

Debugging

What is Debugging and why do we need debug?

Debugging is a very important aspect of programming and engineering. It's about running the code and Finding or Detection the Bug. The debugging Important Factors to Consider are What the bug or fault is, Where it has occurred, How it is caused.

Programming has the power to create any virtual experience anyone can imagine. It's fun as long as we're stuck with bugs. And we can spend up to 50% of our time solving them.

“A debugger is an essential and more powerful tool for the developers to help them find the errors and understand the codes in a much better way.”



What is Debugging in UiPath Automation?

Debugging is the process of identifying and removing errors from a given project. Coupled with logging, it becomes a powerful functionality that offers you information about your project and step-by-step highlighting, so that you can be sure it is error-free.

Logging enables you to display details about what is happening in your project in the **Output** panel. This, in turn, makes it easier for you to debug automation Process.

Breakpoints enable you to pause the execution of a project so that you can check its state at a given point.

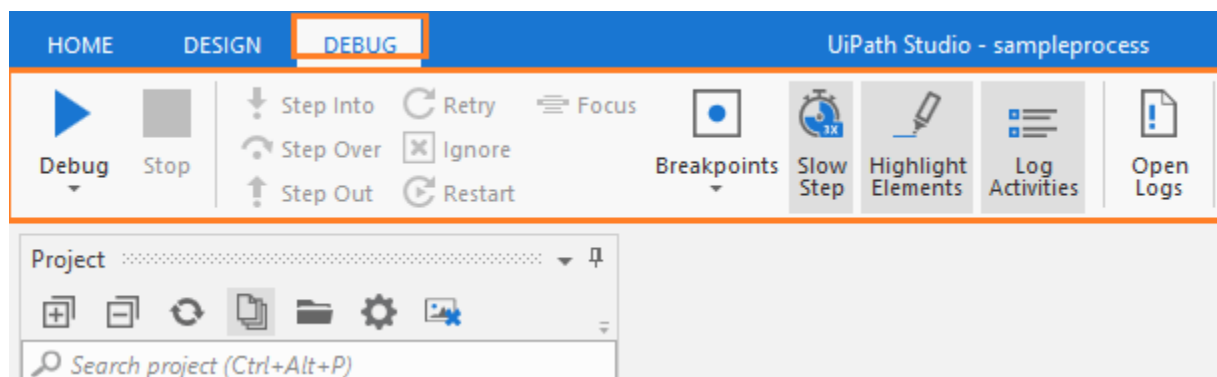
Debugging Actions

Debugging in UiPath is a process of detecting and removing of existing and potential errors (also called as ‘bugs’) in a Single or Multiple Workflows that can cause it to behave unexpectedly or crash.

Debugging of a single file or the whole project can be performed both from the **Design** or **Debug** ribbon tabs. However, the debugging process is not available if project files have validation errors.

When getting started with debugging Actions, keep the following features in mind:

- Debug File, Run File, Debug and Run
- The Ribbon Options
- Breakpoints
- Retry, Ignore and Restart;
- Step Into, Step Over and Step Out
- Break, Continue and Stop
- And the Locals and Output panels.



Main Debugging Actions:

1. **Step Into:** Step into will debug the activities one at a time. This means that it will pause after every step. When you click this action, the debugger opens and highlights the activity before it is executed.

The keyboard shortcut for Step Into is F11.

