

RPA Design and Development

v4.0



Lesson 7 | Exception handling

Error Handling– Exam Topics

Use

1. Try Catch
2. Throw
3. Rethrow
4. Retry Scope

Errors are events that hamper the regular execution of the program. Based on their source, there are different types of errors:

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 (bugs)

Where the program contains no syntax errors but does not produce the expected results

System and Business Exceptions

Exceptions are a subset of errors that are recognized (caught) by the program, categorized, and handled. More specifically, there is a routine configured by the developer that is activated when an exception is caught. Sometimes, the handling mechanism can be simply stopping the execution.

Some of the exceptions are linked to the systems used, while others are linked to the logic of the business process. The two types of exceptions are:

Application (System) Exception

- Describes an error rooted in a technical issue, such as an application that is not responding
- Has a chance of being solved simply by retrying the transaction, as the application can unfreeze
- Can be managed by following good naming conventions for activities and workflows. This helps in tracking the activity that caused the exception

Business Exception

- Describes an error rooted in the fact that certain data which the automation project depends on is incomplete, missing, outside of set boundaries, or does not pass other data validation criteria
- An exception from the usual process flow and the validation is made explicitly by the developer inside the workflow
- The text in the exception should contain enough information for a human user (business user or developer) to understand what happened and what actions need to be taken

Application (System) Exception

The unexpected exceptions are the one known as System Exceptions (SE). They're also called Application Exceptions (AE), but the underlying concept remains the same. They belong to the .NET framework and are categorized under the **System.Exception** class. Proper handling should be in place for these kinds of exceptions as they can cause the process to fail.

The below list encompasses the most common exceptions that you can encounter in projects developed with Studio. They are derived from **System.Exception** as mentioned already, so using this generic type in a TryCatch, for example, will catch all types of errors.

Exception	Description
NullReferenceException	Occurs when using a variable with no set value (not initialized)
IndexOutOfRangeException	Occurs when the index of an object is out of the limits of the collection
ArgumentException	Is thrown when a method is invoked and at least one of the passed arguments doesn't meet the parameter specification of the called method
SelectorNotFoundException	Is thrown when the robot is unable to find the specified selector for an activity in the target app within the Timeout period. As you know, the timeout property specifies the amount of time, in seconds, that the system will wait for an operation to finish before generating an error
ImageOperationException	Occurs when an image isn't found within the Timeout period
TextNotFoundException	Occurs when the indicated text isn't found within the Timeout period
ApplicationException	Describes an error rooted in a technical issue, such as an application that isn't responding

Application (System) Exception

Catches Sequence if Excel is opened

Log Message - Exception Details

Log Level: Warn

Message: exception.Message + " at " + exception.Source

Rethrow the exception

Close excel files

This activity will kill all excel files that are opened.

Process *

A Process type object describing the process to be

ProcessName *

"excel.exe"

Exception

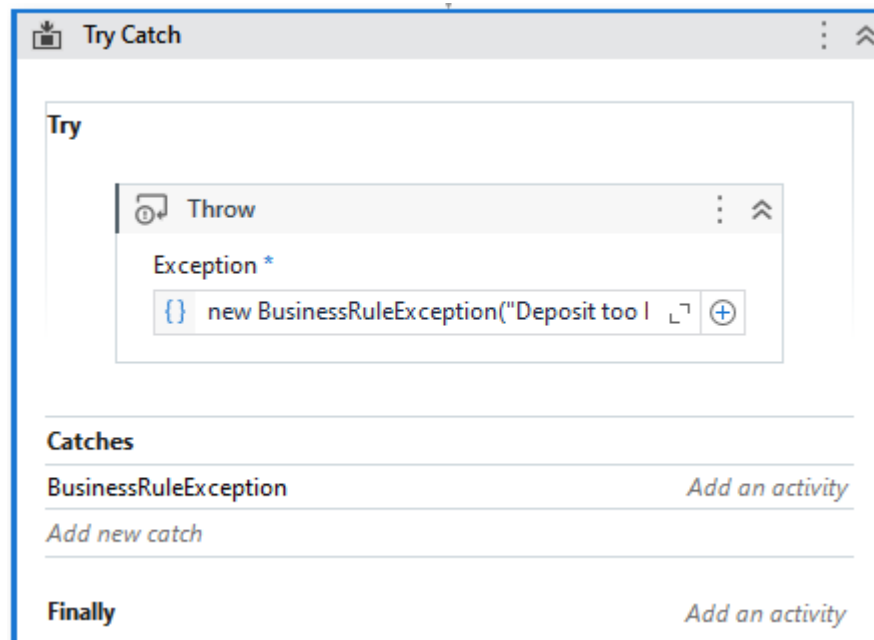
Finally

- System.ArgumentException
- System.NullReferenceException
- System.InvalidOperationException
- System.Exception
- Browse for Types ...

