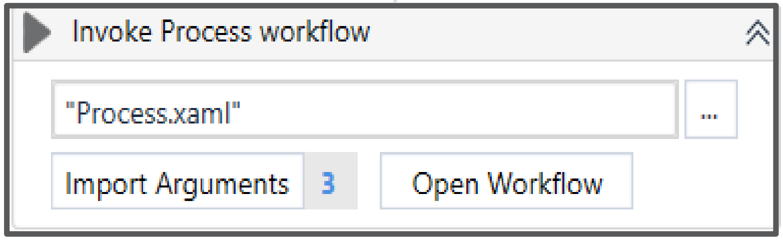
"Framework\SetTransactionStatus.xaml"

Import Arguments 9 | Open Workflow

REF. *Process Transaction* State Workflow (cont.)

1. Process

- Invokes major steps of the business process commonly implemented by multiple sub-workflows
- Exceptions thrown during processing:
  - Business Exception
  - System Exception

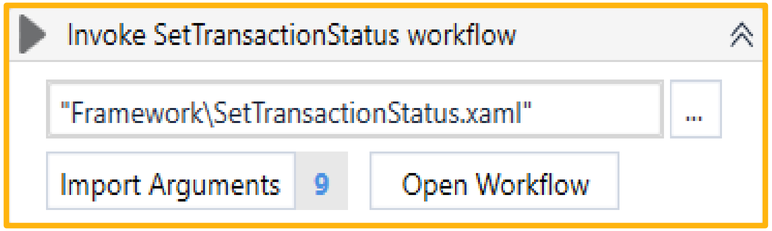


Invoke Process workflow

"Process.xaml"

2. SetTransactionStatus

- Sets and logs each transaction's status
- Possible statuses:
  - Success
  - Business Exception
  - System Exception
- Invokes further workflows:
  - RetryCurrentTransaction.xaml
  - TakeScreenshot.xaml
  - CloseAllApplications.xaml
  - KillAllProcesses.xaml

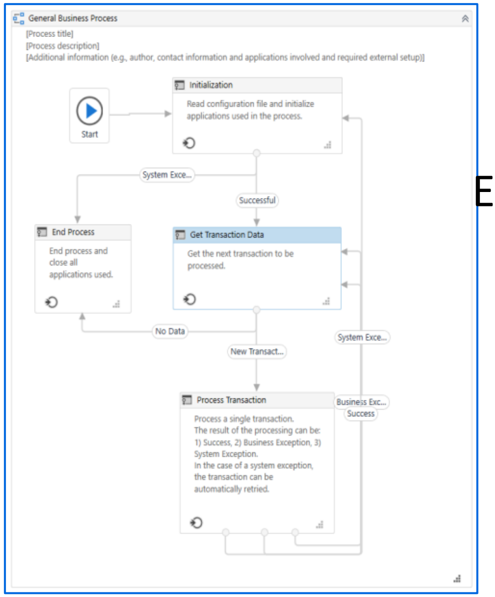


Invoke SetTransactionStatus workflow

"Framework\SetTransactionStatus.xaml"

REF. *End Process* State Workflow

- the workflows invoked in the **End Process** state are:



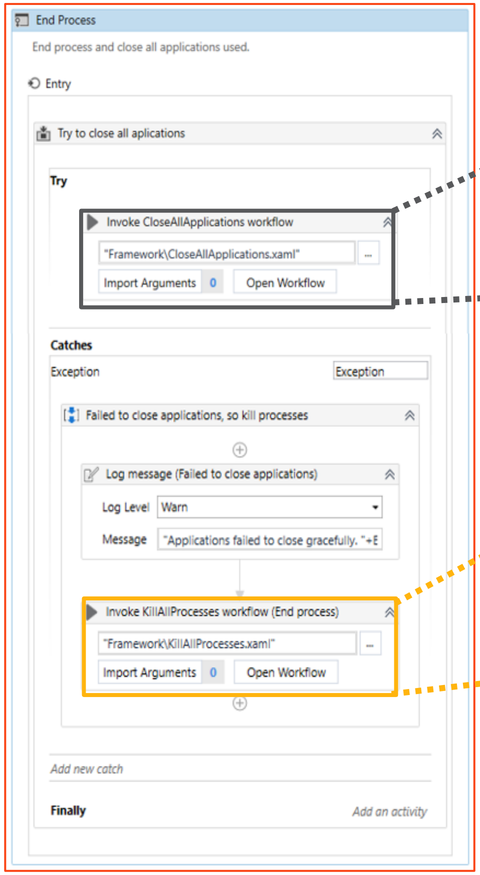
General Business Process

[Process title]
[Process description]
[Additional information (e.g., author, contact information and applications involved and required external setup)]

Start
Initialization
Read configuration file and initialize applications used in the process.
System Exce...
Successful
End Process
End process and close all applications used.
Get Transaction Data
Get the next transaction to be processed.
No Data
New Transact...
System Exce...
Process Transaction
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.
Business Exc...
Success

REFramework Workflow

End Process Workflow



End Process

End process and close all applications used.

Entry

Try to close all aplications

Try

Invoke CloseAllApplications workflow

"Framework\CloseAllApplications.xaml"

Catches

Exception — Exception

Failed to close applications, so kill processes

Log message (Failed to close applications)

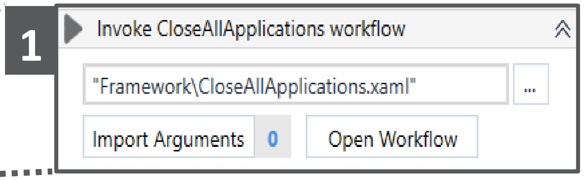
Log Level: Warn

Message: "Applications failed to close gracefully. "+E

Invoke KillAllProcesses workflow (End process)

"Framework\KillAllProcesses.xaml"

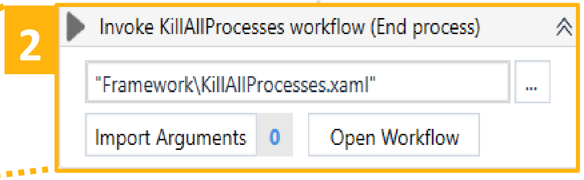
CloseAllApplications Workflow



Invoke CloseAllApplications workflow

"Framework\CloseAllApplications.xaml"

KillAllProcesses Workflow



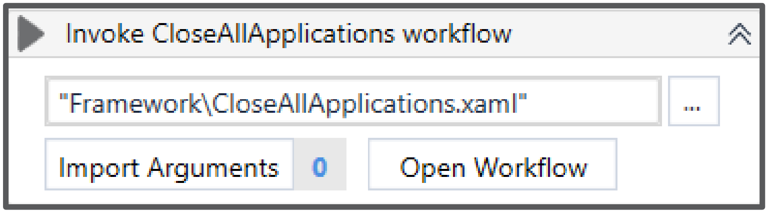
Invoke KillAllProcesses workflow (End process)

"Framework\KillAllProcesses.xaml"

REF. *End Process* State Workflow (cont.)

1. CloseAllApplications

- Ends the process and closes the used applications
- Sub-workflows can be invoked to perform more complex steps, such as logging out of a system

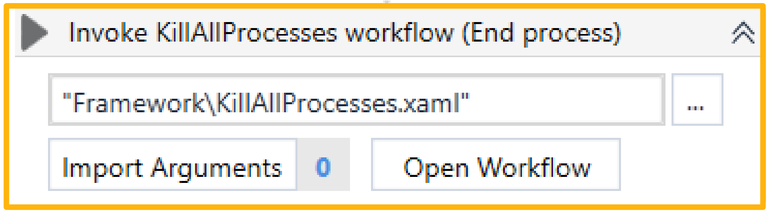


Invoke CloseAllApplications workflow

"Framework\CloseAllApplications.xaml"

2. KillAllProcesses

- Implements cleanup steps
- Kill Process activity forces the termination of a Windows process representing an application used in the business process

