



RAMAIAH
Institute of Technology

ROBOTIC PROCESS AUTOMATION DESIGN & DEVELOPMENT



Subject Code: CSE552 (Credits: 3:0:0)

Error and Exception Handling:

Errors, Error handling approach, Try Catch, Retry Scope, Exception Handling, Types of Exceptions, Global Exception Handler, Best Practice for Error Handling

Orchestrator:

Overview, Orchestrator Functionalities, Orchestrator User Interface-Automations, Management and Monitoring

Errors are events that a particular program can't normally deal with. There are different types of errors, based on what's causing them - for example:

- **Syntax errors**, where the compiler/interpreter cannot parse the written code into meaningful computer instructions;
- **User errors**, where the software determines that the user's input is not acceptable for some reason;
- **Programming errors**, where the program contains no syntax errors, but does not produce the expected results. These are often called bugs.



The Error Handling activity offers 4 options

▲ Error Handling

- 📄 Rethrow
- ⦿ Terminate Workflow
- 📄 Throw
- 📄 Try Catch

Rethrow

Used to make sure activities occur before exception is thrown

Try Catch

Catches a specific exception type

Terminate Workflow

Used to terminate the workflow in case of errors

Throw

Used to simulate an exception

Exception Handling mainly deals with handling errors with respect to various activities in UiPath. The Error Handling activity offers four options:

- **Try Catch activity :**

Try Catch activity is used when you want to test something and handle the exception accordingly. So, whatever you want to test you can put it under the **try section**, and then if any exception occurs, then it can be handled using the **catch section**.

Apart from the try-catch, we also have a **Finally section** that performs the activities irrespective of whether an exception occurs or not.

Try Catch Activity

Try Block




All the possible activities that can create or cause error should be placed in this block.

Catch Block

The user can add multiple type of catches in this block.

FinallyBlock

It is used for the actions to be performed after the Try & Catch blocks.