Business Rule Exceptions

Describes an error rooted in the fact that certain data which the automation project depends on is incomplete, missing, outside of set boundaries, or does not pass other data validation criteria

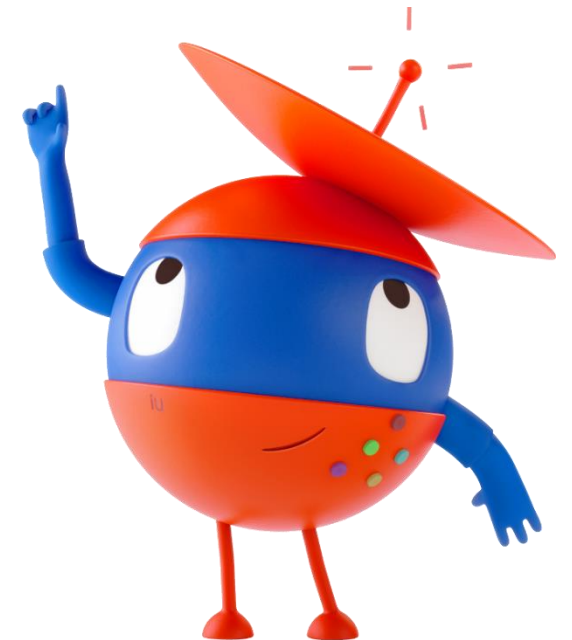
BREs aren't automatically generated as system exceptions.



They must be **defined by a developer by using the Throw Activity and handled inside a TryCatch.**

Business Exception Vs Application Exception

<https://docs.uipath.com/orchestrator/standalone/2023.4/user-guide/business-exception-vs-application-exception>



Error Handling Approach

It is common for automation projects to encounter events that interrupt or interfere with the projected execution. The different approaches to handle errors are:



Stop the
execution



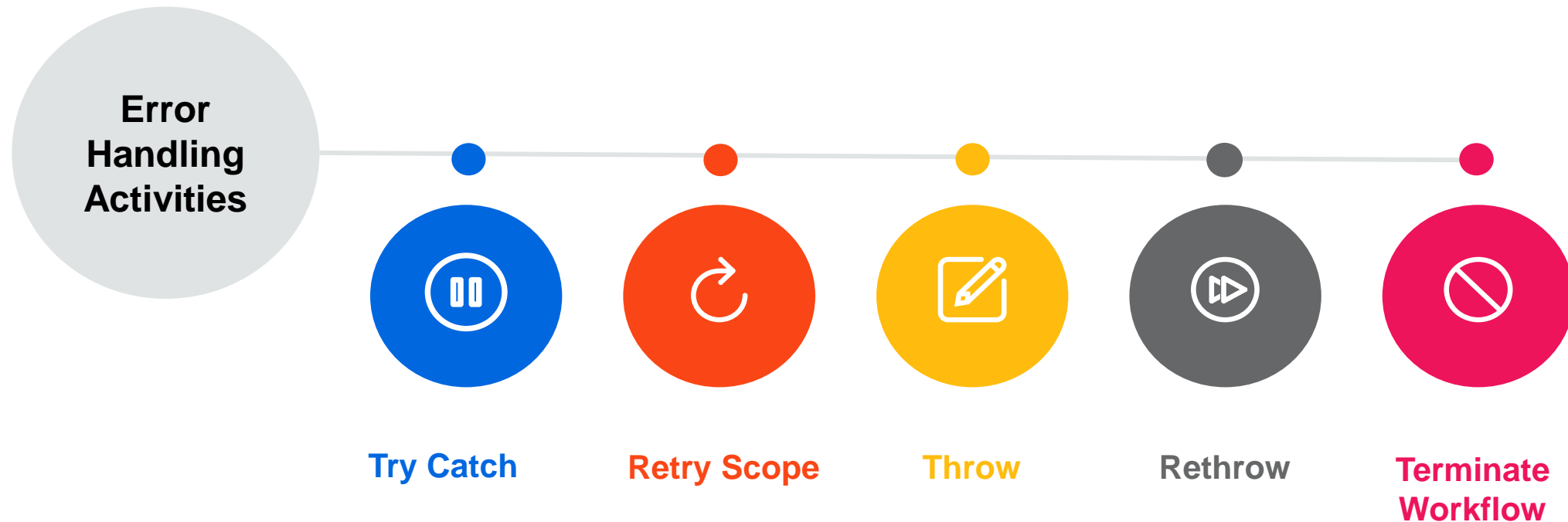
Execute specified
actions within the
workflow



Escalate the
issue to a
human operator

Error Handling Activities

Error handling is the mechanism for identifying and addressing the errors in a program.
Some of the important error handling activities are:



Try Catch Activity

Catches a specified exception type in a sequence or activity, and either displays an error notification or dismisses it and continues the execution.