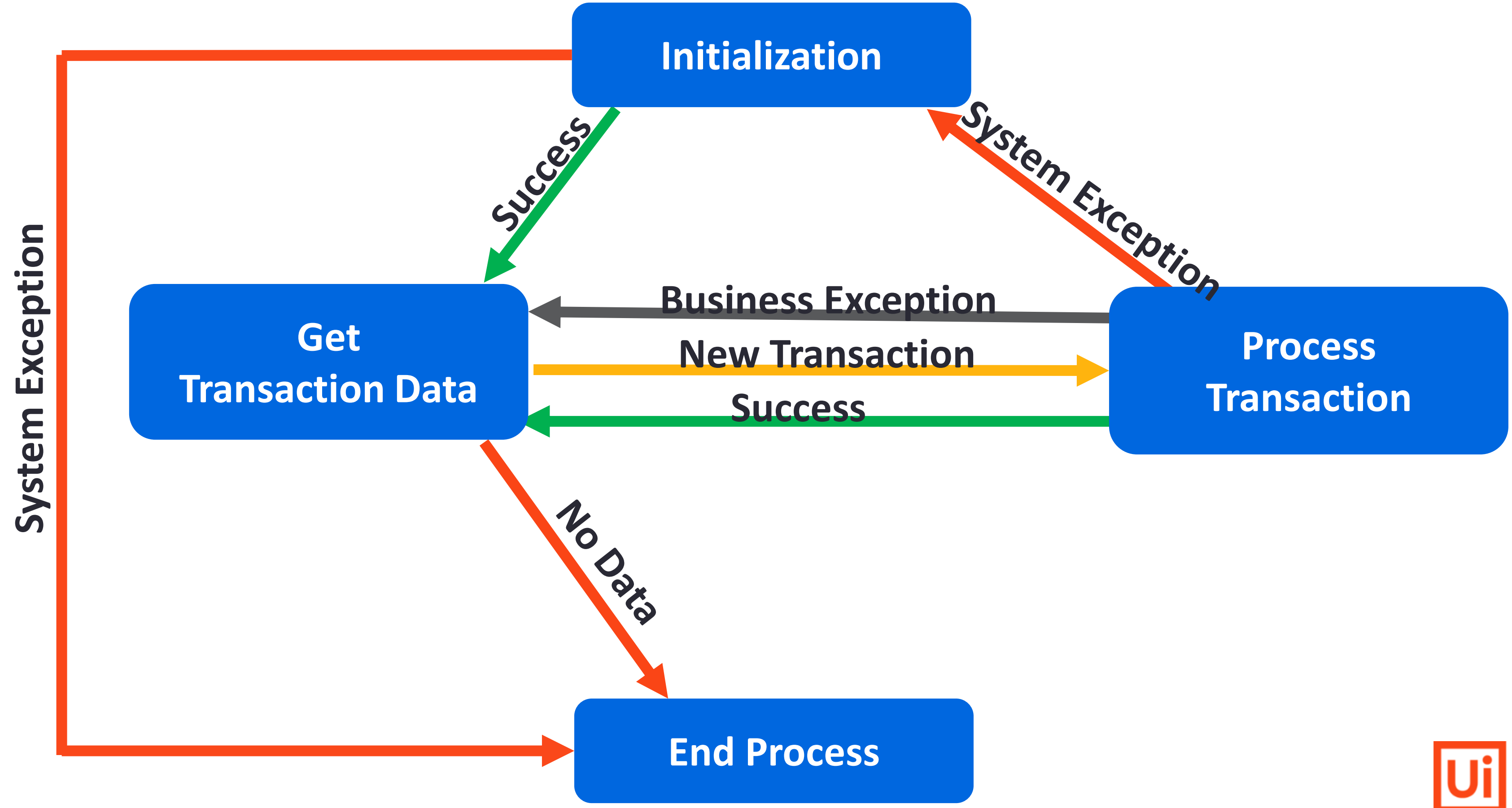


Invoke KillAllProcesses workflow (End process)

"Framework\KillAllProcesses.xaml"

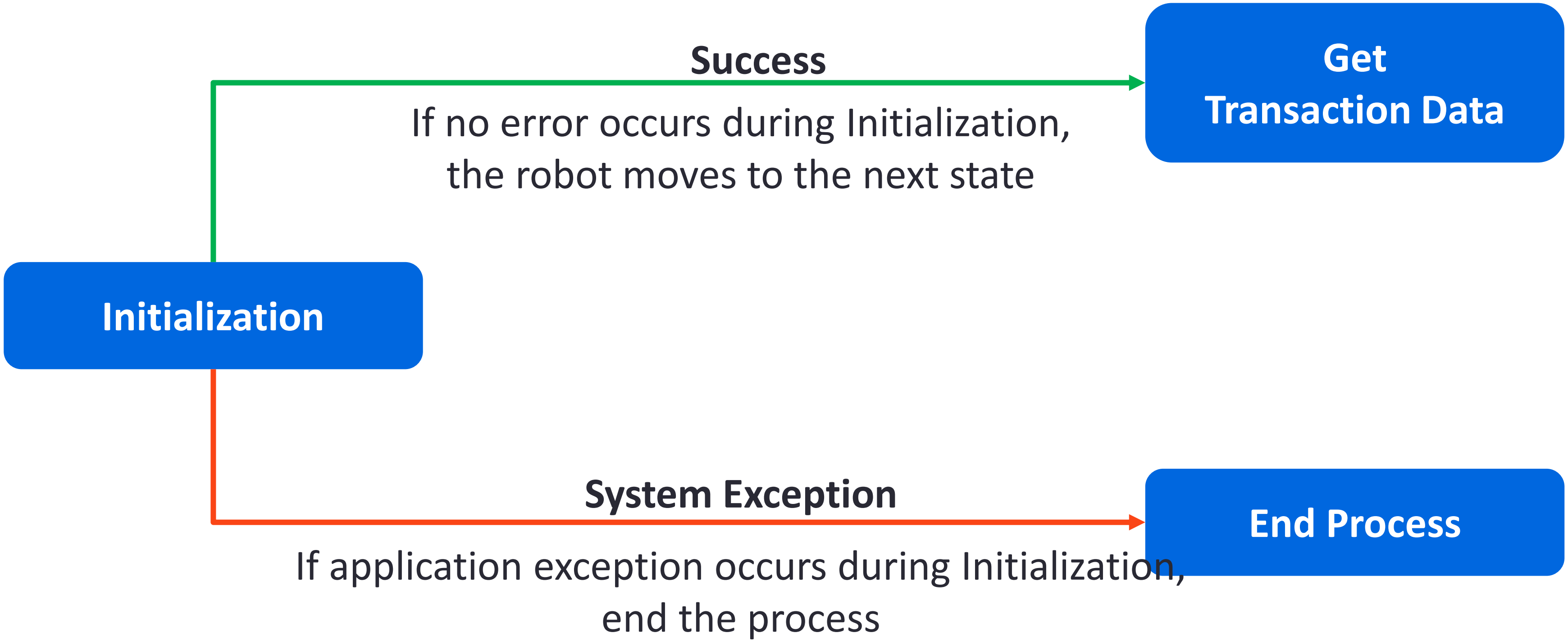
REF. Transitions

- a **transition** refers to
  - the movement of the process from one state to another;
- the transitions in the REFramework workflow are:



REF. Transitions from *Initialization* State

- the transitions from **Initialization** state are:



Success

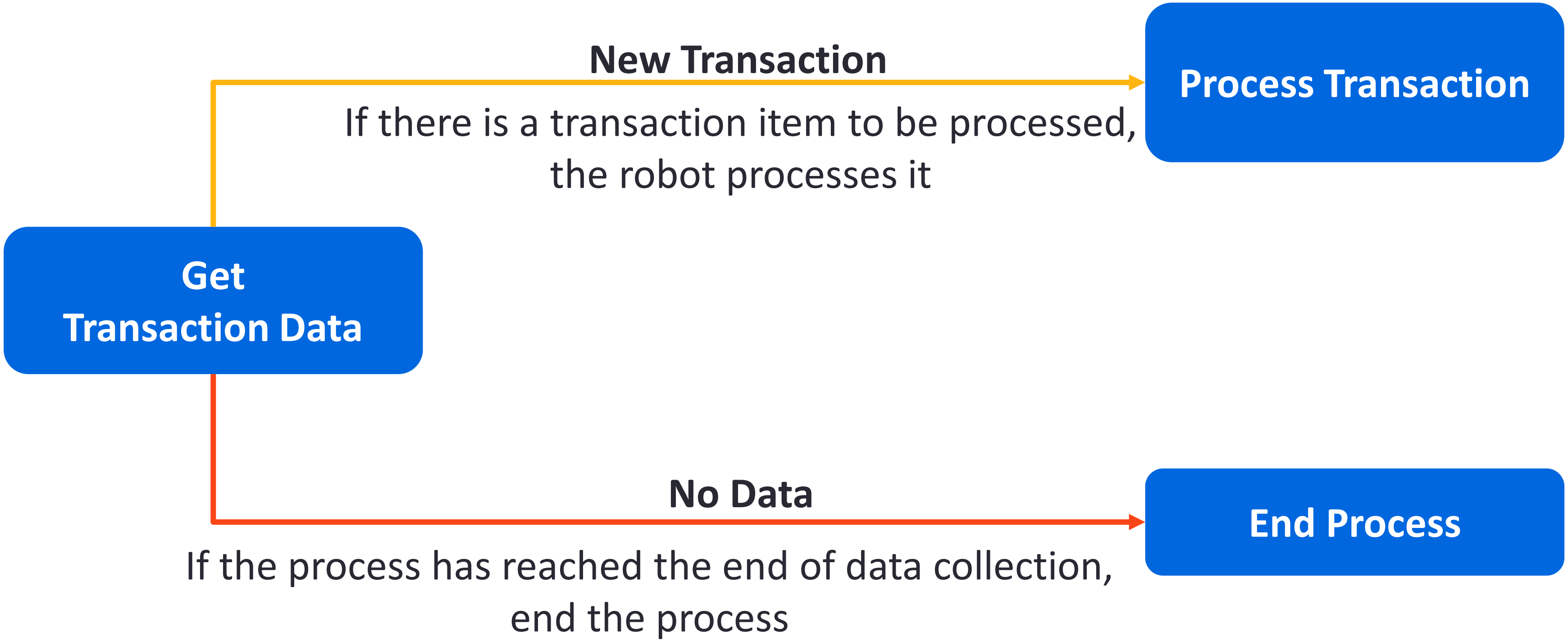
If no error occurs during Initialization, the robot moves to the next state

