Error and Exception Handling in Studio

Errors

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 types of errors are often called bugs.

Exceptions

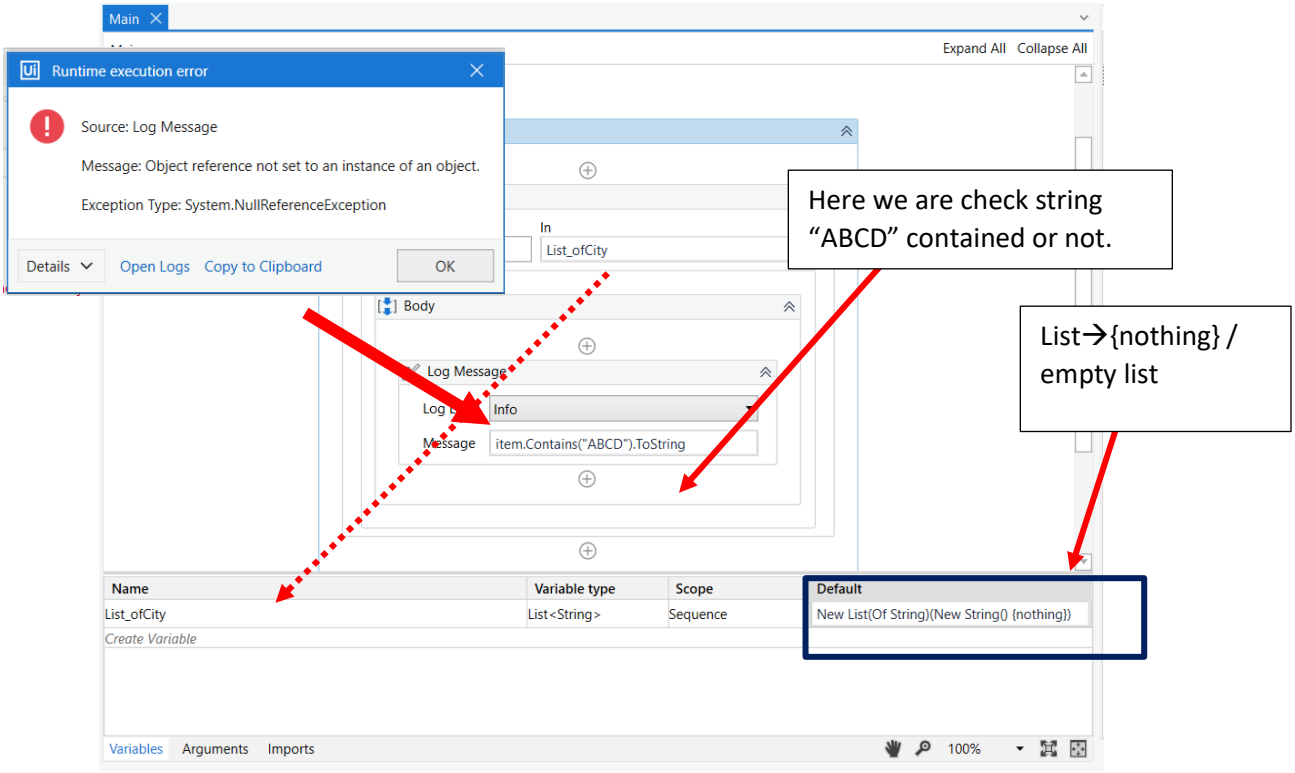
Exceptions are events that are recognized (caught) by the program, categorized, and handled.
More specifically, there is a routine configured by the developer that is activated when an exception is caught.
Sometimes, the handling mechanism can be simply stopping the execution.
Some of the exceptions are linked to the systems used, while others are linked to the logic of the business process.

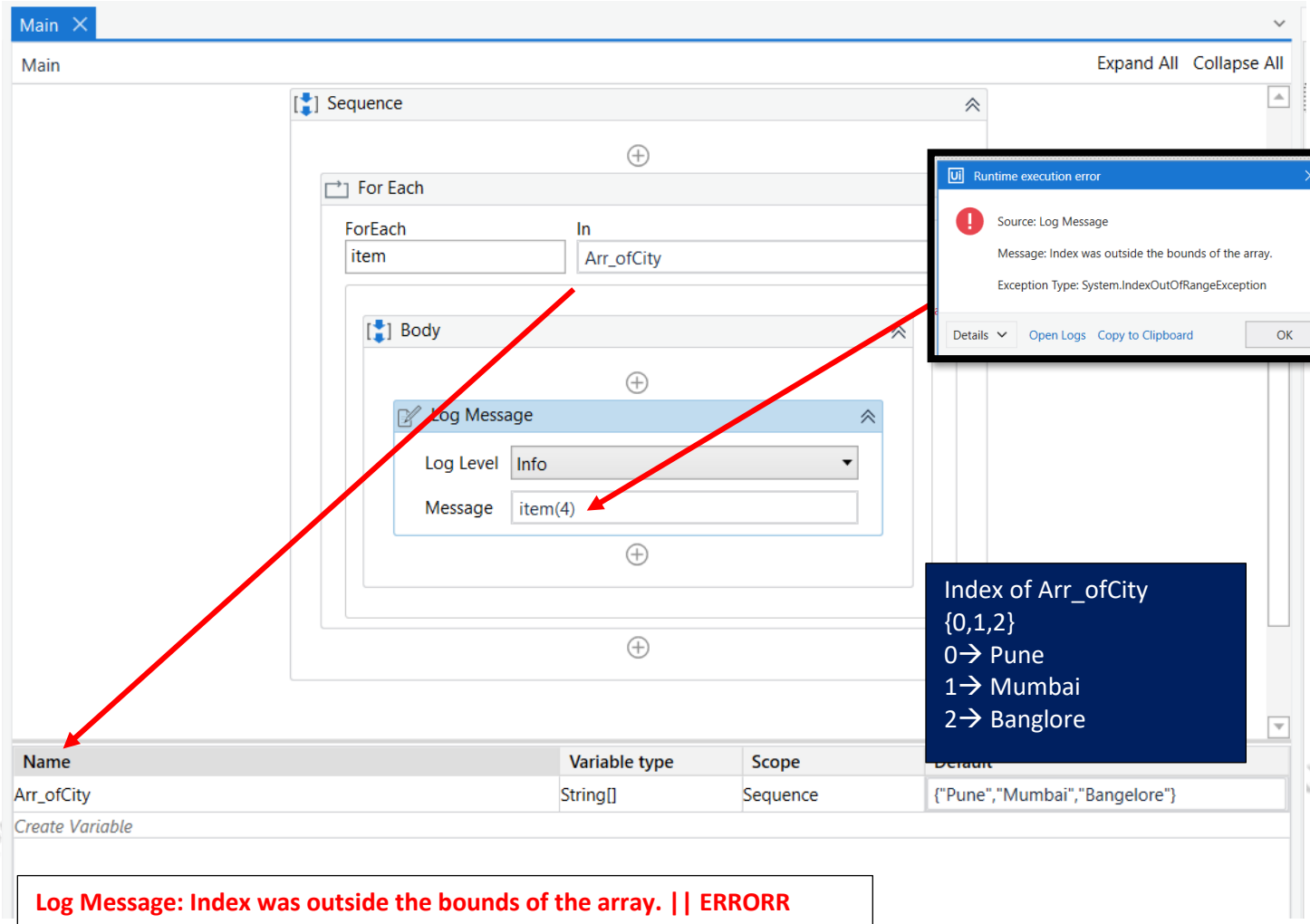
System exceptions

Below is the list of the most common exceptions that you can encounter in projects developed with UiPath.
As a general note, ***all of these exception types mentioned below are derived from System.Exception,*** so using this generic type in a TryCatch, for example, will catch all types of errors.