 Try Catch  

Try

Drop activity here

Catches

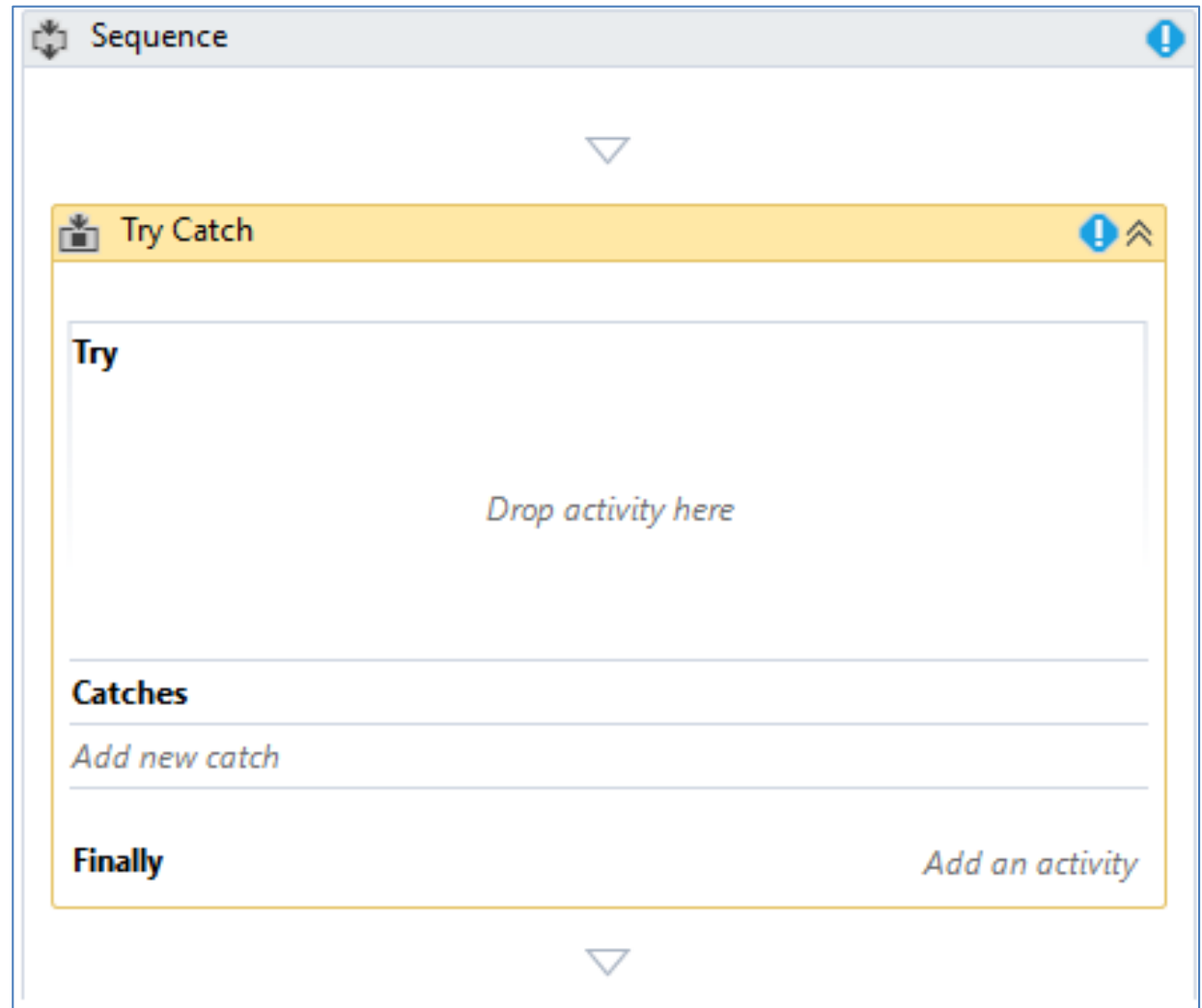
Add new catch

Finally




Add an activity




Step 1: Drag the Try-Catch activity.



Step 2: Drag a sample sequence or workflow inside it.