Try

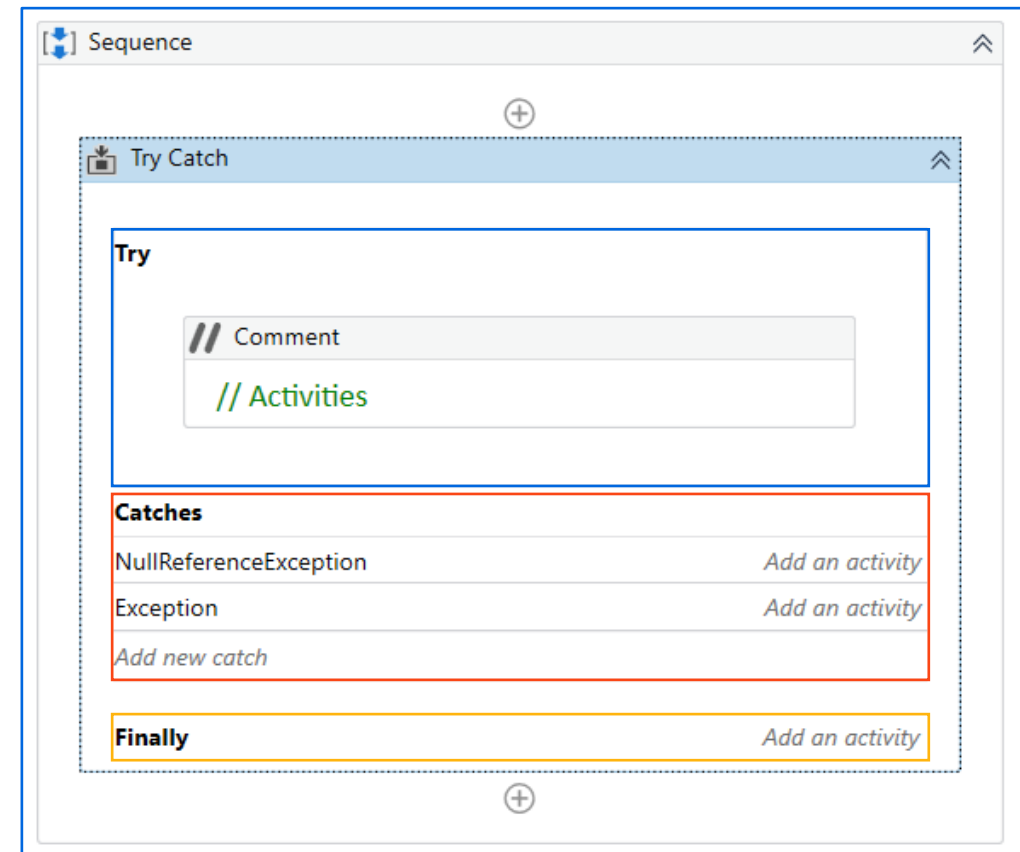
The activities performed which have a chance of throwing an error.

Catches

The activity or set of activities to be performed when an error occurs. Multiple errors and corresponding activities can be added in this block.

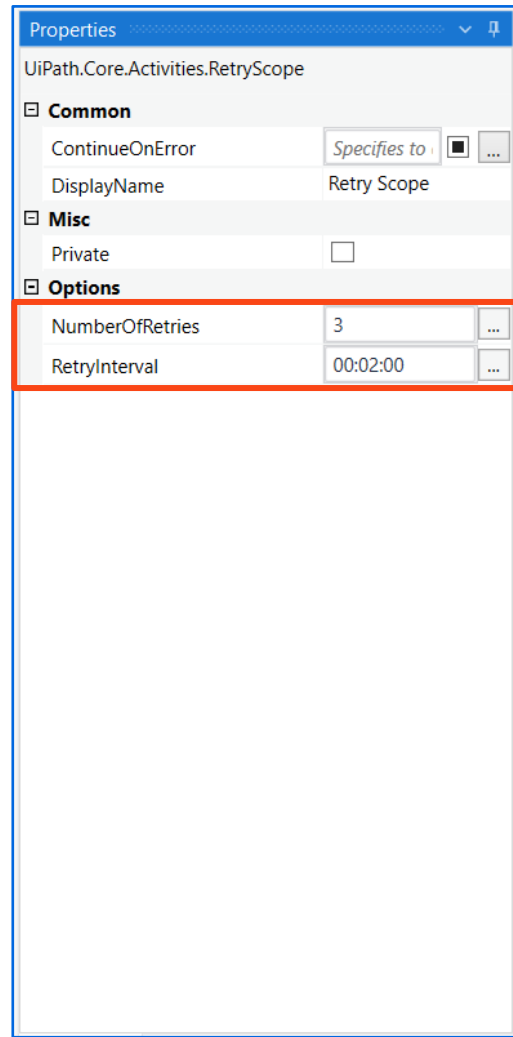
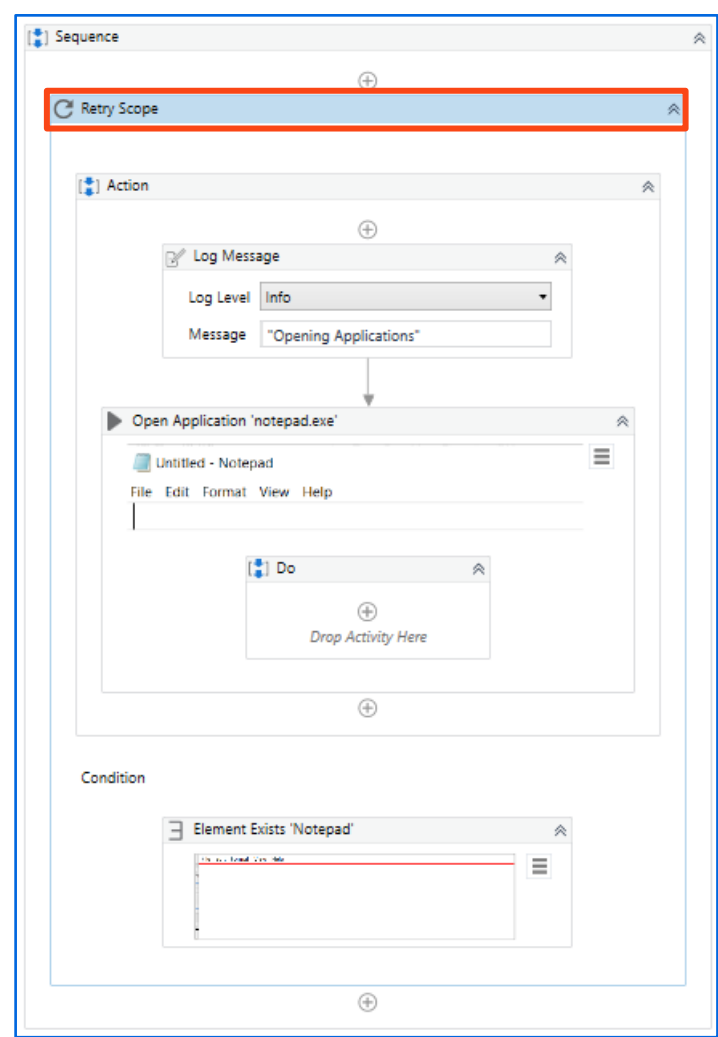
Finally

