**INTRODUCTION**-

UiPath Studio is a complete solution for application integration, and automating third-party applications, administrative IT tasks and business IT processes. One of the most important notions in Studio is the automation project.A project is a graphical representation of a business process. It enables you to automate rule-based processes, by giving you full control of the execution order and the relationship between a custom set of steps, also known as activities in UiPath Studio. Each activity consists of a small action, such as clicking a button, reading a file or writing to a log panel.

The main types of supported projects are:

* [**Sequences**](https://studio.uipath.com/docs/sequences) - suitable to linear processes, enabling you to smoothly go from one activity to another, without cluttering your project.
* [**Flowcharts**](https://studio.uipath.com/docs/flowcharts) - suitable to a more complex business logic, enabling you to integrate decisions and connect activities in a more diverse manner, through multiple branching logic operators.
* [**State Machines**](https://studio.uipath.com/docs/state-machines) – suitable for very large projects; they use a finite number of states in their execution which are triggered by a condition (transition) or activity.

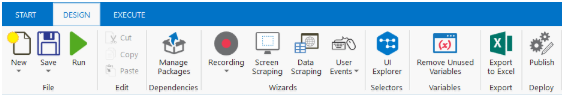
# 1.2 The User Interface-

# UiPath Studio contains multiple panels for an easier access to specific functionalities. They can be docked, act as floating windows, or the **Auto-hide** option can be enabled.

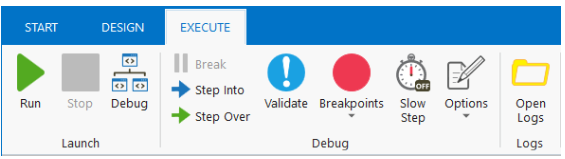
# 1.2.1 The Ribbon-

The ribbon is straightforward and can be minimized or expanded by clicking the Minimize / Expand button . It consists of the following four tabs:

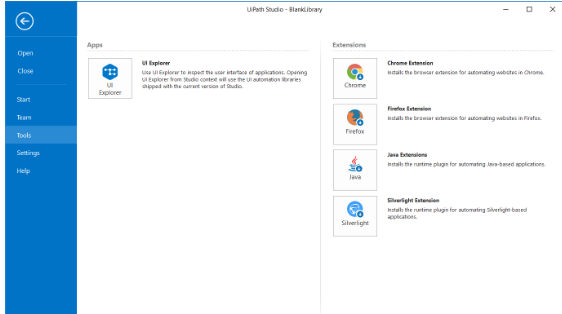
1. **Start** - create a project or open a previously created one, go to the online documentation or submit a request, or for the Community Edition switch to a Beta or Stable version. By default, projects are created in C:\Users\Username\Documents\UiPath.
2. **Design** - create or launch sequences, flowcharts or state machine diagrams, access wizards, manage variables, and inspect user interface elements from third-party apps.



1. **Execute** - run or stop projects, start the debug process, slow down steps and open logs.



1. **Setup** - publish a project or create a shortcut for it, schedule tasks, and install extensions with just one click.



# 1.3 The Quick Access Toolbar

The **Quick Access Toolbar** is located by default on the title bar, above the ribbon, yet you can easily move it below.It provides shortcuts for the most used commands, and you can easily add new ones to it by right-clicking a desired button and selecting the **Add to Quick Access Toolbar** option.

1.The Universal Search Bar-The Universal Search bar enables the user to find anything contained in the Activities, Libraries, Project and Designer panels of Studio.

# 2.The Designer Panel- The **Designer** panel displays your current automation project, enables you to make changes to it, and provides quick access to [variables](https://studio.uipath.com/docs/managing-variables), [arguments](https://studio.uipath.com/docs/managing-arguments) and [imports](https://studio.uipath.com/docs/about-imported-namespaces) .It is possible to navigate within a diagram by double-clicking the activity you want to view. The path is displayed as breadcrumbs in the header of the **Designer** panel.

# The context -menu enables you to perform several operations on your workflow, be it a sequence or a flowchart. You can display it by right-clicking an activity inside your workflow.

## **Field Descriptions for the Context Menu**

**View Parent**

Displays the parent of the target activity in the **Designer** panel.  
**Note:** This option is only displayed for child-activities, if they are opened in the **Designer** panel.

**Open**

Opens the selected activity in the **Designer** panel. Has the same result as double-click.

**Collapse**

Collapses the activity thus reducing the displayed details to "Double-click to view". You can achieve the same result by clicking the **Collapse** button .  
**Note:** Only displayed for sequences.

**Expand in Place**

Expands the content of the activity, displaying its detailed content. You can achieve the same result by clicking the **Expand** button .  
**Note:** Only displayed for sequences.

**Cut**

Removes the selected activity and places it onto the Clipboard.

**Copy**

Copies the selected activity and places it onto the Clipboard.

**Paste**

Inserts the content of the Clipboard at the current location.

**Delete**

Deletes the target activity.

**Note:** Since the **Context** menu is only displayed for a one-activity selection, you can use this command for more activities with the corresponding keyboard shortcut, Delete.

**Annotations**

Add, edit, delete, show and hide annotations added to activities in the workflow.

**Copy as Image**

Makes a screenshot of the content displayed in the **Designer** panel.

**Note:** If you want to make a screenshot of a child activity, make sure to firstly open it in the **Designer** panel.

**Save as Image**

Makes a screenshot of the content displayed in the **Designer** panel and prompts the user to set the details for saving it (Name, Type, Destination). Supported formats: JPG, PNG, GIF, XPS.

**Create Variable**

Creates a variable in the **Variable** panel.

**Remove Sequence**

Removes the selected **Sequence** activity from the **Designer** panel.

**Surround with Try Catch (Ctrl + T)**

Inserts the activity in a [**Try Catch**](https://activities.uipath.com/docs/try-catch) statement.

**Note:** A **Try Catch** statement is used for handling exceptions caused by data or coding errors. The **Try** clause encloses the activity to be checked for exceptions. The **Catches** clause is the exception handler. The **Finally** clause is used for executing an activity regardless of the status of the first two clauses.

**Extract as Workflow**

Creates a new workflow containing the targeted activity with the purpose of breaking down a large project into smaller ones. In the place of the extracted activity an **Invoke <new> workflow** activity is created which enables you to edit and import arguments from the newly created workflow. The arguments are created automatically from the variables used in that activity.

**Open Workflow**

Opens a workflow created using the above option.

**Enable Activity**

Enables a previously disabled activity.

**Disable Activity**

Disables an activity, which is then placed inside a **Comment out**activity.

**Toggle Breakpoint**

Marks the selected activity as a breakpoint for debugging. Breakpoints can also be toggled by clicking the **Breakpoint** button in the **Execute** tab. You can trigger a breakpoint for one activity at a time.

**Show All Conditions**

Displays all conditions defined in your project (**Properties** panel > **Conditions**).

**Note:** This option is only displayed for flowcharts when you right-click an empty space inside the project; it is not displayed if you right-click an activity in your flowchart.

**Hide All Conditions**

Hides all displayed conditions.

**Note:** This option is only displayed for flowcharts when you right-click an empty space inside the project; it is not displayed if you right-click an activity in your flowchart.

**Set as Start Node**

Connects the selected activity with the **Start** node.  
**Note:** Only displayed for flowcharts.

**Set as Initial State**

Connects the State Machine specific activity to with the **Start** node.  
**Note:** Only displayed for state machines.

The context menu is also displayed for tabs in the **Designer** panel. The following options are available:

Option

Description

**Close**

Closes the active tab.

**Close Others**

Closes all tabs but the active one.

**Close All Documents**

Closes all tabs.

**Float**

Undocks the target tab and changes it to a floating state.

**Pin Tab**

Pins the target tab in the **Designer** panel.  
**Note:** The tab pinned last is always moved in front of all other existing tabs (pinned or not). The position of a pinned tab can only be changed relative to other pinned tabs.

**New Horizontal Tab Group**

Splits the screen horizontally, enabling you to see two or more instances of the **Designer** panel within the same screen. The target project is moved onto the panel on the right.  
**Note:** Only displayed when your project contains two or more tabs and, implicitly, workflows.

**New Vertical Tab Group**

Splits the screen vertically, enabling you to see two or more instances of the **Designer** panel within the same screen. The target project is moved onto the panel at the bottom.  
**Note:** Only displayed when your project contains two or more tabs and, implicitly, workflows.

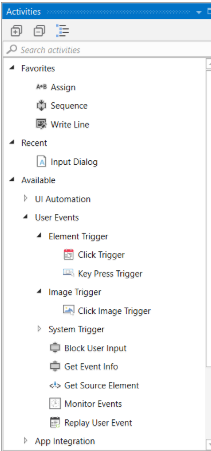
**Move to Previous Tab Group**

Moves the target tab onto the previous **Designer** panel displayed.  
**Note:** Only displayed if you had used the **New Vertical Tab Group** or **New Horizontal Tab Group** options before.

**Move to Next Tab Group**

Moves the target tab onto the next **Designer** panel displayed.  
**Note:** Only displayed if you had used the **New Vertical Tab Group** or **New Horizontal Tab Group** options before.

**The**[**Activities**](https://activities.uipath.com/)**panel** provides quick access to all available activities that can be dragged to the current project.It features a search box, and the **Show Activities** list enables you to hide or show the **Favorites**, **Recent** and **Available** folders of activities.The **Manage Packages** functionality enables you to install additional activity packages.



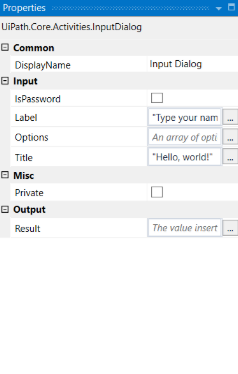
# The Snippets Panel

# enables you to easily reuse automations. A search box is included to facilitate finding items faster.It includes, by default, multiple samples and snippets, and you can add your own by clicking the ****Add Folder****button and selecting a directory from your hard drive.To remove a folder, right-click it and select ****Remove****.

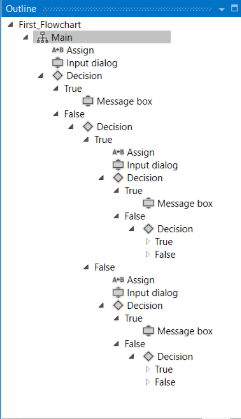
The **Snippets** panel enables you to easily reuse automations. It includes, by default, multiple samples and snippets.You can add your own by clicking the **Add Folder**button and selecting a directory from your hard drive. Empty folders are not displayed.To remove a folder, right-click it and select **Remove**.

**The Project panel** enables you to view the contents of the current project and open the file location in a **Windows Explorer** window.You can connect to a type of source control, either TFS or SVN, and create a new project directly from this panel.Version control is available through the context menu, and it is also possible to view the history of a selected file.

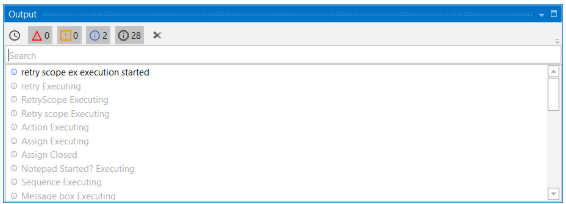
**The Properties panel** is contextual and enables you to view and change the properties of a selected activity.The properties can be sorted alphabetically, while the search box enables you to look for a specific one.



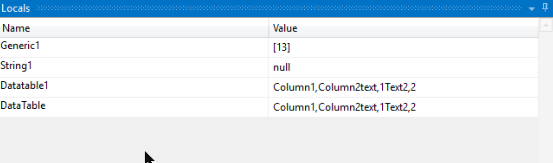
**The Outline panel** displays the project hierarchy, all available variables and nodes, and includes a search box. This enables you to easily navigate through large automations.You can highlight activities in this panel by selecting them in the **Designer** panel, or you can go to a specific activity by selecting it in the **Outline** panel.



**The Output panel** enables you to display the output of the **Log Message** or **Write Line**activities, as well as the logs when the debug mode is activated.When debugging, the logs are more or less detailed, depending on the option you selected in the **Execute** tab, under **Options > Logging Level**.You can hide or show timestamps, errors, warnings, information or trace data, by clicking the buttons in the panel’s header. Additionally, the **Clear All** button erases all info displayed in the **Output** panel.



**The Locals Panel** The **Locals** panel displays all the variables that are in the scope of the activity that is currently running. This panel is only visible at runtime or while debugging.



# 1.4Keyboard Shortcuts

# File Management

**Ctrl + Shift + N** - Creates a new **Blank Project**.

**Ctrl + O** - Enables you to open a previously created workflow. Only .xaml files are supported.

**Ctrl + L** - Opens the folder where the Log files are stored.

**Ctrl + S** - Saves the currently opened workflow.

**Ctrl + Shift + S** - Saves all the workflows that are currently open.

# Comment

**Ctrl + D** - Ignores the activity that is currently selected by placing it into a **Comment Out**container.

**Ctrl + E** - Removes the activity from the **Comment Out** container it was placed in.

# Debugging

**F7** - Runs the currently opened workflow in debug mode.

**F8** - Checks the currently opened workflow for validation errors.

**F9** - Marks the selected activity with a breakpoint.

**Shift + F9** - Removes all the breakpoints in the currently opened workflow.

**F11** - When debugging, enables you to step into a block of activities and executes the first one.

**Shift + F11** - When debugging, steps over the execution of a block of activities in the currently selected workflow.

# Recording

**Alt + Ctrl + W** - Opens the **Web Recording** toolbar.

**Alt + Ctrl + B** - Opens the **Basic Recording** toolbar.

**Alt + Ctrl + C** - Opens the **Citrix Recording** toolbar.

**Alt + Ctrl + D** - Opens the **Desktop Recording** toolbar.

**F2** - Adds delay during a recording activity.

**F3** - Lets you specify a custom recording region.

**F4** - Lets you choose the UI Framework to record, which can be **Default, AA,** and **UIA**.

# Workflow Execution

**F5** - Runs the workflow that is currently open.

**Pause** - Pauses the execution of the current workflow, in both normal and debug mode.

**F12** - Stops the execution of the current workflow, in both normal and debug mode.

# Selected Activity

**Ctrl + T** - Places the activity inside the **Try** section of a **Try Catch** activity.

**Ctrl + N** - Creates a new **Sequence Diagram** in the current project.

**Ctrl + C** - Copies the selected activity or activities to the clipboard.

**Ctrl + V** - Pastes the copied activity or activities inside the selected item.

# Miscellaneous

**F1** - Enables you to access a help topic associated with the currently selected element.

**Alt + Ctrl + F** - Sets the focus to the search box in the **Activities** panel to search for an activity.

**Ctrl + P** - Opens the **Manage Packages** window.

# Customizing Keyboard Shortcuts

All supported keyboard shortcuts are stored in an .xml file, which can be found at this location - %appdata%\UiPath\UiPath/keyboardmappings.xml.

Any keyboard shortcut can be changed by editing the .xml document in **Notepad** and modifying the values between <Key> </Key> and <Modifiers> </Modifiers>.

* <Key> </Key> - Represents the main keyboard button to press.
* <Modifiers> </Modifiers> - Represents special/modifier keys (Control, Alt, Shift, Windows).
* <CommandName> </CommandName> - The target command.
* <KeyboardMapping xmlns="http://schemas.datacontract.org/2004/07/Orchestra.Models">
* <CommandName>NewProjectCommand</CommandName>
* <InputGesture xmlns:ctl1="http://schemas.datacontract.org/2004/07/Catel.Windows.Input" graphid="6">
* <Key>N</Key>
* <Modifiers>Control Shift</Modifiers>
* </InputGesture>
* <Text IsNull="true" />
* <IsEditable>false</IsEditable>
* </KeyboardMapping>

## **Example of Changing a Keyboard Shortcut Binding**

1. Close Studio.
2. Navigate to the %appdata%\UiPath\UiPath/keyboardmappings.xml location.
3. Open the keyboardmappings.xml file with a text editor, such as Notepad.
4. Look for the keyboard shortcut you want to change.In this case, creating a new **Blank Project**:
5. Replace the values between <Key> </Key> and <Modifiers> </Modifiers> with the desired keys. For example <Key>S</Key> <Modifiers>Alt Shift</Modifiers>. The keyboard shortcut should look as in the following
6. screenshot.Save and close the keyboardmappings.xml file.

# 1.4 About Automation Projects

When you start a new project in Studio, a folder is created with your custom name to the selected location. Projects are saved in the %USERPROFILE%\Documents\UiPath directory by default.This folder includes:

* a Main.xaml file is automatically created, which should hold your main workflow.
* all other automation .xaml files included in the project. Please note that these files have to be linked to the Main.xaml through the [**Invoke Workflow File**](https://activities.uipath.com/docs/invoke-workflow-file) activity, because only the Main.xaml file is executed when you run a job.
* a .screenshots folder, if you are dealing with UI automation.
* a project.json file that contains information about your automation project.