System Exception

If application exception occurs during Initialization, end the process

REF. Transitions from *Get Transition Data* State

- the transitions from **Get Transition Data** state are:



New Transaction

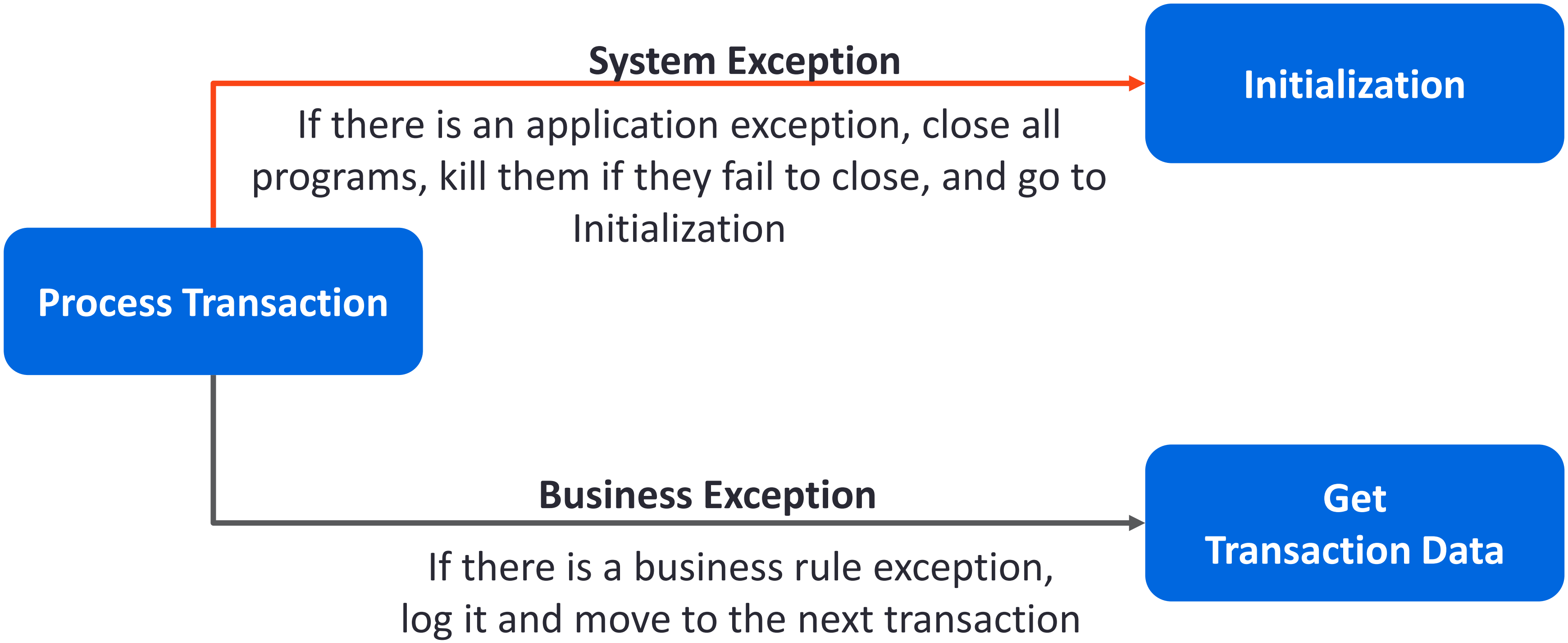
If there is a transaction item to be processed, the robot processes it

No Data

If the process has reached the end of data collection, end the process

REF. Transitions from *Process Transaction* State

- the transitions from **Process Transaction** state are:



If there is an application exception, close all programs, kill them if they fail to close, and go to Initialization

If there is a business rule exception, log it and move to the next transaction

REF. Shared Variables

- shared variables are
  - predefined variables passed as arguments to the workflows invoked in different states, information that will be available throughout the runtime of the process;
- the shared variables in REFramework are:

TransactionItem	Default Type	Written in Workflows	Read in Workflows
Stores the Transaction item to be processed	QueueItem	GetTransactionData.xaml	Process.xaml SetTransactionStatus.xaml

SystemException	Default Type	Written in Workflows	Read in Workflows
Used during transitions between states to represent exceptions other than BusinessRuleException	Exception	Main.xaml	Main.xaml SetTransactionStatus.xaml

BusinessException	Default Type	Written in Workflows	Read in Workflows
Represents a situation that does not conform to the rules of the process being automated	BusinessRuleException	Main.xaml	Main.xaml SetTransactionStatus.xaml

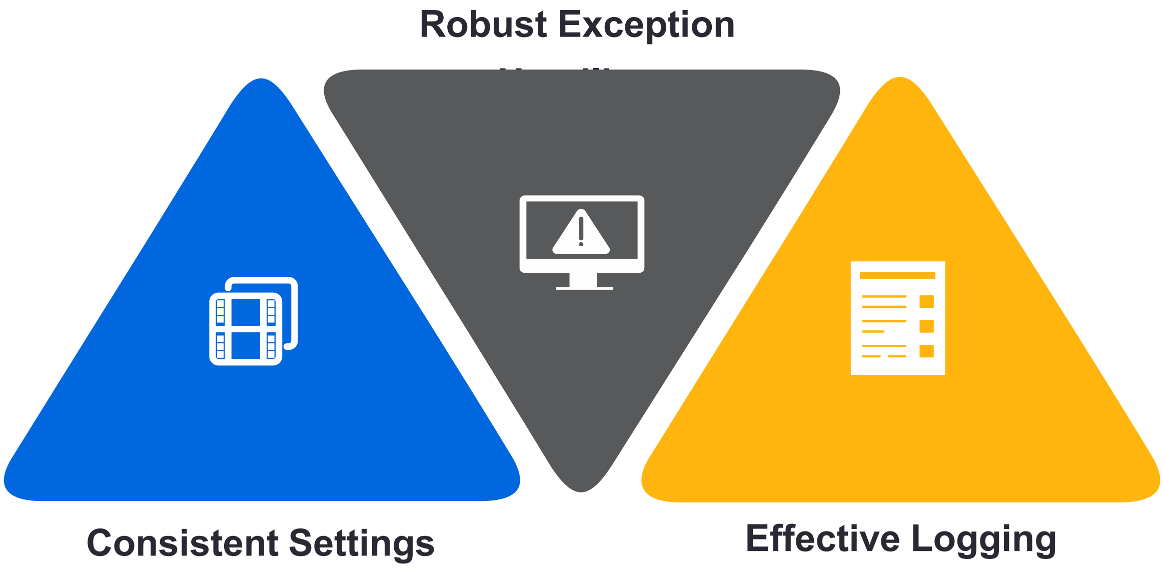
TransactionNumber	Default Type	Written in Workflows	Read in Workflows
Sequential counter of transaction items	Int32	SetTransactionStatus.xaml	GetTransactionData.xaml

REF. Shared Variables (cont.)

	Default Type	Written in Workflows	Read in Workflows
TransactionData Transactions stored in a DataTable	DataTable	GetTransactionData.xaml	GetTransactionData.xaml
Config Dictionary structure to store configuration data of the process read from the Config file	Dictionary(Of String,Object)	InitAllSettings.xaml	InitAllSettings.xaml Process.xaml GetTransactionData.xaml SetTransactionStatus.xaml
RetryNumber Number of retry attempts for transaction processes in case of system exceptions	Int32	SetTransactionStatus.xaml	SetTransactionStatus.xaml
TransactionField1, 2, 3, … Additional information about the transaction item. By default, two transaction fields are available	String	GetTransactionData.xaml	SetTransactionStatus.xaml
TransactionID Unique ID used for information and logging purposes	String	GetTransactionData.xaml	SetTransactionStatus.xaml

REF. Features

- REFramework provides several features that are helpful in the implementation of stable and scalable automation projects;
- the features are:



Robust Exception

Consistent Settings

Effective Logging

Config File. Details

- a configuration file (***Config***) can be used to define the parameters that are used throughout the project and avoid values hardcoded in workflows;
- there are *three sheets* in the **Config.xslx** file:

Config.xslx

S

Settings

C

Constants

A

Assets

Config File. *Settings* Sheet

- **Settings** sheet contains
  - the configuration values to be used throughout the project;
- E.g.: URLs to access web applications, Orchestrator queue names.