Holds an activity that should be executed only if no error occurred or if the error was already caught



Retry Scope Activity

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



The Properties window for the 'Retry Scope' activity is shown. The 'Options' section is highlighted with a red box. It contains two properties: 'NumberOfRetries' (set to 3) and 'RetryInterval' (set to 00:02:00). A red circle highlights these two properties, with a dotted line connecting it to a magnified view on the right.

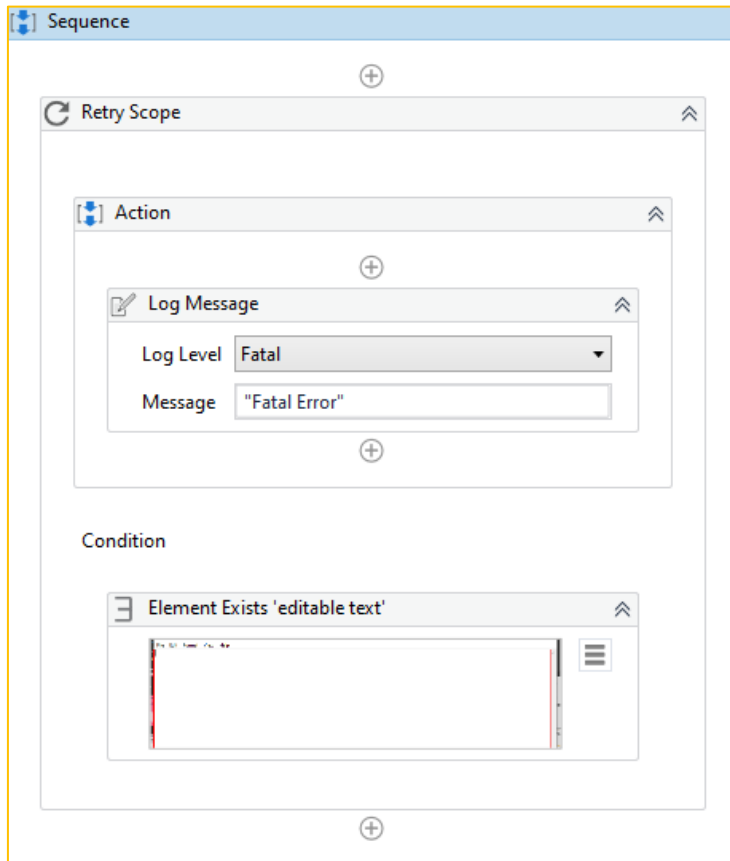
Property	Value
NumberOfRetries	3
RetryInterval	00:02:00



Retry Scope

Retry Scope

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



Retry Scope activity retries the contained activities as long as the condition is not met, or an error is thrown.

It is used to retry the execution in situations in which an error is expected. The execution will be retried until a certain event happens (for a number of times) or without any condition (retried until no exception is thrown).

It is used for catching and handling an error, which is why it's similar to Try Catch. The difference is that this activity simply retries the execution instead of providing a more complex handling mechanism.

Some of the properties of Retry Scope are:

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

For example, consider a website that simply works faulty, and the user just needs to click the same button over and over until it goes to the desired screen

To know more, visit: <https://docs.uipath.com/activities/docs/retry-scope>

The Retry Scope activity in Studio **allows for the repetition of activities until a condition is met or an error is encountered**. It is a helpful mechanism for handling errors and recovering from failures in an automation process.

This activity is a powerful tool in cases where exceptions are thrown sporadically and other measures, like tuning selectors, already took place.

For example, a particular selector isn't found in a certain application in less than 5% of the time the workflow runs, but no further selector improvements are possible. Using Retry Scope in this scenario will make the robot try to access the selector again in case a `SelectorNotFoundException` is thrown.

The Retry Scope will throw an exception if the retry number is exceeded.
Therefore, it should be place is a Try Catch, with a proper Log Message in the Catch section.

Chapter 7:

Error and Exception Handling in Studio

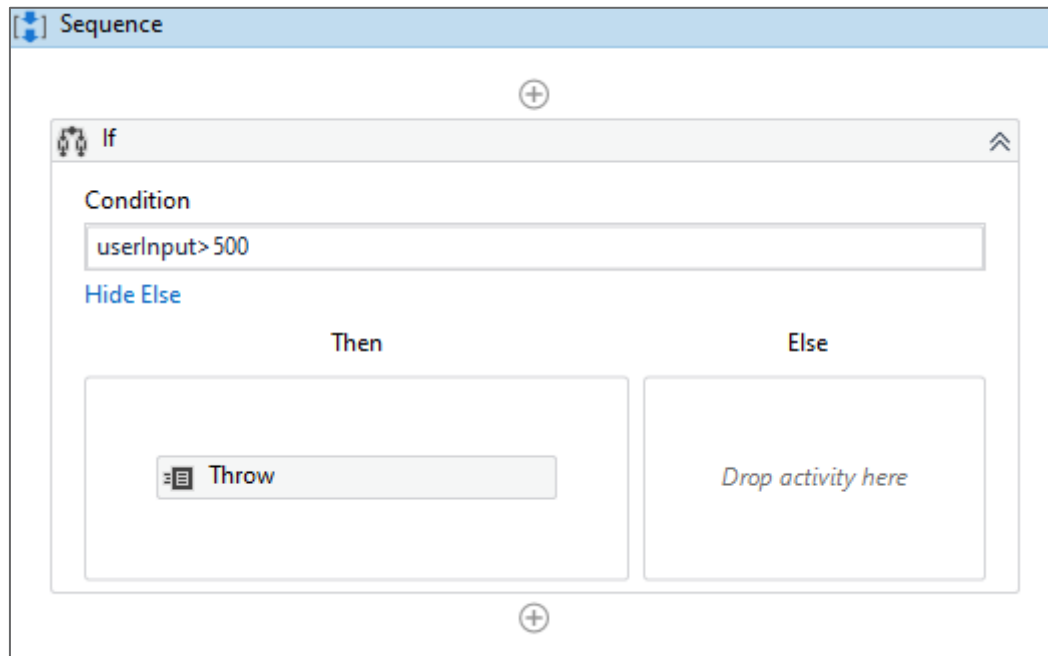
Retry Scope



Throw

Throw

Throws a user-defined exception



This activity throws a user-defined exception.
It can be a system or business exception

Example: `new Exception("Throwing an exception")`
OR `new BusinessException("Throwing a business exception")`

To know more, visit:
<https://docs.uipath.com/activities/docs/try-catch>

Rethrow

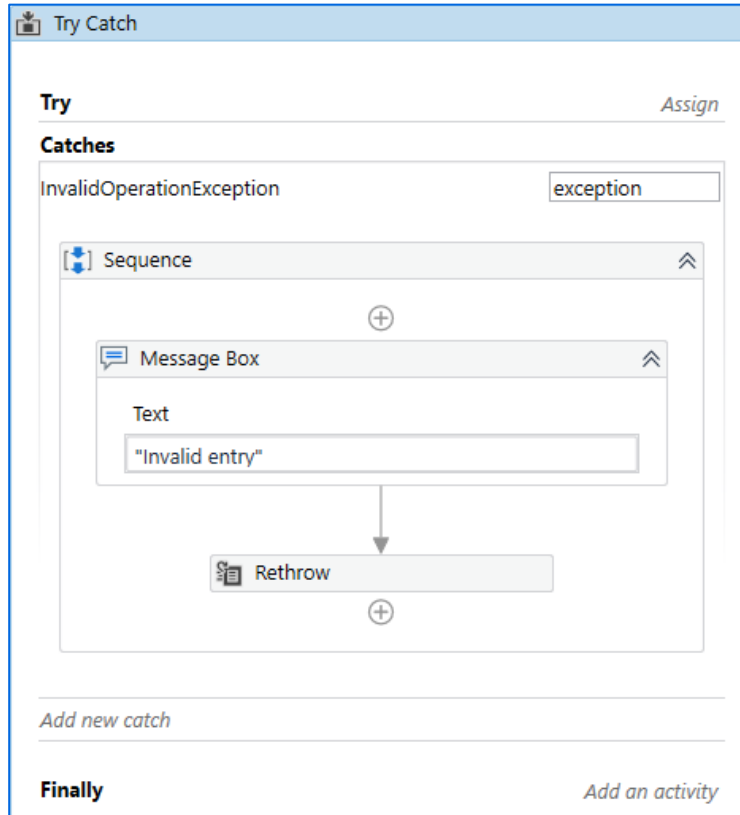
Takes an existing exception that has been encountered and regenerates it at a higher level

This activity takes an existing exception that has been encountered and regenerates it at a higher level.

It always comes in the catch block of Try Catch activity.

It is used when the user wants the activities to occur before the exception is thrown