 Try Catch  

Try

 Sequence 

Double-click to view

Catches

Add new catch

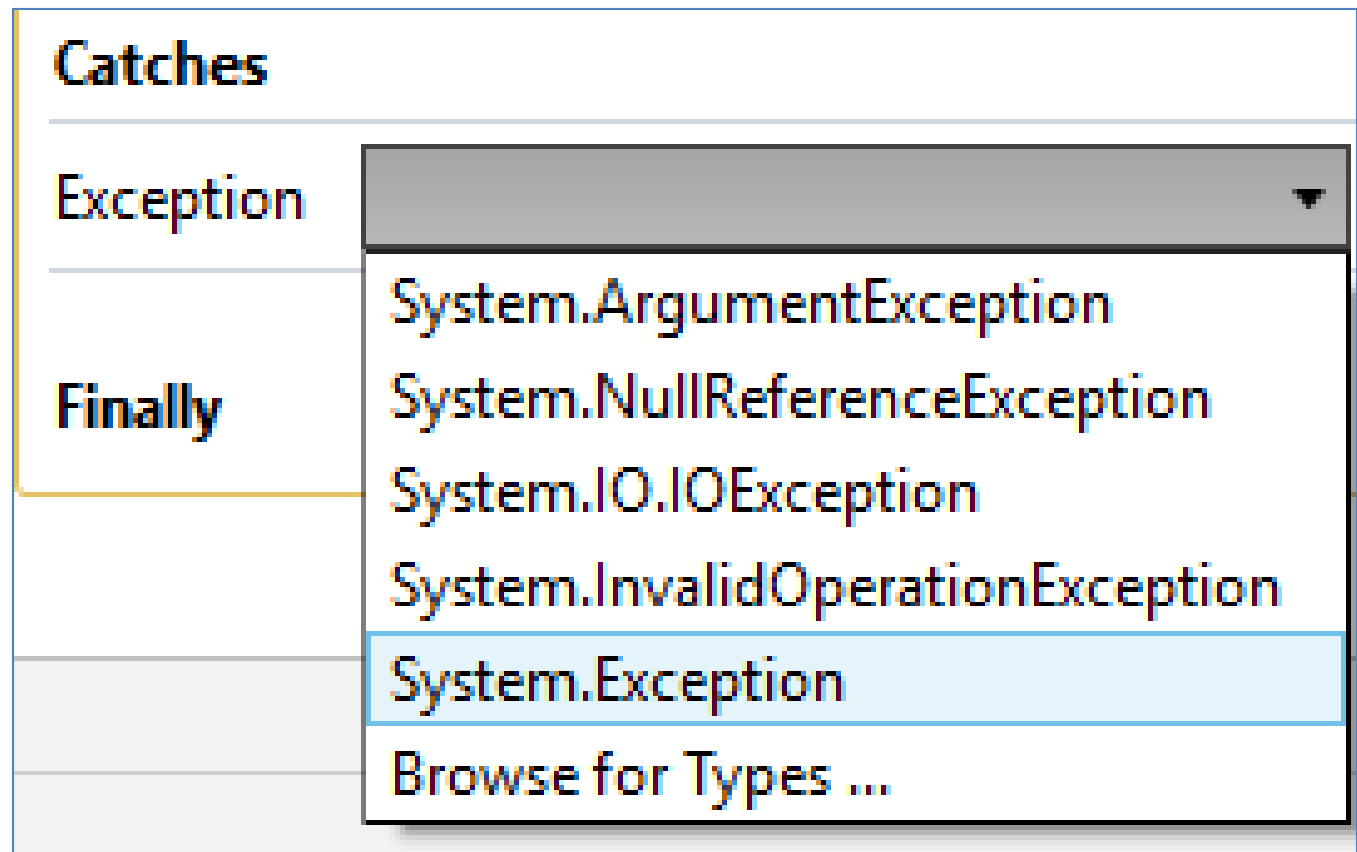
Finally

Add an activity



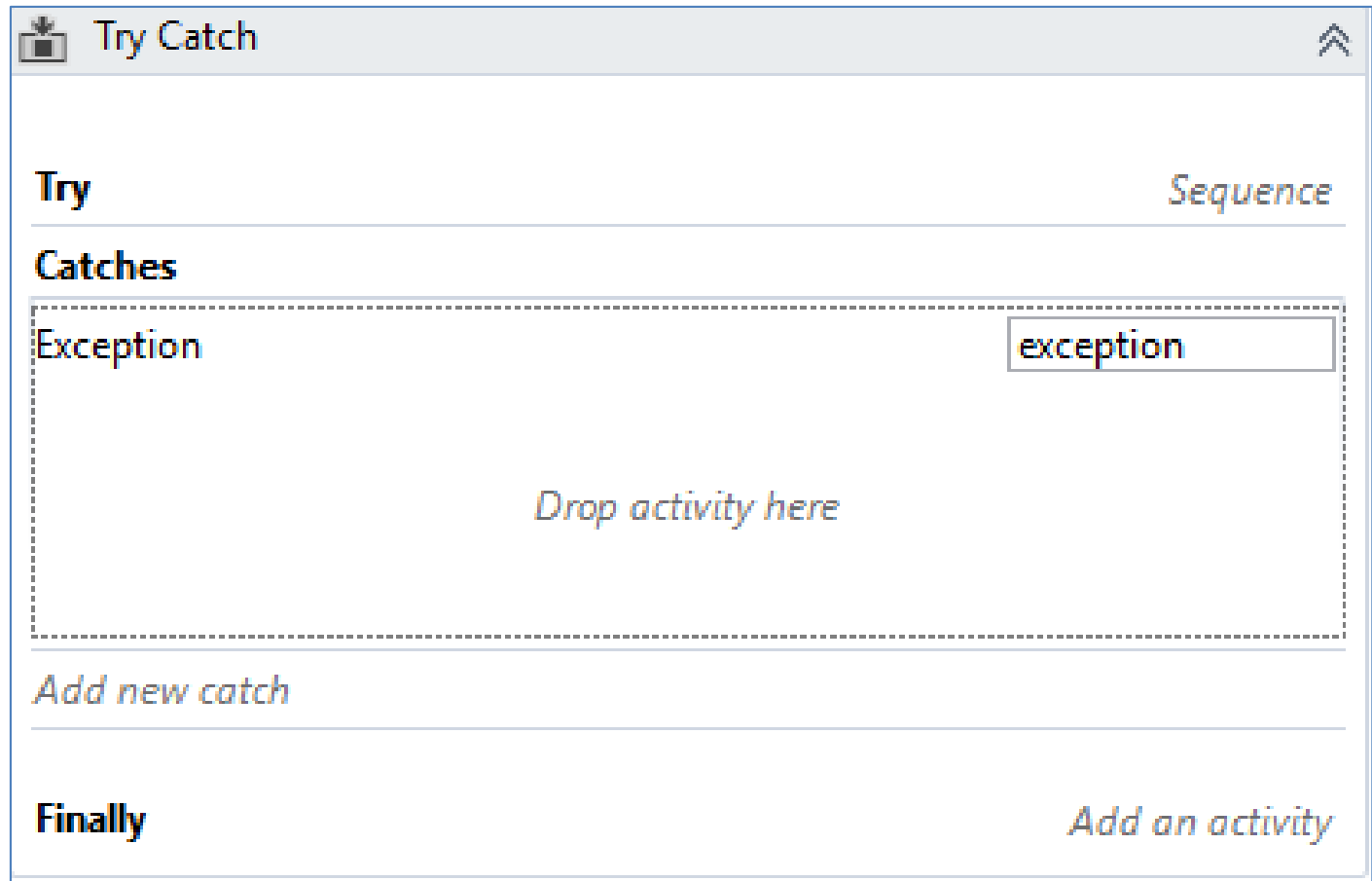
Step 3:

