The Initialization State and the Configuration File

Detail on the **Initialization state**, focusing on its role, **exception handling mechanism**, the **Configuration file**, and the workflows that compose this state.

Initialization state

- In the Initialization state, the robot reads the **configuration settings** and opens the needed **applications**. In our case, this is where it will open and log into the UI Demo application.
- The **first** thing we see in this state is the **Try Catch activity**.
- It's **important to note that the overall exception handling is already implemented throughout the framework**. Thus, we can find Try Catch activities in every state.
- In the **Try block**, the robot attempts to initialize and **if an exception is encountered**, it transitions to **the End Process state**.

So how does this work?

When an **exception occurs in this state**, the exception object is stored in the SystemException variable.

SystemException variable = Exception Object

The **SystemException variable is initialized** with the value **Nothing** at the beginning of the Try block.

SystemException variable = Nothing

Afterward, if this is the first run of the state

- The robot reads the **configuration file** and attempts **to open all applications**.
- If an **exception is caught**, the **Catches block is executed** where we **assign** the **Exception object to the SystemException variable.**
- **The value of the variable determines the transitions from this state.**
- If the value is not Nothing, meaning it holds an Exception object, the robot logs a Fatal level message and transitions to the End Process state.

if this is the first run of the state→ ROBOT READS→ Config file & open All application → if(exception is caught)→ Catches block is excuted → Assign(ExceptionObject = SystemException variable)

Configuration file

Why We are using Configuration file?

- To make it easier to **maintain a project and quickly change configuration values**.
- A configuration file is used to **define parameters** that are used **throughout the project** and to **avoid values hardcoded in workflows**.

Location: -

- The REFramework offers a configuration file, named **Config.xlsx** and **located in the folder named Data**, which can be used to define project configuration parameters.

In which state file will be read?

- This file is read in the Initialization state.
- You may remember that the Initialization state is run at the start of the process execution and whenever a System Exception is encountered in the Process Transaction state if retries are available.

How many times Config File will Read and what condition?

- It's important to note that the Config file is read only **once per job**, at the **start of the process execution**.
- The condition is that the value of the Config variable is Nothing.

  Condition→Config Variable should be Nothing.

Config Variable

Config is a dictionary where the key is of type String and the value of type Object. This way we can store any type of value such as number, string, datetime, and so on.

The Config Excel file is read in the InitAllSettings workflow which is invoked in the Then block.

Arguments for the invoked workflow:

We have two In arguments and an Out argument.

- **In_ConfigFile** is of type string and passes the path of the Configuration file.

In_ConFigFile(String)→ Passes→ Path of Configuration File

- **In_ConfigSheets** is of type array of string and passes the names of the sheets which contain configuration data.

In_ConfigSheets(Array of String)→Passes→ names of the sheets which contain configuration data

- **Out_Config** is a dictionary of type String, Object which passes the data extracted from the Excel file to be used throughout the process**.**

Out_Config(dictionary of type String, Object)→ Passes → data extracted from the Excel file to be used throughout the process

If an exception occurs during the execution of InitAllSettings workflow - for example, if the configuration file is not found, it is caught by the Try Catch activity in the Initialization state, and the execution transitions into the End Process state.

It is named Config.xlsx and can be used to define project configuration values. These values are then read into the Config dictionary.

Configuration file sheets(Settings, Constants, and Assets)

Reading the Configuration file is one of the key functions of the Initialization state. Let's find out what settings we can define in this file and how the process handles them

The configuration file is called Config.xlsx and can be found in the Data folder of the project.

By default, it contains three sheets - Settings, Constants, and Assets.

Settings sheet.

- Here we can store any configuration related to the business process.
- This is where we add **URLs, file paths, credential names, and any process specific piece of information**.
- For implementations where we use **Queue Items**, this is where we **add the name** of the **Orchestrator queue** which will be used in the process.
- This is also where we **add the business process name**.

Note *credentials are added as assets in Orchestrator, they are included in the Settings sheet in the Config file*.

But how is this information read?

There are three columns: **Name, Value, and Description**.

The **Name column** always contains a **String**. The **robot assigns this information** as the **key in the Config dictionary**.

The information in the **Value column** is assigned as the **dictionary value tied to the key from the Name field**.

The Description column provides a detailed account of each setting, but it’s not used in the process. Its role is to **keep developers informed**.