- **NullReferenceException** - *Occurs when using a variable with no set value (not initialized).*

The NullReferenceException is an exception that will be thrown while accessing a null object.

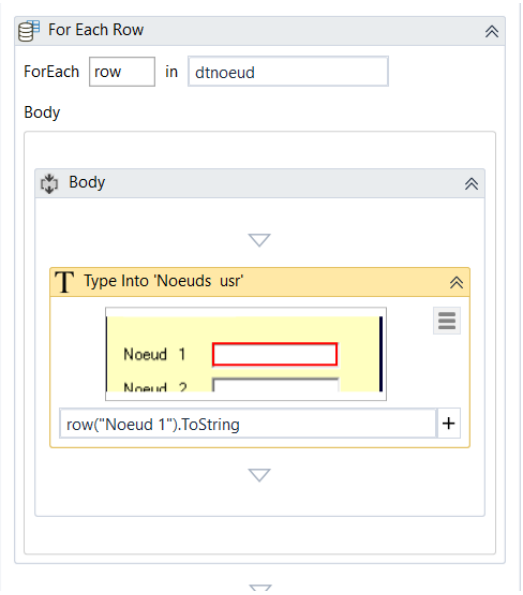


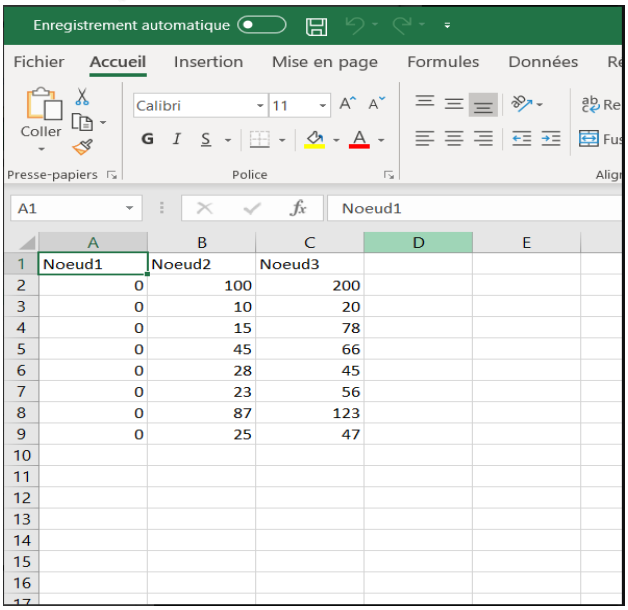


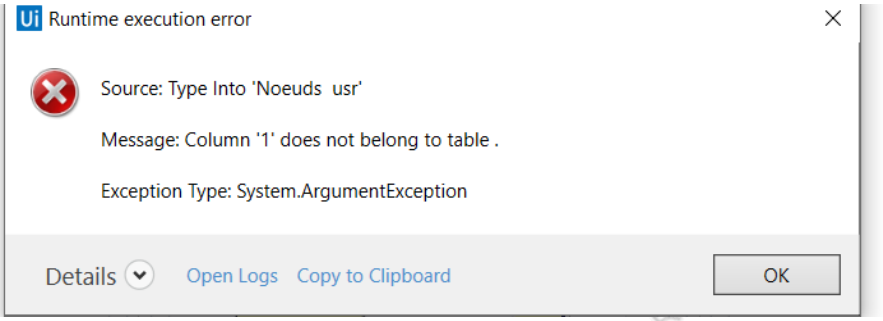
Log Message: Index was outside the bounds of the array. || ERRORR

- **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 does not meet the parameter specification of the called method.







SelectorNotFoundException - Is thrown when the robot is unable to find the designated selector for an activity in the target app within the TimeOut period.

ImageOperationException - Occurs when an image is not found within the TimeOut period.

TextNotFoundException - Occurs when the indicated text is not found within the TimeOut period.

ApplicationException - Describes an error rooted in a technical issue, such as an application that is not responding.

****

Business exceptions

A business exception mainly refers to information used in an automated process. Either *it may be incomplete or incorrect from a business perspective.*

When **business exceptions occur**, **robots stop the process,** and it **requires human intervention to address a thrown exception**.

Some of the examples of business exceptions are certain pieces of data that the automation project depends on are incomplete, missing, outside set boundaries (like trying to extract more from the ATM than its daily limit), or don't pass other data validation criteria.

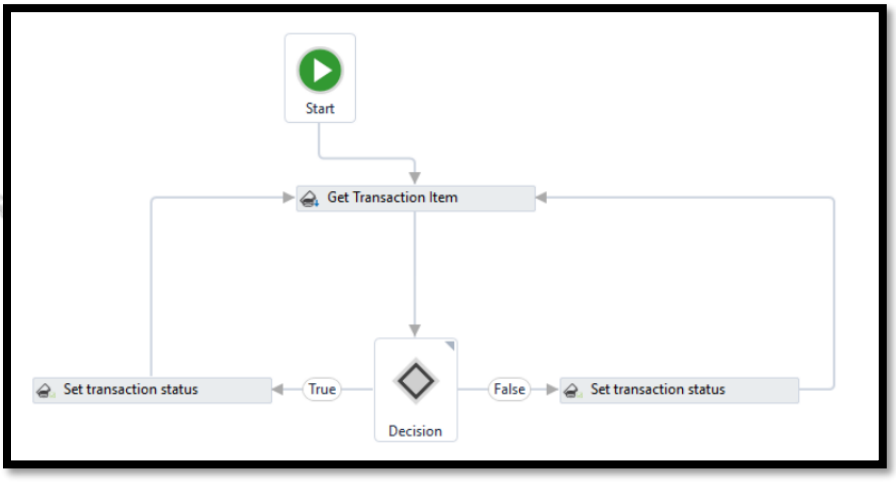
Business rule exceptions aren't automatically generated as system exceptions.