- Click on “Catches”
- Choose “System.Exception”
- Press Enter



Step 4: The user will get room for the actions in the Catches section.

Step 5: In the Catch block, add activities to take place when an error occurs in the main workflow.





The screenshot shows a 'Try Catch' configuration window. It has a title bar with a folder icon and the text 'Try Catch'. The main area is divided into sections: 'Try' (with a 'Sequence' label), 'Catches' (containing an 'Exception' section with a text input field labeled 'exception' and a 'Drop activity here' prompt), 'Add new catch' (a button), and 'Finally' (with an 'Add an activity' label). The 'Catches' section is highlighted with a dashed border.

Try	Sequence
Catches	
Exception	exception
Drop activity here	
Add new catch	
Finally	Add an activity



- Drag and drop the Log Message command inside the Catch block.
- Choose the level “Warn”.
- In the message column, write, “an error”+exception.Message.

Exception exception

 Log Message 

Level Warn

Message "an error"+exception.Message

Global Exception Handler

The **Global Exception Handler** is a type of workflow designed to determine the project's behavior when encountering an execution error.

- Only one Global Exception Handler can be set per automation project.
- The Global Exception Handler has two arguments, that should not be removed.
- The first argument is **errorInfo** with the **In direction** and it stores information about the error that was thrown and the workflow that failed. The level of the error to be logged can be set in the **Log Message activity**.

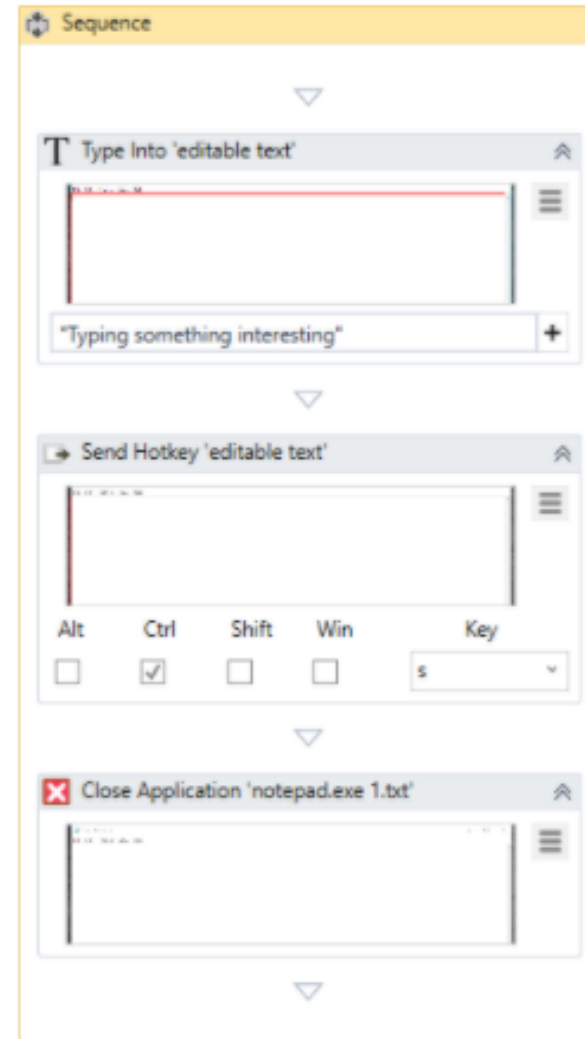
The second argument, **result** has the **Out direction** and it is used for determining the next behavior of the process when it encounters an error.