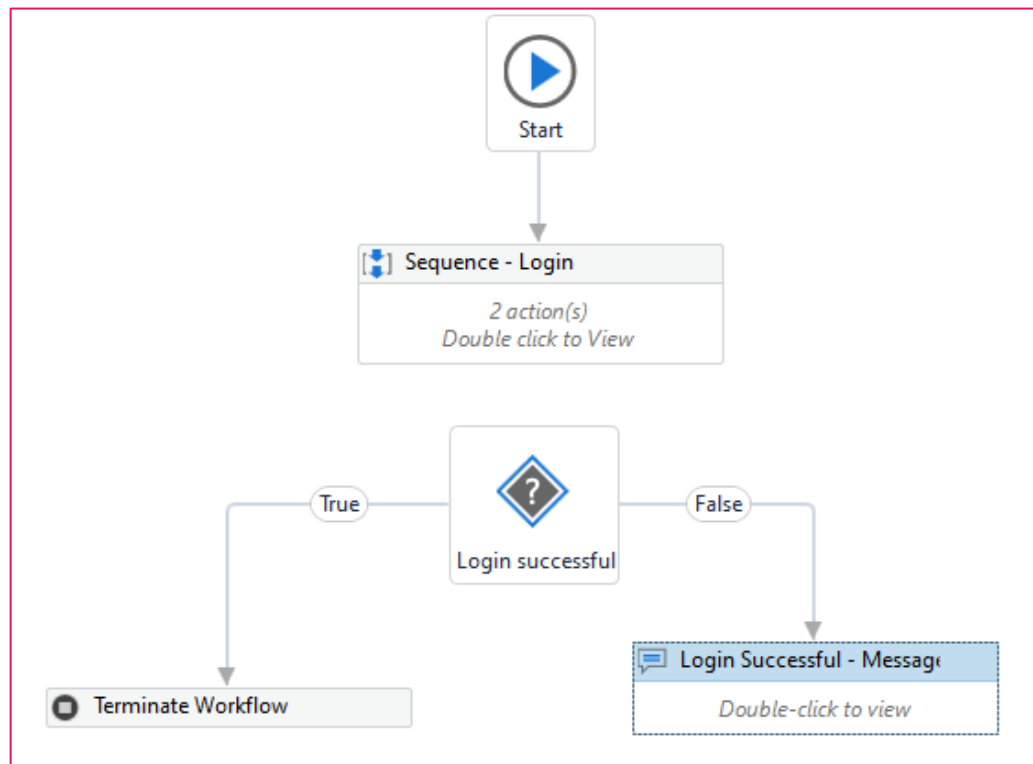
Example: “FileNotFoundException” error



Terminate Workflow

Terminate Workflow

Terminates the workflow when the task encounters an error



This activity terminates the workflow when the task encounters an error.

Example: If login is unsuccessful, the workflow is terminated

Chapter 7:

Error and Exception Handling in Studio

TryCatch, Throw and ReThrow



Topic	Link
Try Catch	https://docs.uipath.com/activities/other/latest/user-guide/try-catch
Error Handling in Project Organization	https://docs.uipath.com/studio/standalone/2023.4/user-guide/project-organization#error-handling
Logging Levels in UiPath	https://docs.uipath.com/orchestrator/standalone/2023.4/user-guide/logging-levels
Retry Scope	https://docs.uipath.com/activities/other/latest/user-guide/retry-scope
UI Activities Properties	https://docs.uipath.com/studio/standalone/2023.4/user-guide/ui-activities-properties

Classroom Exercise - Try Catch



Build a workflow using **Try Catch** that:

- Inserts a text into a Notepad file in the first attempt
- Encounters an error when attempting to write a second text
- Catches and displays the error in a message box
- Continues to save and close the Notepad file

Classroom Exercise - Retry Scope



Build a workflow using a **Retry Scope** that simulates a failing Notepad window.

Your goal is to create a workflow that handles different scenarios based on the value of a random variable.

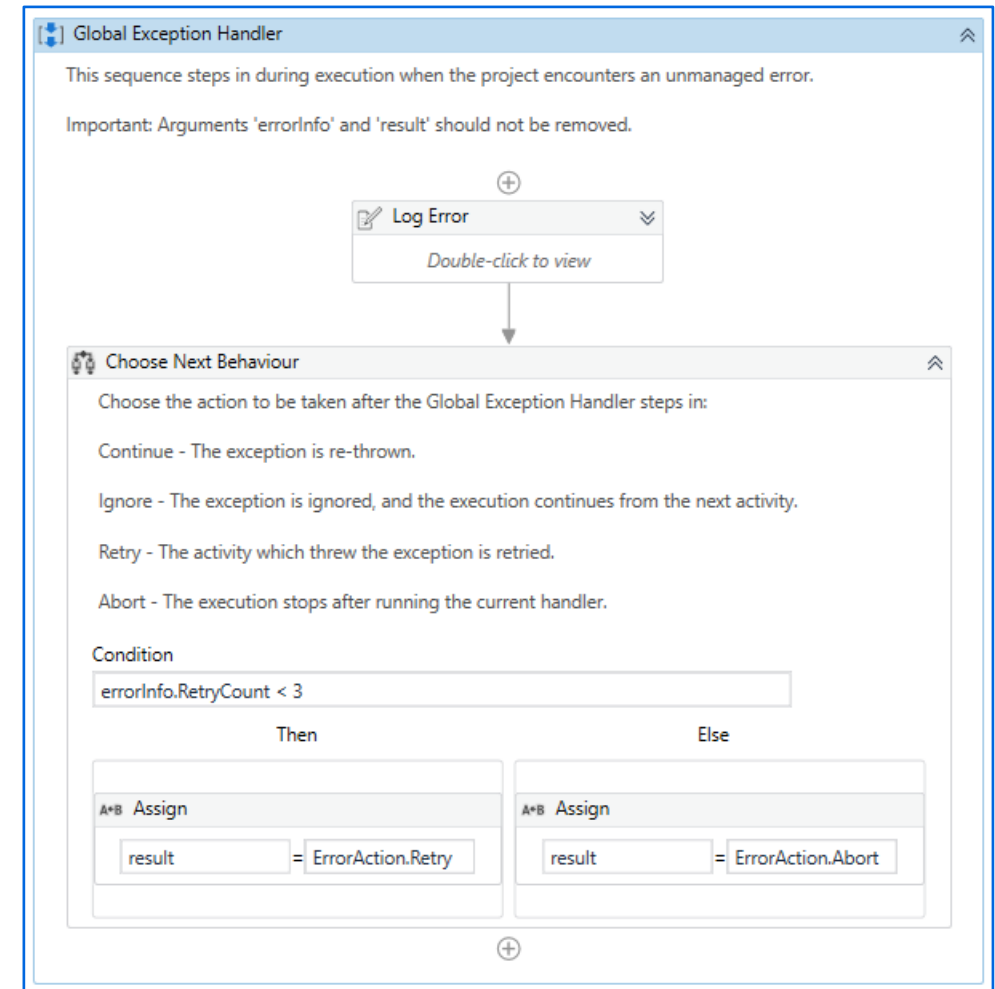
If the value of the Random variable is not 0 for three consecutive times, display the message "Notepad Window failed to start" and terminate the workflow with the error message "Notepad failed to start."