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

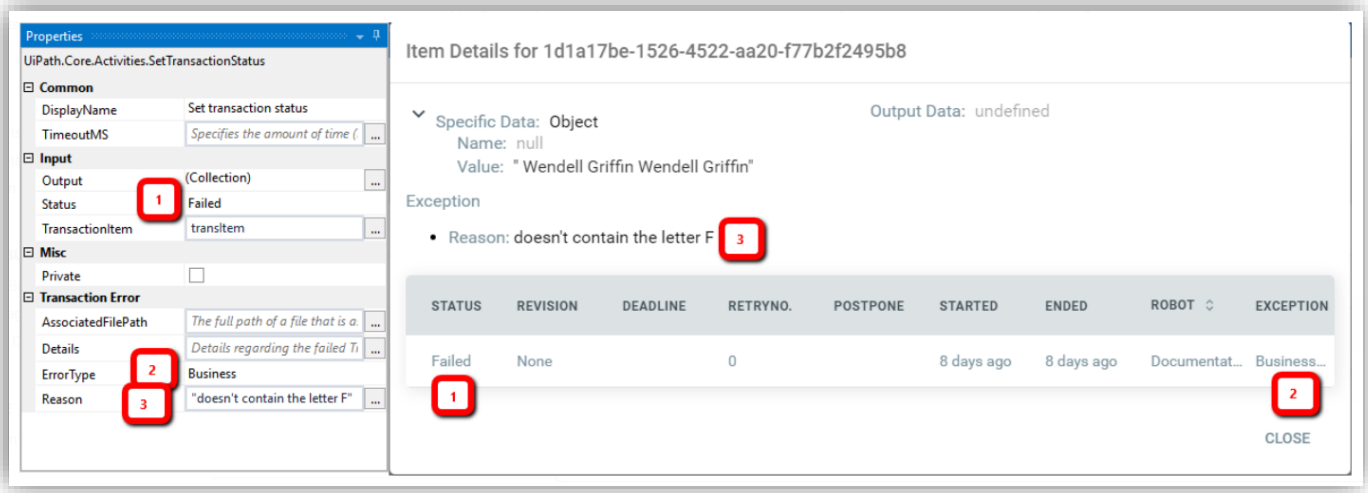
The Set Transaction Status activity can be used to shape the logic of your project in a way that encapsulates this distinction in several ways:

- If the Set Transaction Status activity fails the transaction with an Application Exception and the Auto Retry option in the Create Queue page is set to Yes when the queue is created, the queue item is retried.
- By default, Orchestrator does not retry transactions which are failed due to Business Exceptions.
- This happens because an inconsistency between the transaction value and the business requirement means that there might be errors in the initial data which the queue items were created from. Additional actions might be required to fix this type of issue, and logging this type of exception can be useful.
- If or Flow Decision activity can be used to take different courses of action if a transaction is failed with a certain type of exception, such as using the Log Message activity to log a custom message or the Message Box activity to display a window containing information about the event.

Below you can see an example of such a project:



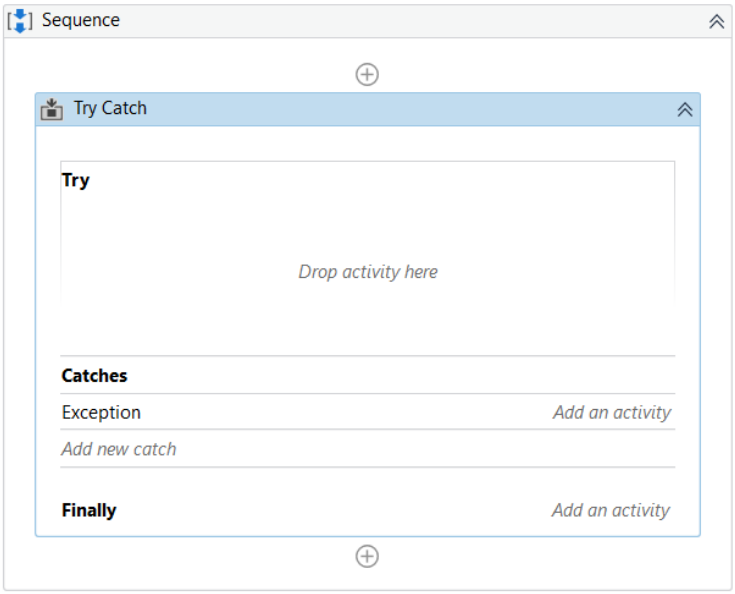
The screenshot below shows the mapping of the properties in the Set Transaction Status activity (on the left) and their corresponding fields in the Transaction Details window in Orchestrator.



- The True branch of the Flow Decision activity sets the transaction status to Failed with a Business Exception, while the False branch sets it to Failed with an Application Exception.

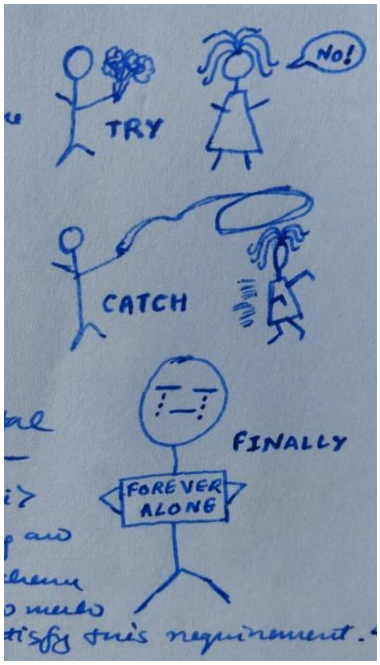
TryCatch, Throw, and Rethrow

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



- TryCatch runs the activities in the Try block and, if an error takes place, executes the activities in the Catches block.

- The Finally block is only executed when no exceptions are thrown or when an exception is caught and handled in the Catches block (without being re-thrown).



Retry Scope

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

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

The activity has two main sections Actions and Conditions.

It can be used without a termination condition; in which case it'll retry the activities until no exception occurs or the provided number of attempts is exceeded.

Retry scope has two additional properties NumberOfRetries and RetryInterval.
For example, *consider the case of clicking a button before it's loaded and checking if the resulting pop-up comes up. Or another case of downloading a file and trying to read it before it finishes downloading. In all these cases, the retry scope is going to run the action and check the condition like a do-while loop.*
----------*----------*----------
First is the Action Section.
This section is just a sequence where we can put in as many activities as we want. These are the activities that will be carried out at least once.

What happens to the activities within the action section?
The activities are retried if there is an error. Or they're retried if the condition is set, and it isn't true. Basically, we can have one to an infinite number of activities in the Action section.
The next section is the Condition

It can only have zero or one activity. It cannot have more activities. This activity output needs to be a Boolean value, like true or false. So in the end, after running all of the actions, if the condition is checked and is False, then, there will be another retry.
If the condition checked is True, then there won't be a Retry., it'll move to the next activity in sequence.
--------------------*----------*----------*----------*----------*----------*----------*

what kind of activities can be used inside a Condition?
The basic or common activities used in this are, Element Exists, File exists, Check True, and Check false. But any activity that turns a boolean is okay.
Element Exists activity. The condition activity returns a True or a False

The Retry Scope activity has two key properties: NumberOfRetries and RetryInterval.
First is the NumberOfRetries, it is the number of retries that the activities inside the action section could be retried.

It specifies the maximum number of times the contained activities should be executed if an exception is continually thrown, or the condition isn't met.