The following values can be assigned to the result argument:

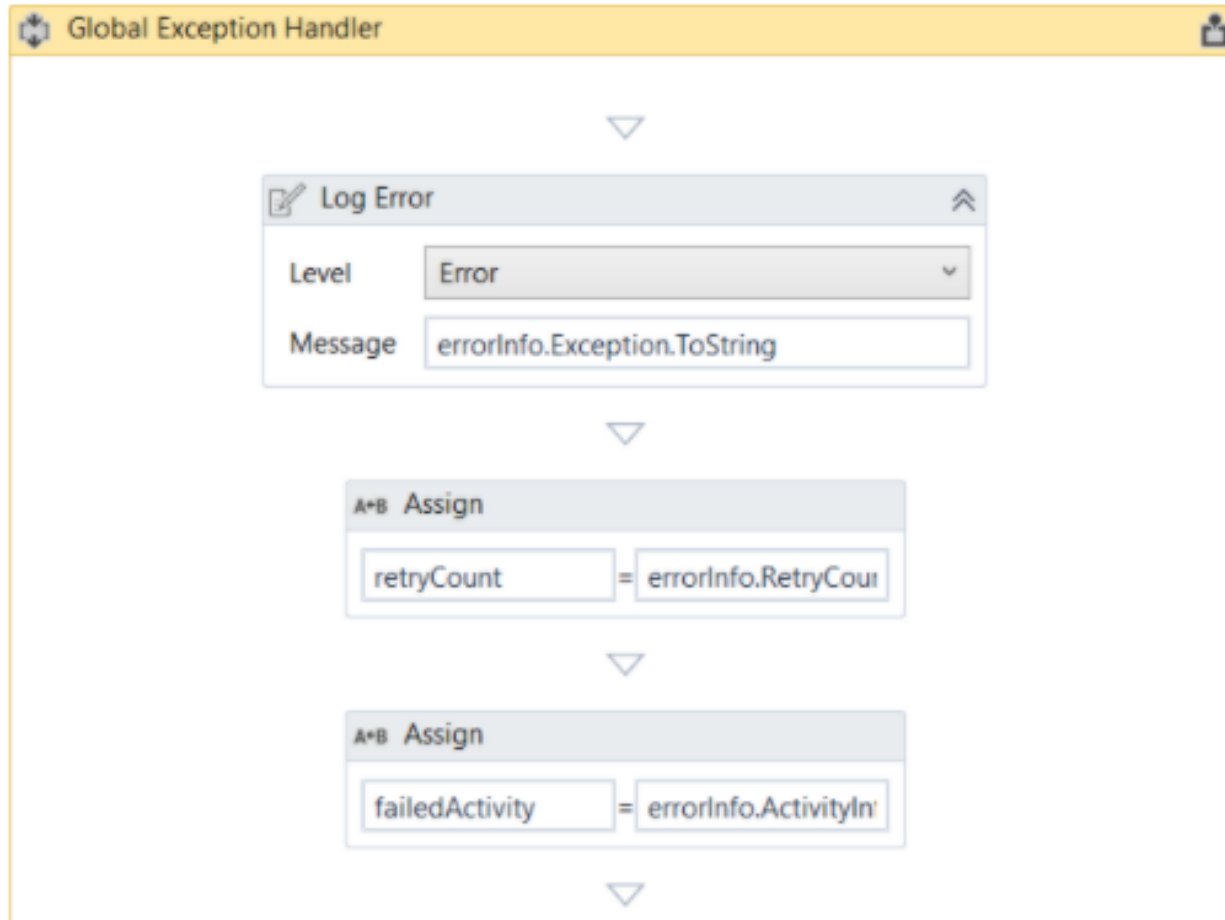
- **Continue** - The exception is re-thrown.
- **Ignore** - The exception is ignored, and the execution continues from the next activity.
- **Retry** - The activity which threw the exception is retried. Use the RetryCount method for errorInfo to count the number of **times the activity is retried**.
- **Abort** - The execution stops after running the current Global Exception Handler.

Example of Using the Global Exception Handler

Creating the Workflow



Adding a Global Exception Handler



Global Exception Handler

Log Error

Level: Error

Message: errorInfo.Exception.ToString




A*B Assign

retryCount = errorInfo.RetryCoun

A*B Assign

failedActivity = errorInfo.ActivityIn



 If  

Condition

retryCount < 3

Then

Else

A*B Assign

result = ErrorAction.retry

A*B Assign

result = ErrorAction.Abort



- 1. Organizing your project**
- 2. Choose the right layout**
- 3. Break up your workflow**
- 4. Exception handling and logging**
- 5. Readability**
- 5. Cleanup**
- 6. Naming conventions**

Organizing your project

- **Reliable** — create solid, robust workflows that can handle errors.
- **Efficient** — create a structure that:
 - Helps to short down development time.
 - Helps to reduce errors in production.
- **Maintainable** — create a structure that makes it easy to update when needed.
- **Extendable** — new requirements should be able to be implemented easily.

