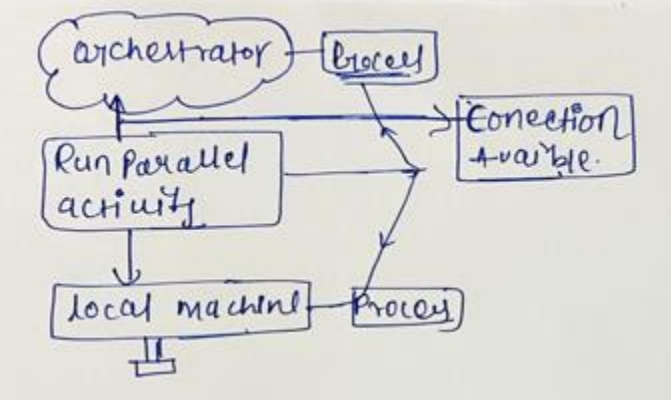
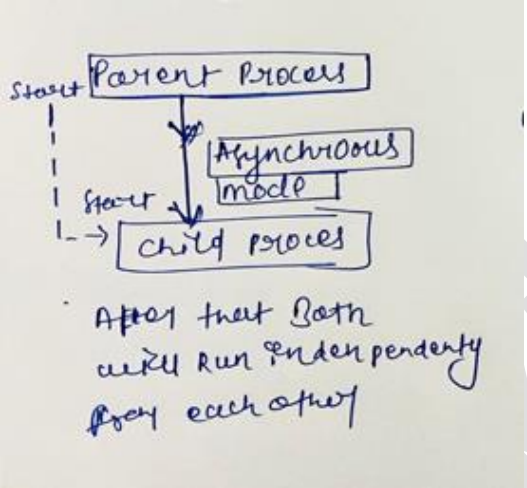
Run Parallel Process activity: -

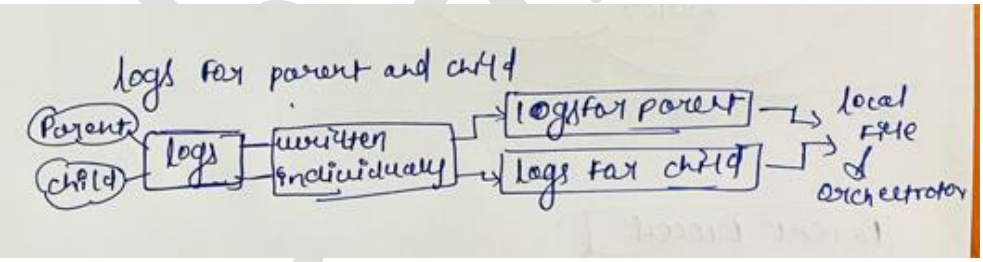
Executes a UiPath process that is available for the local machine.
Can be used to run local packages, as well as processes in Orchestrator, if a connection is available.



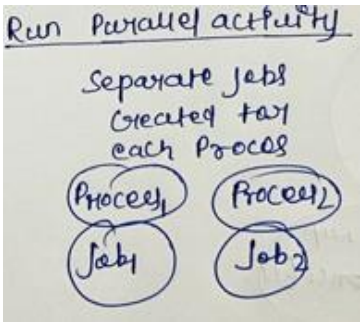
The Run Parallel Process activity runs in Asynchronous mode, meaning that the parent process is only responsible for starting the child ones.
After that happens, both processes run independently from each other.



Logs generated for the execution of the parent and child processes are written individually in the local log files and Orchestrator.