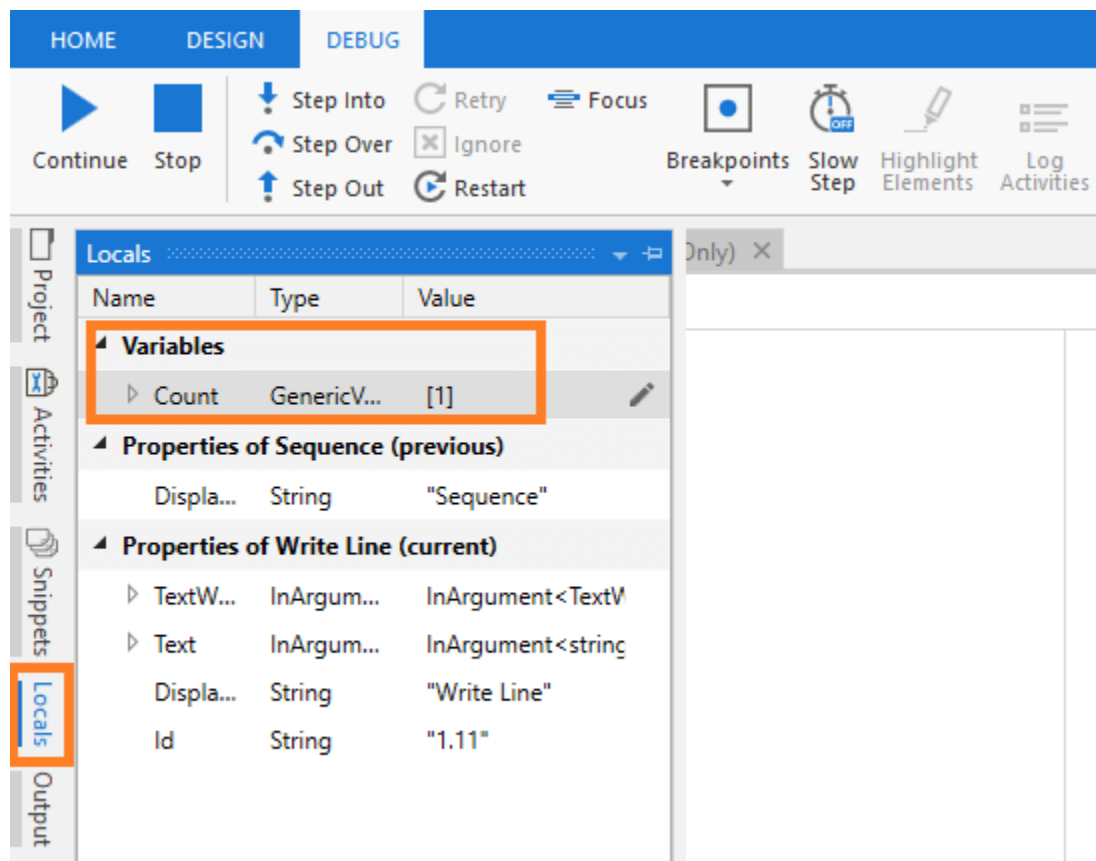
2. **Step Over:** While Step Into executes the activities step by step, Step Over will not open the current container. When used, this action debugs the next activity, highlighting containers without opening them. This action is useful when you want to quickly go through large containers which are unlikely to trigger any issues during execution. The keyboard shortcut for Step Into is F10.
3. **Step Out:** Clicking Step Out will run the activities in the current container, before pausing the debugging. This option works well with nested sequences. The keyboard shortcut for Step Out Shift + F11.
4. **Retry:** Retry re-executes the previous activity, and throws the exception if it's encountered again. The activity which threw the exception is highlighted and details about the error are shown in the **Locals** and **Call Stack** panels.
5. **Restart:** Restart is available after an exception was thrown and the debug process is paused. The action is used for restarting the debugging process from the first activity of the project.
6. **Ignore:** The Ignore action can be used to ignore an encountered exception and continue the execution from the next activity so that the rest of the workflow can be debugged.
7. **Breakpoints:** The breakpoints allow you to pause the debugging process at any given moment. The activity which is being debugged remains highlighted when paused. Once this happens, you can choose to Continue, Step Into, Step Over, or Stop the debugging process. You can place a breakpoint on any activity either by selecting it and clicking the Breakpoints button on the Execute tab, from the context menu, or by pressing F9 while the activity of interest is selected.
8. **SlowSetp:** Slow Step enables you to take a closer look at any activity during debugging. While this action is enabled, activities are highlighted in the debugging process. Moreover, containers such as flowcharts, sequences, or Invoke Workflow File activities are opened. This is similar to using Step Into, but without having to pause the debugging process. Although called Slow Step, the action comes with 4 different speeds. The selected speed step runs the debugging process slower than the previous one. For example, debugging with Slow Step at 1x runs it the slowest, and fastest at 4x. In other words, the speed dictates how fast the debugger jumps from one activity to the next.

Debugging Panel

Debugging Panel is an important part of debugging to identify and remove errors. Generally, it is recommended to perform it during the design stage of the automation project to assure high quality. As we will discover there are several panels that makes it easier to view the debugging process, add values or monitor variables and arguments.

There are 6 types of Debugging Panel:

1. **Local Panel:** Locals panel knows everything about a process such as variables, variable values, runtime values, and arguments, and you can easily view the values when you are running the workflow. To easily understand what Locals panel is used for; we can relate the concept to a real-life example. Let's suppose a stranger comes to your local area and asks you for a place to visit. You will easily be able to give directions and guidance.



2. **Watch Panel:** Watch Panel (as the name suggests) is used to watch something specific. In Studio, it is used to watch the variables and arguments values. And this is a particularly useful panel, while debugging, we do not have to search for the variables value in the output/local panel since we can directly monitor specific variables.

Watch		
Expression	Value	Type
FirstVariable	8	Int32
Result	12	Int32
SecondVariable	4	Int32
FirstVariable/SecondVariable	2	Double
Add Watch		
Call Stack Watch Breakpoints		

Ways in which we can variables in watch panel :

A. Add via the add-in Watch panel

Name	Variable type	Scope	Default
Value1	Int32	Sequence	Enter a VB expression
Value2	Int32	Sequence	Enter a VB expression
Sum	Int32	Sequence	Enter a VB expression
val	String	Sequence	Enter a VB expression

Convert to Argument
Copy
Paste
Delete
Add Annotation
Edit Annotation
Delete Annotation
Add Watch
Find References