Name

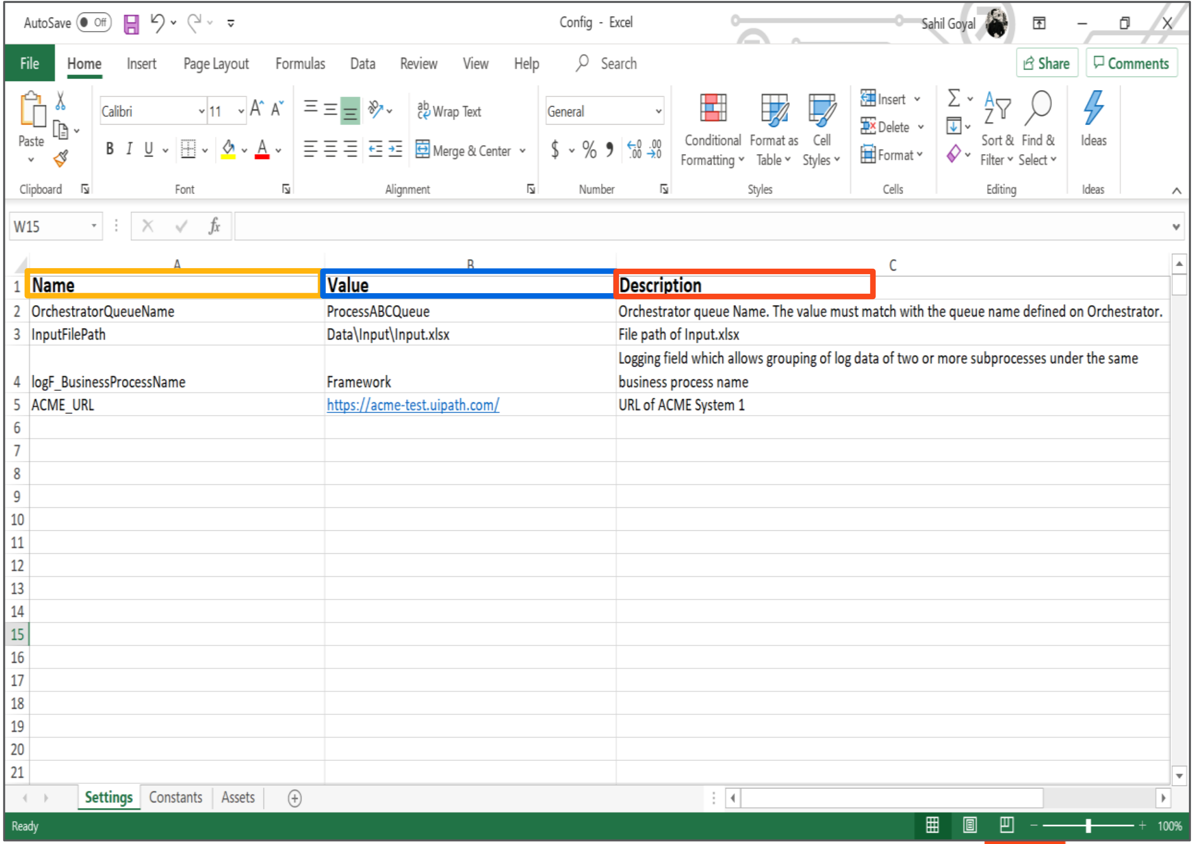
Specifies a key for a config dictionary (Example: OrchestratorQueueName, logF_BusinessProcessName, ACME_URL)

Value

Defines the value associated with the key (Example: ProcessABCQueue, Framework, https://acme-test.uipath.com/)

Description

Gives an explanation about the key_value pair



Name	Value	Description
OrchestratorQueueName	ProcessABCQueue	Orchestrator queue Name. The value must match with the queue name defined on Orchestrator.
InputFilePath	Data\Input\Input.xlsx	File path of Input.xlsx
logF_BusinessProcessName	Framework	Logging field which allows grouping of log data of two or more subprocesses under the same business process name
ACME_URL	https://acme-test.uipath.com/	URL of ACME System 1

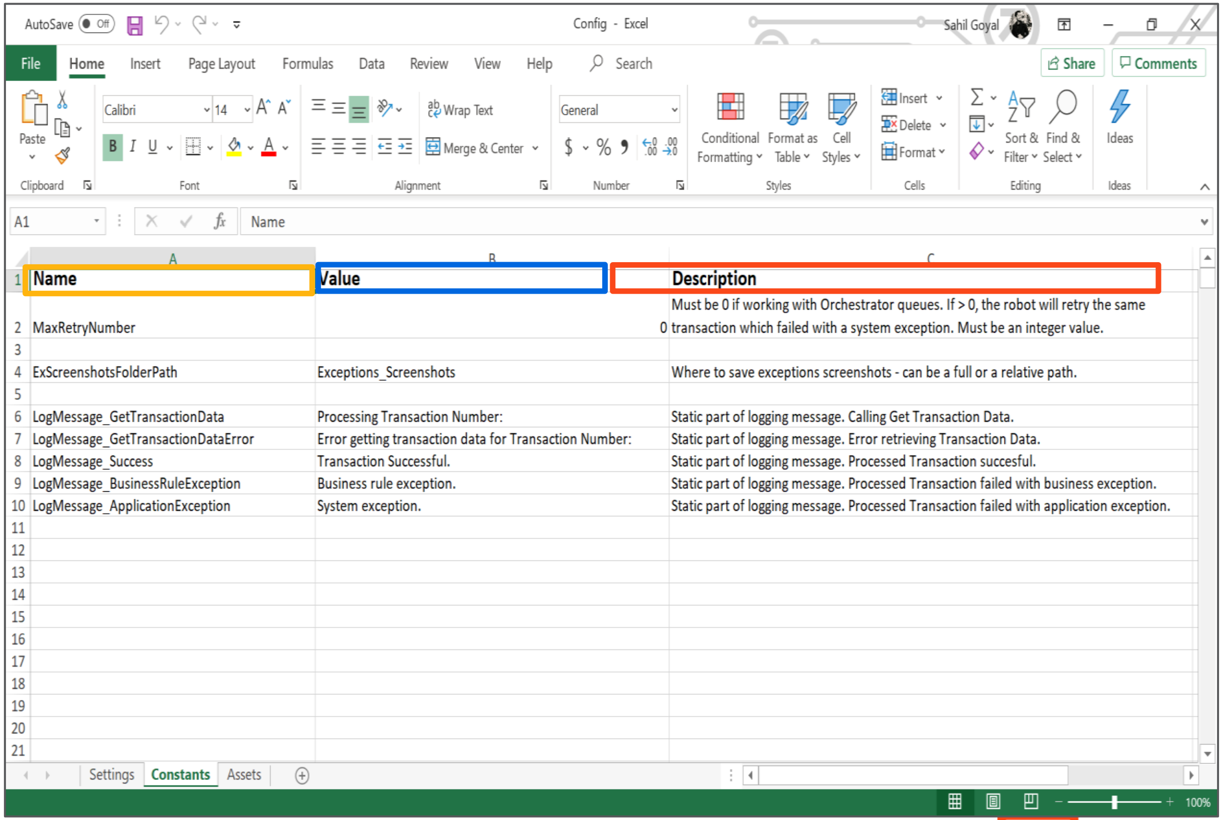
Config File. *Constants* Sheet

- **Constants** sheet contains
    - the values that are supposed to be the same across all deployments of the workflow;
- E.g.: **MaxRetryNumber**, default folder paths and default logging messages.

Name
Specifies a key for a config dictionary (Example: MaxRetryNumber, ExScreenshotsFolderPath,and LogMessage_Success)

Value
Defines the value associated with the key (Example: Exceptions_Screenshots, Transaction Successful)

Description
Gives an explanation about the key_value pair



	A	B	C
1	Name	Value	Description
2	MaxRetryNumber	0	Must be 0 if working with Orchestrator queues. If > 0, the robot will retry the same transaction which failed with a system exception. Must be an integer value.
3			
4	ExScreenshotsFolderPath	Exceptions_Screenshots	Where to save exceptions screenshots - can be a full or a relative path.
5			
6	LogMessage_GetTransactionData	Processing Transaction Number:	Static part of logging message. Calling Get Transaction Data.
7	LogMessage_GetTransactionDataError	Error getting transaction data for Transaction Number:	Static part of logging message. Error retrieving Transaction Data.
8	LogMessage_Success	Transaction Successful.	Static part of logging message. Processed Transaction succesful.
9	LogMessage_BusinessRuleException	Business rule exception.	Static part of logging message. Processed Transaction failed with business exception.
10	LogMessage_ApplicationException	System exception.	Static part of logging message. Processed Transaction failed with application exception.