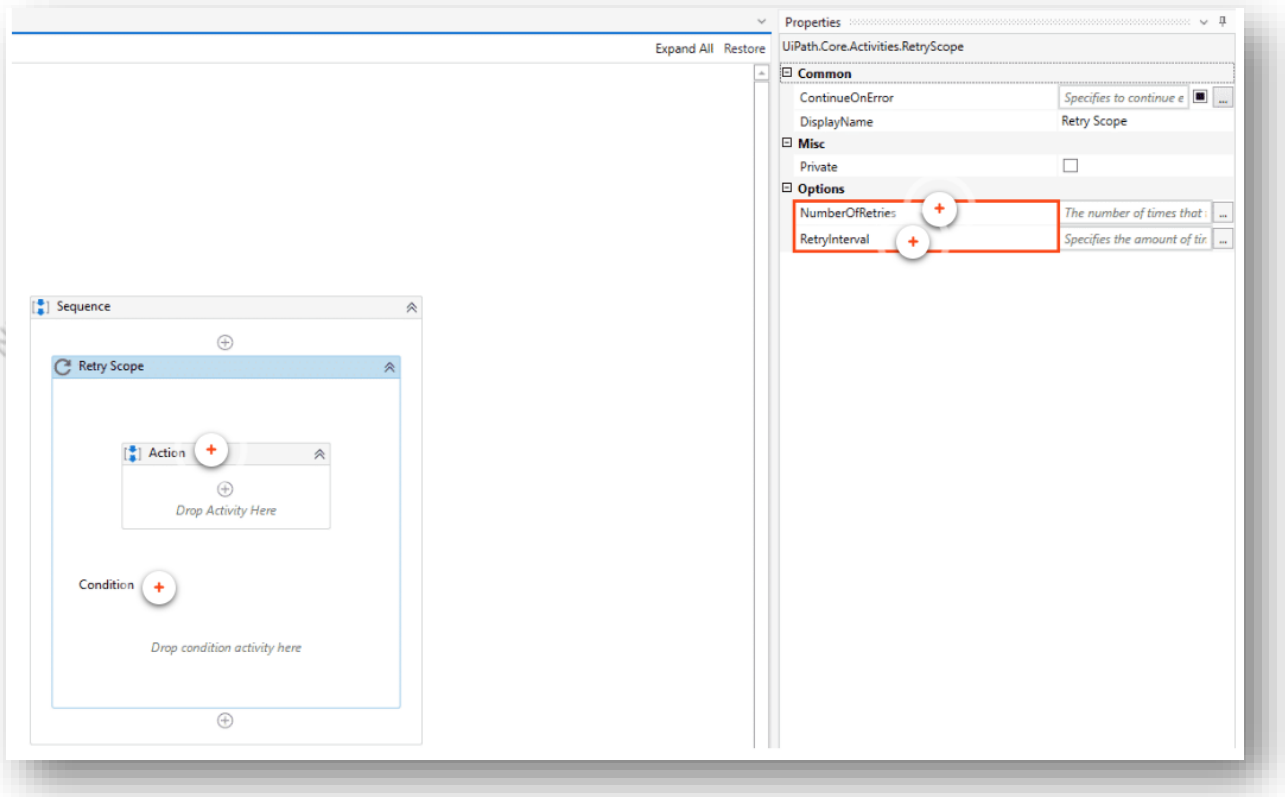
For example, if you put the number of retries equal to two, the total number of times, or the maximum number of times the action sequence can run is two.
We can enter five times. If you don't set any value in the number of retries field, then the default is three.
Then we have the Retry interval, it's duration.
The RetryInterval specifies the interval of time between each retry attempt.
These are written in an hour, colon minute, and colon second format. We can set any value, the default is five seconds.

Note:

If this activity is included in Try Catch **and the value of the** ContinueOnError **property is True, no error is caught when the project is executed**

The main takeaways:

- The retry scope is an activity with features in between a try-catch and a do-while loop.
- It performs error handling within the action section. If there's an error, it'll just stop wherever the error comes. It'll wait for the duration that's set. Otherwise, it'll wait five seconds and then retry and consume that reading.
- NumberOfRetries is just the maximum number an action sequence can run. If the number of retries is set to two, it'll only retry once, so that the action executes a total of two times.
- RetryInterval specifies the amount of time between each retry.
- The condition can have zero or one activity, if the condition checked is False, there'll be another Retry.
- If the condition checked is True, there won't be another retry.



The ContinueOnError Property

The ContinueOnError is a property that specifies if the execution of the activity should continue even when the activity throws an error.

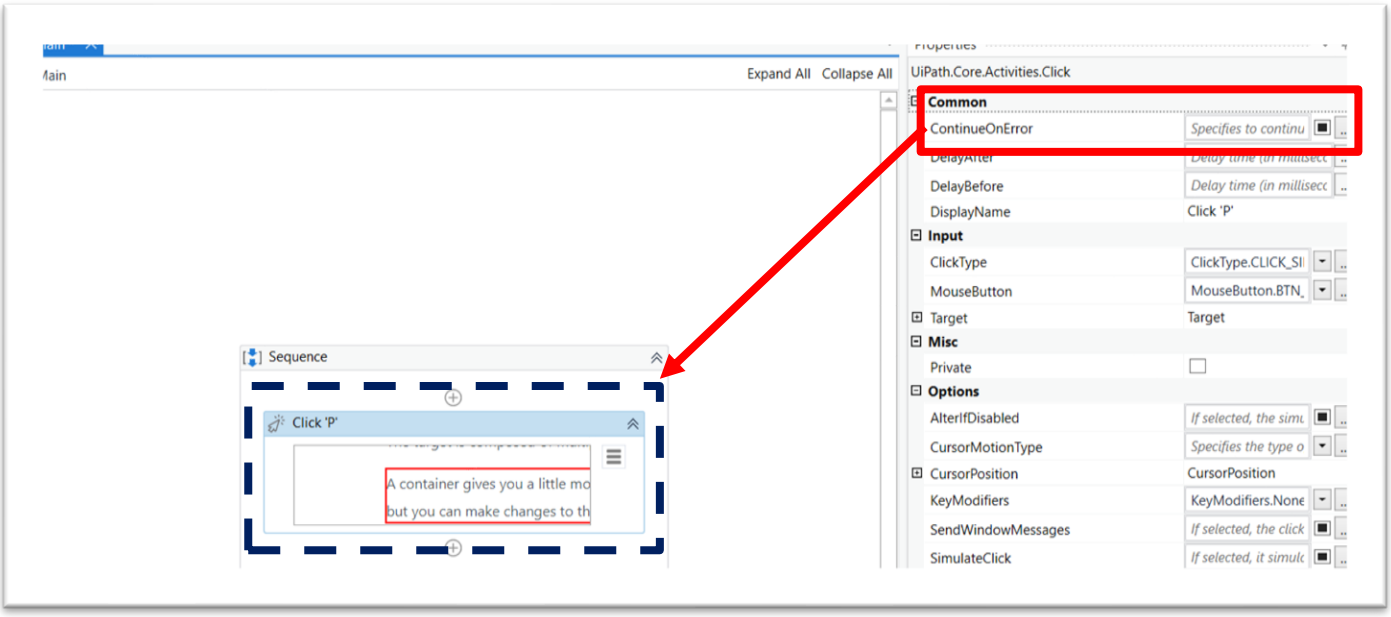
This field only supports Boolean values (True, False). The default value is False, thus, if the field is blank and an error is thrown, the execution of the project stops. If the value is set to True, the execution of the project continues regardless of any error.