# 1.5 Sequences

Sequences are the smallest type of project. They are suitable to linear processes as they enable you to go from one activity to another seamlessly, and act as a single block activity.One of the key features of sequences is that they can be reused time and again, as a standalone automation or as part of a state machine or flowchart.For example, you can create a sequence to take information from a .pdf file and add it to a spreadsheet, and reuse it in a different setting, while changing just a few properties.Note:Sequences do not use connectors.

# Example of a Sequence

To create a sequence that asks the user for his first and last name, and his hair color, and then displays his answers, do the following:

1. Create a blank project and, on the **Design** tab, in the **File** group, select **New > Sequence**. The **New Diagram** window is displayed.

### Note:You can also add a [**Sequence**](https://activities.uipath.com/docs/sequence) activity to the **Designer** panel to create a new sequence.

1. In the **Name** field type a name for the automation, such as "First Sequence," and click **Create**. The **Designer** panel is updated accordingly.
2. Create three **String** variables such as strFirstName, strLastName, and strHairColor, so that you can store data from the user in them. Leave the **Default** field empty, to indicate that there is no default value.
3. Drag three [**Input Dialog**](https://activities.uipath.com/docs/input-dialog) activities to the **Designer** panel, one under the other.
4. Select the first **Input Dialog** and, in the **Properties** panel, add a **Label** asking for the first name of the user, and a custom **Title**.
5. In the **Result** field add the strFirstName variable. This indicates that this variable is going to be updated with the value added by the user at this point.
6. Repeat steps 6 - 7 for the second and third **Input Dialog** activities to ask the user for his last name and hair color, and store them in the strLastName and strHairColor variables.
7. Add a [**Message Box**](https://activities.uipath.com/docs/message-box) activity under the third **Input Dialog**.
8. Select the **Message Box** and, in the **Properties** panel, in the **Text** field, add the variables and a string to enable you to display all information gathered from the user, such as:  
   strFirstName + " " + strLastName + " has " + strHairColor + " hair."

### Note: Remember to add spaces between variables and within strings for an optimal output.

The final project should look as in the following screenshot’

1. On the **Design** tab, in the **File** group, click **Run**. The automation is executed. The final output message should look as in the following screenshot.

# Flowcharts

Flowcharts can be used in a variety of settings, from large jobs to small projects that you can reuse in other projects.

The most important aspect of flowcharts is that, unlike sequences, they present multiple branching logical operators, that enable you to create complex business processes and connect activities in multiple ways.

# Example of a Flowchart

To exemplify the properties of a flowchart, we are going to build a guessing game that generates a random number from 1 to 999 that the user must guess. To create such an automation, do the following:

1. Create a blank project and from the **Design** tab, in the **File** group, select **New > Flowchart**. The **New Diagram** window is displayed.

### Note:

You can also add a [**Flowchart**](https://activities.uipath.com/docs/flowchart) activity to the **Designer** panel to create a new flowchart project.

1. In the **Name** field type a name for the automation, such as "First Flowchart," and click **Create**. The **Designer** panel is updated accordingly.
2. Create two **Int32** variables (intRandomNumber, intGuessNumber) and a **String** one (strMessage).
3. Set the default value of the strMessage variable to "Guess a number from 1 to 999." The intRandomNumber stores a random number between 1 and 999, intGuessNumber stores the user’s guess and strMessage stores the message that is going to be displayed to prompt the user.
4. Add an [**Assign**](https://activities.uipath.com/docs/assign) activity to the **Designer** panel, and connect it to the **Start** node.
5. In the **Properties** panel, in the **To** field add the intRandomNumber variable.
6. In the **Value** field, type new Random().Next(1,999).

### Note:

This field uses the Random() function to generate a random number between 1 and 999. For more information on variables, see [Variables](https://studio.uipath.com/docs/managing-variables).

1. Add an [**Input Dialog**](https://activities.uipath.com/docs/input-dialog) activity to the **Designer** panel and connect it to the **Assign** one.
2. In the **Properties** panel, in the **Label** field, add the strMessage variable.
3. In the **Result** field, add the intGuessNumber variable. This activity asks and stores the user’s guesses in the intGuessNumber variable.
4. Add a [**Flow Decision**](https://activities.uipath.com/docs/flow-decision) activity and connect it to the **Input Dialog**. This activity enables you to tell the user if he correctly guessed the number or not.
5. In the **Properties** panel, in the **Condition** field, type intGuessNumber = intRandomNumber. This enables you to verify if the number added by the user is the same as the randomly-generated one.
6. Add a [**Message Box**](https://activities.uipath.com/docs/message-box) activity and connect it to the **True** branch of the **Flow Decision**.
7. In the **Properties** panel, in the **Text** field, type "Congratulations! You guessed correctly! The number was " + intRandomNumber.ToString + ".". This is the message that is going to be displayed if the user correctly guessed the number.
8. Add a new **Flow Decision** activity and connect it to the **False** branch of the previously added **Flow** Decision.
9. In the **Properties** panel, in the **Condition** field, type intGuessNumber > intRandomNumber. This activity enables you to check if the number the user added is bigger than the randomly-generated one.
10. In the **DisplayName** field, type **Comparison**. This enables you to easily to tell the difference between the two **Flow Decisions** used.
11. Add an **Assign** activity and connect it to the **True** branch of the **Comparison** activity.
12. In the **To** field, type the strMessage variable, and in the **Value** field, type a message indicating that the guess was too high, such as "Too big. Try again.".
13. Select the **Assign** activity and press Ctrl+C. The entire activity and its properties are copied to the Clipboard.
14. Press Ctrl + V. A duplicate of the previous **Assign** activity is displayed.
15. Connect it to the **False** branch of the **Comparison** activity and, in the **Properties** panel, in the **Value** field, type "Too small. Try again.".
16. Connect the **Assign** activities created at steps 18-22 to the **Input Dialog**. A loop is created, asking the user to type a smaller or bigger number, until he guesses correctly.  
    The final project should look as in the screenshot below.

# State Machines

[**SUGGEST EDITS**](https://studio.uipath.com/docs/state-machines)

A state machine is a type of automation that uses a finite number of states in its execution. It can go into a state when it is triggered by an activity, and it exits that state when another activity is triggered.

Another important aspect of state machines are transitions, as they also enable you to add conditions based on which to jump from one state to another. These are represented by arrows or branches between states.

There are two activities that are specific to state machines, namely [**State**](https://activities.uipath.com/docs/state) found under **System > Activities > Statements**, and [**Final State**](https://activities.uipath.com/docs/final-state), under **Core > Presentation > Factories**.

### Note:

You can only create one initial state, yet it is possible to have more than one **Final State**.

The **State** activity contains three sections, **Entry**, **Exit** and **Transition(s)**, while the **Final State** only contains one section, **Entry**. Both of these activities can be expanded by double-clicking them, to view more information and edit them.

The **Entry** and **Exit** sections enable you to add entry and exit triggers for the selected state, while the **Transition(s)** section displays all the transitions linked to the selected state.

Transitions are expanded when you double-click them, just like the **State** activity. They contain three sections, **Trigger**, **Condition** and **Action**, that enable you to add a trigger for the next state, or add a condition under which an activity or sequence is to be executed.

# Example of How to Use a State Machine

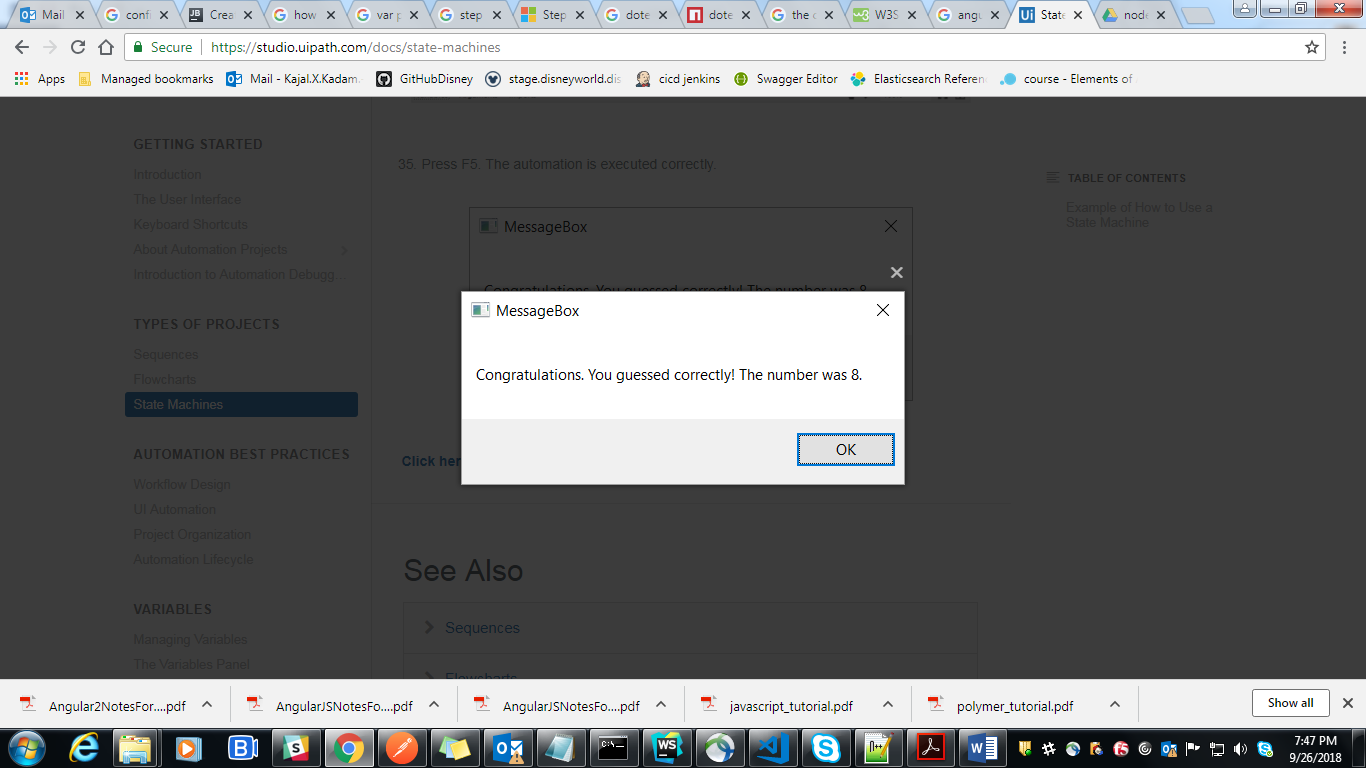
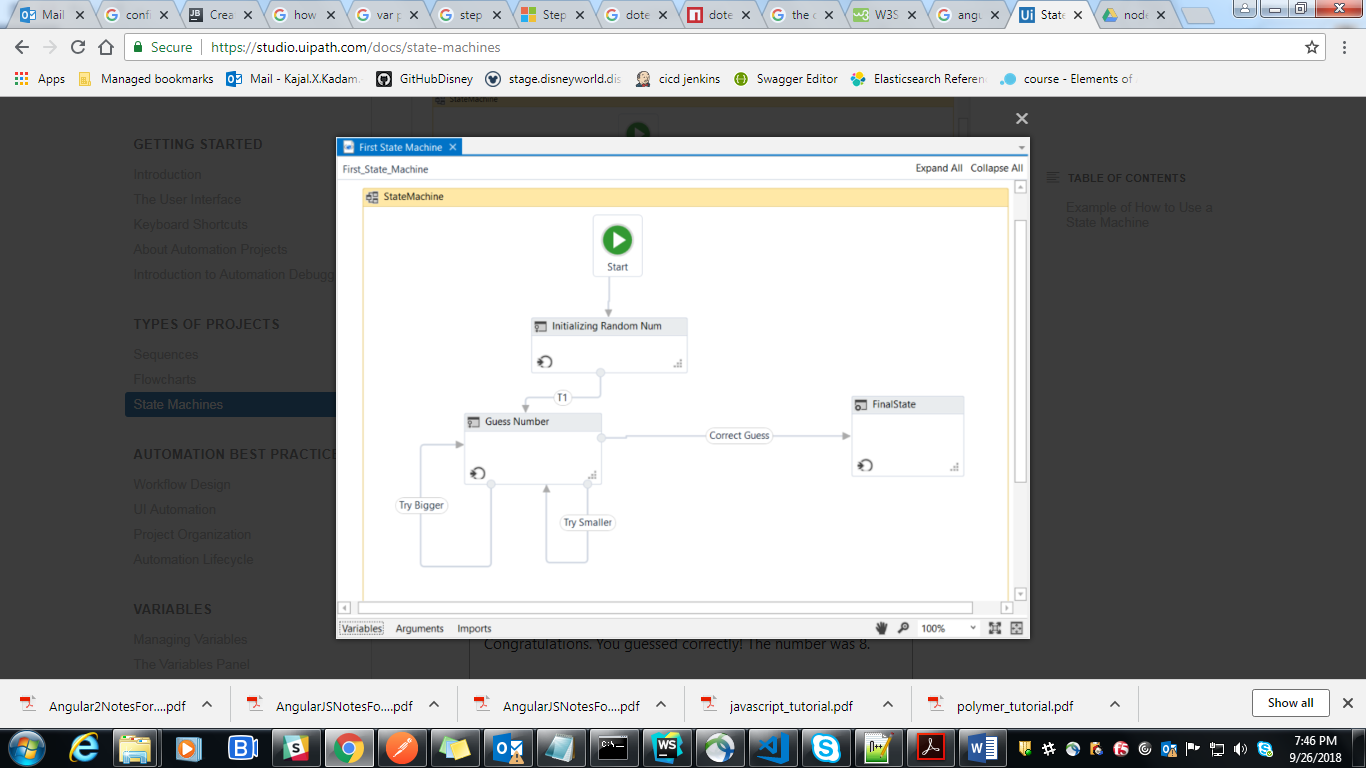
To exemplify how to use a state machine, we are going to build the guessing game we did in the previous chapter, the only difference being that we will try to guess a number between 1 and 100.

1. Create a blank project and, on the **Design** tab, in the **File** group, select **New > State Machine**. The **New Diagram** window is displayed.

### Note:

You can also add a [**State Machine**](https://activities.uipath.com/docs/state-machine) activity to the **Designer** panel to create a new state machine automation.

1. Create two integer variables, intGuess and intRandomNumber. The first variable stores your guess, while the second stores the random number.
2. Add a **State** activity to the **Designer** panel and connect it to the **Start** node. This is the initial state, and it is used to generate a random number.
3. Double-click the activity. This **State** activity is displayed expanded in the **Designer** panel.
4. In the **Properties** panel, in the **DisplayName** field, type Initializing Random Number. This enables you to easily tell states apart.
5. In the **Entry** section, add an [**Assign**](https://activities.uipath.com/docs/assign) activity.
6. In the **To** field, add the intRandomNumber variable.
7. In the **Value** field, type new Random().Next(1,100). This expression generates a random number.
8. Return to the main project view and add a new **State** activity.
9. Connect it to the previously added activity.
10. Double-click the last added **State** activity. This activity is displayed expanded in the **Designer** panel.
11. In the **Properties** panel, in the **DisplayName** field, type Guess Number. This state is used to prompt the user to guess a number.
12. In the **Entry** section, add an [**Input Dialog**](https://activities.uipath.com/docs/input-dialog) activity.
13. Select the **Input Dialog**, and in the **Properties** panel, add an appropriate **Label** and **Title**to prompt the user to guess a number between 1 and 100.
14. In the **Result** field, add the intGuess variable. This variable stores the user’s guess.
15. Return to the main project view and create a transition that points from the Guess Number state to itself.
16. Double-click the transition. The transition is displayed expanded in the **Designer** panel.
17. In the **Properties** panel, in the **DisplayName** field, type Try Smaller. This message is displayed on the arrow, enabling you to run through your automation easier.
18. In the **Condition** section, type intGuess > intRandomNumber. This verifies if the user’s guess is bigger than the random number.
19. In the **Action** section, add a [**Message Box**](https://activities.uipath.com/docs/message-box) activity.
20. In the **Text** field, type something similar to "Your guess is too big. Try a smaller number." This message is displayed when the user’s guess is bigger than the random number.
21. Return to the main project view and create a new transition that points from the **Guess Number** state to itself
22. Double-click the transition. The transition is displayed expanded in the **Designer** panel.
23. In the **Properties** panel, in the **DisplayName** field, type "Try Bigger". This message is displayed on the arrow, enabling you to run through your automation easier.
24. In the **Condition** section, type intGuess < intRandomNumber. This verifies if the guess is smaller than the random number.
25. In the **Action** section, add a **Message Box** activity.
26. In the **Text** field, type something similar to "Your guess is too small. Try a bigger number." This message is displayed when the users guess is smaller than the random number.
27. Return to main project view and add a **Final State** activity to the **Designer** panel.
28. Connect a transition from the **Guess Number** activity to the **Final State**.
29. In the **Properties** panel, in the **DisplayName** field, type "Correct Guess".
30. In the **Condition** field, type intGuess = intRandomNumber. This is the condition on which this automation steps to the final state and end.
31. Double-click the **Final State** activity. It is displayed expanded in the **Designer** panel.
32. In the **Entry** section, add a **Message Box** activity.
33. In the **Text** field, type something similar to "Congratulations. You guessed correctly! The number was " + intRandomNumber.ToString + "." This is the final message that is to be displayed, when the user correctly guesses the number.  
    The final project should look as in the following screenshot.
34. Press F5. The automation is executed correctly.