Choose the right layout

- The **main workflow works** best as a Flowchart or a State Machine.
- **Business logic works** best in Flowcharts as it is easier to visualize.
- Consider using ReFramework if you are using Orchestrator-queues.
- **UI interactions work** best as sequences as it does not involve much logic.
- If one sees that there is a need for much if-else checks and the like, then one should consider using a Flowchart.
- Avoid nested IF statements, use Flowcharts instead; It gives a tidier flow.

Break up your workflow

- Keywords: Extract as Workflow.
- Makes it possible to develop and test the various parts separately.
- Allows you to reuse workflows.
- Makes it easier to cooperate in a team when working on different files (use Invoke Workflow File).
- Easiest if you send pure string parameters and not lists and the like (is easier to set default parameters on primitive types like string, int32, etc.).

Exception handling and logging

- Put problematic workflows in try/catch blocks.
- Implement sequences within the Try / Catch blocks that allow you to restore the workflow.
- Use the Retry Scope for fragile(breakable) parts in the workflow.

Readability

- Give good, explanatory names to all components (workflows, activities, variables, etc.).
- Enter notes and comments where necessary. Annotations are very useful.
- Log real-time execution.

Cleanup

- Close applications, windows and web pages. If you do not, many interactions will have numerous instances of these running, which can eventually crash the machine.
- Always try soft-close applications first if the killing process is not used

Naming conventions

- When creating very large automation, it can be very easy to forget what every variable does. That is why it is important to have a good naming system in place.
- It is recommend that always use descriptive and accurate names, such as `userName` for a variable that stores the name of a user. This allows you to easily identify the information the variable stores.

An Application Exception describes an error rooted in a technical issue, such as an application that is not responding. Such a situation is, for example,

- a project which extracts phone numbers from an employee database, creating queue items for each of them.
- These items are then to be processed and inserted into a financial application.
- If, when the transaction is attempted, the financial application freezes, the Robot cannot find the field where it should insert the phone number, and eventually throws an error.

Note: System exceptions are the one which happens due to issue in application you are interacting with

A Business Exception describes an error rooted in the fact that certain data which the automation project depends on is incomplete or missing. Such a situation is, for example,

- a project which extracts phone numbers from an employee database, creating queue items for each of them.
- These items are then to be processed and inserted into a financial application.
- If a certain phone number is missing a digit due to human error, the queue item containing it becomes invalid.
- This causes the automation to throw an exception, as the Phone Number field in the financial application does not accept a queue item that contains an incomplete number.

Note: Business exceptions are the one which happens when some business rules are not satisfied.

Retries the contained activities as long as the condition is not met or an error is thrown.

Properties

Options

- **NumberOfRetries** - The number of times that the sequence is to be retried.
- **RetryInterval** - Specifies the amount of time (in seconds) between each retry.

Common

- **DisplayName** - The display name of the activity.
- **ContinueOnError** - Specifies if the automation should continue even when the activity throws an error. **This field only supports Boolean values (True, False). The default value is False.**
 - As a result, if the field is blank and an error is thrown, the execution of the project stops. If the value is set to True, the execution of the project continues regardless of any error.

- **Rethrow** : Rethrowing an exception If a catch block cannot handle the particular exception it has caught, you can rethrow the exception. The rethrow expression (throw without assignment_expression) causes the originally thrown object to be rethrown.
- **Terminate workflow** : Terminate workflow is used to terminate the workflow the moment the task encounters an exception.
- **Throw activity** : Throw activity is used when you want to throw an exception.

Orchestrator is the heart of your automation management. It gives you the power to provision, deploy, trigger, monitor, measure, track, and ensure the security of every robot

Orchestrator can be defined as a web application where people or end users can deploy their robots, monitor the health of the robots, manage their functioning and keep a log of tasks the robots

Orchestrator is a centralized application for deploying a robot, managing its functionality, monitoring the tasks it is undertaking and logging the reports in storage.