*Keep in mind that, if the ContinueOnError is set to True on an Activity that has a scope such as (**Attach Window** or **Attach Browser**, in the classic, and **Use Application Browser** in the modern), then error is thrown by all the activities inside the DO container of respective scope Activity will also be ignored.*

Setting this property to True is not recommended in all situations. But there are some in which it makes sense, such as:

- While using data scraping - So that the activity doesn't throw an error on the last page when the selector of the 'Next' button is no longer found.
- When we are not interested in capturing the error but simply in the execution of the activity.

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.



The Global Exception Handler

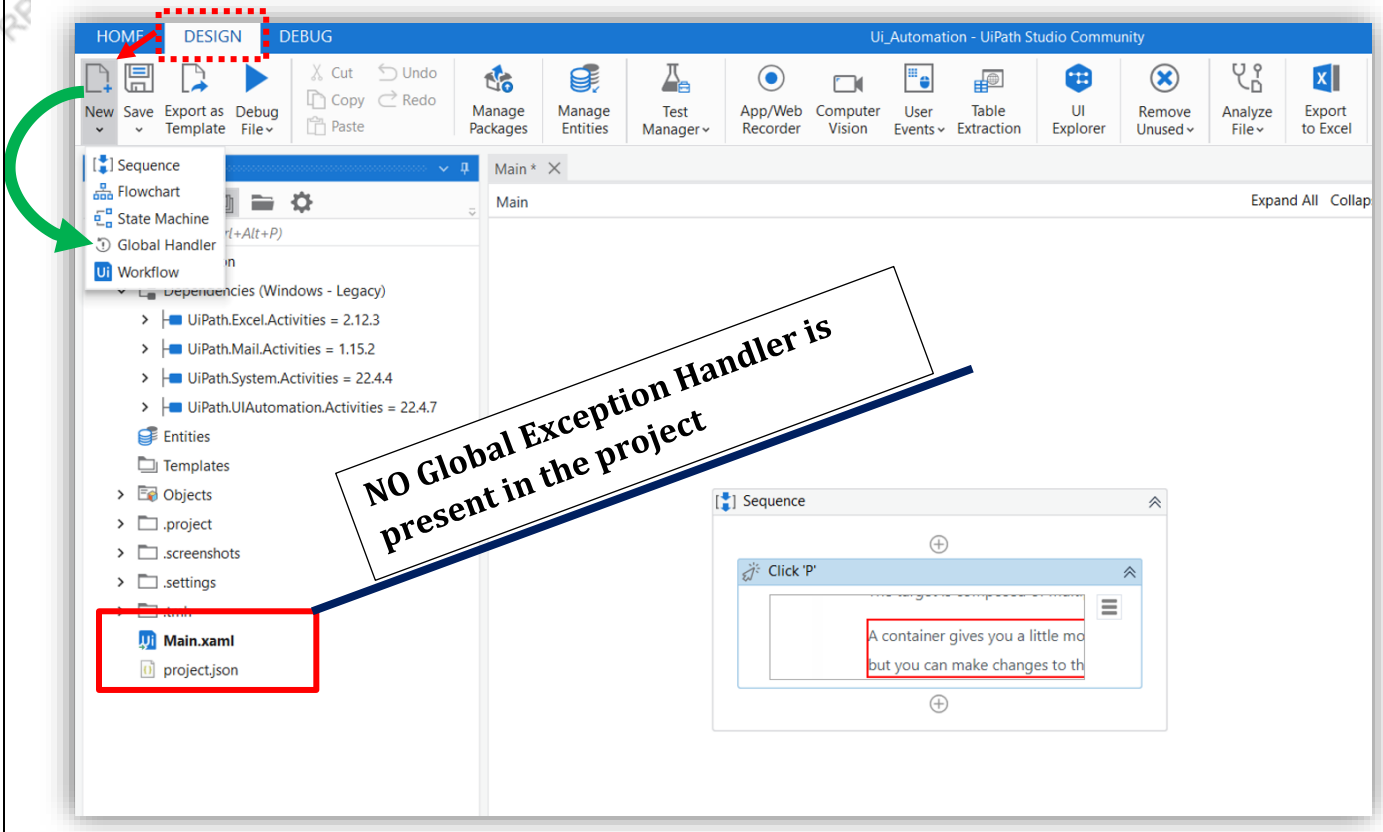
Use the Global Exception Handler in both attended and unattended scenarios.

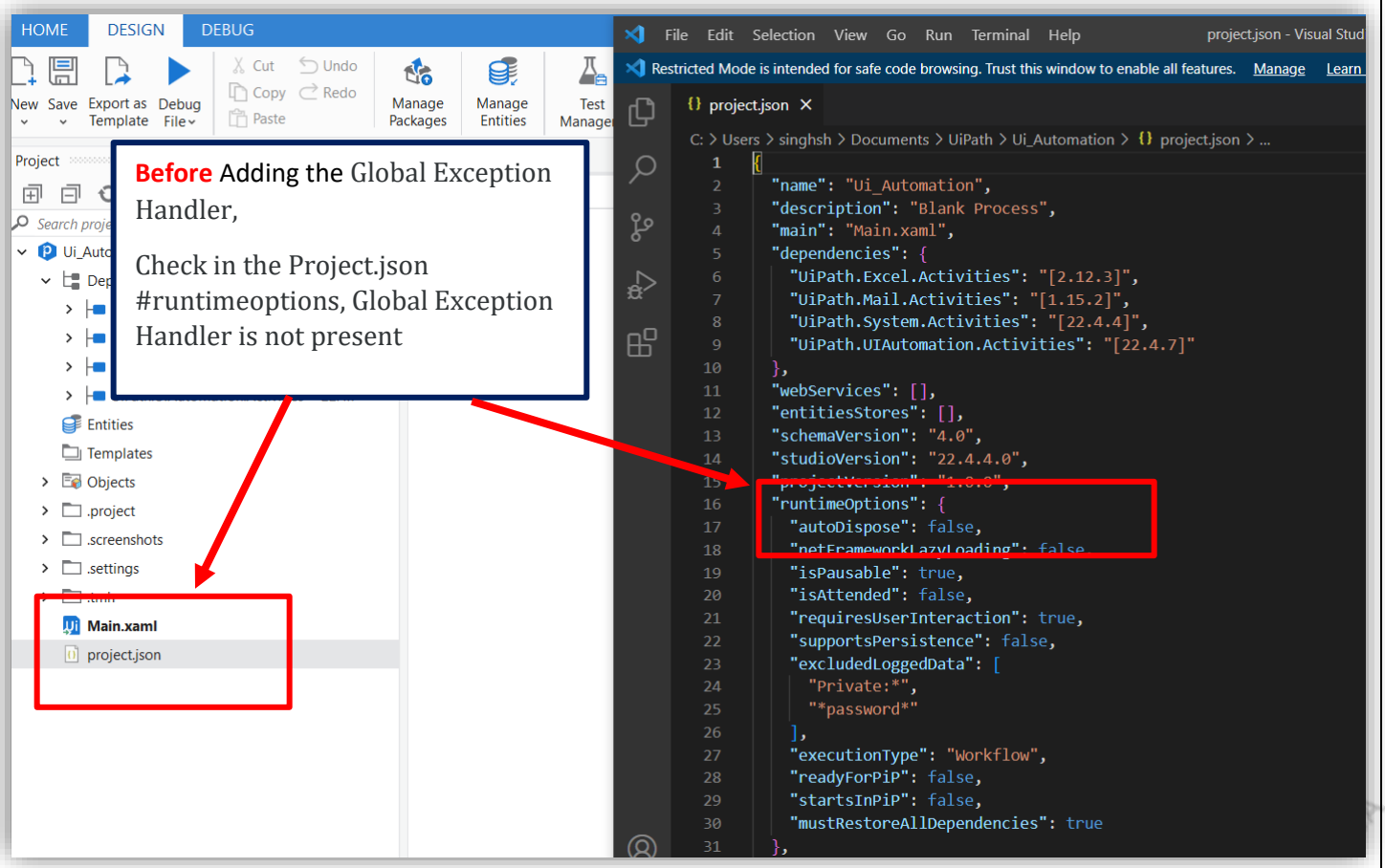
The Global Exception Handler is a type of workflow designed to determine the process' behavior when encountering an unexpected exception. This is why only **one** Global Exception Handler can be set per automation project.