# Workflow Design

[**SUGGEST EDITS**](https://studio.uipath.com/docs/workflow-design)

# Layout Diagrams

UiPath offers three diagrams for integrating activities into a working structure when  
developing a workflow file:

* **Flowchart**
* **Sequence**
* **State Machine**

## **Sequence**

Sequences have a simple linear representation that flows from top to bottom and are best suited for simple scenarios when activities follow each other. For example, they are useful in UI automation, when navigation and typing happens one click/keystroke at a time. Because sequences are easy to assemble and understand they are the preferred layout for most workflows.

## **Flowchart**

Flowcharts offer more flexibility for connecting activities and tend to lay out a workflow in a plain two-dimensional manner. Because of its free form and visual appeal, flowcharts are best suited for showcasing decision points within a process. Arrows that can point anywhere closely resemble the unstructured [**GoTo** programming statement](https://en.wikipedia.org/wiki/Goto) and therefore make large workflows prone to chaotic interweaving of activities.

## **State Machine**

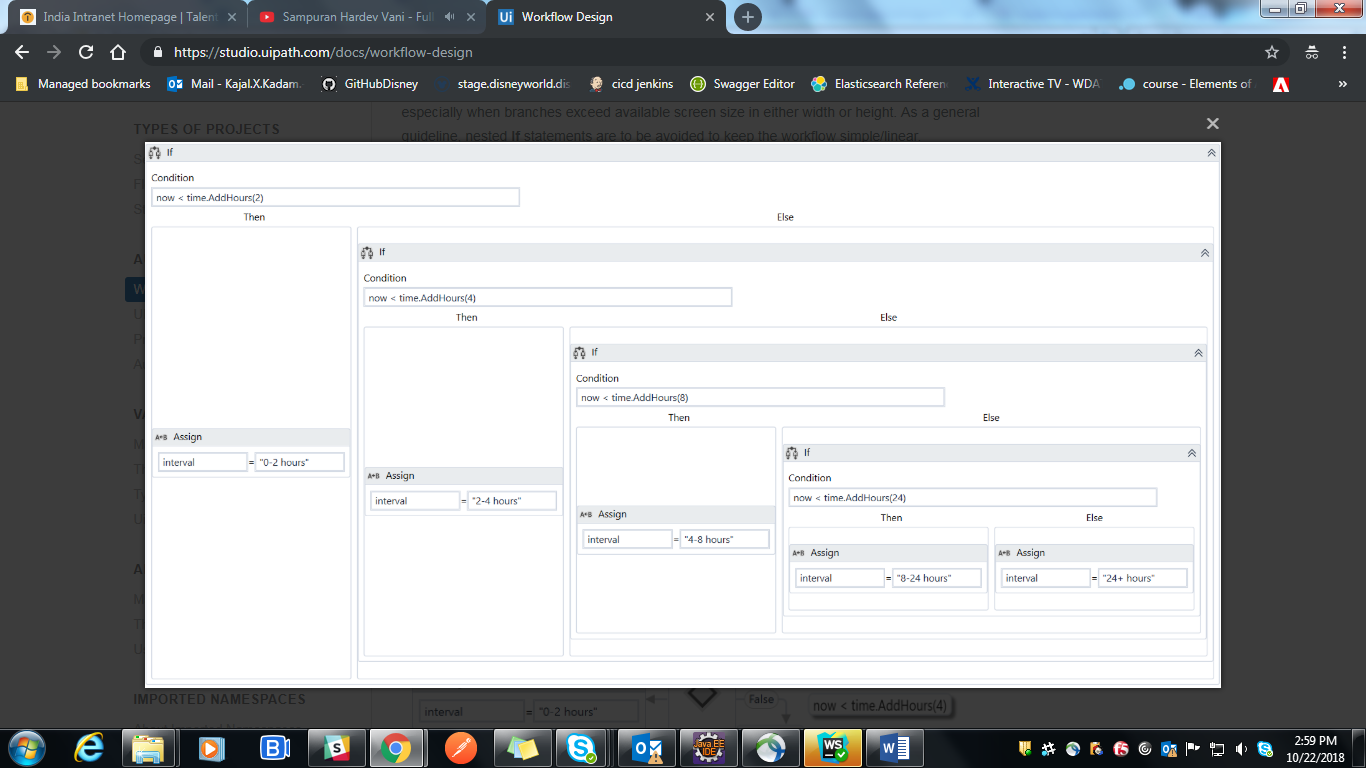
State Machine is a rather complex structure that can be seen as a flowchart with conditional arrows, called transitions. It enables a more compact representation of logic and we found it suitable for a standard high-level process diagram of transactional business process templates.

# Choices

Decisions need to be implemented in a workflow to enable the Robot to react differently in various conditions in data processing and application interaction. Picking the most appropriate representation of a condition and its subsequent branches has a big impact on the visual structure and readability of a workflow.

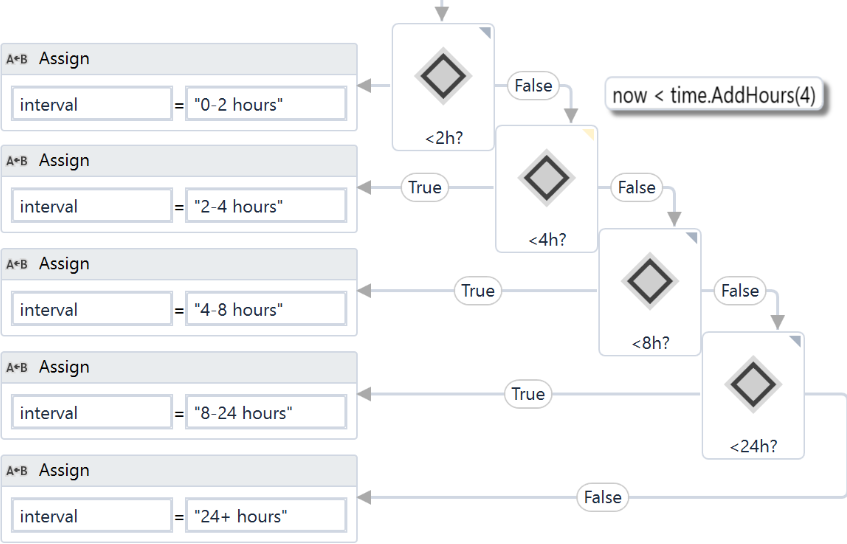
## **If Activity**

The **If** activity splits a sequence vertically and is perfect for short balanced linear branches. Challenges come when more conditions need to be chained in an **If… Else If** manner, especially when branches exceed available screen size in either width or height. As a general guideline, nested **If** statements are to be avoided to keep the workflow simple/linear.



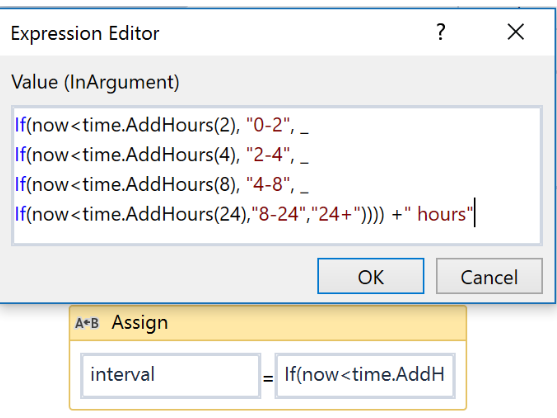
## **Flow Decision**

Flowchart layouts are good for showcasing important business logic and related conditions like nested **If** statements or **If… Else If** constructs. There are situations where a Flowchart may look good even inside a Sequence.



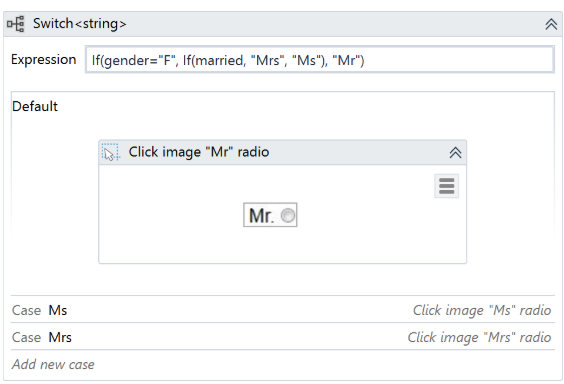
## **If Operator**

The [VB **If** operator](https://docs.microsoft.com/en-us/dotnet/visual-basic/language-reference/operators/if-operator) is very useful for minor local conditions or data computing, and it can sometimes reduce a whole block to a single activity.



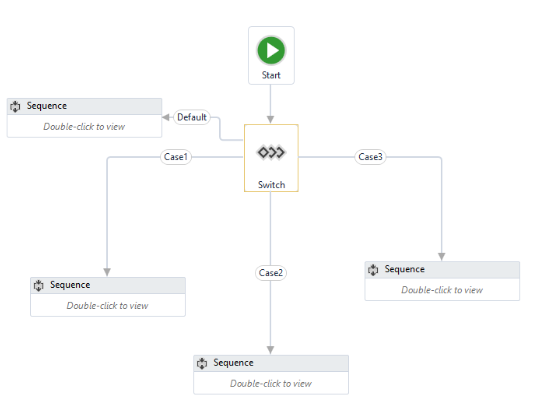
## **Switch Activity**

The **Switch** activity may be sometimes used in convergence with the **If** operator to streamline and compact an **If… Else If** cascade with distinct conditions and activities per branch.



## **Flow Switch**

The **Flow Switch** activity selects the next node depending on the value of an expression; **Flow Switch** can be seen as the equivalent of the procedural **Switch** activity in flowcharts. It can match more than 12 cases by starting more connections from the same switch node.



# Data

Data comes in two flavors when it comes to visibility and life cycle: arguments and variables. While the purpose of arguments is to pass data from one workflow to another, variables are bound to a container inside a single workflow file and can only be used locally.

## **Variable Scope**

Unlike arguments, which are available everywhere in a workflow file, variables are only visible inside the container where they are defined, called scope.

Variables should be kept in the innermost scope to reduce the clutter in the **Variables** panel and to show only, in autocomplete, what is relevant at a particular point in the workflow.

### Note:

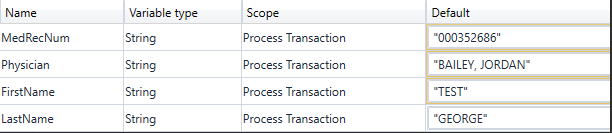
If two variables with the same name exist, although we highly recommend against it, the one defined in the most inner scope has priority.

## **Arguments**

Keep in mind that when invoking workflows with the **Isolated** option (which starts running the workflow in a separate [system process](https://en.wikipedia.org/wiki/Process_(computing))), only serializable types can be used as arguments to pass data from a process to another. For example, **SecureString**, **Browser** and **Terminal Connection** objects cannot safely cross the inter-process border.

## **Default Values**

Variables and input arguments have the option to be initialized with some default static values. This comes in very handy when testing workflows individually, without requiring real input data from calling workflows or other external sources.



# Naming Conventions

Meaningful names should be assigned to workflow files, activities, arguments, and variables in order to accurately describe their usage throughout the project.

Firstly, projects should have meaningful descriptions, as they are also displayed in the Orchestrator user interface and might help in multi-user environments.

Only argument names are case sensitive, but to improve readability, variables should also align to a naming convention:

* Variables should be upper CamelCase, such as FirstName, LastName.
* Arguments should be in upper CamelCase with a prefix stating the argument type, such as in\_DefaultTimeout, in\_FileName, out\_TextResult, io\_RetryNumber.
* Activity names should concisely reflect the action taken, such as Click the **Save** Button. Keep the part of the title that describes the action (**Click**, **Type Into**, **Element Exists**, etc.).
* Except for Main, all workflow names should contain the verb describing what the workflow does, such as GetTransactionData, ProcessTransation, TakeScreenshot.

# Comments and Annotation

The **Comment** activity and **Annotations** should be used to describe in more detail a technique or the particularities of a certain interaction or application behavior. Keep in mind that other people may, at some point, come across a robotic project and you can try to ease their understanding of the process.

# UI Automation

[**SUGGEST EDITS**](https://studio.uipath.com/docs/ui-automation)

Sometimes the usual manual routine is not the optimal way for automation. Carefully explore the application’s behavior and UiPath’s integration/features before committing to a certain approach.

### Important! Using the [Parallel](https://docs.microsoft.com/en-us/dotnet/api/system.activities.statements.parallelforeach-1?redirectedfrom=MSDN&view=netframework-4.7.2) activity with UI Automation activities is not supported and will often result in unforeseen consequences.

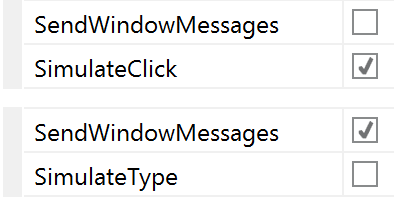
# Desktop Automation

UI automation is best utilized when Robots and applications run on the same machine because UiPath can integrate directly with the technology behind the application to identify elements, trigger events and get data behind the scenes.

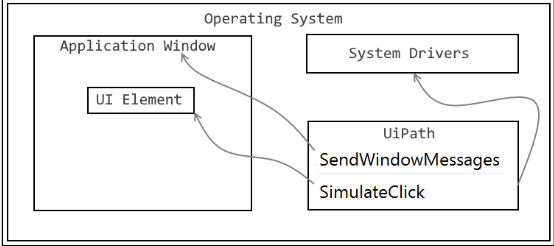
## **Input Methods**

There are three methods UiPath uses for triggering a **Click** or a **Type Into** activity on an application. These are displayed as properties in all activities that deal with UI automation.

* If the **SimulateType** or **SimulateClick** properties are selected, Studio hooks into the application and triggers the event handler of an indicated UI element (button, text box).
* If the **SendWindowMessages** property is selected, Studio posts the event details to the application message loop and the application’s window procedure dispatches it to the target UI element internally.



* Studio signals system drivers with hardware events if none of the above option are selected and lets the operating system dispatch the details towards the target element.



These methods should be tried in the order presented, as the **SimulateClick** and **WindowMessages** properties are faster and also work in the background, but they depend mostly on the technology behind the application.

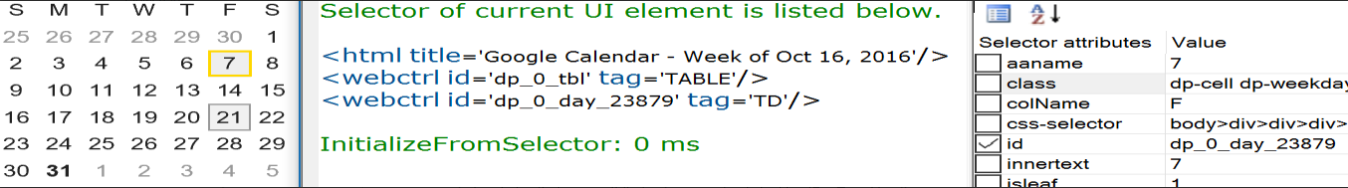
Hardware events work 100% as Studio performs actions just like a human operator, such as moving the mouse pointer and clicking at a particular location. However, in this case, the application being automated needs to be visible on the screen. This can be seen as a drawback since there is the risk that the user can interfere with the automation.

## **Selectors**

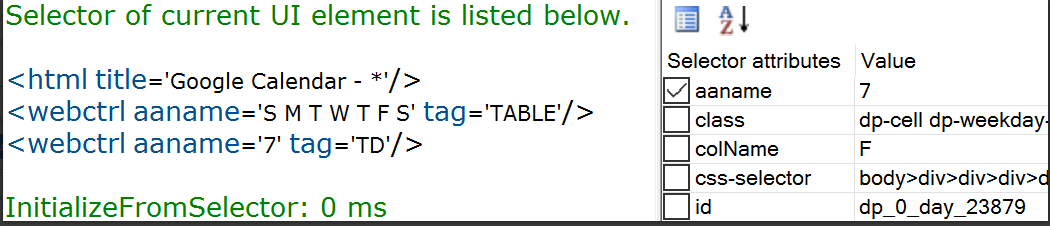
Sometimes the automatically generated selectors propose volatile attribute values to identify elements and manual intervention is required to calibrate the selectors. A reliable selector should successfully identify the same element every time in all conditions, in development, test and production environments and no matter the usernames logged on to the applications.