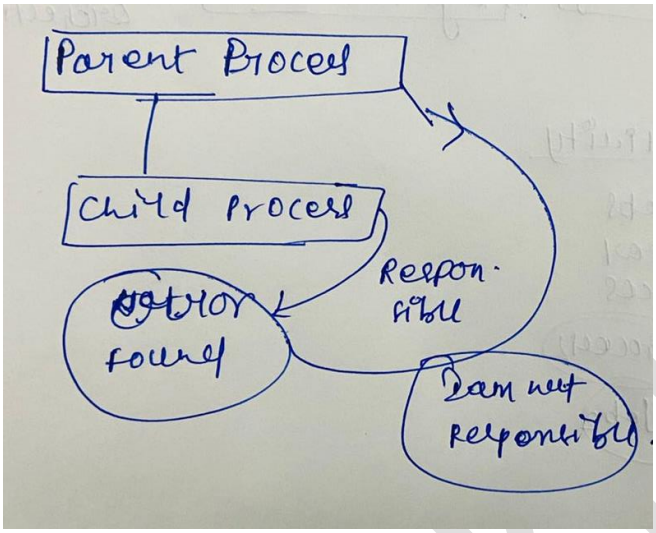


They both contain complete information about each execution.
When using the Run Parallel Process activity, a separate job is created for each process, as opposed to the Invoke Process activity which creates only one job.



If a child process is invoked but not found, an error is logged containing the name of the child process, the path where Robot searched for it and additional details such as WindowsIdentity and machineName. When this happens, an exception is thrown and additional details are being logged.

If an error occurs in the child process after it has been invoked, the parent is not aware of it and continues its normal execution.



Note:
The Run Parallel Process activity only works with Attended Processes.

Run Parallel Process lets us invoke a local process which executes alongside the caller process. In the Process Name property of the activity, we must enter the package name. We can pass data to the invoked parallel process, but we cannot retrieve data, the only available argument direction is in

---

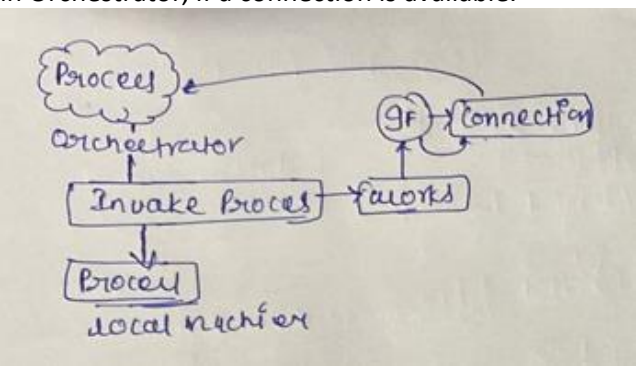
We can use 'Invoke Process' to execute a process which is available for the local machine. The parent process waits for the child process to complete before resuming. The Process Name property should contain the name of the Package. Data can be passed between the parent and child processes by using arguments.

Invoke Process

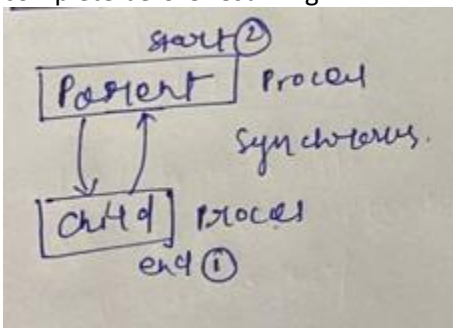
UiPath.Core.Activities.InvokeProcess

About the InvokeProcess activity

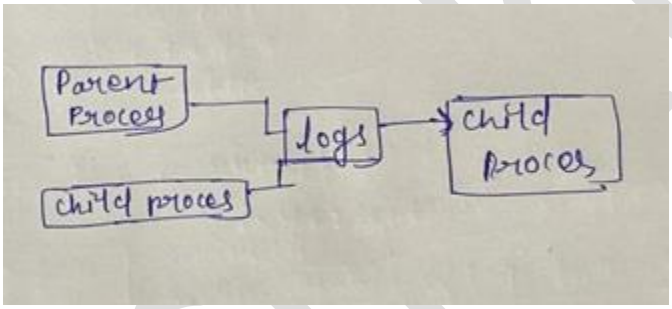
Executes a UiPath process that is available for the local machine. Can be used to run local packages, as well as processes in Orchestrator, if a connection is available.