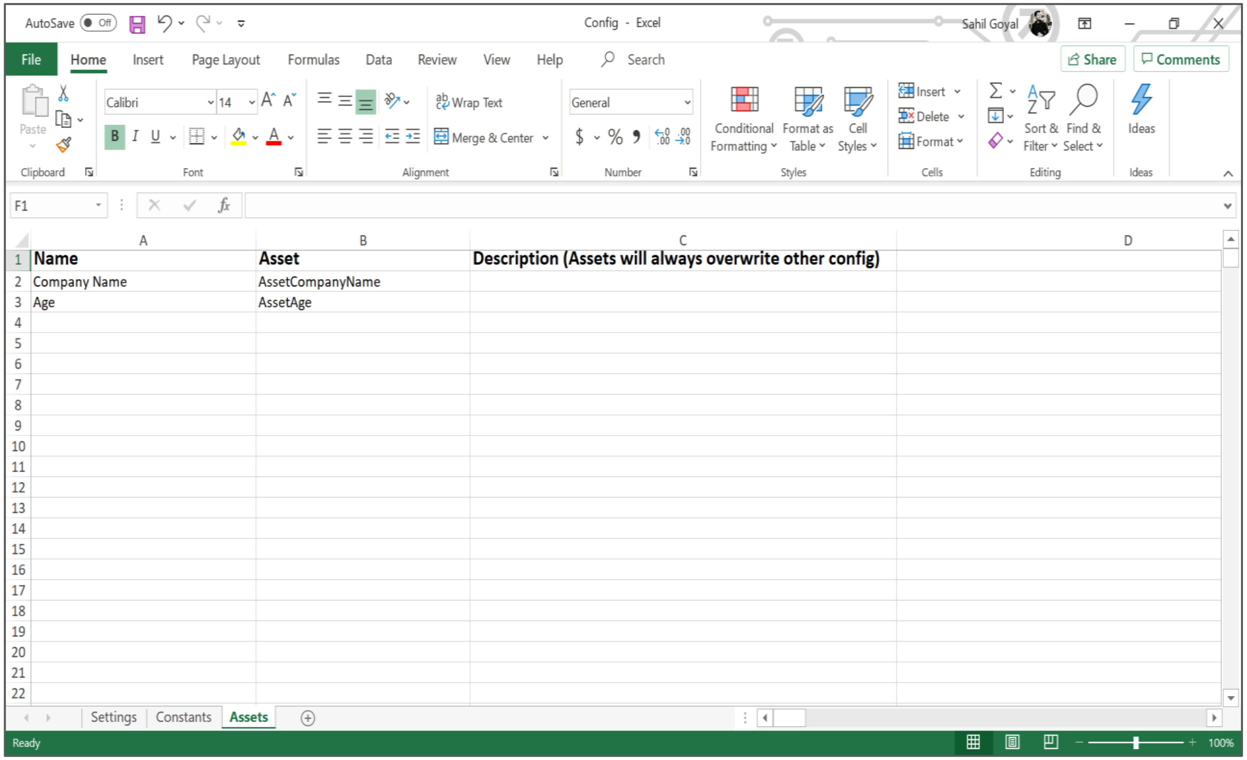
Config File. *Assets* Sheet

- **Assets** sheet contains
  - the values defined as assets in **Orchestrator**;
- it shows the relationship between assets defined in **Orchestrator**, their definition in the **Assets** sheet of the **Config.xlsx** file and their usage in workflows.

Name
Specifies a key for a config dictionary (Example: Company Name, Age)

Asset
Determines the name of the asset as defined in Orchestrator (Example: AssetCompanyName, AssetAge)

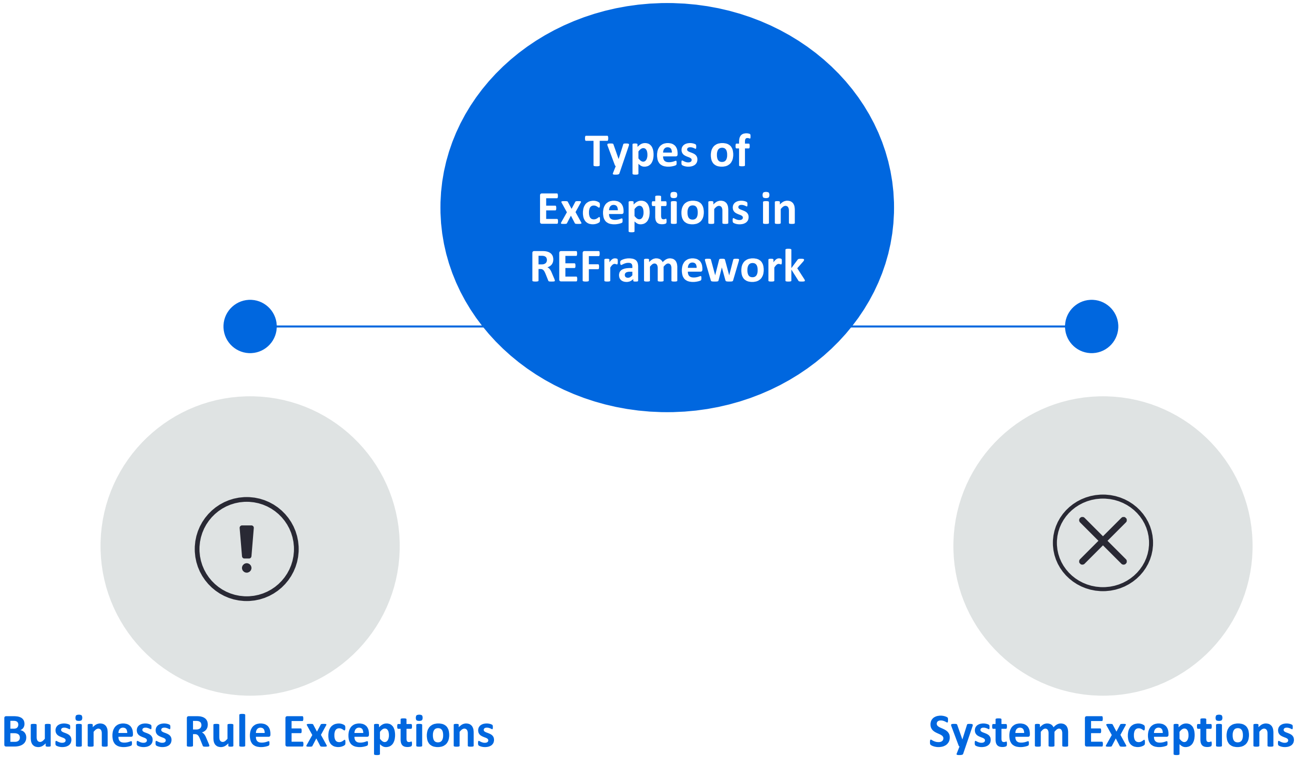
Description
Gives an explanation about the asset



Name	Asset	Description (Assets will always overwrite other config)
Company Name	AssetCompanyName	
Age	AssetAge	

Exception Handling. Details

- REFramework enables the recovery from exceptions by:
  - ***attempting*** to process the transaction again (i.e., retrying) or
  - ***skipping*** that transaction depending on the type of exception.

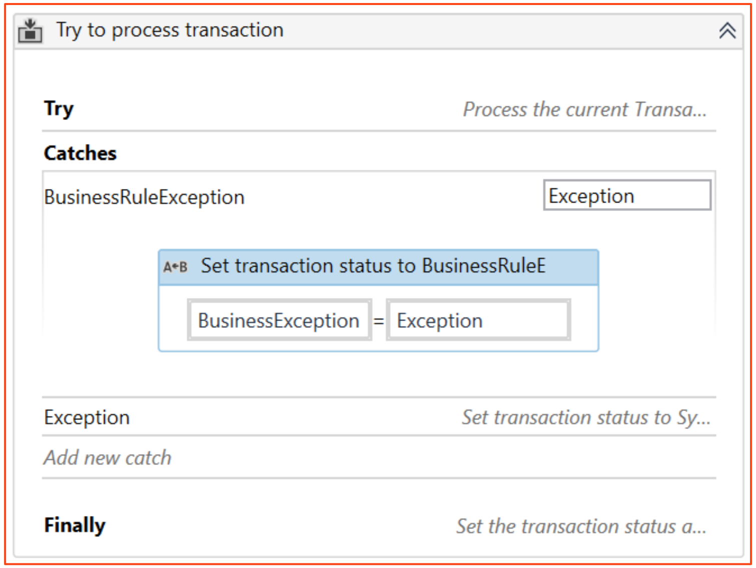


Exception Handling. Business Rule Exceptions

- a **Business Rule Exception** describes
  - an error that occurs when
    - some crucial data is incomplete or
    - missing from the automation project or
    - when the developer encounters unknown scenarios;
- it is **manually** triggered by the developer using the **Throw** activity.

Handling the Business Exception in REFramework:

- the **Try** section of the **Try Catch** activity invokes the **Process.xaml** file;
- when a Business Exception is thrown by the process, the **TransactionStatus** is set to *BusinessException*;
- the transaction is skipped and the framework proceeds to the next transaction.



Try to process transaction

Try *Process the current Transa...*

Catches

BusinessRuleException Exception

Set transaction status to BusinessRuleE

BusinessException = Exception

Exception *Set transaction status to Sy...*

Add new catch

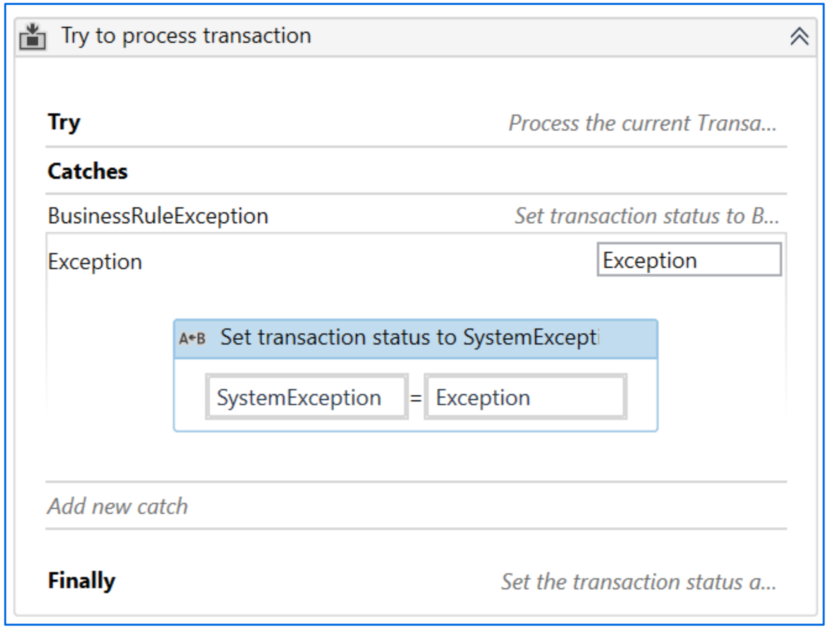
Finally *Set the transaction status a...*

Exception Handling. System Exceptions

- a **System Exception** describes
  - an error based on a technical issue, such as an application that is not responding;
- it is triggered **automatically** by activities that fail, or **manually** by the developer using the **Throw** activity.