Here are some tips on how to improve a selector in the [**Selector Editor**](https://studio.uipath.com/v2018.3/docs/about-selectors) or [**UI Explorer**](https://studio.uipath.com/v2018.3/docs/uipath-explorer):

* Replace attributes with volatile values with attributes that look steady and meaningful.
* Replace variable parts of an attribute value with wildcards (\*).
* If an attribute’s value is all wildcard (e.g. name=’\*’) then the attribute should be removed.
* If editing attributes doesn’t help, try adding more intermediary containers.
* Avoid using the idx attribute unless its value is a very small number like 1 or 2.



In the selector above, we notice the page title has a reference to the time when the selector was recorded and also that some attributes have randomly looking IDs. Tweaking the attributes, we can come up with a better selector than UiPath recorder proposed.



## **Containers**

Similar to file paths, selectors can be full or partial (relative). Full selectors start with a window or an HTML identifier and have all the necessary information to find an element on the whole desktop, while partial selectors work only inside an attach/container that specifies the top-level window where the elements belong:

* **OpenBrowser**
* **OpenApplication**
* **AttachBrowser**
* **AttachWindow**

Here is a bit more info on [Full versus Partial Selectors](https://studio.uipath.com/docs/full-versus-partial-selectors).

There are several advantages to using containers with partial selectors instead of full selectors:

* Visually groups activities that work on the same application.
* Is slightly faster, not seeking for the top window every time.
* Makes it easier to manage top-level selectors in case manual updates are necessary.
* Essential when working on two instances of the same application.

# Image Automation

Image recognition is the last approach to automating applications if nothing else works to identify UI elements on the screen (like selectors or keyboard shortcuts). Because image matching requires elements to be fully visible on the screen and that all visible details are the same at runtime as during development, when resorting to image automation extra care should be taken to ensure the reliability of the process. Selecting more/less of an image than needed might lead to an image not found or a false-positive match.

## **Resolution Considerations**

Image matching is sensitive to environment variations like desktop theme or screen resolution. When the application runs in Citrix, the resolution should be kept greater or equal than when recording the workflows. Otherwise, small image distortions can be compensated by slightly lowering the captured image Accuracy factor. Check how the application layout adjusts itself to different resolutions to ensure visual elements proximity, especially in the case of coordinate based techniques like relative click and relative scrape.

If the automation supports different resolutions, parallel recordings can be placed inside a **PickBranch** activity and the Robot uses either match.

## **OCR Engines**

If OCR returns good results for the application, text automation is a good alternative to minimize the environment influence. The Google Tesseract engine works better for smaller areas, while Microsoft MODI for larger ones.

Using the MODI engine in loop automations can sometimes create memory leaks. This is why it is recommended that scraping done with MODI be invoked via a separate workflow, using the **Isolated** property.

# UI Synchronization

Unexpected behavior is likely to occur when the application is not in the state the workflow assumes it to be. The first thing to watch for is the time the application takes to respond to the Robot interactions.

The **DelayMS** property enables you to wait a while for the application to respond. However, there are situations when an application’s state must be validated before proceeding with certain steps in a process. Measures may include using extra activities that wait for the desired application state before other interactions. Activities that might help include:

* [**Element Exists**](https://activities.uipath.com/docs/ui-element-exists), [**Image Exists**](https://activities.uipath.com/docs/image-found), [**Text Exists**](https://activities.uipath.com/docs/text-exists), [**OCR Text Exists**](https://activities.uipath.com/docs/ocr-text-exists).
* [**Find Element**](https://activities.uipath.com/docs/wait-ui-element-appear), [**Find Image**](https://activities.uipath.com/docs/wait-image-appear), [**Find Text Position**](https://activities.uipath.com/docs/find-text).
* [**Wait Element Vanish**](https://activities.uipath.com/docs/wait-ui-element-vanish), [**Wait Image Vanish**](https://activities.uipath.com/docs/wait-image-vanish).
* [**Wait Screen Text**](https://activities.uipath.com/docs/terminal-wait-screen-text) (in terminals).

# Background Automation

If an automation is intended to share the desktop with a human user, all UI interaction must be implemented in the background. This means that the automation has to work with UI element objects directly, thus allowing the application window to be hidden or minimized during the process.

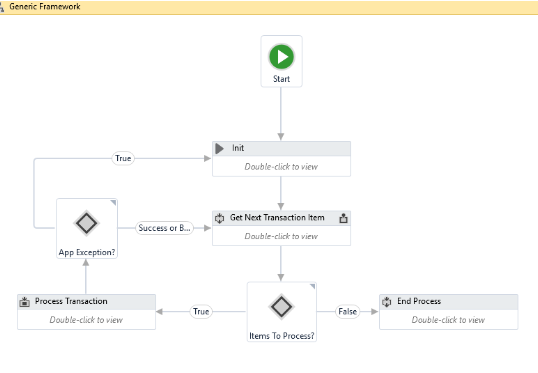
* Use the **SimulateType**, **SimulateClick**, and **SendWindowMessagesoptions** for navigation and data entry via the [**Click**](https://activities.uipath.com/docs/click) and [**Type Into**](https://activities.uipath.com/docs/type-into) activities.
* Use the [**Set Text**](https://activities.uipath.com/docs/set-value), [**Check**](https://activities.uipath.com/docs/check), and [**Select Item**](https://activities.uipath.com/docs/select-item) activities for background data entry.
* The [**Get Text**](https://activities.uipath.com/docs/get-value), [**Get Full Text**](https://activities.uipath.com/docs/get-full-text), and **WebScraping** activities are the outputs that run in the background.
* Use the [**Element Exists**](https://activities.uipath.com/docs/ui-element-exists) activity to verify application state.

# Project Organization

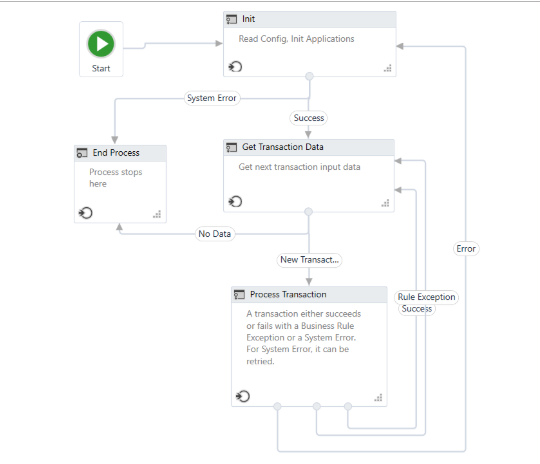
[**SUGGEST EDITS**](https://studio.uipath.com/docs/project-organization)

# High-Level Frameworks

Starting from a generic (and process agnostic) framework ensures you deal in a consistent and structured way with any process. A framework helps you start with the high-level view, then you go deeper into the specific details of each process.

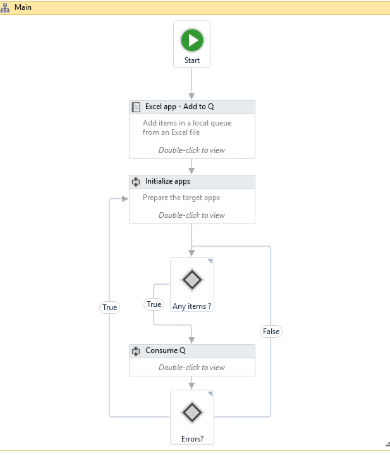


The **Robotic Enterprise Framework** Template proposes a flexible high-level overview of a repetitive process and includes a good set of practices described in this guide and can easily be used as a solid starting point for RPA development with UiPath. The template is built on a [**State Machine**](https://studio.uipath.com/v2018.3/docs/state-machines) structure.

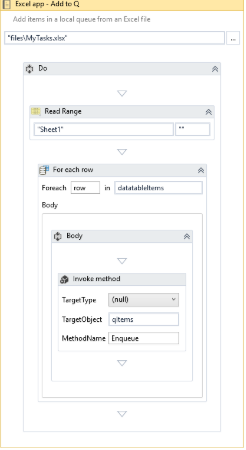


How it works:

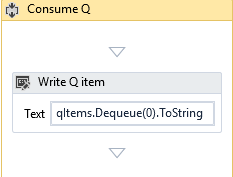
* The Robot loads settings from the config file and Orchestrator assets, keeping them in a dictionary that is to be shared across workflows.
* The Robot fetches the required credentials and logs into all applications.
* It retries a few times if any errors are encountered, then succeeds or aborts.
* The Robot checks the input queue or other input sources to start a new transaction.
* If no (more) input data is available, configure the workflow to either wait and retry, or end the process.
* The UI interactions to process the transaction data are executed.
* If the transactions are processed successfully, the transaction status is updated and the Robot continues with the next transaction.
* If any validation errors are encountered, the transaction status is updated and the Robot moves to the next transaction.
* If any exceptions are encountered, the Robot either retries to process the transaction a few times (if configured), or it marks the item as a failure and restarts.
* At the end, an email is sent with the status of the process, if configured.



For transaction-based processes (such as processing all the invoices from an Excel file) which are not executed through Orchestrator, local queues can be built ([**using .NET enqueue/ dequeue methods**](https://msdn.microsoft.com/en-us/library/t249c2y7(v=vs.110).aspx)).



Then, the flow of the high-level process (exception handling, retrial, recovery) could be easily replicated - easier than by having the entire process grouped under a **For Each Row** loop.



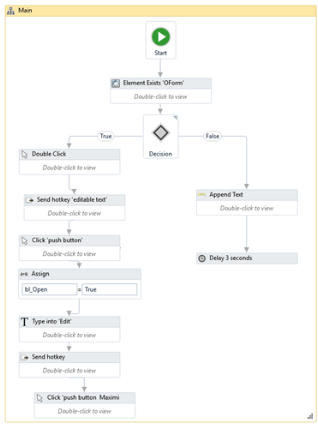
# Design Principles

**Breaking the process in smaller workflows** is paramount to good project design. Dedicated workflows allow independent testing of components while encouraging team collaboration by developing working on separate files.

Choose the layout type wisely (flowcharts and sequences). Normally the **logic** of the process stays in flowcharts while the **navigation and data processing** is in sequences.

By developing complex logic within a sequence, you will end up with a labyrinth of containers and decisional blocks, very difficult to follow and update.

On the contrary, UI interactions in a flowchart make it more difficult to build and maintain.



Project related files (such as email templates) could be organized in local folders or shared drives.

### Note:

If placed inside the project folder, they are replicated during the deployment process (together with the projects workflows) on all the Robot machines in the lib/net45folder.

These folders could be also stored on a shared drive, so all the Robots connect to the same unique source. This way, the process related files could be checked and maintained by the business users entirely, without support from the RPA team. However, the decision (shared or local folders) is complex and should take into consideration various aspects related to the process and environment: size of the files, frequency of changes, concurrency for editing the same file, security policies, etc.

# Source Control

In order to easily manage project versioning and share the work on more developers, we recommend using a Version Control System. UiPath Studio is directly integrated with **TFS** and **SVN**. You can find a tutorial explaining the connection steps and functionalities [here](https://studio.uipath.com/v2018.3/docs/connecting-your-project-to-a-source-control).

# Control Settings

To avoid hard-coding external settings (like file paths, URLs, etc.) in the workflows, we recommend keeping them in a .config file (.xlsx, .xml, or .json) or in Orchestrator, as assets, if they change often.

Generally speaking, the final solution should be extensible in order to allow variation and changes in input data without developer intervention. For example, lists with customers that are allowed a certain type of transaction, emails of people to receive notifications, etc. should be stored in external files (like Excel) where business people or other departments can alter them directly (add/remove/update).

For any repetitive process, all workflow invocations from the main loop should be marked with the **Isolated** option to defend against potential Robot crashes (such as Out of memory).

# Credentials

No credentials should be stored in the workflow directly, but rather loaded from safer places like local Windows Credential Store or Orchestrator assets. You can use them in workflows via the **GetCredential** activity.

# Error Handling

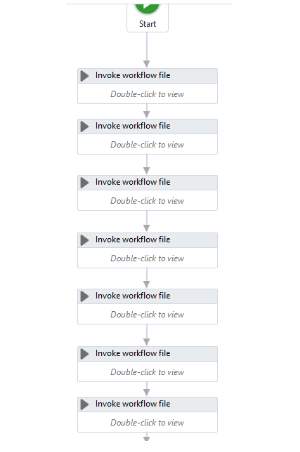
Two types of exceptions may happen when running an automated process: somewhat predictable or totally unexpected. Based on this distinction there are two ways of addressing exceptions, either by explicit actions executed automatically within the workflow or by escalating the issue to human operators.

Exception propagation can be controlled by placing susceptible code inside **Try/Catch** blocks where situations can be appropriately handled. At the highest level, the main process diagram must define broad corrective measures to address all generic exceptions and to ensure system integrity.

Contextual handlers offer more flexibility for Robots to adapt to various situations and they should be used for implementing alternative techniques, cleanup or customization of user/log messages. Take advantage of the vertical propagation mechanism of exceptions to avoid duplicate handlers in catch sections by moving the handler up some levels where it may cover all exceptions in a single place.

Enough details should be provided in the exception message for a human to understand it and take the necessary actions. Exception messages and sources are essential. The source property of the exception object indicates the name of the activity that failed (within an invoked workflow). Again, naming is vital, because poor naming gives no clear indication about the component that crashed.

As you can see below, choosing not to rename the **Invoke** activity makes the exception source meaningless in case of a crash (such as **Invoke Workflow File > Invoke Workflow File > Invoke Workflow File > Type Into**).



# Keep it Clean

In the process flow, make sure you close the target applications (browsers, apps) after the Robots interact with them. If left open, they use the machine resources and may interfere with the other steps of automation.

Before publishing the project, take a final look through the workflows and do some clean-up:

* Remove unreferenced variables.
* Delete temporary **Write Line** outputs.
* Delete disabled code.
* Make sure the naming is meaningful and unique.
* Remove unnecessary containers (**Right-click > Remove Sequence**).

The project name is also important. This is how the process is seen in Orchestrator, so it should be in line with your internal naming rules. By default, the project ID is the initial project name, but you can modify it from the project.json file.

The description of the project is also important (it is visible in Orchestrator). It might help you differentiate easier between processes, so choose a meaningful description as well.

# Code Reusability

When developing, we often need to automate the same steps in more than one workflow/project, so it should be a common practice to create workflows that contain small pieces of occurring automation and add them to the Library.

There is no universal recipe that tells you how to split any given process.

However, separation of business logic from the automation components is a good principle that helps with building a code that can be reused effectively.

## **Example**

Let’s assume that a part of your process requires reading the customer info, then, based on that info and internal business rules, update the customer details.

**Get Customer Info** and **Change Customer Info** should be two distinct automation components, completely agnostic of any process. The logic (update the customer type only when the total amount is greater than 100k in the last 12 months) should be kept separated from automation. Both components could be used later, separately, in the same project or in a different one, with a different logic. If needed, specific data could be sent to these components through arguments.

**Change Customer Info** should **not** be invoked from within **Get Customer Info**, as this makes it more difficult to test, handle exceptions, and reuse.

### Recommended:

Use separate components for **Get Info** and **Change Info**.



When the separation between actions is not that obvious, copy-pasting existing code from one workflow to another (or from one project to another) is also a good indication that you should build a separate component (workflow) for the code and invoke it when needed.

## **Where to Store Reusable Components**

Dragging and dropping existing code from the Library to a workflow is easier than recreating the code from scratch, again and again. Dealing with data (sorting, filtering) or with text (splitting, Regex patterns) are examples of what could be added to the sample library. Please take into account that once the code is added to the workflow, it becomes static, so if you update the workflow in the Library, it won’t be reflected in the existing live processes.

Common (reusable) components (such as App Navigation, Log In, Initialization) are better stored and maintained separately, on network shared drives. From that drive, they can be invoked by different Robots, from different processes. The biggest advantage of this approach is that any change made in the master component is reflected instantly in all the processes that use it.

# How to Check Quality Automation

* Modularity:
  + Separation of concerns with dedicated workflows allows fine granular development and testing;
  + Extract and share reusable components or workflows between projects.
* Maintainability:
  + Good structure and development standards.
* Readability:
  + Standardized process structure encouraging clear development practices;
  + Meaningful names for workflow files, activities, arguments, and variables.
* Flexibility:
  + Keep environment settings in external configuration files or Orchestrator instances, making it easy to run automation in both testing and production environments.
* Reliability:
  + Exception handling and error reporting;
  + Real-time execution progress update.
* Extensible:
  + Ready for new use cases to be incorporated.