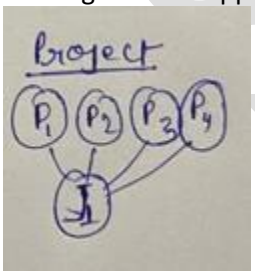
The Invoke Process activity runs in Synchronous mode, meaning that the parent process waits for the child process to complete before resuming.



Logs generated by the child processes only contain the outcome of the execution and errors, and are written in the same place as the ones from the parent and can be differentiated by the ProcessName field in local logs and Process column in Orchestrator.



When using the Invoke Process activity, even though the project contains multiple processes, a single job is created for running them as opposed to the Run Parallel Process activity which creates separate jobs for each process in the project.



If there are multiple processes added, the active version on the current folder is the one that will be used.

Note:

Unserializable data types cannot be passed between workflows (e.g. UIBrowser, UIElement, etc).

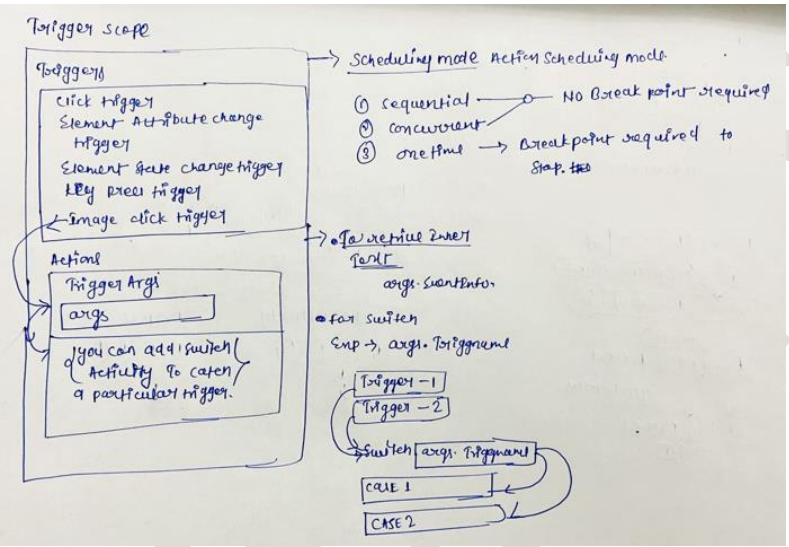
If a child process is invoked but not found, an error is logged containing the name of the child process, the path where Robot searched for it and additional details such as WindowsIdentity and machineName.

Triggers

Multiple triggers can be added in the Trigger Block of the Trigger Scope activity.

File change triggers allow us to monitor file or folder changes.
Click triggers allow us to respond to mouse events.
Key press and hotkey triggers allow us to respond to keyboard input.
And a click image trigger allows us to respond when a user interacts with a graphic image.

Triger Scopes have three scheduling options:
One Time - Execute one action and exit.
Sequential: Actions are executed one after another.
Concurrent: Action executions can overlap.
Information is passed from the Trigger block to the Action block via Trigger Arguments.
A few key properties are Trigger Name, Trigger Type and Event Info. We can define different courses of action in the Action block by using a Switch activity.