Constants sheet

- The Constants sheet stores "technical" settings that are useful for developers.
- It contains information such as the number of **retries, default paths, and static log message parts**.
- **Instead of hard-coding values**, we can use the **Config object**, which **enables an easy global value adjustment**.

Why All These required ?

This way, we can **fine-tune projects**, **improve their performance**, and **smoothly switch their environments, from Dev to Test and then to Production**.

MaxRetryNumber:-

It is used to **determine how many times the Robot will retry a transaction** which has **failed with an Application Exception**.

The template is designed to work with **Orchestrator Queues by default**, but it can easily be adapted to suit other types **of input data, such as Excel workbooks, emails, files, folders, and so on**.

By default, the **MaxRetryNumber setting in the Config file is zero** as the **number of retries is set in Orchestrator**.

We strongly recommend not to set a different number in the Config file for projects which use queue items as input data.

In these situations, the value set in Orchestrator overrides the one set in the Config file.

if the process does not use the Queues functionality, we can set the value of MaxRetryNumber here, allowing the Robot to retry failed transactions.

Assets sheet

- The **Assets sheet defines the assets stored in Orchestrator used** in the **current automation process**.
- OrchestratorAssetFolder, we can specify the folder in which the asset is stored. If we leave it empty, the robot uses the default folder.
- ***The robot processes the Assets sheet after the Settings sheet and stores the results in the same Config dictionary.***
- ***This means that if we define an Asset with the same name as a Setting, the Asset value will overwrite the setting value if the asset is in Orchestrator***.
- If the **asset is not found in Orchestrator**, **no error is thrown**, and the **process uses the value defined in Settings sheet**.

<u>For the second note</u>, keep in mind that we can't use this mechanism for Assets of type Credential. Our Config dictionary stores a single value for each Asset, while a Credential type asset holds two values, one for the username and another for the password. We define Credential Assets in the Settings sheet because they require special handling.

- Setting up a config file for your workflows is an efficient way to configure the initial settings of your workflow environment. Variables that refer to path locations, URLs, and other config information that you need to set up when moving your workflow from one environment to another.
- The above example offers a detailed explanation of the three sheets - Settings, Constants, and Assets.
- Think of it as not having to go inside UiPath Studio to update your variables every now and then. Instead, you can just create a config file, and read it in your workflow.
- As a final note about Config.xlsx, since the configuration file is not encrypted, it should not be used to directly store credentials. Instead, it is safer to use Orchestrator assets or Windows Credential Manager to save sensitive data

<u>let's discover how the framework can perform actions to make sure that the system is in a clean state before the main process starts: -</u>

Above the role of the Initialization state, the Exception handling mechanism, the InitAllSettings workflow, and the Configuration file.

Now, we will focus on the other two workflows invoked in this state: **KillAllProcesses and InitAllApplications.**

After successfully read the settings from the Config file in the **InitAllSettings workflow and passed the data back to Main.xaml** to the **Config dictionary variable**.

In the next step, the Framework offers the option to **overwrite the queue name** ... by using a **Process Configuration or Runtime Argument**.

A Process Configuration or Runtime Argument is an argument with the direction In, declared in the project entry point, Main.xaml.

But where would an Input argument get its value from in Main?

Values can be assigned from Orchestrator. All we need to do is go to the Processes section, select our process, click Edit and provide values for the Runtime arguments.

Note that for attended processes, we can also assign values to Runtime arguments from UiPath Assistant.

KillAllProcesses workflow file. This workflow is meant to force close all applications used in the process.

We do this to make sure that the robot starts in a clean and controlled environment.

Example, in case, **when the process starts, an instance of Application may already be open in an incorrect state or even frozen.** This would likely **cause an exception**. The **placement of KillAllProcesses here addresses this potential issue**.

InitAllApplications.xaml workflow

is used to initialize any applications operated during the execution of the process. Note that the config dictionary object is passed to it via the In_Config argument.

By default, the workflow is empty. We recommend placing the steps to open and log into applications in separate workflows, one per application, invoked inside InitAllApplications.

Usually, the activities to open and log into applications are placed in separate workflows, one per application, so we can call these activities wherever is needed.

You might ask what is the difference between KillAllProcesses and CloseAll Applications. The first force closes the applications while the second gracefully follows the procedure to close the applications. CloseAllApplications is always used in a Try block with KillAllApplications in the Catches block as a failsafe.

The best part of UiPath ReFramework is that it is built in such a way that you can initialize