Handling the System Exception in REFramework:

- the **Try** section of the **Try Catch** activity invokes the **Process.xaml** file;
- when a System Exception is thrown by the process, the **TransactionStatus** is set to *SystemException;*
- the framework automatically restarts the applications and tries to process the same transaction again.



Try to process transaction

Try *Process the current Transa...*

Catches

BusinessRuleException *Set transaction status to B...*

Exception Exception

Set transaction status to SystemExcept

SystemException = Exception

Add new catch

Finally *Set the transaction status a...*

Setting the Transaction Status

- the **Finally** clause of the **Try Catch** activity invokes the **SetTransactionStatus** workflow which is used to set the transaction status of an item to either **Success**, **Business Exception** or **System Exception**.

Success

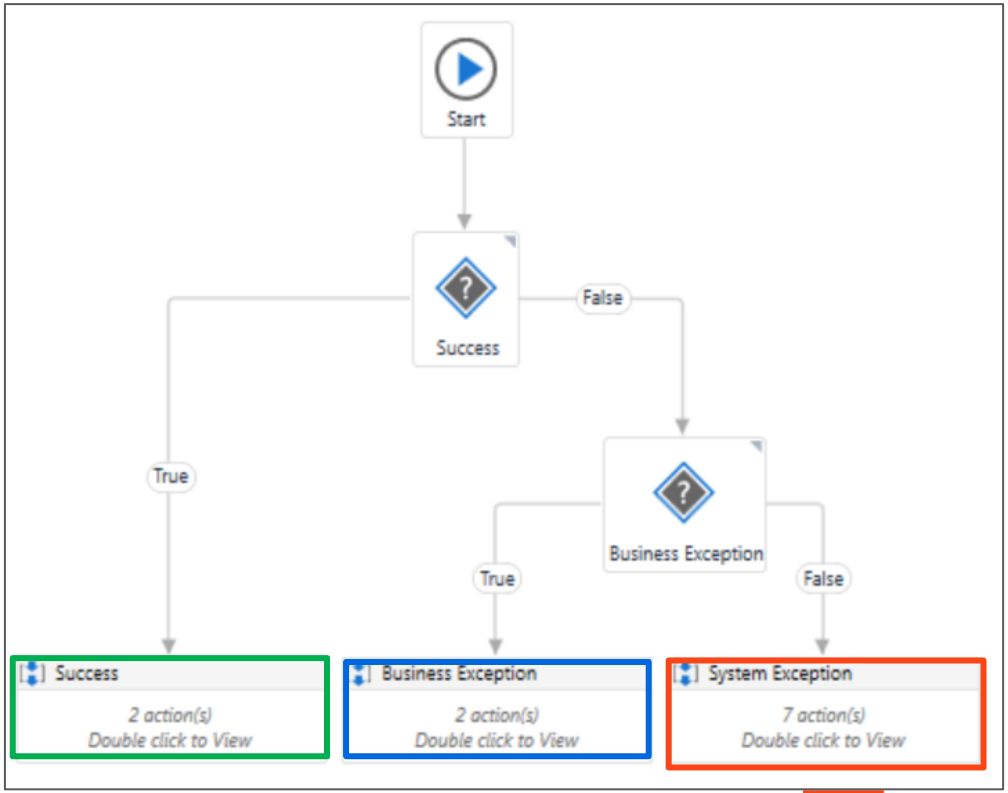
If the transaction item is processed without any exception, its status is updated as Successful

Business Exception

If a **Business Exception** is thrown during the process, the transaction item's status is updated as Failed

System Exception

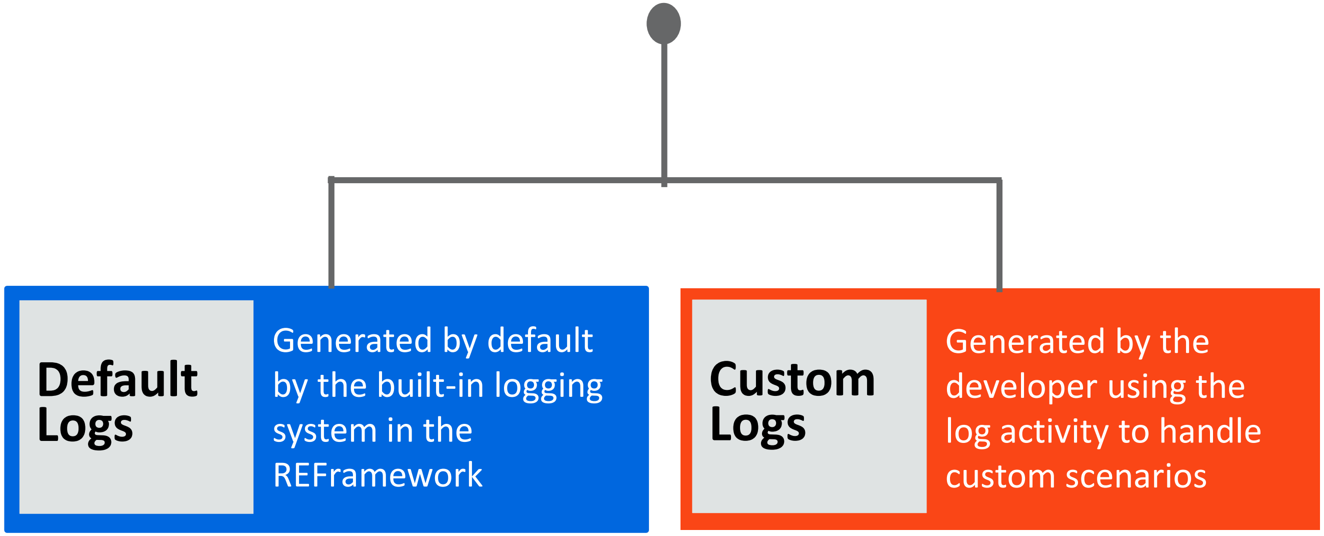
If a **System Exception** occurs during the process, the transaction item's status is updated as Failed



Logging. Details

- REFramework has a **logging** structure that uses different levels of the **Log Message** activity to output the statuses of *transactions*, *exceptions*, and *transitions* between states.

Types of Logs



Default Logs

Generated by default by the built-in logging system in the REFramework

Custom Logs

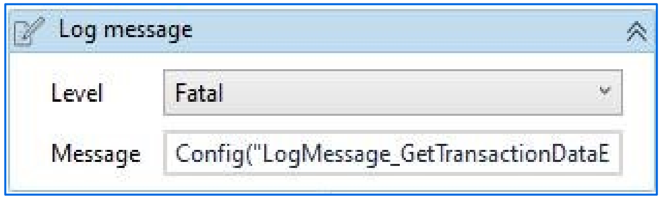
Generated by the developer using the log activity to handle custom scenarios

Default Logs. Details

- the default **logs** according to the logging levels are:

Fatal Level

Log	Workflow
1. Error in extracting the transaction data from a specific transaction number	Main.xaml
2. SetTransactionStatus.xaml failed	Main.xaml
3. System error at initialization	Main.xaml



Log message

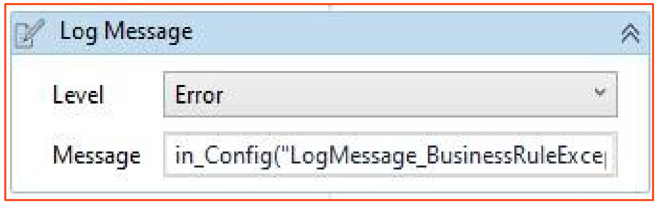
Level: Fatal

Message: Config("LogMessage_GetTransactionDataE

Log message for error in extracting the transaction data from a specific transaction number

Error Level

Log	Workflow
1. Business Rule Exception	Main.xaml
2. System Exception after reaching Max number of retries with the Error message and source information/description	Main.xaml
3. System Exception with an error message and source information/description	Main.xaml



Log Message

Level: Error

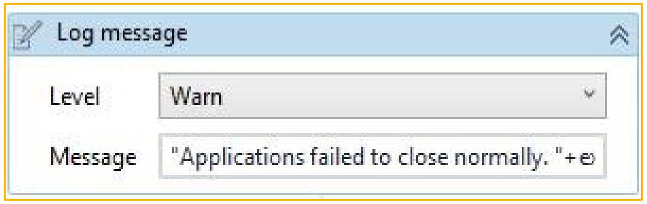
Message: in_Config("LogMessage_BusinessRuleExce

Log message for Business Rule Exception

Default Logs. Details (cont.)