If the value of the Random variable is 0, open the Notepad application. Since the exist condition of the loop is to find the Notepad window, consider this scenario as a successful completion of the workflow.

Global Exception Handler

A type of workflow designed to determine the behavior **when encountering an execution error at the project level.**

- Only one Global Exception Handler can be set per automation project
- It is used in conjunction with Try Catch and only uncaught exceptions will reach the Exception Handler
- It is only available for processes and not for library projects



Global Exception Handler

The Global Exception Handler has a predefined structure.

Predefined Arguments (shouldn't be removed)

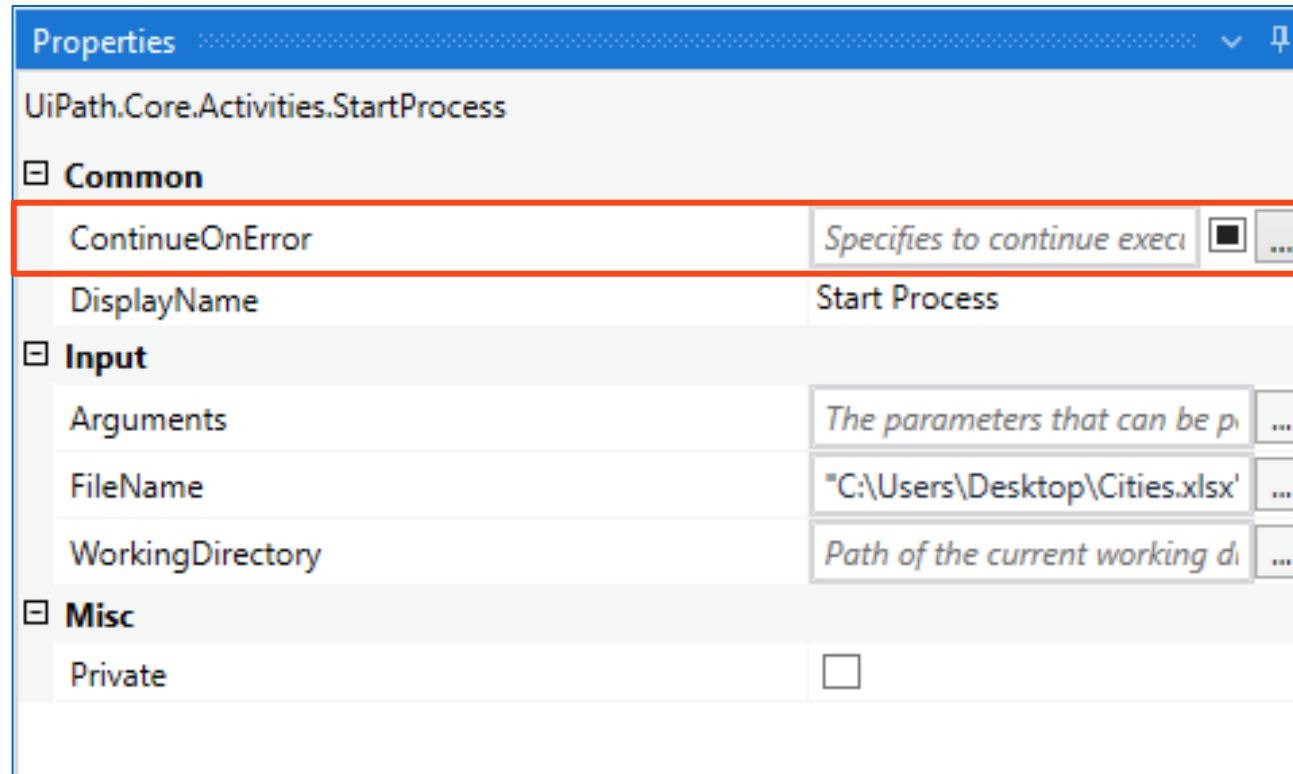
- errorInfo, with the In direction:
 - Contains the error that was thrown & the workflow that failed
- result, with the Out direction:
 - Stores the next behavior of the process when it encounters the error

Predefined Actions (can be removed)

- Log Error:
 - Logs the error
 - Choose the logging level: Fatal, Error, Warning, Info, and so on
- Choose Next Behavior:
 - Choose the action to be taken when an error is encountered during execution
 - Continue, Ignore, Retry, Abort

ContinueOnError

Specifies if the automation should continue, even if the activity throws an error.