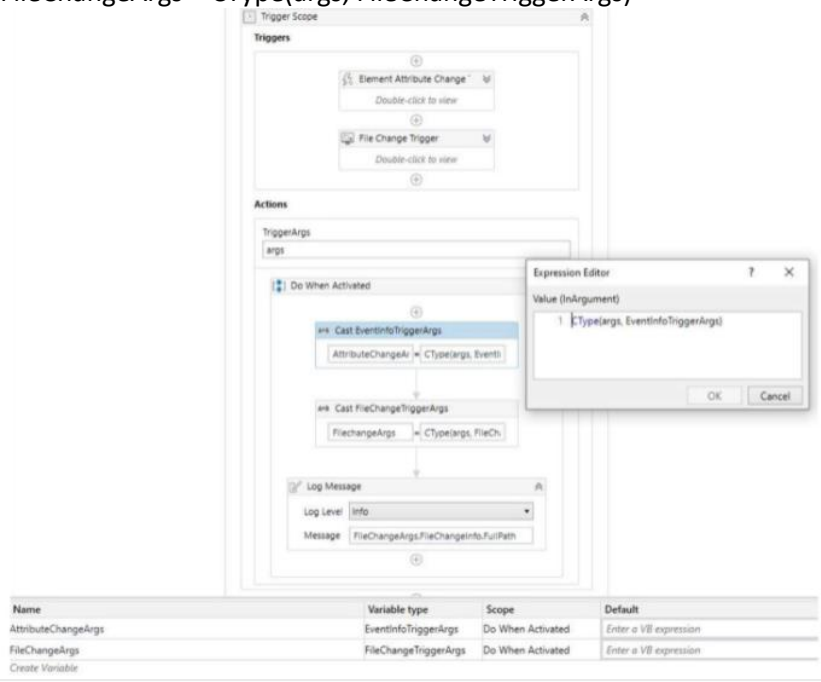
Casting Trigger Arguments

Now that we've learned about Trigger Scope, Trigger activities, and Trigger Arguments there's one specific situation we need to cover.
When adding different types of triggers in the same Trigger Scope activity (e.g. Element Attribute Change Trigger and File Change Trigger), the args type defaults to TriggerArgs.
This means trigger arguments lose the specific event properties (in our example, EventInfo, and FileChangeInfo).

While it's still possible to identify the trigger name or type, by default, we cannot pass specific information about the event to the Actions container. Fortunately, there is a way to solve this, by casting the specific type of argument to new variables. Here's how this works:

In a scope included in the Actions container, create a new variable for each type of trigger. (e.g. AttributeChangeArgs and FileChangeArgs).
Set each variable type to the specific argument type associated to each trigger activity. (e.g. UiPath.Core.Activities.EventInfoTriggerArgs and UiPath.Core.Activities.FileChangeTriggerArgs).

Inside the scope of the new variables, add Assign activities, convert the default args to the new types and assign it to the new variables. Like this:
AttributeChangeArgs = CType(args, EventInfoTriggerArgs)
FileChangeArgs = CType(args, FileChangeTriggerArgs)