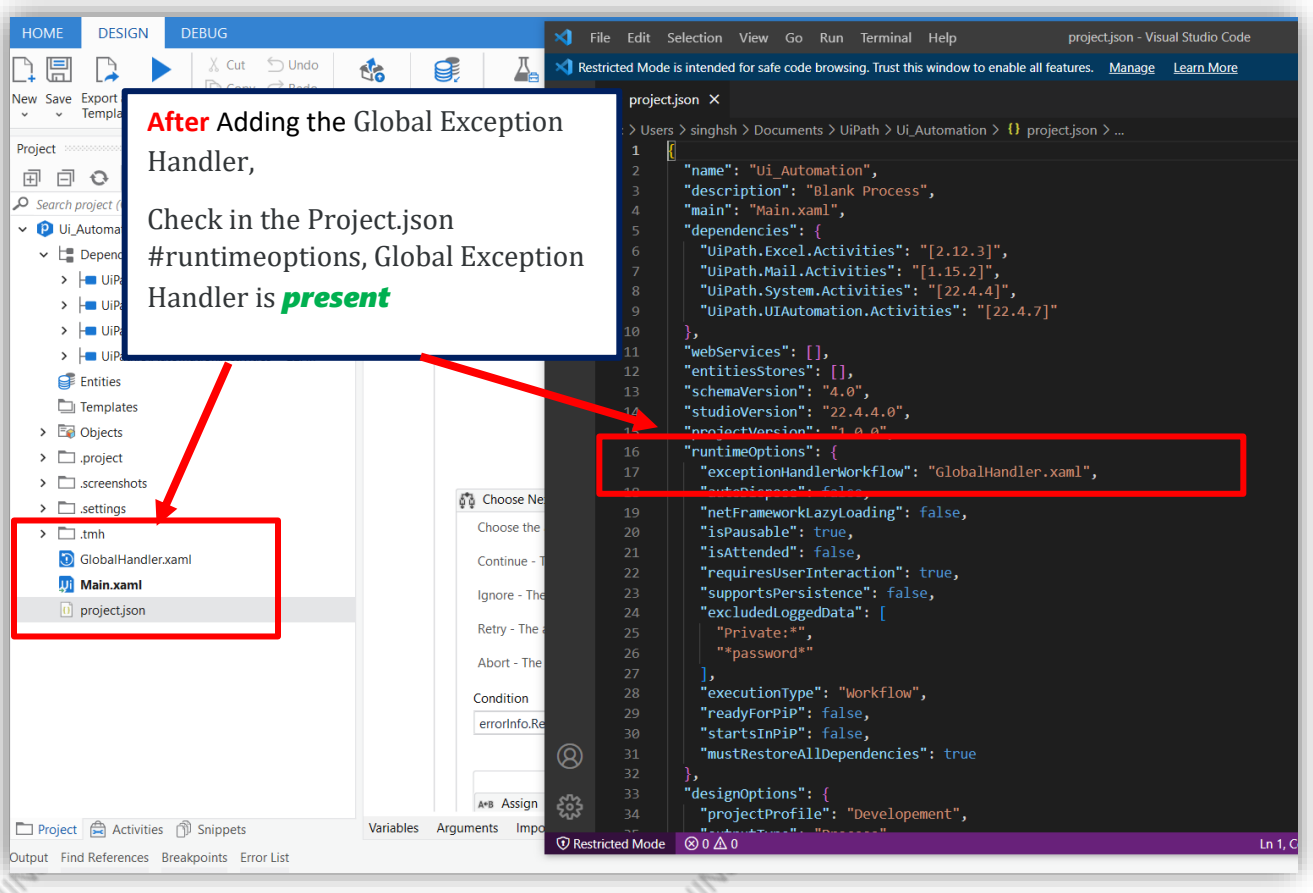
In its default configuration, the Global Handler catches the exceptions thrown by any activity in the process at runtime and executes a standard response - ignore, retry, abort or continue, as predefined at design time. For attended scenarios, it can be configured to let the user select the action.

A Global Exception Handler can be created by adding a new workflow file with this type. It can also be created by setting an existing workflow as Global Exception Handler from the Project panel.

How to get **Global Exception Handler in the project:**




HOME
DESIGN
DEBUG
New
Save
Export as Template
Debug File
Cut
Copy
Paste
Undo
Redo
Manage Packages
Manage Entities
Test Manage
Project
Before Adding the Global Exception Handler,
Check in the Project.json #runtimeoptions, Global Exception Handler is not present
Entities
Templates
Objects
.project
.screenshots
.settings
Main.xaml
project.json
File Edit Selection View Go Run Terminal Help
project.json - Visual Stud
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn
project.json
C: > Users > singhsh > Documents > UiPath > Ui_Automation > project.json > ...
{
"name": "Ui_Automation",
"description": "Blank Process",
"main": "Main.xaml",
"dependencies": {
"UiPath.Excel.Activities": "[2.12.3]",
"UiPath.Mail.Activities": "[1.15.2]",
"UiPath.System.Activities": "[22.4.4]",
"UiPath.UIAutomation.Activities": "[22.4.7]"
},
"webServices": [],
"entitiesStores": [],
"schemaVersion": "4.0",
"studioVersion": "22.4.4.0",
"runtimeOptions": {
"autoDispose": false,
"isPausable": true,
"isAttended": false,
"requiresUserInteraction": true,
"supportsPersistence": false,
"excludedLoggedData": [
"Private:*",
"*password*"
],
"executionType": "Workflow",
"readyForPiP": false,
"startsInPiP": false,
"mustRestoreAllDependencies": true
},