100%

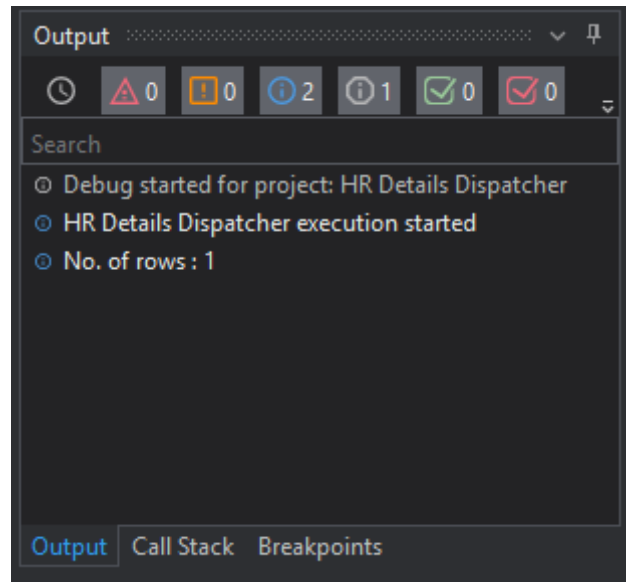
B. Add via right-click from Locals

Locals		
Name	Type	Value
Variables		
Value1	Int32	0
Value2	Int32	0
Sum	Int32	0
value	String	null
Properties of Sequence (previous)		
DisplayName	String	Sequence
Properties of Assign: Value 1 (current)		
To	OutArgument	OutArgument<int> ArgumentType...
Value	InArgument	50

3. **Immediate Panel:** Immediate panel in debugging is used to act immediately. The Immediate panel goal is to see values, variables, or arguments and to evaluate an expression and output the result. For example, if you have two integer values A & B and you want to see the value of A+B, you can do that in the immediate panel.

Immediate	
> A	< 12
> B	< 12
> A + B	< 24
>	
Locals Watch Immediate	

- 4. Output Panel:** Output panel is used to see the log message or any custom value that we are writing using Log Message or Write Line activity. It is also used to notify information regarding packages installation, remote debugging states, etc. Note: The output panel is also available outside the debugging process. It also shows the execution start and end time in the screenshot below. This panel also categorizes the log messages into distinct categories like warning, information, trace, error, and test results in Failed and Passed Assertions.

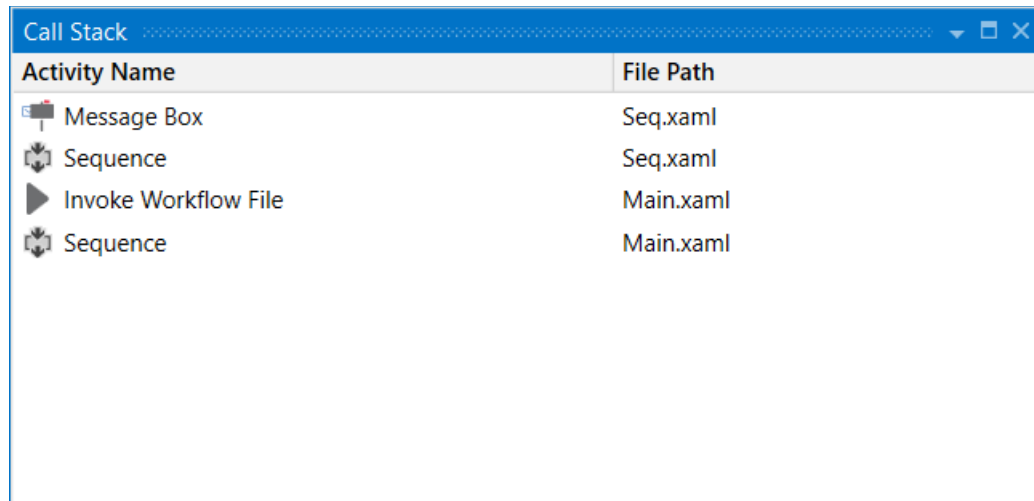


- 5. Breakpoints Panel:** Breakpoints are used to pause workflow at any given point and monitor the values or check if the workflow is running as expected. Breakpoints panel helps to monitor the breakpoints in the workflow. It shows which activity has the breakpoint and is present in which Extensible Application Markup Language. It is also used to give conditions at break points.

Breakpoints				
	Activity Name	File Path	Condition	Log Message
●	Click 'editable text'	Main.xaml		
⊕	Assign	Main.xaml	FirstVariable=1	"Should be different than 1" ⚙
○	Input Dialog	Main.xaml		
●	Type Into 'editable text'	Main.xaml		

- 6. Call Stack Panel:** The Call Stack panel is used to see the upcoming task to be performed within a particular sequence. It shows three things: activity name, activity type, file path.

The Call Stack panel is immensely helpful when there are heavy workflows. We can use it to highlight the currently executing activity within the workflow.



Activity Name	File Path
Message Box	Seq.xaml
Sequence	Seq.xaml
Invoke Workflow File	Main.xaml
Sequence	Main.xaml