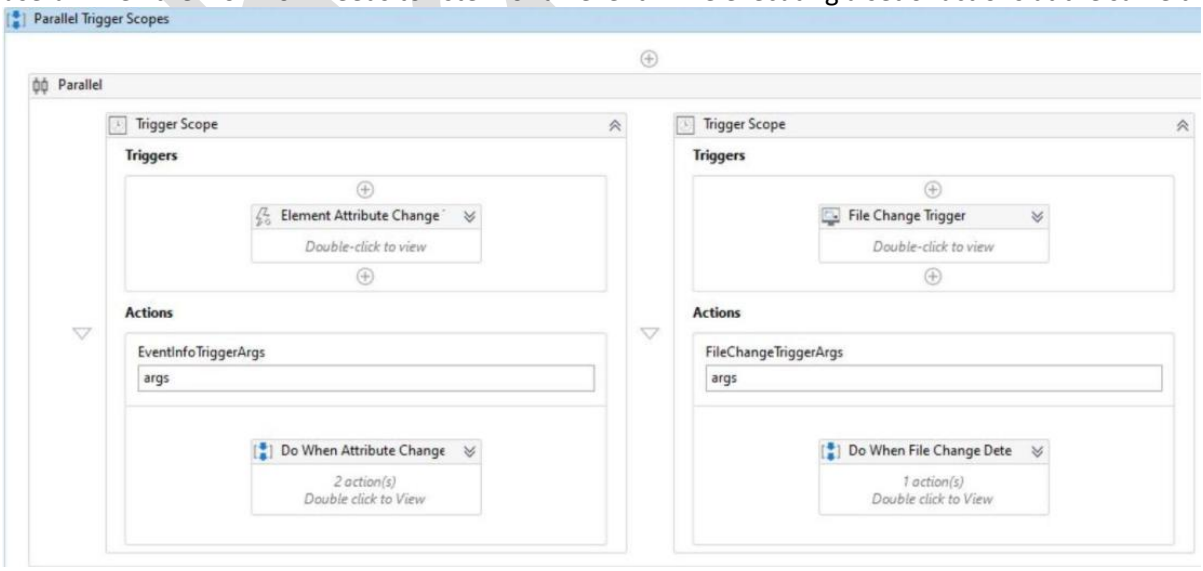


Use the two variables to pass specific event data.
Note**: You can identify the data type of an argument by adding a separate Trigger Scope activity, adding the specific trigger we want to get the data type for and then hovering the mouse over the label of the args field.**

Trigger Scopes in Parallel:-

Note that a different way to organize your triggers is to use one Trigger activity per Trigger Scope and to include them in different lanes of the Parallel activity.

We don't recommend using this approach when you need to use more than two Trigger Scopes, however, it can be very useful when the workflow needs to listen for an event while executing a set of actions at the same time.



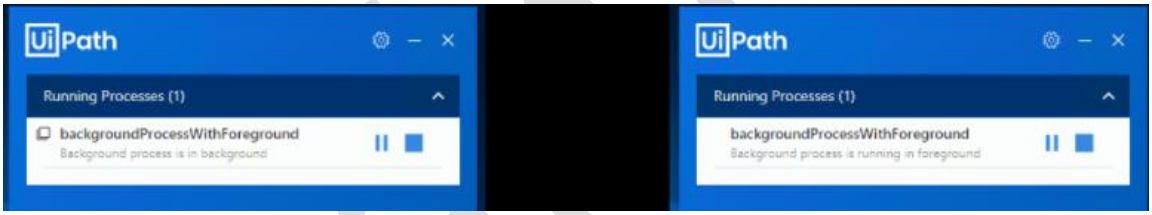
Background Processes and Use Foreground :-

Moves the current background process into the foreground, executing all the activities it contains.
After the execution is complete, the process is moved back into the background.

if another process is already running in the foreground?
If we start a background process with a foreground scope and another entity is running in the foreground (either a process or another foreground scope),
the background process goes into a queue and waits, determined by the 'Wait for foreground' property, to acquire foreground.

Only background processes with a foreground scope make use of this queue, not standalone foreground processes.
This means that a foreground process fails if started while either another foreground process or a background process with a foreground scope is running.

If we need to use UI Interactions in a background process, we should surround them with a "Use Foreground" activity.
If the foreground is busy when this activity executes, it will be added to a queue of processes waiting for the foreground to be freed up.
Orchestrator lets us disable the stop button for individual processes in Assistant and to set processes to start automatically when Assistant is first started on the user's machine.



Background Processes and Triggers

Use Triggers to monitor local events.

- The Background Process is a template for creating processes that can run in parallel on the same Robot, together with one foreground process.
- For this reason, background processes must not contain activities that require UI Interaction activities like Click and Type Into.

Note that this feature is only available for Attended Robots.

- When using Unattended Robots to run multiple Processes at the same time, each Running Process consumes a separate license.
- Triggers let us monitor certain events on the user's machine like UI element changes, keyboard actions, or mouse actions and define activities to be executed in response to these events.
- A background process is one that can run concurrently with other background processes and potentially a single foreground process on the same Robot.

- Background processes can pass data back and forth between other foreground and background processes.
- This activity allows us to define a set of triggers in the Triggers container and a sequence of activities to execute based on trigger response in the Actions container.