Warn Level

Log	Workflow
1. Applications failed to close normally	Main.xaml
2. Failure in loading assets from Orchestrator	InitAllSettings.xaml
3. System Exception with an error message and source information	SetTransactionStatus.xaml
4. Take screenshot failed with error	SetTransactionStatus.xaml
5. CloseAllApplications failed	SetTransactionStatus.xaml
6. KillAllProcesses failed	SetTransactionStatus.xaml



Log message

Level: Warn

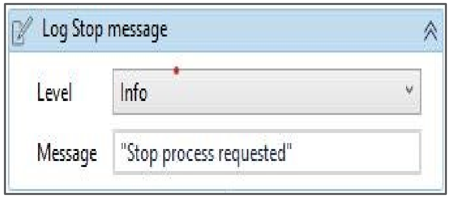
Message: "Applications failed to close normally. "+e

Log message for Applications failed to close normally

Default Logs. Details (cont.)

Information Level

Log	Workflow
1. Stop Process Requested	Main.xaml
2. Information about the current transaction number	Main.xaml
3. Process finished due to no more transaction data	Main.xaml
4. Opening applications	InitAllApplications.xaml
5. Successful transactions	SetTransactionStatus.xaml
6. Exception screenshot saved to the folder specified in the Config.xlsx/Constants sheet	TakeScreenshot.xaml
7. Closing applications	CloseAllApplications.xaml
8. Killing processes	KillAllProcesses.xaml


Log Stop message

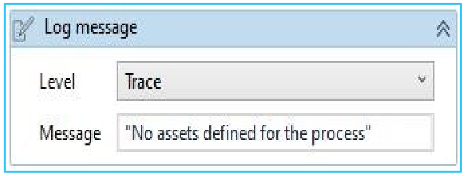
Level: Info

Message: "Stop process requested"

Log message for Stop Process Requested

Trace Level

Log	Workflow
1. No assets defined for the process	InitAllSettings.xaml


Log message

Level: Trace

Message: "No assets defined for the process"

Log message for No assets defined for the process

Custom Logs. Details

- **Custom** log messages can be added in REFramework to include additional information about transactions;
- some of the custom log fields are:

logF_BusinessProcessName
Holds the name of the business process

logF_TransactionStatus
Holds the status of the transaction

logF_TransactionNumber
Holds the number of the transaction index, TransactionNumber

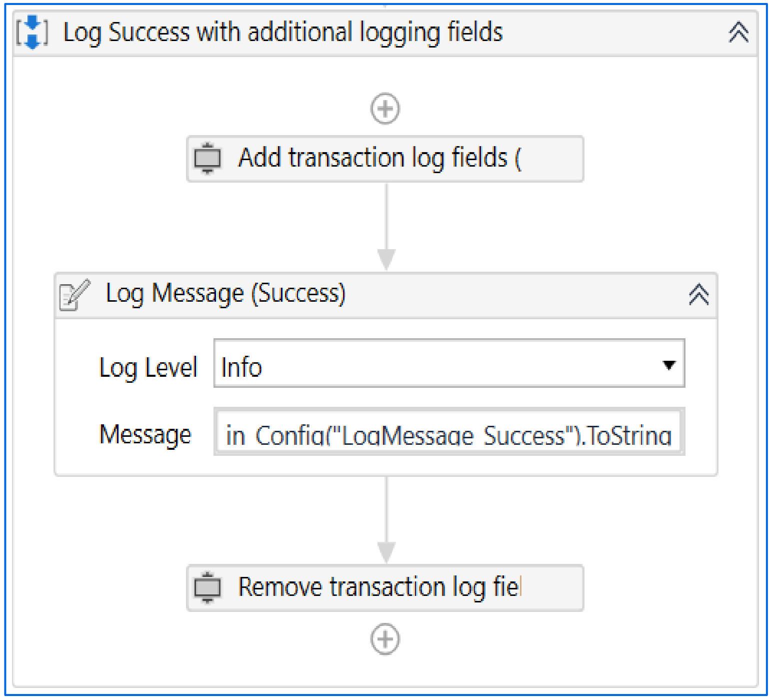
logF_TransactionID
Holds the value of the variable TransactionID

logF_TransactionField1
Holds the value of the variable TransactionField1

logF_TransactionField2
Holds the value of the variable TransactionField2

Adding Custom Logs. Details

- **Custom log messages** are added in REFramework using the **Add Log Fields** activity.



Log Success with additional logging fields

Add transaction log fields (

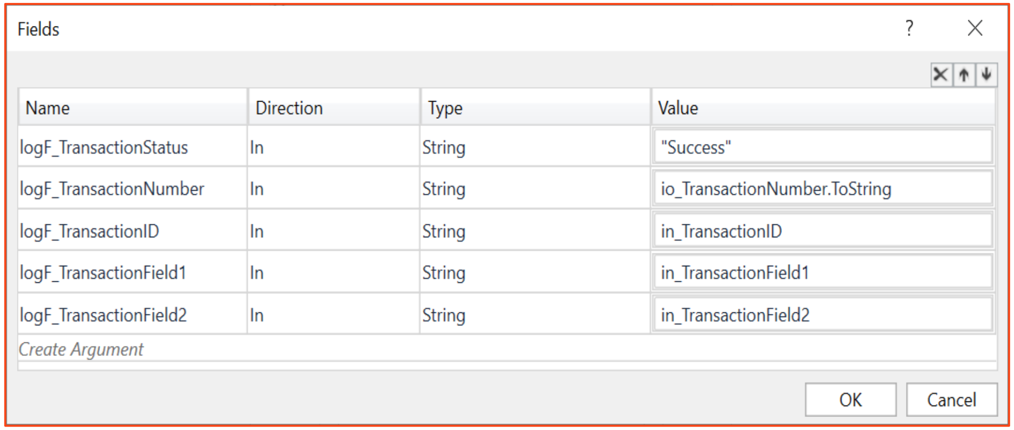
Log Message (Success)

Log Level: Info

Message: in_Config("LogMessage_Success").ToString

Remove transaction log fiel

Using the **Add Log Fields** activity in the **SetTransactionStatus.xaml** file



Fields

Name	Direction	Type	Value
logF_TransactionStatus	In	String	"Success"
logF_TransactionNumber	In	String	io_TransactionNumber.ToString
logF_TransactionID	In	String	in_TransactionID
logF_TransactionField1	In	String	in_TransactionField1
logF_TransactionField2	In	String	in_TransactionField2

Create Argument

Adding custom log fields to **Robot Execution Logs**

REF Implementation. Details

- the implementation of REFramework can be done in two ways:

Without Orchestrator

- Orchestrator **Queue** is not used, and the variable type of input transaction item should be matched to the variable type of the transaction in the process (Example: **DataRow**, **MailMessage**, etc.)

With Orchestrator

- Orchestrator **Queue** is used, and the predefined variable type, **QueueItem**, needs no modification

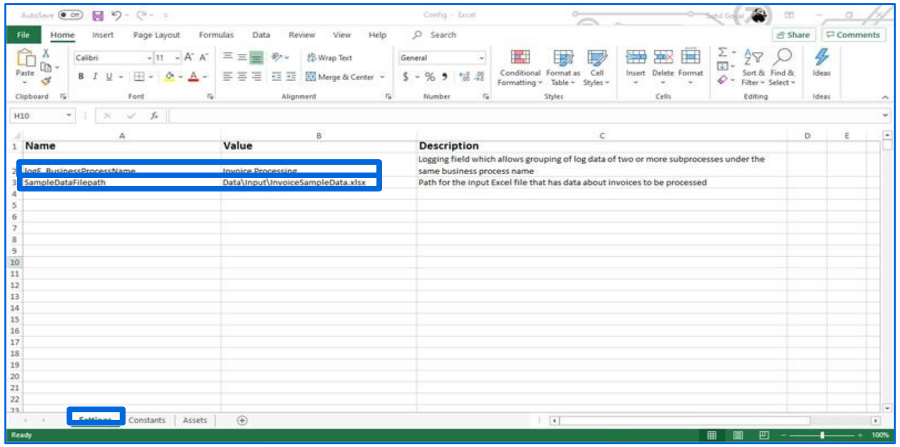
REF Implementation. Without Orchestrator

- when REFramework is used without **Orchestrator**, the modifications required in the **Config.xlsx** file are:

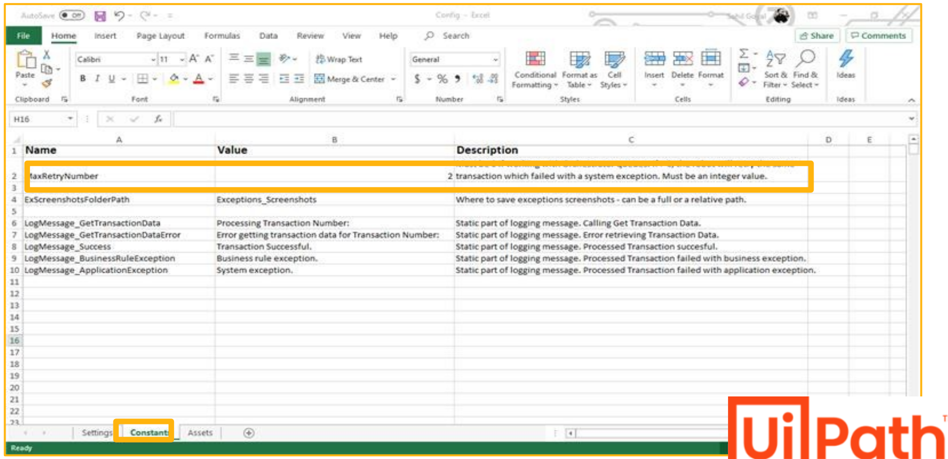
Config.xlsx

1. Change the value of the **logF_BusinessProcessName** setting to match the name of the process
   Example: If using an Excel file (that has data to be processed), specify the path for the input Excel file, by adding a new setting parameter with **SampleDataFilepath** as the name and, Data\Input\InvoiceSampleData.xlsx as the value