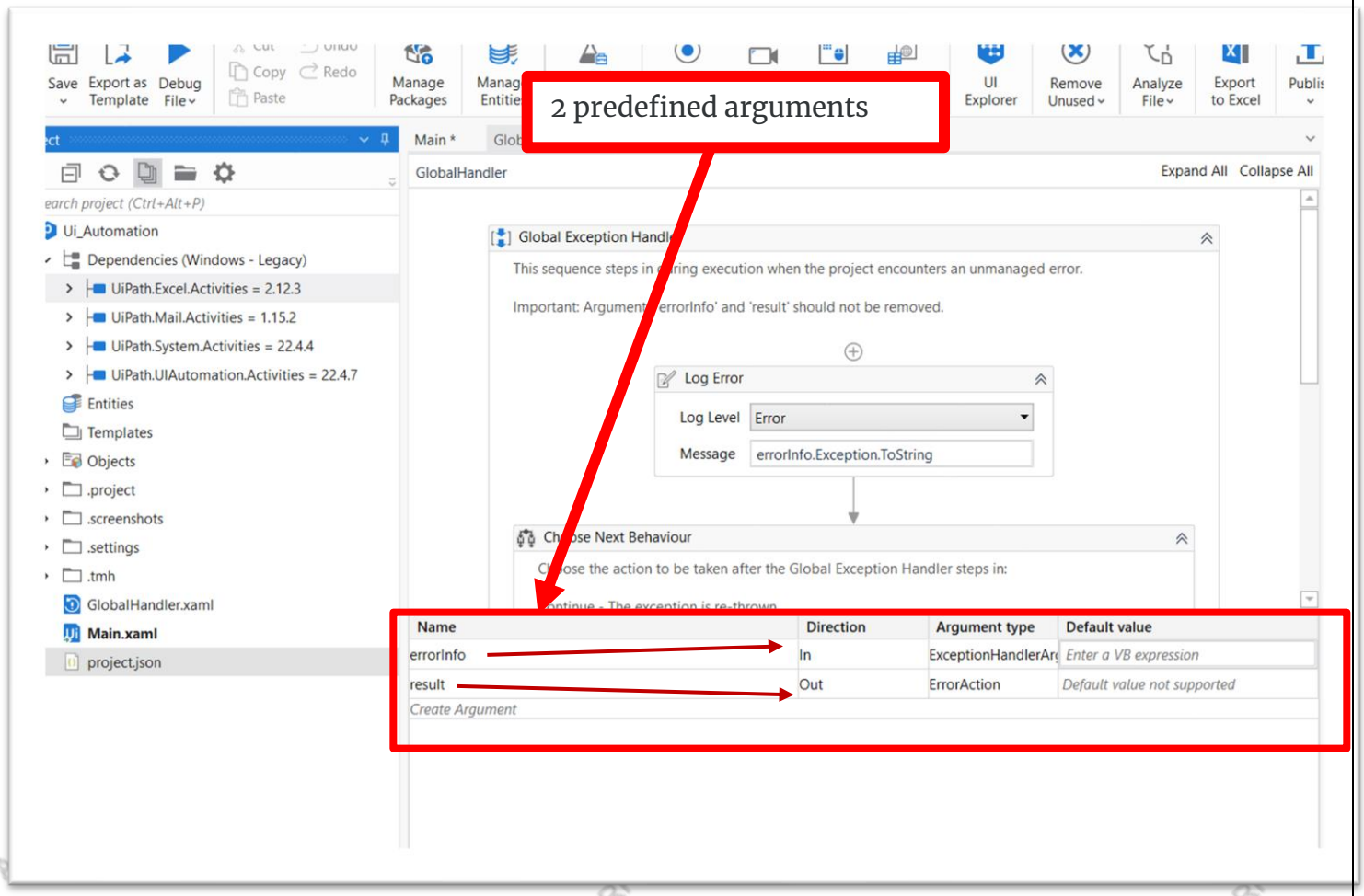


How does it work?

The Global Exception Handler has 2 predefined arguments, that shouldn't be removed:

errorInfo with the In direction - contains the information about the error that was thrown and the workflow that failed.

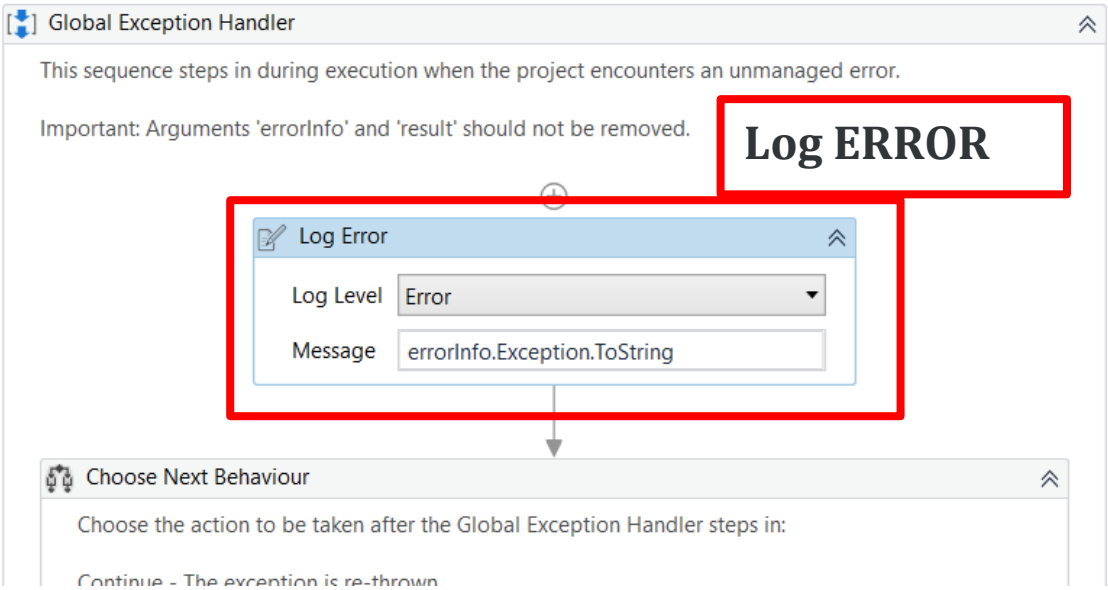
result with the Out direction - used for determining the next behavior of the process when it encounters the error.