# Automation Lifecycle

[**SUGGEST EDITS**](https://studio.uipath.com/docs/automation-lifecycle)

# Process Understanding

Deciding between an automation for Attended Robots or Unattended Robots is the first important decision that impacts how developers build the code, as the general running framework (Robot triggering, interaction, exception handling) is different. Switching to the other type of Robots later may be cumbersome.

For time-critical, live, humanly triggered processes, like in a call center an Attended Robot working side by side with a human might be the only possible answer.

But not all processes that need human input are supposed to run with Attended Robots. If a purely judgmental decision (not rule-based) during the process could not be avoided, evaluate if a change of flow is possible, like splitting the bigger process into two smaller sub-processes, when the output of the first sub-process becomes the input for the second one.Although human intervention takes place in between, such as validating/modifying the output of the first sub-process, both sub-processes could be triggered automatically and run unattended.

A typical case would be a process that requires a manual step somewhere during the process, such as checking the unstructured comments section of a ticket and, based on that, assign the ticket to certain categories.

Generally speaking, going with an Unattended Robot ensures a more efficient usage of the Robot load and a higher ROI, a better management and tracking of robotic capacities.

But these calculations should take into consideration various aspects, such as the fact that an Attended Robot could usually only run in the normal working hours of a person, or it may keep the machine and user busy until the execution is finished. Input types, transaction volumes, time restrictions, the number of Robots available, and others also play a role in this decision.

# Documenting the Process - DSD

The process documentation guides the developer's work and provides help in tracking the requests and the application maintenance. Of course, there might be lots of other technical documents, but one is critical for a smooth implementation, namely **DSD** (Development Specification Document).

The Development Specification Document should contain the automated process details and focus on two main categories: **Runtime Guide** and **Development Details**.

The **Runtime Guide** should contain a high-level runtime diagram, as well as details about the functionality of the Robot, such as sub-processes, schedules, configuration settings, input files, output files, temporary files, and performed actions. Additional details about the master process should be specified, such as prerequisites, automatic and manual error handling, process resuming in case of failure, Orchestrator usage, logging and reporting, credential management, and any other relevant information related to security or function.

The **Development Details** should contain information about the packages in use, the development environment, the logging level, the source code repository and versioning, a list of workflow components with their description and argument list, a list of reusable components, the workflow invoke tree, defined custom logs and log fields, relevant snapshots of the process flowchart, the level of background versus foreground automation, and any other relevant or outstanding development items.

# Development and Code Review

The RPA Solution Architect is responsible for continuously coaching developers on the best practices. Hence, frequent and thorough code reviews are a must, to enforce a very high quality of the developed workflows. This way, the developers are motivated to build robust workflows and to follow the best practices guide.

# Test

After each component is built, unit testing should be conducted. If every component is thoroughly tested, the integration runs more smoothly, and debugging lasts for a shorter period of time. The **REFramework** contains a **Test\_Framework** folder where all the test files should be placed. Using the RunAllTests.xaml file, a developer can test a sequence containing a lot of .xaml files automatically, thus being able to try out small integrations between components and to run stress tests. A report is generated at the end of each test. Typically, these kinds of tests should be executed outside of office hours, in testing environments, to optimize the developer’s time.

The recommended UiPath architecture includes **Dev** and **Test** environments that allow the processes to be tested outside the live production systems.

Sometimes applications look or behave differently between the **Dev**, **Test**, or **Production**environments and extra measures must be taken, sanitizing selectors or even conditional execution of some activities.

Use the UiPath.config file or Orchestrator assets to switch flags or settings for the current environment. A test mode parameter (Boolean) could be checked before interacting with live applications. This could be received as an asset (or argument) input. When it is set to **True**, during debug and integration testing, it follows the test route, does not execute the case fully. For example, the test patch can skip sending notifications, skips the **OK** or **Save** button or press the **Cancel** or **Close** button instead. When set to **False**, the normal **Production** mode route is followed.

This allows you to make modifications and test them in processes that work directly in live systems.

# Release

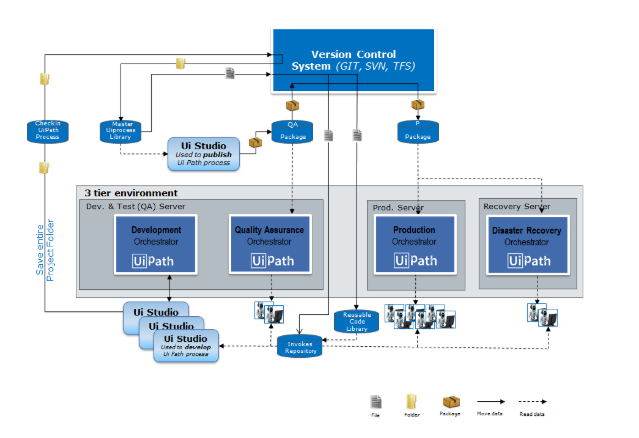
There are various ways of designing the architecture and release flow, considering the infrastructure setup, concerns about the segregation of roles, etc.

In this proposed model UiPath developers can build their projects and test them on **Development** environments in Orchestrator. They are allowed to check in the project to a drive managed by a version control system (VCS), such as GIT, SVN, or TFS.

Publishing the package and making it available for **Test** and **Production** environments is the work of a different team, such as IT.

The deployment paths on Orchestrator have been changed from their default to folders managed by the **VCS**, by changing the **NuGet.Packages.Path** value in web.config file in the **UiPath.Server.Deployment** section.

The model also contains a repository of reusable components.

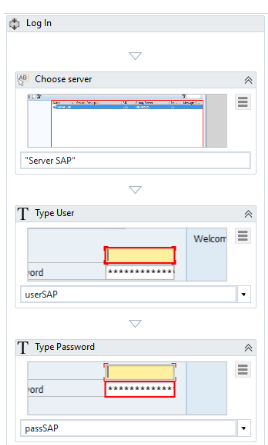


Here is the project publishing flow, step by step:

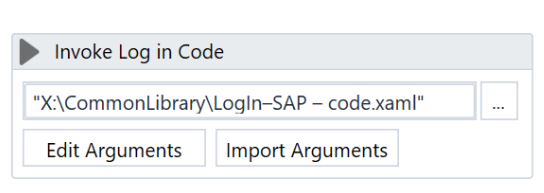
1. Developers build the process in Studio and test it with the **Development** environment in Orchestrator. Once done, they check in the workflows (not packaged) to a **Master UiProcess** Library folder (on **VCS**).
2. The IT team creates the package for QA. This is stored on a **QA Package** folder on **VCS**, from which QA runs the process on dedicated machines.
3. If any issue is revealed during the tests, the steps above are repeated.
4. Once all QA tests are passed, the package is copied to a **Production** environment (**P Package**).
5. The Process is going live and is run by the production Robots.

**Reusable Content** is created and deployed separately, as UiPath code (**Reusable Code Library**) and **Invokes** (**Invokes Repository**).

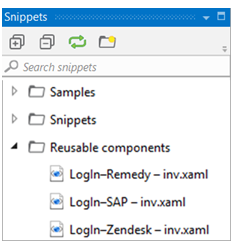
Workflows with source code are .xaml files containing activities for automating common processes, such as Lon in SAP:



**Invokes** represent workflows composed only of one **Invoke** activity of the code workflows mentioned above.



The Snippet panel of a Studio developer should point to this **Invoke** repository in order to provide easy access (drag & drop) to **Reusable Content**.



The local design authority in charge with maintaining the **Reusable Content** updates (due to a change in process, for instance) the workflows with code. The **Invokes** remain unchanged.

The advantage of this approach (as opposed to working directly with the library of source code) is that when a change is done to a reusable component, all the running projects reflect this change as well, as they only contain an **Invoke** of the changed workflow.

# Monitoring

Using **Log Message** activities to trace the evolution of a running process is essential for supervising, diagnosing and debugging a process. Messages should provide all relevant information to accurately identify a situation, including the transaction ID and state.

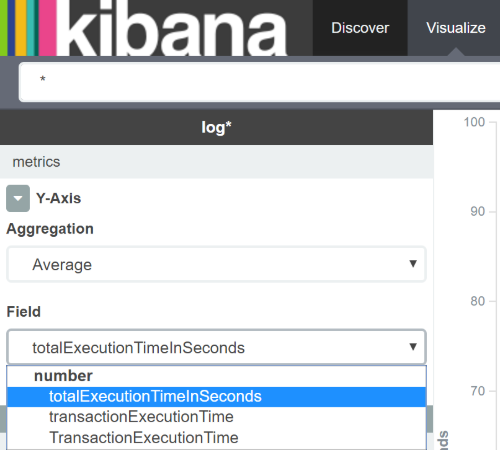
Logging should be used:

* At the beginning and the end of every workflow;
* When data is coming in from external sources;
* Each time an exception is caught at the highest level.

Messages are sent with the specified priority (e.g. **Info, Trace, Warning**) to Orchestrator and are also saved in the local .nlog file.

## **Custom Log Fields**

To make data easily available in Kibana for reporting purposes, the Robot may tag log messages with extra values using the **Add Log Fields** activity. By default, any UiPath log output has several fields already, including message, timestamp, level, processName, fileName, and the Robot’s Windows Identity. Log Fields are persistent so if we do not need to mark all messages with a tag, fields should be removed immediately after logging, using the **Remove Log Fields** activity. **Do not use a field name that already exists.** It’s important to specify the proper type of argument the first time when you add the field. This is how Elasticsearch indexes it.



# About Debugging

[**SUGGEST EDITS**](https://studio.uipath.com/docs/about-debugging)

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.

Before attempting to debug a workflow, it’s a good idea to validate it by simply clicking **Validate** in the **Execute** tab.

All functions you can use while debugging are found in the **Execute** tab.

Breakpoints enable you to pause the execution of a project so that you can check its state at a given point. After a breakpoint has been triggered you can stop, go to the next step of the automation or continue the debugging process by clicking **Stop**, **Step Over**, **Step Into** or **Continue Debug**.

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 an automation.

# Break

### Note:

This action is only available while debugging is in progress.

**Break** allows 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.

It is recommended to use **Break** along with **Slow Step** so that you know exactly when debugging needs to be paused. This is because activities are highlighted during debugging only when **Slow Step** is active, or when using **Step Into** and **Step Over**.

An alternative to using **Slow Step** in this situation is to keep an eye on the **Output** panel and use **Break** on the activity that is currently being debugged.

# Step Into

### Note:

This action is only available while debugging is paused.

**Step Into** is the functionality to be use when you want to closely analyze your activities while debugging step-by-step. When this action is triggered, the debugger opens and highlights activities in any container you might have in your workflow, such as flowcharts, sequences, or **Invoke Workflow** activities. In the latter case, the invoked workflow is opened in a new tab in a Read Only mode.

If you’re sure such a container is free of possible issues, you can use **Step Over** to continue debugging without showing you what’s happening inside the container.

# Step Over

### Note:

This action is only available while debugging is paused.

Unlike the **Step Into** functionality, **Step Over** does not open the current container. When used, the action debugs the next activity, highlighting containers (such as flowcharts, sequences, or **Invoke Workflow** activities) without opening them.

This action comes in handy for skipping analysis of large containers which are unlikely to trigger any issues during execution.

# Validate

The **Validate** action ensures that all variables, arguments, and imports are properly configured and used across the workflow. Validation issues can easily be identified by this icon.

Validation should be one of the first steps to take before executing workflows. You can think of **Validate** as a simple reminder to check variables, arguments, and imports.

# Breakpoints

**Breakpoints** are used to purposely pause the debugging process on an activity which may trigger execution issues. You can place a breakpoint on any activity either by selecting it and clicking the **Breakpoints** button on the **Execution** tab, from the context menu, or by pressing F9 while the activity of interest is selected.

A single activity needs to be selected for a breakpoint to be toggled. You can, however, toggle as many breakpoints as you see fit.

### Note:

**Breakpoints** are not saved in your workflows. This means that any workflow file is invoked without any breakpoints. If you want to closely analyze activities inside such a container or invoked workflow, it is recommended to use **Slow Step** to highlight activities, or **Step Into** so that containers are opened during debugging.

The **Remove All Breakpoints** option enables you to delete all the breakpoints from the opened project.

# Slow Step

**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** activities are opened. This is similar to using **Step Into**, but without having to pause the debugging process.

**Slow Step** can be activated both before or during the debugging process. Activating the action does not pause debugging.

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.

Each time you click **Slow Step** the speed changes by one step. You can easily tell by the icon, which updates accordingly.

# Options

Debugging options allow you to focus on fragile parts in your workflow. As such, you can have UI elements highlighted during debugging, as well as all activities logged in the **Output** panel as they are debugged.

Note that these options can only be toggled before debugging.

## **Highlight Elements**

If enabled, UI elements are highlighted during debugging. The option can be used both with regular and step-by-step debugging.

## **Log Activities**

If enabled, debugged activities are displayed as Trace logs in the **Output** panel. Please note that this option is enabled by default. Logs are automatically sent to Orchestrator if connected, but you can have them stored locally by disabling the **Allow Development Logging** option from [the **Settings** tab](https://orchestrator.uipath.com/v2018.3/docs/field-descriptions-robots#section-the-settings-tab) in the Add or Edit Robot window.

Disabling **Log Activities** can be a way to send smaller log files to Orchestrator.

By default, the debugger logs activities so that each step appears in the **Output** panel. We recommend leaving it enabled for easier tracing, as you can see in the image below:

The issue here is that one or more input fields from the User Input sequence are blank, which is a True condition for the Flow Decision. You can tell this by the fact that, during debugging, the User Input sequence is executed twice, meaning that one or more fields were left blank during the first execution.

If you decide to disable the **Log Activities** option for debugging, Trace logs are not displayed in the **Output** panel. In the case of a normal execution with no errors, you only get to see the debug execution start and end times. However, adding a **Log Message** can help you determine where issues might occur.

For example, you can add a **Log Message** activity to inform you that, in this case, one or more input fields are empty. This message appears in the **Output** panel during debugging, even if the **Log Activities** option is disabled, as you can see below:

Remember that you can always filter the messages displayed in the [**Output** panel](https://studio.uipath.com/docs/the-user-interface#section-the-output-panel) by simply selecting the alert types of interest, or even clear all messages.

Note that by default all debugging logs are sent to Orchestrator. You can disable this by clearing the **Allow Development Logging** option from [the **Settings** tab](https://orchestrator.uipath.com/v2018.3/docs/field-descriptions-robots#section-the-settings-tab) in the Add or Edit Robot window. If this option is disabled, debugging logs are only stored locally.

# Open Logs

Clicking **Open Logs** brings up the C:\Users\Your\_User\AppData\Local\UiPath\Logs folder where logs are locally stored. The naming format of log files is YYYY-DD-MM\_Component.log(such as 2018-09-12\_Execution.log, or 2018-09-12\_Studio.log).

# The UiPath.DiagTool Utility

[**SUGGEST EDITS**](https://studio.uipath.com/docs/the-uipathdiagtool-utility)

UiPath.DiagTool.exe is a command line utility that collects various debugging information related to the OS, session, or UiPath installations such as identification information, settings, and logs. It is meant to be used by support, QA, and development teams.

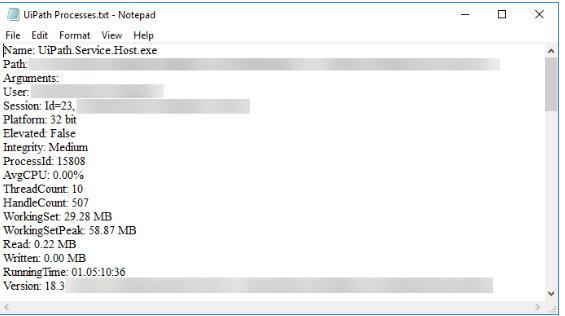
The utility is located in the Studio installation folder (C:\Program Files (x86)\UiPath\Studio) and needs to be run from the command line.

The amount of information which is provided depends on your system privileges. For instance, if you are the system administrator, the generated log contains full details about the system, whereas a user with limited privileges only receives information at a lower depth level. These logs also come in handy if you contact support, and it is recommended to attach them if you [submit a ticket](https://www.uipath.com/company/contact-us/contact-technical-and-activations).

Information is delivered by providers. They collect logs from different types of UiPath Platform events so that you better understand possible automation issues, debug processes, or attach logs if you contact support. You can find out more [about providers here](https://studio.uipath.com/v2018.3/docs/argument-descriptions-for-the-uipathdiagtool-utility#section-available-providers), but in short, they are as follows: **UiPath Installations, Computer Information, UiPath Processes, Settings File, Terminal Sessions, Screen Information, Third Party Installations, Environment Variable, Log Files, and Event Logs**.