2. Change the value of **MaxRetryNumber** to an integer greater than zero



Name	Value	Description
logF_BusinessProcessName	Invoice Processing	Logging field which allows grouping of log data of two or more subprocesses under the same business process name
SampleDataFilepath	Data\Input\InvoiceSampleData.xlsx	Path for the input Excel file that has data about invoices to be processed



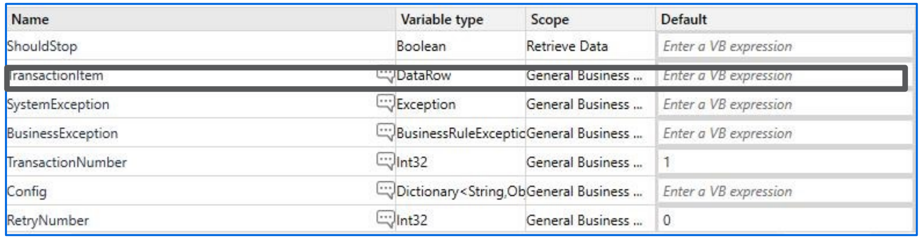
Name	Value	Description
MaxRetryNumber	2	transaction which failed with a system exception. Must be an integer value.
ExScreenshotsFolderPath	Exceptions_Screenshots	Where to save exceptions screenshots - can be a full or a relative path.
LogMessage_GetTransactionData	Processing Transaction Number:	Static part of logging message. Calling Get Transaction Data.
LogMessage_GetTransactionDataError	Error getting transaction data for Transaction Number:	Static part of logging message. Error retrieving Transaction Data.
LogMessage_Success	Transaction Successful.	Static part of logging message. Processed Transaction succesful.
LogMessage_BusinessRuleException	Business rule exception.	Static part of logging message. Processed Transaction failed with business exception.
LogMessage_ApplicationException	System exception.	Static part of logging message. Processed Transaction failed with application exception.

REF Implementation. Without Orchestrator (cont.)

- when REFramework is used without **Orchestrator**, the modifications required in the **Main.xaml** workflow are:

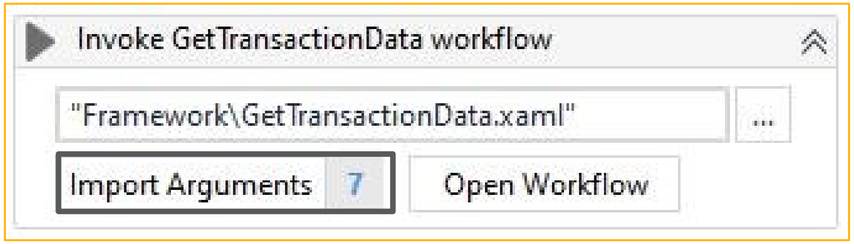
Main.xaml

1. Match the variable type of input transaction item to the variable type of the transaction in the process
    - Example: Use DataRow in case rows are being read from an Excel file, or MailMessage in case emails are retrieved from an email account



Name	Variable type	Scope	Default
ShouldStop	Boolean	Retrieve Data	*Enter a VB expression*
TransactionItem	DataRow	General Business ...	*Enter a VB expression*
SystemException	Exception	General Business ...	*Enter a VB expression*
BusinessException	BusinessRuleExceptic	General Business ...	*Enter a VB expression*
TransactionNumber	Int32	General Business ...	1
Config	Dictionary<String,Ob	General Business ...	*Enter a VB expression*
RetryNumber	Int32	General Business ...	0

2. Use the Import Arguments button of the Invoke Workflow File activity to update the arguments according to the type of the transaction in the process



Invoke GetTransactionData workflow

"Framework\GetTransactionData.xaml"

REF Implementation. Without Orchestrator (cont.)

- when REFramework is used without **Orchestrator**, the modifications required in the workflows are:

GetTransactionData.xaml

- Match the variable type of **out_TransactionItem** to the variable type of the transaction in the process (Example: DataRow, MailMessage, etc.)
- Replace the **Get Transaction Item** activity with the appropriate data retrieval method

Process.xaml

Match the variable type of input transaction item to the variable type of **TransactionItem** in **Main.xaml**

SetTransactionStatus.xaml

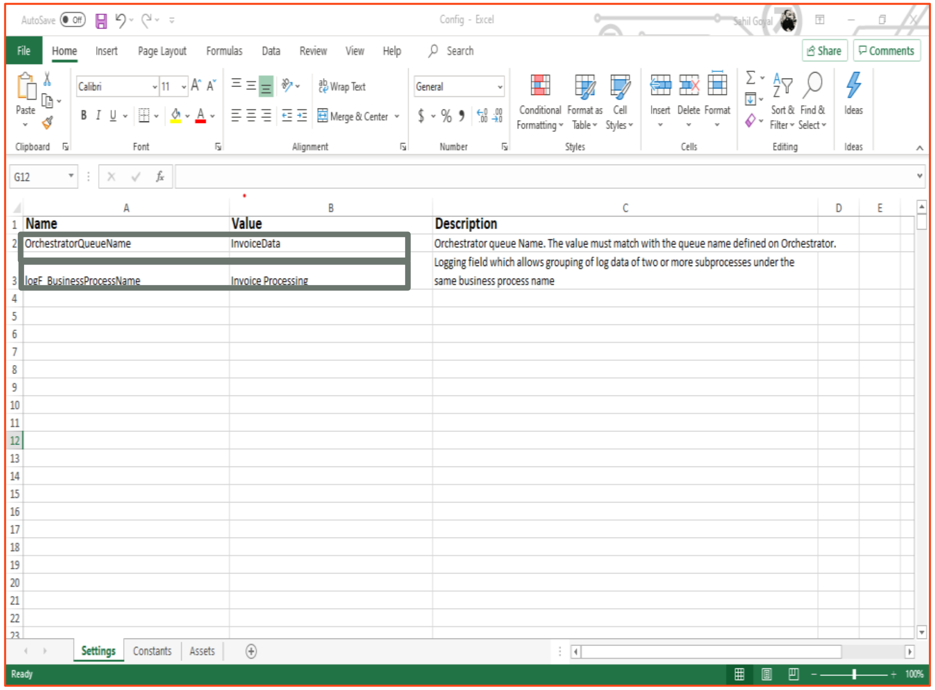
- Match the variable type of input transaction item to the variable type of **TransactionItem** in **Main.xaml**
- Set the transaction status according to the process

REF Implementation. With Orchestrator

- when REFramework is used with **Orchestrator**, the modifications required in the **Config.xlsx** file are:

Config.xlsx

- Change the value of the **OrchestratorQueueName** setting to match the name of the queue as defined in the Orchestrator
  - Example: Suppose the name of the Orchestrator queue is InvoiceData. So, the value of the field OrchestratorQueueName is changed from default value to InvoiceData
- Change the value of the **logF_BusinessProcessName** setting to match the name of the process
  - Example: Suppose the name of the process is Invoice Processing. So, the value of the field logF_BusinessProcessName is changed from default value to Invoice Processing



Name	Value	Description
OrchestratorQueueName	InvoiceData	Orchestrator queue Name. The value must match with the queue name defined on Orchestrator.
logF_BusinessProcessName	Invoice Processing	Logging field which allows grouping of log data of two or more subprocesses under the same business process name

REF Implementation. With Orchestrator (cont.)

- when REFramework is used with **Orchestrator**, no modifications are required in the workflows:

Main.xaml
No modifications required in the workflow as the default type of **Transaction Item** is **QueueItem**

GetTransactionData.xaml
No modifications required in the workflow as the transaction retrieval is handled by the **Get Transaction Item** activity included by default

Process.xaml
No modifications required in the workflow as each transaction item is accessible via the argument **in_TransactionItem**

SetTransactionStatus.xaml
No modifications required in the workflow as the status of the queue item is updated by the **Set Transaction Status** activity by default

Best Practices in Using REFramework

- the best practices for using REFramework are:

✓ Always open the applications in **InitAllApplications.xaml** workflow

✓ Always close the applications in **CloseAllApplications.xaml** workflow

✓ Always kill the applications in the **KillAllApplications.xaml** workflow

✓ Separate configuration values from workflows by keeping them in a configuration file

✓ Assign the null pointer, **Nothing**, to the **TransactionItem** at the end of the process

✓ Use the **TransactionNumber** index to loop through **TransactionData** and obtain new **TransactionItem**

Next lecture…

- week 08 –
  - Lecture 08
    - Image and Text automation
    - Data Tables. Excel Automation

References

- UiPath Docs
  - https://docs.uipath.com/studio/docs
- UiPath Forum
  - https://forum.uipath.com/
- UiPath Academy
  - https://academy.uipath.com/