The Global Exception Handler contains the predefined activities below. *If required, we can also add other activities in our Handler.*

Log ERROR

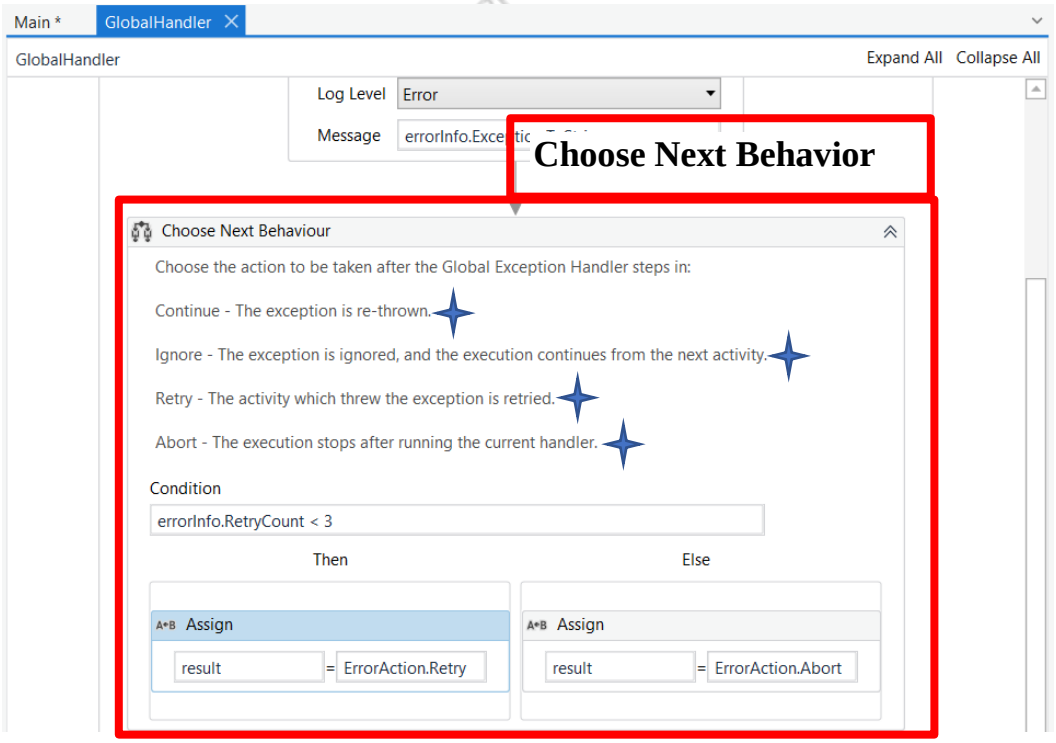
This part simply logs the error. The developer gets to choose the logging level - Fatal, Error, Warn, Info, Trace, and so on.



Choose Next Behavior

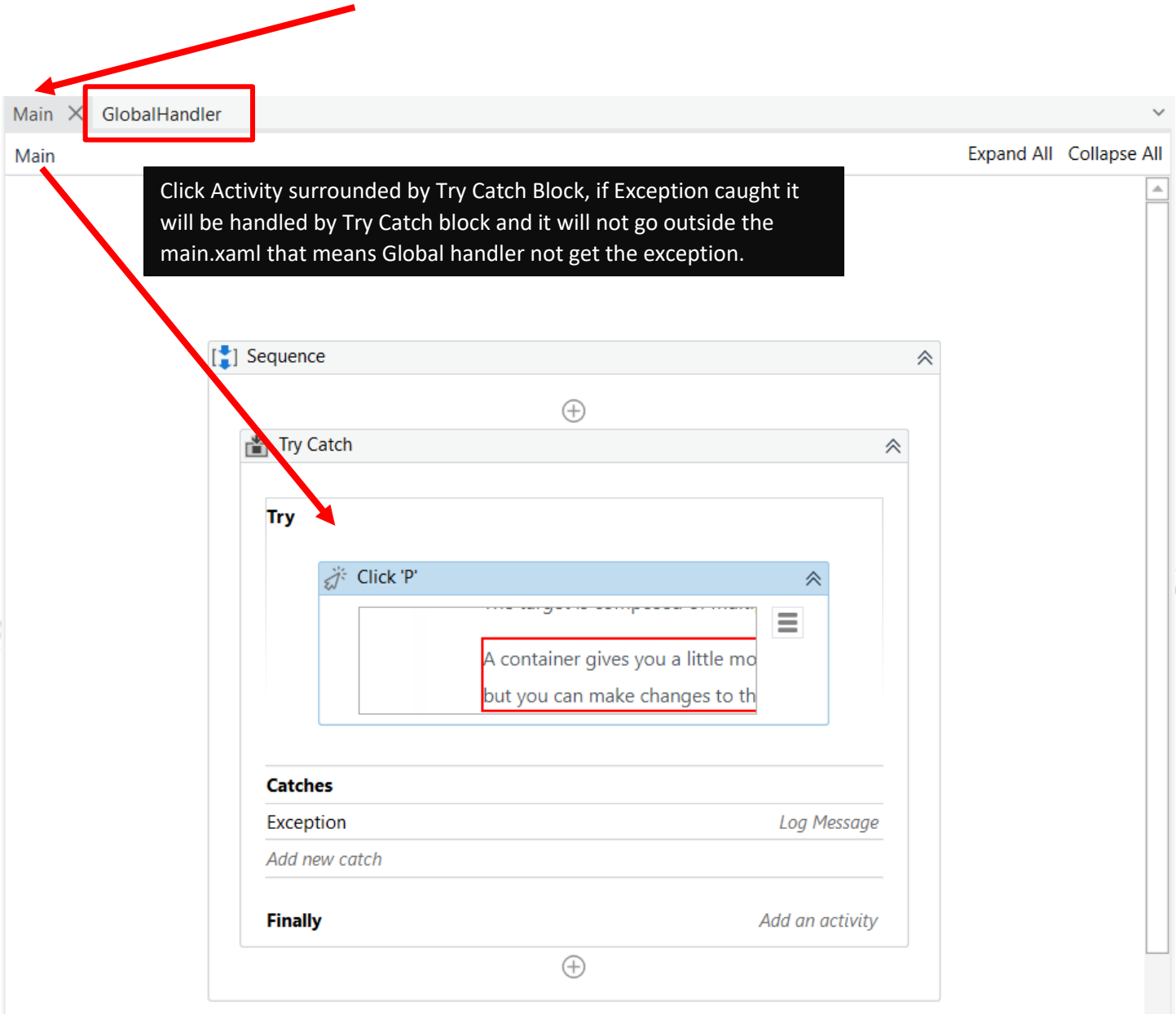
Here the developer can choose the action to be taken when an error is encountered during execution:

- **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.
- **Abort** - The execution stops after running the current handler.



The Global Exception Handler is not available for library projects, only processes.

Only uncaught exceptions will reach the exception handler. If an exception occurs inside a TryCatch activity and is successfully caught and handled inside the Catches block (and not re-thrown), it will not reach the Global Exception Handler.



- In attended scenarios, we can use the Global Exception Handler to let the user decide what course of action the robot should take when it encounters an error.
- We can do this by using a multiple-choice input dialog in the Handler workflow to let the user select between Continue, Ignore, Retry and Abord. Then we assign the corresponding action to the result argument inside a Switch activity.