Please note that, to interrupt the Trigger Scope activity handling sequence, the Break activity should be used. This allows the workflow to continue execution as intended.
three background processes running and one foreground process running. However, does not interfere with the behavior of our automation, as multiple UiPath background processes are allowed to run at the same time.

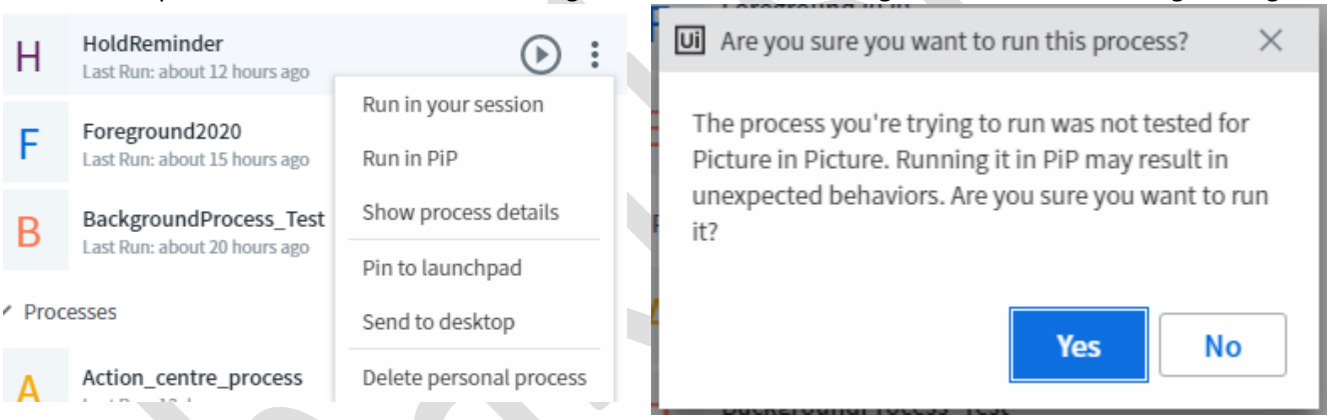
Picture-in-picture

Picture-in-picture, or PIP, is a powerful feature that helps simplify side-by-side automations.
Any process that starts in the picture-in-picture mode runs in its own, isolated, Windows session.
This allows automation to run without interfering with the side-by-side user.

This mode lets us configure the entire process or individual workflows to run in a separate session.
However, this does not mean the logged-in user cannot intervene. When needed, they can take control and provide input or recover from errors.

There are a number of ways to execute a process in Picture-in-Picture mode.
One way is to choose this option in UiPath Assistant.

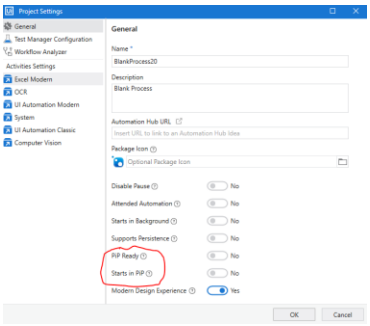
Notice that when the start menu is selected, one of the options is to 'Start in PiP.'
Because this process was never tested and designed to run in PiP, we are greeted with this warning message.



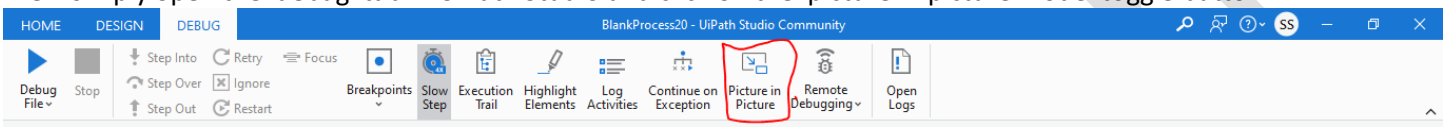
We can run it, but it's important to note that PiP does have limitations,
and automations not designed with PiP in mind may encounter runtime issues.
click 'Yes' to tell Assistant that we do indeed want to run the automation in this mode.