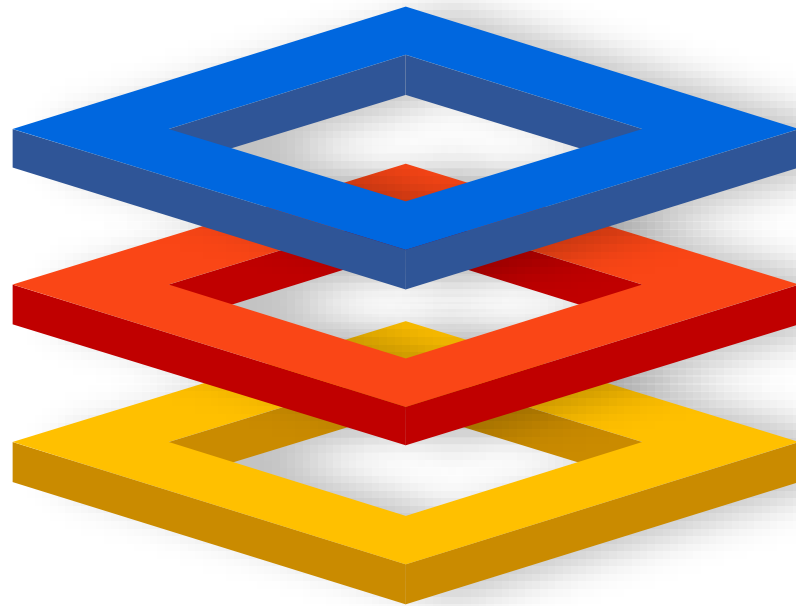


Properties	
UiPath.Core.Activities.StartProcess	
Common	
ContinueOnError	Specifies to continue execution <input type="checkbox"/>
DisplayName	Start Process
Input	
Arguments	The parameters that can be passed <input data-bbox="1753 778 1793 801" type="button" value="..."/>
FileName	"C:\Users\Desktop\Cities.xlsx" <input data-bbox="1753 849 1793 872" type="button" value="..."/>
WorkingDirectory	Path of the current working directory <input data-bbox="1753 921 1793 943" type="button" value="..."/>
Misc	
Private	<input type="checkbox"/>

NOTE: If an activity is included in Try Catch, in the Try section, and the value of the ContinueOnError property is True, no error is caught when the project is executed.

Best Practices for Error Handling

The best practices for handling errors are:



Breaking the process into smaller workflows



Using Try Catch blocks



Using Global Exception Handler

Best Practices: TryCatch Donts

While the Try Catch activity can be helpful, it's important to **avoid excessive usage**.

Catching an exception should only be done when there's a valid reason to do so.

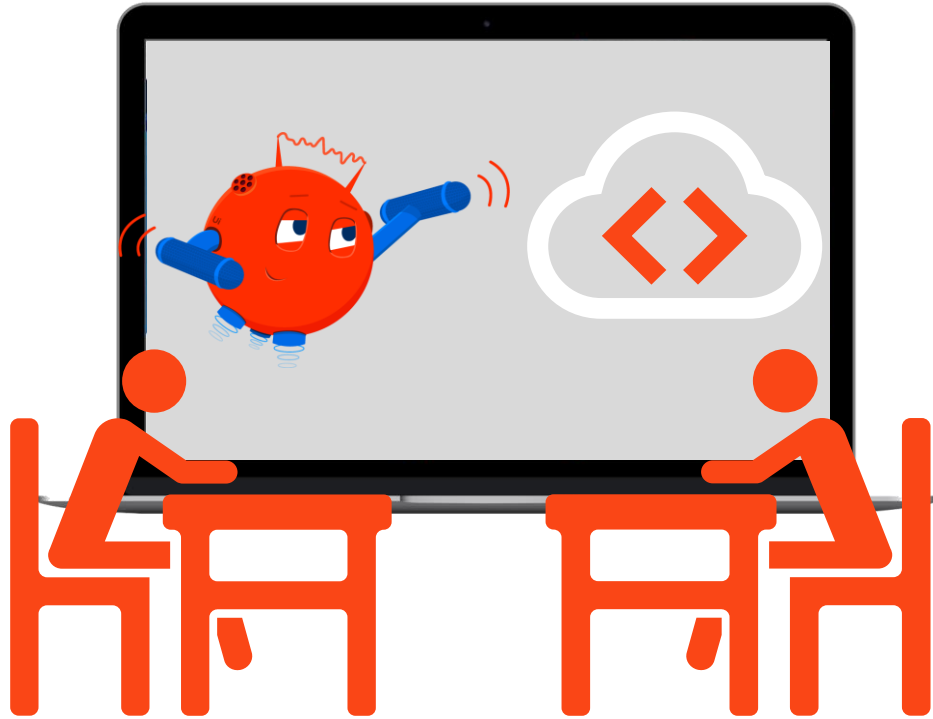
Typically, the Catch block is responsible for handling the exception and enabling error recovery.

However, there are instances where an exception is caught performing certain actions, for the purpose of logging it, before being rethrown to higher levels.

Here are a few examples where the Try Catch activity shouldn't be used:

- When dealing with an Assign activity that involves filtering a DataTable. Instead of using it in a Try block, it's advisable to use an If statement. This Assign can throw an exception if the input DataTable (in_Datatable) is empty or if the Select operation doesn't return any results. By using an If statement, we can first check the necessary requirements before executing the Assign activity.
- In case of UI interaction, avoid placing Click or Type Into activities in Try Catch. Instead, use synchronization activities to check the availability of the target elements, such as: Element Exists, Find Element, Check App State etc.

Practice Exercise



Build a workflow using a **Try Catch** activity that does the following:

- Take the Name, Gender, and Age as the user input
- Subtract current year with Age value to get the Year of Birth
- Handle an error that occurs due to a reckless user input of an incorrect age containing the 11-digit number
- Continue the process to display the Name, Gender, and Year of Birth of the user in a message box