# Using the UiPath.DiagTool Utility

1. Open an elevated command prompt instance.
2. Use the cd C:\Program Files (x86)\UiPath\Studio command to navigate to the designated folder.
3. Run the utility with the UiPath.DiagTool.exe -providers=procs --procs.extra=UiPath.Service.Host, installs, logs -file C:\Automation Logs\ExecStop.zipcommand. This generates a standard report file which contains information collected by the specified providers and creates theExecStop.ziparchive in theC:\Automation Logs` folder. You can find out more about providers [here](https://studio.uipath.com/v2018.3/docs/argument-descriptions-for-the-uipathdiagtool-utility#section-available-providers).
4. Navigate to the folder provided at the previous step and extract the ExecStop.zip archive. Inside you find the log files for UiPath Processes, UiPath Installation, as well as Studio and Robot. Check out the table of [available providers](https://studio.uipath.com/v2018.3/docs/argument-descriptions-for-the-uipathdiagtool-utility#section-available-providers) for more details. The image below is an example of a generated log file, in this case for the UiPath Process provider:



# Argument Descriptions for the UiPath.DiagTool Utility

[**SUGGEST EDITS**](https://studio.uipath.com/docs/argument-descriptions-for-the-uipathdiagtool-utility)

To use the tool launch an elevated instance of the Command Prompt and navigate to the tool location by using the cd C:\Program Files (x86)\UiPath\Studio command. You can use it with the following arguments:

Argument

Description

-providers=[Value]

Retrieves info about the specified provider(s). Here, [Value] represents one or more comma-separated providers. Please check the [available providers](https://studio.uipath.com/docs/link) to learn more.  
**Examples:**

* -providers=all - Retrieves info about all providers.
* -providers=procs.extra=UiPath.Service.Host, screen, computer - Retrieves info about the **UiPath.Service.Host** process, the screen information (setup, resolution), and general OS and computer ID information.

-file=File\_Path\Logs.zip

The path of the Logs.zip file which contains all requested info from providers.  
**Example:**

* -file=C:\WorkLogs\ProcsAndSettings.zip - Creates the ProcsAndSettings.zip file in the C:\WorkLogsfolder, containing all your requested logs.  
  This command can also be used with -cwd. However, the -file command only needs to contain the name of the .zip file.  
  **Example:**
* -file=ProcsAndSettings.zip -cwd - Creates the ProcsAndSettings.zip file in the Studio installation folder.

-console

Displays the full report in the console instead of a summary.

-cwd

Used along with the -file=ReportFiles.zip command to create the .zip file to the Studio installation folder instead of a custom path.

-q, -quiet

Silently retrieves requested logs, with minimal console output.

-h, -help

Displays the list of supported commands.

# Available Providers

Provider Name

Command

Retrieved Information

UiPath Installations

-providers=installs

Retrieved info about UiPath product installations. Such as installed Studio version, Studio and Robot installation paths, and License information.

Computer Information

-providers=computer

Retrieves general OS and computer ID information, such as computer name, domain, OS type and version, and user global proxy.

UiPath Processes

-providers=procs  
--procs.extra=RunningProcess

Retrieves information about UiPath running processes, such as Robot Service, Agent, and Executor diagnostics and resource impact.  
**Examples:**  
-providers=procs - Retrieves info about all running UiPath processes, such as path, arguments, user, session, platform, resource consumption.-providers=procs --procs.extra=UiPath.Service.Host - Retrieves info only about the **UiPath.Service.Host** process, such as path, arguments, user, session, platform, resource consumption.

Settings File

-providers=settings  
--settings.timeout=ValueInMs

Retrieves info about the connection between Orchestrator and the Robot at based on a specific timeout value, such as the UiPath.settings file. By default, the timeout value is set to 4000 milliseconds if the -providers=settings command is used alone.  
**Examples:**  
-providers=settings - Retrieves info with the default timeout value.  
-providers=settings --settings.timeout=30000 - Retrieves info with the timeout value of 30000 milliseconds.

Terminal Sessions

-providers=sessions

Retrieves info about running Terminal Server sessions.

Screen Information

-providers=screen

Retrieves screen information (such as setup and resolution) for the current desktop.

Third Party Installations

-providers=other

Retrieves info about third party product installations.

Environment Variable

-providers=envvars

Retrieves environment variables pertaining to UiPath, such as variable name, path, and value.

Log Files

-providers=logs  
--logs.exec\_cnt=Value  
--logs.studio\_cnt=Value  
--logs.etl\_cnt=Value

Retrieves log files related to the Robot Executor, Studio, and .etlfiles. By default, only the last 10 Robot Executor logs, 10 Studio logs, and the latest .etl file are retrieved from the desktop if you use the -providers=logscommand alone.  
**Examples:**  
-providers=logs - Retrieves the default number of log files for Robot Executor, Studio, and the latest .etl file from the desktop.  
You can also specify the exact number of logs to retrieve:  
-providers=logs --logs.exec\_cnt=50 --logs.studio\_cnt=30 --logs.etl\_cnt=3 - Retrieves the last 50 Robot Executor logs, 30 Studio logs, and 3 .etl files from the desktop.

Event Logs

-providers=events

Retrieves relevant events from Windows Event Log.

# Variables

# Managing Variables

[**SUGGEST EDITS**](https://studio.uipath.com/docs/managing-variables)

In UiPath Studio, variables are used to store multiples type of data. Another key aspect of variables is that their value can change so that you can, for example, control how many times the body of a loop is executed.

### Note:

Variables need to be created with different names, even if used in different Scopes. You can check out our [Workflow Design Naming Conventions](https://studio.uipath.com/docs/workflow-design#section-naming-conventions) recommendations.

The data stored within a variable is called a value, and it can be of multiple types. In UiPath, we support a large amount of types, ranging from generic value, text, number, data table, time and date, UiElements to any .Net variable type.

# Creating Variables

### Note:

Variables cannot be created if the **Designer** panel does not contain at least one activity.

## **From the context menu or with a keyboard shortcut (Ctrl+K):**

1. In the **Properties** panel of any activity, right-click a field that can be edited, and select **Create Variable** from the context menu, or press Ctrl+K. A **Set Name** field is displayed.
2. Fill in the name and press Enter. The variable is created and you can view and edit it in the **Variables** panel. The scope of activities created like this always belongs to the smallest container it is part of.

### Note:

When creating variables like this, the type is automatically generated, depending on the selected property.

## **From the Variables panel:**

# 

1. In the **Designer** panel, click **Variables**. The **Variables** panel is displayed.
2. Click the **Create Variable** line. A new variable with the default values is displayed.

### Note:

By default, all new variables are of **String**type if you create them from the **Variables**panel.

# Removing Variables

* In the **Variable** panel, right-click a variable and select the **Delete** option.
* In the **Variable** panel, select a variable and press the Delete key.

### Note:

If you want to undo this action, press Ctrl+Z.

You can also remove all the variables that are not used in your currently opened project:

1. On the **Design** ribbon tab, in the **Variables** group, select **Remove Unused Variables**. Note that the **Variables** panel only contains the variables used in your automation.

### Note:

If you remove or upgrade a package that may lead to a variable or argument being undefined, an **Unknown Type** is added in its place so that the file can be opened and edited in Studio.

# Browsing for .Net Variable Types

To search for types of variables that are not displayed by default in the **Variable Type** list, do the following:

1. In the **Variable** panel, from the **Variable Type** drop-down list, select **Browse for Types**. The **Browse and Select a .Net Type** window is displayed.
2. In the **Type Name** field, type a keyword for the variable you are looking for, such as excel. Note that the result section is updated, displaying all the .Net variable types that contain your keyword.
3. Select one and click **OK**. A new variable is created with the selected type and is displayed in the **Variables** panel.



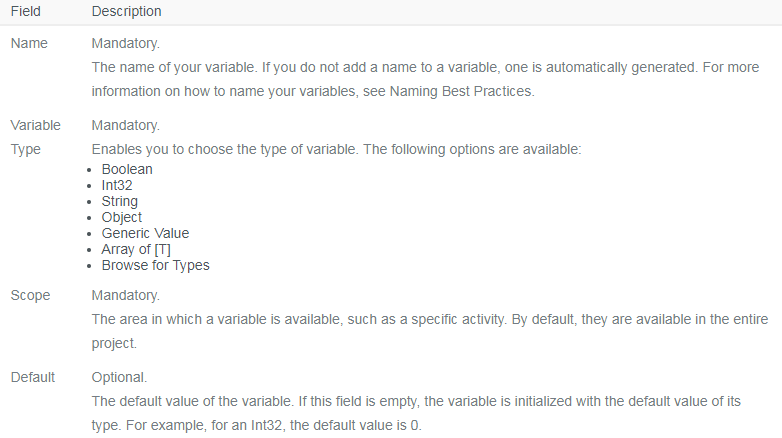
### Note:

After first using a type of variable from the Browse and Select a .Net Type window, it is displayed in the Variable Type drop-down list, in the Variables panel.

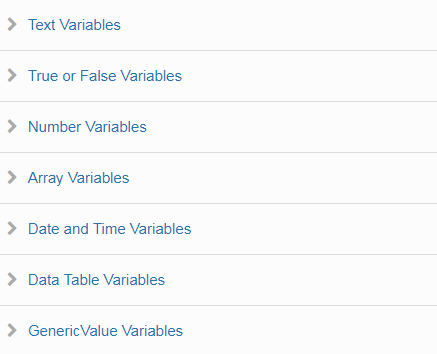
# The Variables Panel

[**SUGGEST EDITS**](https://studio.uipath.com/docs/the-variables-panel)

The **Variables** panel enables you to create variables and make changes to them.



# Types of Variables



# Text Variables

[**SUGGEST EDITS**](https://studio.uipath.com/docs/text-variables)

A text or string variable is a type of variable that can store only [strings](https://en.wikipedia.org/wiki/String_%28computer_science%29). These types of variables can be used to store any information such as employee names, usernames or any other string.

### Note:

All strings in UiPath have to be placed in between quotes.

# Example of Using a Text Variable

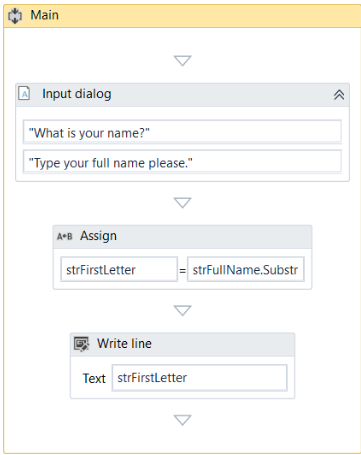
To exemplify how you can work with text variables, we are going to create a project that asks for the user’s name, stores it and displays only the first letter of his name in the **Output** panel.

1. Create a sequence.
2. Create two simple string variables, strFullName and strFirstLetter.
3. Add an [**Input Dialog**](https://activities.uipath.com/docs/input-dialog) activity to the **Designer** panel.
4. In the **Properties** panel, in the **Label** field, type "Type your full name please.".
5. In the **Title** field, type "What is your name?".
6. In the **Result** field, add the StrFullName variable. This variable stores whatever the user writes when prompted with the **Input Dialog** activity.
7. Add an [**Assign**](https://activities.uipath.com/docs/assign) activity under the **Input Dialog** one.
8. In the **Properties** panel, in the **To** field, add the strFirstLetter variable.
9. In the **Value** field, type strFullName.Substring(0,1). The strFirstLetter variable is assigned the new value created by the strFullName.Substring(0,1) expression.

### Note:

This field uses the Substring() function to find the first character from the string added by the user in the **Input Dialog**.

1. Add a [**Write Line**](https://activities.uipath.com/docs/write-line) activity under the **Assign** one.
2. In the **Properties** panel, in the **Text** field, enter the strFirstLetter variable. This means that the **Output** panel is going to display the first letter of what the user wrote in the **Input Dialog**. The project should look as in the following screenshot.



1. Press F5. The **What is your name?** window is displayed.
2. Type your name in the text field and click **OK**. In **UiPath Studio**, in the **Output** panel, note that the first letter of your name is displayed.

# True or False Variables

[**SUGGEST EDITS**](https://studio.uipath.com/docs/true-or-false-variables)

The true or false variable, also known as boolean, is a type of variable that only has two possible values, true or false. These variables enable you to make decisions, and thus have a better control over your flow.

# Example of Using a True or False Variable

To exemplify how you can work with true or false variables, we are going to create an automation that asks the user for his name and gender, and displays the results in another window.

1. Create a new project.
2. Create two string variables, strName and strGender. The first is going to be used to store the name of the user, and the second to store the user’s gender.
3. Create a boolean variable, boolMale. This variable is going to be used to verify if the user is a male.
4. Add an [**Input Dialog**](https://activities.uipath.com/docs/input-dialog) activity to the **Designer** panel and connect it to the **Start** node.
5. In the **Properties** panel, in the **Label** field type "What is your name?".
6. Add a title and, in the **Result** field, add the strName variable.
7. Add another **Input Dialog** activity and connect it to the previous one.
8. In the **Properties** panel, in the **Label** field, type "What is your gender?".
9. Add a title and, in the **Result** field, add the strGender variable.
10. Add a [**Flow Decision**](https://activities.uipath.com/docs/flow-decision) activity to the **Designer** panel, and connect it to the second **Input Dialog**.
11. In the **Properties** panel, in the **Condition** field, type strGender = "Male" or strGender = "male". This activity checks if the user is a male or female.
12. Add two [**Assign**](https://activities.uipath.com/docs/assign) activities.
13. Connect one to the **True** branch of the **Flow Decision**activity.
14. In the **Properties** panel, in the **To** field enter the boolMale variable.
15. In the **Value** field, type **True**. This assigns the **True** value to the boolMale variable when the strGender = "Male" or strGender = "male" condition is met.
16. Connect the second **Assign** activity to the **False** branch of the **Flow Decision**.
17. In the **Properties** panel, in the **To** field, enter the boolMale variable.
18. In the **Value** field, type **False**. This assigns the **False** value to the boolMale variable when the strGender = "Male" or strGender = "male" condition is not met.
19. Add a new **Flow Decision** and connect the previously added **Assign** activities to it.
20. In the **Properties** panel, in the **Condition** field, type boolMale = True.
21. Add a [**Message Box**](https://activities.uipath.com/docs/message-box) activity and connect it to the **True** branch of the **Flow Decision**.
22. In the **Properties** panel, in the **Text** field, type strName + " is a " + strGender + ".". This message displays the name of the user and its gender, if boolMale is true.
23. Add another **Message Box** activity and connect it to the **False** branch of the **Flow Decision**.
24. In the **Properties** panel, in the **Text** field, type strName + " is a " + strGender + ".". This message displays the name of the user and its gender, if boolMale is false. The final project should look like in the following screenshot.
25. Press F5. The automation is executed. Note that the final **Message Box** displays the message as expected.

# Number Variables

[**SUGGEST EDITS**](https://studio.uipath.com/docs/number-variables)

Number variables are also known as integer or Int32, and are used to store numeric information. They can be used to perform equations or comparisons, pass important data and many others.

# Example of Using a Number Variable

To exemplify how you can work with number variables, we are going to create an automation that asks the user for the year in which he or she was born and displays the age in a window.

1. Create a new sequence.
2. Create two Int32 variables, intBirthYear and intAge. The first stores the user’s birth year and the second, the user’s age.
3. Add an [**Input Dialog**](https://activities.uipath.com/docs/input-dialog) activity to the sequence.
4. In the **Properties** window, type an appropriate title and label.
5. In the **Result** field, add the intBirthYear variable.
6. Add an [**Assign**](https://activities.uipath.com/docs/assign) activity under the **Input Dialog**.
7. In the **Properties** panel, in the **To** field, add the intAge variable.
8. In the **Value** field, type 2016 – intBirthYear. This assigns the value of the subtraction (2016 – user’s birth year) to the intAge variable.
9. Add a [**Message Box**](https://activities.uipath.com/docs/message-box) activity under the **Assign** one.
10. In the **Properties** panel, in the **Text** field, type "Congratulations! You are " + intAge.ToString + ".".

### Note:

The .ToString method converts the integer stored in the intAge variable to a string and displays it as such.

The final project should look as in the following screenshot.

1. Press F5. The automation is executed. Note that the **Message Box** displays your age, as expected.

# Array Variables

[**SUGGEST EDITS**](https://studio.uipath.com/docs/array-variables)

The array variable is a type of variable which enables you to store multiple values of the same type.

UiPath Studio supports as many types of arrays as it does types of variables. This means that you can create an array of numbers, one of strings, one of boolean values and so on.