Notice how the picture-in-picture mode executes the automation in an isolated window that blocks all interactions unless the 'Take Control' button is enabled.

Another way to have processes start in PiP mode is to configure it as the default run mode from Project Settings.
For this, we will simply open a project's 'Settings' and enable the 'Starts in PiP' toggle.
Notice how there is also a 'PiP ready' toggle.
If an automation has been developed and tested in picture- in picture mode, we should enable this option.

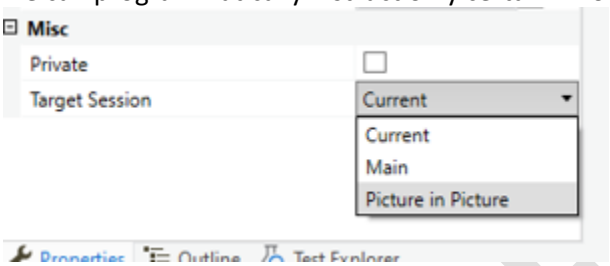


It's also possible to run processes in PiP mode during development.
We'll simply open the 'debug' tab in UiPath Studio and click on the 'picture in picture mode' toggle button.



Picture-in-picture then becomes the default run mode during development and testing.

we can programmatically instruct only certain invoked workflows to run in an isolated, Picture-in-Picture window.



The options are:
Current - The child process opens in the same session as the parent one
Process Default - The child process uses the Process Settings
Main - The child process starts in the main session regardless to where the parent process runs
Picture-in-Picture - The child process starts in the Picture-in-Picture session regardless to where the parent process runs

Microsoft Office Automation
Automations that use Microsoft Office resources do not run successfully in Picture-in-Picture if the resources are already open in the main session. In order to make sure that automations run smoothly in PiP, you can do the following:

- **Close the resource used by Microsoft Office applications from the main session so they can be opened in the PiP session.**
- **Use an InvokeIsolatedWorkflow activity to invoke the part of the automation using Microsoft Office and set its Target Session to Picture-in-Picture from Studio.**
- **It's worth noting that Microsoft Office applications in general do not run successfully in Picture-in-Picture mode if they are already open in the user's main session**.

For automations which use both the PiP mode and Office products, either ensure they are closed in the user's session, or change the Target Session of the workflow that uses the apps to Main

Note that running a sequence in isolated mode introduces some limitations, **one of which is the inability to use non-serializable data types, such as the 'Browser' type, for arguments.**

Data Types such as Int and String, however, can be passed without issue.
There are other limitations pertaining to web browsers we should be aware of when using PiP.

Working with Browsers in different sessions

The User Data browser folder contains essential profile data such as history, bookmarks, cookies, as well as other pre-installation local states.
Each profile is a subdirectory (often Default) within the user data directory.

We cannot have a specific browser (let's say Chrome) be open with the same user profile **both in the PiP and the Main session at the same time.**