Orchestrator Main Capabilities

- **Provisioning** - creates and maintains the connection between Robots and the web application
- **Deployment** - assures the correct delivery of package versions to the assigned Robots for execution
- **Configuration** - maintains and delivers Robot environments and processes configuration

Orchestrator Main Capabilities

- **Queues** - ensures automatic workload distribution across Robots
- **Monitoring** - keeps track of Robot identification data and maintains user permissions
- **Logging** - stores and indexes the logs to an SQL database and/or Elasticsearch (depending on your architecture and configuration)
- **Inter-connectivity** - acts as the centralized point of communication for 3rd party solutions or applications

1.Processes: A process represents the association between a package and an environment. Each time a package is linked to an environment, it is automatically distributed to all the Robot machines that belong to that environment.

2.Assets: Assets usually represent shared variables or credentials that can be used in different automation projects. They give you the opportunity to store specific information so that the Robots can easily have access to it.

3.Queues: A queue is a container that enables you to hold an unlimited number of items. New queues created in Orchestrator are empty by default and can store multiple types of data.

4.Schedules: Schedules enable you to execute jobs in a pre-planned manner, at regular intervals on Robots. Input values for processes which support input and output parameters can be managed at this level as well

5. Robots: A Robot is an execution host that runs processes built in UiPath Studio. The Robots page enables you to add robots, edit them, view their status and other settings.

6. Jobs: A job is the execution of a process on one or multiple Robots. After creating a process the next step is to execute it by creating a job. When creating a new job, you can assign it to specific Robots.

7. Transactions: The Transactions page displays the transactions from a given queue. It also shows their statuses, the dates when they should be processed, the Robot that processed them, and the type of exception thrown or assigned reference, if any.

- Deployment:** One can deploy their robots on this platform once they are created and developed completely.
- Scheduling:** One can schedule when and how the robot must run and what and all should be logged.
- Management:** If a developer deploys numerous number of robots on the portal, the Orchestrator offers multiple features for the management of these robots.
- Maintenance:** It is a very useful feature. Once developers have deployed their robots, they can manage and maintain these robots using the Orchestrator.
- Monitoring:** One can monitor the real-time run of the robots using UiPath Orchestrator and see how the robot executes each process defined by the developer.

- Checks the Performance of the Robot:** The concept of a robot's heartbeat is unique to the Orchestrator. Here the robots which are running fine or are in good health, send constant heartbeats (or signals) to the central server. Once the heartbeats become less frequent, the server is alerted for the declining health of the robot and thereby alerts the developer/manager too.
- Common and Shared Assets:** The robots are designed in such a way that they need some resources to execute the tasks defined by the developers. These resources are called assets. They are reusable and stored within the Orchestrator. They are provided to the robots whenever needed.

- Exception is Handled Smartly:** If there occurs any exception in the process run of the robot, it doesn't immediately send out a notification. the Orchestrator makes the robot attempt the task one more time as a precautionary check. If it functions properly in the second run, the schedule remains as is. If it still encounters the exception, an alert is sent out to the developer to check the issue.

- Choice Amongst Multiple Clouds:** There are various choices for clouds offered by the Orchestrator. It can be either on one's personal cloud or the cloud offered by the Orchestrator.

- Remote Monitoring:** A developer can easily monitor the robots from anywhere using their cell phones.

Attended Robots

Supervised Robots that run under human supervision. Can be further classified according to their licensing type as follows:

- **Attended** - Works on the same workstation as a human user and is launched through user events.
- **Studio** - Connects your Studio to Orchestrator for development purposes.

Unattended Robots

Autonomous Robots that don't require human supervision to execute jobs. Can be further classified according to their licensing type as follows:

- **Unattended** - Runs without human supervision in virtual environments and can automate any number of processes.
 - Has all the capabilities of an attended robot plus remote execution, monitoring, scheduling, and support for work queues. Can execute any process type except for test cases.

Connect a Robot to Orchestrator

Step 01

- Log in to Orchestrator & navigate to Services

Step 02

- Create a new machine

Step 03

- Name the machine as per the Orchestrator Settings of the robot

Step 04

- Navigate to Robots and add a Standard Robot

Step 05

- In Orchestrator settings, add Orchestrator URL & machine key

Publish a Robot to Orchestrator

Step 01

- Open a robot in UiPath Studio.

Step 02

- Click on Publish in Design ribbon

Step 03

- Write a description of changes made in the project in Release Notes section



RAMAIAH
Institute of Technology