# Example of Using an Array Variable

To exemplify how you can work with array variables, we are going to create an automation that asks the user for his first and last name and age, stores the information in an array and then writes it in a .txt file.

1. Create a new sequence.
2. Create three string variables, strFirstName, strLastName and strAge, in which to store the information gathered from the user.
3. Create an array of strings variable called arrStringNameAge.
4. Add an [**Input Dialog**](https://activities.uipath.com/docs/input-dialog) activity to the **Designer** panel.
5. In the **Properties** panel, fill in the **Label** and **Title** fields to ask for the user’s first name.
6. In the **Result** field, type the strFirstName variable. This variable stores the first name of the user.
7. Add another **Input Dialog** activity under the previous one.
8. In the **Properties** panel, fill in the **Label** and **Title** fields to ask for the user’s last name.
9. In the **Result** field, type the strLastName variable. This variable is going to store the last name of the user.
10. Add another **Input Dialog** activity under the previous one.
11. In the **Properties** panel, fill in the **Label** and **Title** fields to ask for the user’s age.
12. In the **Result** field, type the strAge variable. This variable is going to store the age of the user.

### Note:

We use a string variable and not an integer to store the age, so that we do not have to convert it later on, when we add it to the string array variable.

1. Add an [**Assign**](https://activities.uipath.com/docs/assign) activity under the last **Input Dialog**.
2. In the **Properties** panel, in the **To** field, type the arrStringNameAge variable.
3. In the **Value** field, type {strFirstName,strLastName,strAge}. This **Assign** activity enables you to store all the values from the initial string variables in the arrStringNameAgeone.
4. Add a [**Write Text File**](https://activities.uipath.com/docs/write-text-file) activity under the **Assign** one.
5. In the **Properties** panel, in the **FileName** field, type the path of the file you want to write to between quotation marks, such as "%HOMEPATH%\Desktop\array\_variable".

### Note:

If the file does not exist at the provided path, it is created.

1. In the **Text** field, type arrStringNameAge(0) + " " + arrStringNameAge(1) + " " +arrStringNameAge(2) + " ".

### Note:

By adding the index number of the array items you can access their values and write them, in this example, to a text file.

The final project should look as in the following screenshot.

1. Press F5. The automation is executed.
2. Navigate to the file provided at step 17 and double-click it. A **Notepad** window is displayed with the information you added at step 20.

# Date and Time Variables

[**SUGGEST EDITS**](https://studio.uipath.com/docs/date-and-time-variables)

The date and time variable is a type of variable that enables you to store information about any date and time. This type of variable can be found in the **Browse and Select a .Net Type**window, under the System namespace (System.DateTime). For more information, see [Browsing for .Net Variable Types](https://studio.uipath.com/docs/managing-variables#section-browsing-for-net-variable-types).

For example, they can be used to append dates to invoices or any other documents you may be working with and are time-sensitive.

# Example of Using a Date and Time Variable

To exemplify how you can work with a date and time variable, we are going to build an automation that gets the current date and time, subtracts a specific amount of time and writes the result to a Microsoft Excel spreadsheet.

1. Create a new sequence.
2. Create two DateTime variables, timToday and timLastTime.
3. Create a TimeSpan variable, called timSpan, and in the **Default** field type 1.02:10:04.

### Note:

The default value attributed to thetimSpan variable uses the day.hh:mm:ss format.

1. Add an [**Assign**](https://activities.uipath.com/docs/assign) activity to the **Designer** panel.
2. In the **Properties** panel, in the **To** field, add the timToday variable.
3. In the **Value** field, type **Now**. This gives you the date and time when the project is executed, in the dd/MM/yyyy and hh:mm:ss formats.
4. Add another **Assign** activity under the previous one.
5. In the **Properties** panel, in the **To** field, add the timLastTime variable.
6. In the **Value** field, type timToday.Subtract(timSpan). This is going to subtract the default value of the timSpan variable from the current date and time, stored in the timTodayvariable.
7. Add an [**Excel Application Scope**](https://activities.uipath.com/docs/excel-application-scope) activity under the last **Assign** one.

### Note:

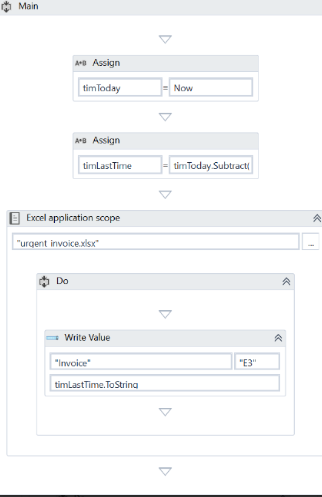
Use the [Manage Packages](https://studio.uipath.com/docs/managing-activities-packages#section-managing-packages) feature to download the Excel activities, if you do not already have them installed.

1. In the **Properties** panel, in the **WorkbookPath** field, type the path of the Excel file you want to write to, between quotation marks. In our case, "%HOMEPATH%\Desktop\urgent\_invoice.xlsx".

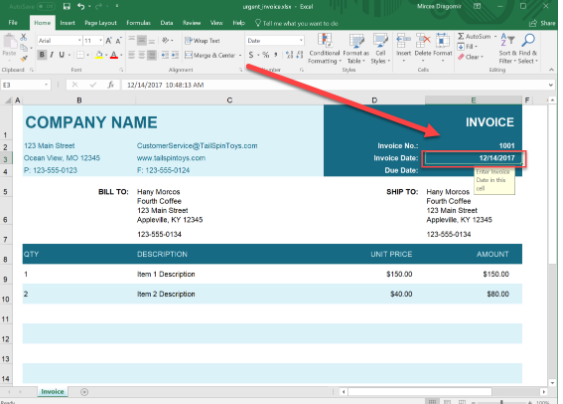
### Note:

If the file does not exist at the provided path, it is going to be created.

1. Add a [**Write Cell**](https://activities.uipath.com/docs/write-cell) activity in the **Excel Application Scope** activity.
2. In the **Properties** panel, in the **Range** field, type the coordinates of an Excel cell between quotation marks. In our case, "E3."
3. In the **Sheet Name** field, type the name of the sheet in which you want to write. In our case, "Invoice". Note that if the sheet does not exist, it is going to be created.
4. In the **Value** field, type timLastTime.ToString. This transforms the value of the timLastTime variable to a string and writes it to the coordinates previously given.  
   The final project should look as in the following screenshot:



1. Press F5. The automation is executed.
2. Navigate to your Excel file and double-click the cell in which you added the date. Note that the time and date information is displayed in the cell you pointed towards.



# Data Table Variables

[**SUGGEST EDITS**](https://studio.uipath.com/docs/data-table-variables)

DataTable variables represent a type of variable that can store big pieces of information, and act as a database or a simple spreadsheet with rows and columns. They can be found in the **Browse and Select a .Net Type** window, under the System.Data namespace (System.Data.DataTable). For more information, see [Browsing for .Net Variable Types](https://studio.uipath.com/docs/managing-variables#section-browsing-for-net-variable-types).

These variables can be useful to migrate specific data from a database to another, extract information from a website and store it locally in a spreadsheet and many others.

# Example of Using DataTable Variables

To exemplify how you can use DataTable variables, we are going to create an automation that reads only two out of multiple columns from an Excel spreadsheet, and then transfers them to another spreadsheet that already contains other information.

The initial file is a database of people, transactions, dates, and products. In this example, we are going to extract their names and order dates and append them to an Excel spreadsheet that already contains similar information.

1. Create a new sequence.
2. Add an [**Excel Application Scope**](https://activities.uipath.com/docs/excel-application-scope) activity to the sequence. This activity is required for most of the Excel-related activities.

### Note:

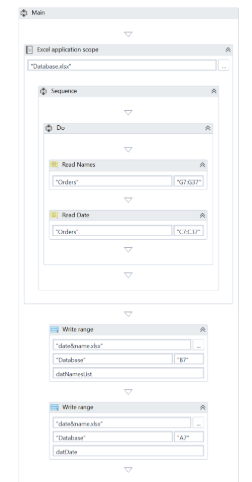
If you do not have Excel activities installed on your version of UiPath, use the [Manage Packages](https://studio.uipath.com/v2018.2/docs/managing-activities-packages) functionality to get them.

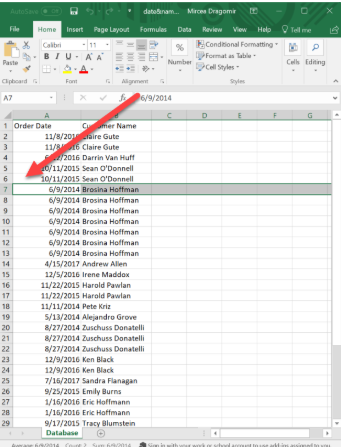
1. Create two DataTable variables, datNamesList and datDate. These are going to be used to store information from the initial Excel spreadsheet.
2. In the **Properties** panel, in the **WorkbookPath** field, type the path of the initial Excel file to be used, between quotation marks.
3. Add two [**Read Range**](https://activities.uipath.com/docs/read-range) activities and place them one under the other, in the **Excel Application Scope** activity. These are used to get information from the initial spreadsheet.
4. Select the first **Read Range** activity and, in the **Properties** panel, in the **Range** field, type "G7:G37". These are the Excel table coordinates that tell UiPath Studio from where to extract information.
5. In the **SheetName** field, do not make any changes as the name of our sheet is the default one, Orders.
6. In the **DataTable** field, type the name of the first DataTable variable, datNamesList. This variable stores all the information available between the G7 and G37 rows.
7. (**Optional**) Change the value in **DisplayName** field to Read Names, so you can easily tell apart this activity from the second one.
8. Select the second **Read Range** activity, and in the **Properties** panel, in the **Range** field, type "C7:C37". These are the Excel table coordinates that contain the order date we want to extract.
9. In the **DataTable** field, specify the datDate variable. This variable retains all the date information we require.
10. Add a [**Write Range**](https://activities.uipath.com/docs/write-range) activity to the **Designer** panel, under the **Excel Application Scope**. This activity is used to write the stored information to another Excel file.

### Note:

The file used with the **Write Range** activity has to be closed when you run the project. If it is not closed, an error is displayed and the automation execution stops.

1. In the **Properties** panel, in the **WorkbookPath** field, type the path of the Excel file to be used to store all the information gathered at the previous steps.
2. In the **DataTable** field, type the datNamesList variable.
3. In the **SheetName** field type Database, and in the **StartingCell**, type "B7.“ This is the starting cell in which information from the initial file is to be added.
4. Add another **Write Range** activity and place it under the first one.
5. In the **Properties** panel, fill in the **WorkbookPath** and **SheetName** fields as for the previous **Write Range** activity.
6. In the **Starting Cell** field, type "A7".
7. In the **DataTable** field, type the datDate variable.



1. Press F5. Your automation is executed.
2. Double-click the final Excel file. Note that the copied information is available, and correctly updated.
3. 

# GenericValue Variables

[**SUGGEST EDITS**](https://studio.uipath.com/docs/genericvalue-variables)

The GenericValue variable is a type of variable that can store any kind of data, including text, numbers, dates, and arrays, and is particular to UiPath Studio.

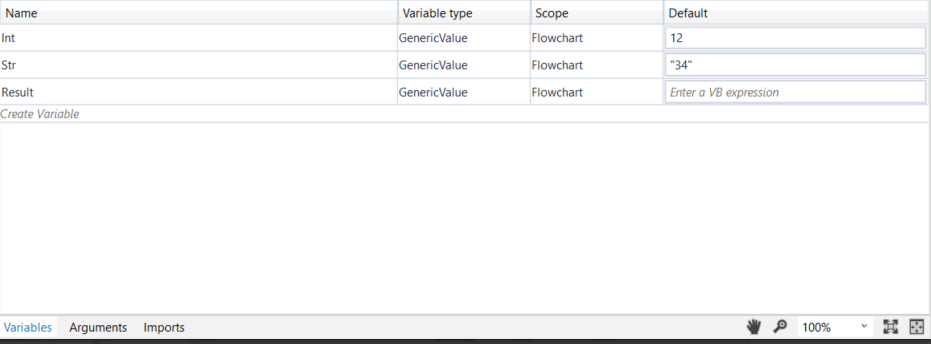
GenericValue variables are automatically converted to other types, in order to perform certain actions. However, it is important to use these types of variables carefully, as their conversion may not always be the correct one for your project.

UiPath Studio has an automatic conversion mechanism of GenericValue variables, which you can guide towards the desired outcome by carefully defining their expressions. Take into account that the first element in your expression is used as a guideline for what operation Studio performs. For example, when you try to add two GenericValue variables, if the first one in the expression is defined as a String, the result is the concatenation of the two. If it is defined as an Integer, the result is their sum.

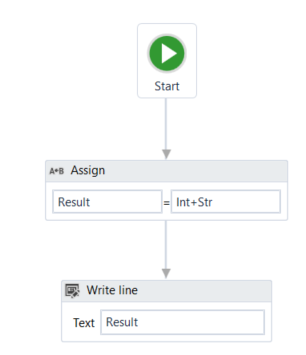
# Example of Using a GenericValue Variable

To demonstrate how a GenericValue variable works, let us create an automation that performs different operations whose results depend on the way we define their expressions. We create two GenericValue variables of different data types and display the results in the **Output** panel.

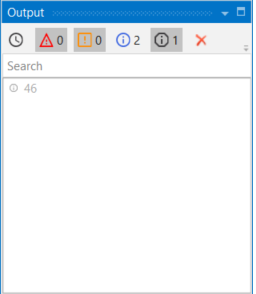
1. Create a new blank project.
2. Create three GenericValue variables: Int, Str, and Result.
3. In the **Default** column for the Int variable, type 12, and for the Str variable, type "34". The first variable is interpreted as an integer, while the second one is interpreted as a string.



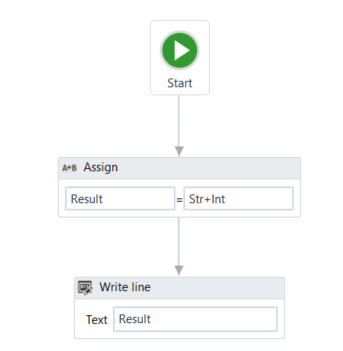
1. Add an [**Assign**](https://activities.uipath.com/docs/assign) activity to the **Designer** panel and connect it to the **Start** node.
2. In the **Properties** panel, in the **To** field, enter the Result variable.
3. In the **Value** field, type Int + Str.
4. Add a [**Write Line**](https://activities.uipath.com/docs/write-line) activity and connect it to the **Assign** one.



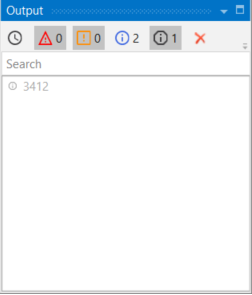
1. In the **Properties** panel, in the **Text** field, enter the Result variable.  
   The project should look as in the following screenshot.
2. Press F5 to execute your automation. Note that, in the **Output** panel, the sum of the two numbers is displayed.



1. Go back to the previously added [**Assign**](https://activities.uipath.com/docs/assign) activity and change the **Value** field to Str + Int, to reverse the order of the variables. The project should look as in the following screenshot.



1. Press F5 to execute your automation. Note that, in the **Output** panel, the concatenation of the two numbers is displayed.



This means that UiPath Studio takes the first element defined in your expression as a guideline for what operation to perform. If the first element in your expression is a GenericValue variable filled in as integer, UiPath Studio will perform the sum of the elements.  
If the first element in your expression is a string or a GenericValue variable filled in as string, UiPath Studio will perform the concatenation of the elements.

# Arguments-

# Managing Arguments

[**SUGGEST EDITS**](https://studio.uipath.com/docs/managing-arguments)

Arguments are used to pass data from a project to another. In a global sense, they resemble variables, as they store data dynamically and pass it on. Variables pass data between activities, while arguments pass data between automations. As a result, they enable you to reuse automations time and again.

UiPath Studio supports a large number of argument types, which coincide with the types of variables. Therefore, you can create Generic Value, String, Boolean, Object, Array, or DataTable arguments and you can also browse for .Net types, just as you do in the case of [variables](https://studio.uipath.com/docs/managing-variables).

Additionally, arguments have specific [directions](https://studio.uipath.com/docs/the-arguments-panel) (**In**, **Out**, **In/Out**, **Property**) that tell the application where the information stored in them is supposed to go.

# Creating Arguments

### Note:

Argument names should be in upper CamelCase with a prefix stating the argument type, such as in\_DefaultTimeout, in\_FileName, out\_TextResult, io\_RetryNumber. You can check out our [Workflow Design Naming Conventions](https://studio.uipath.com/docs/workflow-design#section-naming-conventions) recommendations for more details.