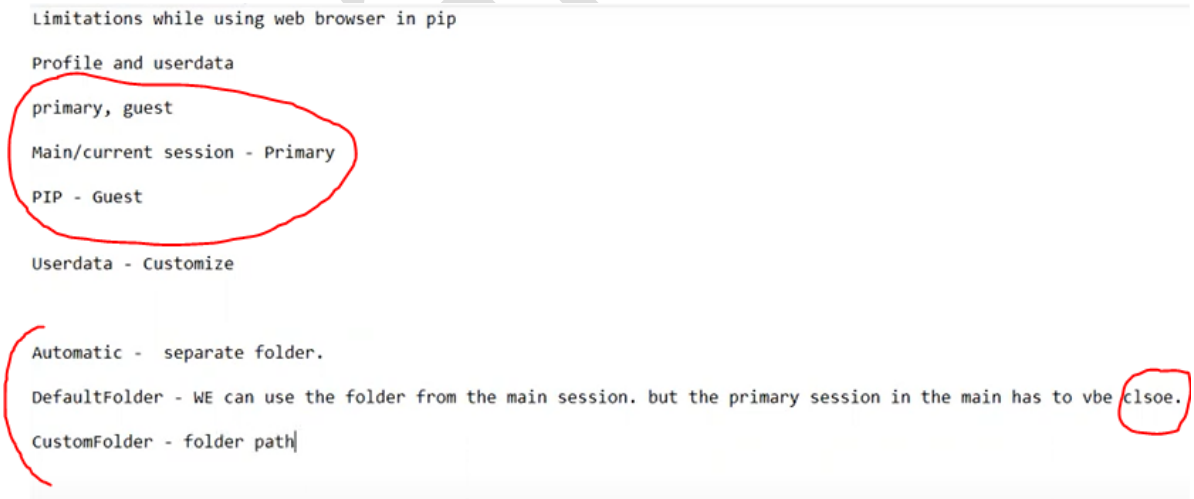
For this reason, when working with Browsers in the PiP session, it's good to consider the UserDataFolderMode and UserDataFolderPath properties of the Open Browser and Use Application/Browser activities.

UserDataFolderMode is used to start the browser with a specific user data folder. It contains the following options:
UserDataFolderMode.Automatic
Picture In Picture mode uses a different folder than the default mode, automatically generated if UserDataFolderPath is not set. The implication is that the same browser can run concurrently in both the Main and the PiP sessions with different user profiles. However, we cannot use settings stored in the Default browser profile in the PiP session.

UserDataFolderMode.DefaultFolder
Uses the default browser folder, no matter if it runs in the main or PIP session. The implication is that before opening the browser in the PiP session, we need to make sure it is not open and is using the default profile in the Main session.

UserDataFolderMode.CustomFolder
Uses the folder specified in UserDataFolderPath or an auto generated path if UserDataFolderPath is not set. This setting lets us create and control a custom user profile for our automations running in the PiP session.

UserDataFolderPath defines the user data folder that the browser will use. Defaults to %LocalAppData%\UiPath\PIP Browser Profiles\BrowserType if not set. This property is used if UserDataFolderMode is set to CustomFolder.



Input method capabilities

Capability Method	Compatibility	Background Execution	Speed	Hotkey Support	Auto Empty Field
Default	100%	no	50%	yes	yes
SendWindowMessages	80%	yes	50%	yes	no
Simulate Type/Click	99% - web apps 60% - desktop apps	yes	100%	no	yes

Block User Input

UiPath.System.Activities.BlockUserInput

A container which disables the mouse and keyboard when activities inside it run. Can be configured to block either mouse, keyboard, or both, and permits you to designate a hotkey combination to re-enable them.

Important!
Certain user input devices such as touchpads, trackpads or touchscreens may not be completely blocked by this activity. Due to the wide variety of device models and drivers as well as the complex commands their gestures can produce, some multi-finger gestures or touchscreen specific interactions are sent to the operating system as software events. Since in UI Automation scenarios, Robots use software events to simulate human interaction, they cannot be blocked.

Note:
Using Block User Input with Parallel activities (such as Parallel For Each) is not supported.

Note:
When running an automation process that uses the Block User Input activity as a job from Orchestrator, and the Executor does not have administrator privileges on the machine you run the process on, any application that has a higher privilege than the Executor and is in focus cannot be blocked by the activity. As a workaround, offering administrator privileges to the Executor enables the Block User Input activity to work as expected.

Note:
Using Debug inside a Block User Input scope enables the effects of the scope. To disable the block, you can use the hotkey combination specified in the Unblock Options section of the properties. Once turned off, the state of all elements can be inspected as usual in a Debug context. Please note that execution can only be continued without the effect of the scope and workflow functionality might be broken.