To create a new argument:

1. In the **Designer** panel, click **Arguments**. The **Arguments** panel is displayed.

# 

1. Click the **Create Argument** line. A new argument with the default values is displayed.

### Note:

By default, all arguments are of **String** type and have an **In** direction.

# Removing Arguments

To remove an argument:

* In the **Arguments** panel, select an argument and press Delete.
* In the **Arguments** panel, right-click an argument and select the **Delete** option.

# 

# The Arguments Panel

The **Arguments** panel enables you to create arguments and make changes to them.

# 

# Using Arguments

[**SUGGEST EDITS**](https://studio.uipath.com/docs/using-arguments)

Due to the nature of arguments, you are going to use them a lot in relation with the **Invoke Workflow File** and **Launch Workflow Interactive** activities. They can be found in the **Activities** panel, under **Workflow > Invoke** and they enable you to browse for a project, and import and edit their arguments.

# Example of Using an Argument

To exemplify how to use an argument in an automation with an **Invoke Workflow File** activity, we are going to create two separate sequences. A very simple one in which to assign a value to an argument, and a second that invokes it and displays the value in a message box.

1. Create a new sequence.
2. In the **Arguments** panel, create an argument, StoreValue.
3. From the **Direction** list, select **Out**, and do not change the **Argument Type** from **String**.
4. Add an [**Assign**](https://activities.uipath.com/docs/assign) activity to the **Designer** panel.
5. In the **Properties** panel, in the **To** field, add the StoreValue argument.
6. In the **Value** field, type a string, such as "How does one use an argument?".  
   The first project should look like in the screenshot below.

# 

1. Create a new sequence.
2. Create a string variable, strFinalValue.
3. Add an [**Invoke Workflow File**](https://activities.uipath.com/docs/invoke-workflow-file) activity to the **Designer** panel.
4. On the activity, click the **Browse (…)** button and browse for the previously created sequence.
5. Click **Import Arguments**. The **Invoked Workflow’s Arguments** window is displayed. Note that the argument of the first sequence is displayed here.
6. In the **Value** field, add the strFinalValue variable and click **Ok**. The argument is imported and the value from it is going to be stored in the current project through the strFinalValue variable.
7. Add a [**Message Box**](https://activities.uipath.com/docs/message-box) activity under the **Invoke Workflow File**.
8. In the **Properties** panel, in the **Text** field, type strFinalValue + " This is how", for example.  
   The second automation should look like in the screenshot below.

# 

1. Press F5 in the second sequence. The automation is executed correctly and the message box displays the desired text.

# 

# About Imported Namespaces

[**SUGGEST EDITS**](https://studio.uipath.com/docs/about-imported-namespaces)

VB.NET namespaces in UiPath Studio represent containers that store different types of data. They enable you to easily define the scope of your expressions, variables and arguments.

For example, if you have the System.Data namespace imported, you can further use DataTable, DataView, DataColumn, DataRow and other classes that are available in it, without having to always type System.Data.DataTable and so on.

All imported namespaces are displayed in the **Imports** panel. Note that some namespaces are automatically imported when you browse for a .Net type variable or argument.

To open this panel, click **Imports** in the **Designer** panel.

# 

# Importing New Namespaces

[**SUGGEST EDITS**](https://studio.uipath.com/docs/importing-new-namespaces)

To add new namespaces to your library:

1. Open the **Imports** panel.
2. In the **Enter or Select namespace** field, start typing the namespace that interest you. Note that suggestions are provided while you type, in case you are not exactly sure what you are looking for.

# 

1. (**Optional**) Click the drop-down arrow to view and browse all available namespaces.
2. Select the desired namespace. The namespace is added to the **Imported Namespaces** list.

To remove a namespace, select it and press Delete. Note that namespaces can only be deleted if they aren't used. For example, you can delete a namespace if the assembly that contains it is no longer referenced by the project.

# About Recording

[**SUGGEST EDITS**](https://studio.uipath.com/docs/about-recording)

Recording is an important part of UiPath Studio, that can help you save a lot of time when automating your business processes. This functionality enables you to easily capture a user’s actions on the screen and translates them into sequences.

These projects can be modified and parameterized so that you can easily replay and reuse them in as many other processes as you need.

All user interface elements are highlighted while you record, as you can see in the following screenshot, so that you can be sure the correct buttons, fields or menus are selected.

nteractions with UI elements yield informative screenshots in the automation. These can be changed, hidden, removed or shown in full size by selecting the respective action from the **Options** menu.

All screenshots are automatically saved as .png files in the same location as your project, in a separate folder named ".screenshot" By default, the path is: C:\Users\your\_user\_name\Documents\UiPath\your\_project\_name\.screenshots.

Regardless of the type of recording selected, some actions are recordable and some are not.

**Recordable-**

* Left-click on buttons, check boxes, drop-down lists and other GUI elements
* Text typing

**Non-Recordable-**

* Keyboard shortcuts
* Modifier keys
* Right-click
* Mouse hover

# Recording Types

[**SUGGEST EDITS**](https://studio.uipath.com/docs/about-recording-types)

There are four types of recordings available in UiPath Studio:

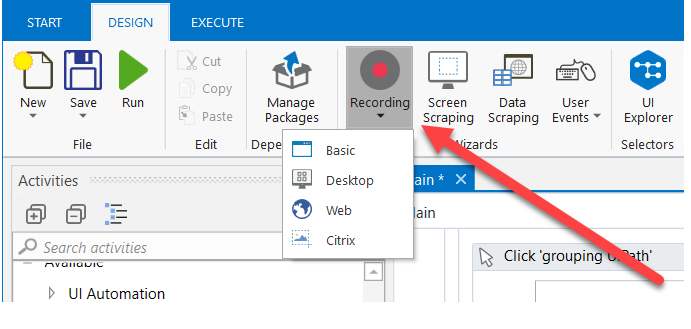
* **Basic** – generates a full selector for each activity and no container, the resulted automation is slower than one that uses containers and is suitable for single activities.
* **Desktop** – suitable for all types of desktop apps and multiple actions; it is faster than the **Basic** recorder, and generates a container (with the selector of the top-level window) in which activities are enclosed, and partial selectors for each activity.
* **Web** – designed for recording in web apps and browsers, generates containers and uses the **Simulate Type/Click** input method by default.

### Note:

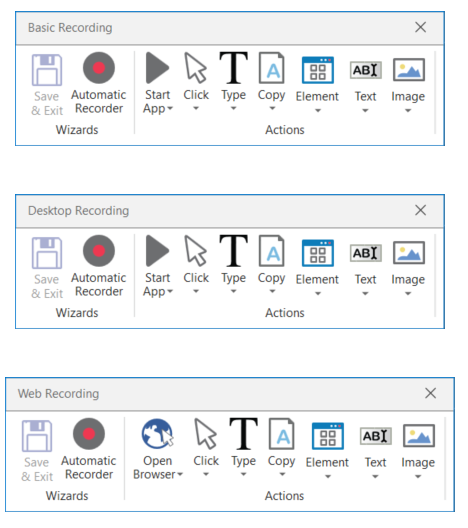
It is recommended to run your web automations on Internet Explorer 11 and above, Mozilla Firefox 50 or above, or the latest version of Google Chrome.

* **Citrix** – used to record virtualized environments (VNC, virtual machines, Citrix, etc.) or SAP, permits only image, text and keyboard automation, and requires explicit positioning.

To see all the available types of recordings and select the one most suited for your project, click **Record**, in the **Wizards** group of the **Design** ribbon tab.



All recording types come with their own controllers (or toolbars) that give you access to actions particular to each environment, but also some common ones.



The **Desktop**, **Basic** and **Web Recording** toolbars are quite similar and enable you to:

* Automatically record multiple actions performed on the screen
* Manually record single actions, such as:
  + Starting or closing an application or web browser
  + Clicking an interface element
  + Selecting an option from a drop-down list
  + Selecting a check box
  + Simulating keystrokes or keyboard shortcuts
  + Copying text from a UI element or performing screen scraping
  + Looking for elements or waiting for them to vanish
  + Finding an image
  + Activating a window

The **Citrix Recording** toolbar enables you to:

* Click an image or text
* Simulate keystrokes or hotkeys
* Select and copy text from a window
* Scrape UI elements
* Look for elements or wait for them to vanish
* Find an image or wait for it to vanish
* Activate a window

### Note:

The **Citrix Recording** toolbar supports only manual recording (single actions).

To figure out if you should use automatic or manual recording in your project, you should better understand the differences between the recording types and their capabilities.

**Automatic Recorder**

* Left-click on windows, buttons, check boxes, drop-down lists etc.
* Text typing

**Manual Recorder**

* Keyboard shortcuts
* Modifier keys
* Right-click
* Mouse hover
* Getting text
* Find elements and images
* Copy to Clipboard

Keyboard shortcuts that you can use while recording:

* F2 – pauses for 3 seconds. A countdown timer is displayed in the bottom left corner of the screen. Can be useful with menus that automatically hide.
* Esc – exits the automatic or manual recording. If you press the Escape key again, the recording is saved as a sequence, and you return to the main view.
* Right-click – exit the recording.

# Automatic Recording

[**SUGGEST EDITS**](https://studio.uipath.com/docs/automatic-recording)

Automatic recording is very useful and time-saving as it can provide you with a skeleton for your business processes, and can be easily customized and parameterized.

# Automatically Generated Activities

As you can see in the examples below, for the actions that are recordable with the **Automatic Recording**, some activities are automatically generated:

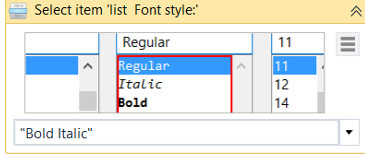
* [**Click**](https://activities.uipath.com/docs/click) – generated when you click a button (Basic and Desktop) or a link (Web). The options available in the **Properties** panel enable you to add a time delay before or after the action, change the click type and add key modifiers.

# 

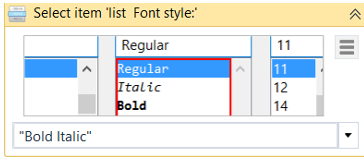
* [**Type Into**](https://activities.uipath.com/docs/type-into) – generated when typing into a text field or any editable UI element. The options available in the **Properties** panel enable you to add a time delay before or after the action or between key strokes, change the text at any point, and erase the entire field before writing to it (**EmptyField**).

# 

* [**Select Item**](https://activities.uipath.com/docs/select-item) - generated when you select an item from a drop-down list or combo box. The options available in the **Properties** panel enable you to add a time delay before or after the action, and change the selected item.



* [**Check**](https://activities.uipath.com/docs/check) - generated when a radio button or check box is clicked. The options available in the **Properties** panel enable you to add a time delay before or after the action, and select or unselect the check box.



# The Active UI Framework

While recording, the **Active UI Framework** can be changed, for different results in the detection of different application UI Elements.

The following options are available:

## **Default**

This is the UiPath proprietary method. Usually works fine with all types of user interfaces.

## **Active Accessibility**

This is an earlier solution from Microsoft for making apps accessible. It is recommended that you use this option with legacy software, when the **Default** one does not work.

## **UI Automation**

This is the improved accessibility model from Microsoft. It is recommended that you use this option with newer apps, when the Default one does not work.

### Note:

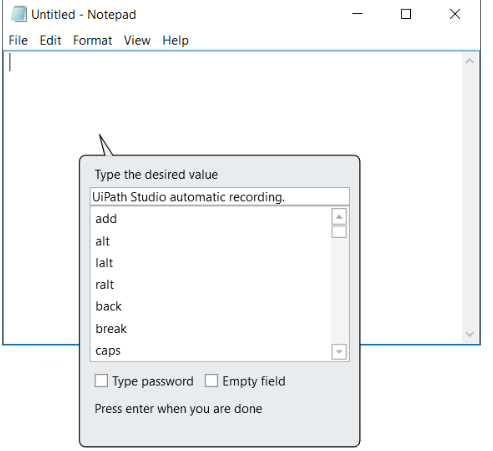
While recording, the **Active UI Framework** can be switched by using the **F4** shortcut key.

# Example of Automatic Recording with Basic and Desktop

[**SUGGEST EDITS**](https://studio.uipath.com/docs/example-of-automatic-recording-with-basic-and-desktop)

To exemplify how you can use the automatic recording and understand more about the differences between **Basic** and **Desktop**, let’s create the same project for the two.

1. Open Notepad.
2. In UiPath Studio, create a new sequence.
3. a. In the **Design** ribbon tab, in the **Wizards** group, **select Record > Basic**. The **Basic Recording** toolbar is displayed and the main view is minimized.  
   b. In the **Design** ribbon tab, in the **Wizards** group, **select Record > Desktop**. The **Desktop Recording** toolbar is displayed and the main view is minimized.
4. In the **Wizards** group, click **Automatic Recorder**. The automating recording process starts.
5. In Notepad, click on the main panel. A pop-up window is displayed.



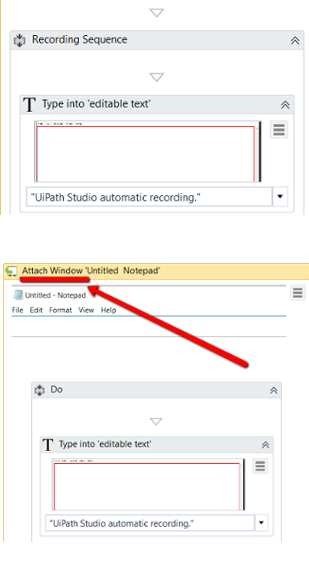
1. Type a custom text and press Enter. The string is displayed in Notepad.

### Note:

Select the **Empty field** check box to delete previously existing text. You can also select this option after the recording is finished, in the **Properties** panel of the [**Type Into**](https://activities.uipath.com/docs/type-into) activity.

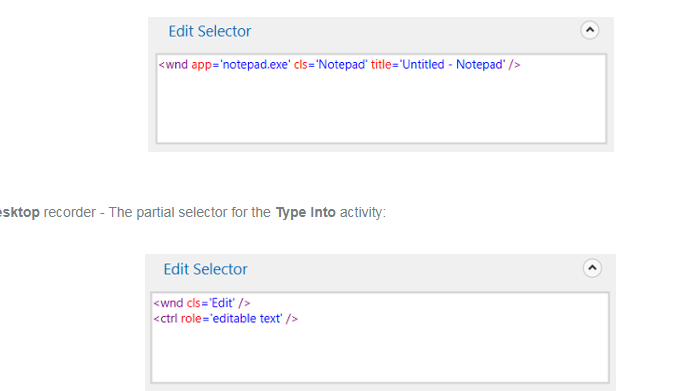
1. From the **Format** menu, select **Font**. The **Font** window is displayed.
2. Select a different font style, such as Bold Italic, and click **OK**.
3. Press Esc two times. You exit the recording view and the saved project is displayed in the **Designer** panel.
4. Press F5. The automation is executed as expected.

The two screenshots below display part of the resulted projects for the **Basic** (first screenshot) and **Desktop** (second screenshot) automatic recordings. As you can see, the second one generates an [**Attach Window**](https://activities.uipath.com/docs/window-scope) container, while the **Basic** one does not.



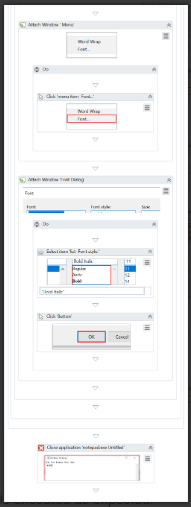
**Desktop** recorder - The top level window selector from the **Attach Window** container:

**Desktop** recorder - The partial selector for the **Type Into** activity:

You can also customize and parameterize this project after it is recorded. To exemplify this, let’s take the **Desktop** recording example and build upon it.

For example, let’s assume that we want to extract the number of an invoice from an Excel file, copy it to a new Notepad application which we close in the end.

1. Add an [**Excel Application Scope**](https://activities.uipath.com/docs/excel-application-scope) activity before the recording sequence.
2. In the **WorkbookPath** field, type the path of the Excel file you need to extract information from.
3. Add a [**Read Cell**](https://activities.uipath.com/docs/read-cell) activity in the **Excel Application Scope**.
4. In the **Properties** panel, add the **Sheet Name** and **Cell** information from the Excel file used.
5. Right-click in the **Result** field, and click **Create Variable**. The **Set Name** field is displayed.
6. Fill in the name, such as genInvoiceNumber, and press Enter. The variable of type 'GenericValue' is created and displayed in the **Result** field and **Variables** panel.
7. Change the scope of the variable to **Main**.
8. In the recording sequence, in the **Type Into** activity, in the **Text** field, add the genInvoiceNumber variable. This copies the value stored in the variable to Notepad.  
   What was added to the project should look as in the following screenshot.



1. Add an [**Open Application**](https://activities.uipath.com/docs/open-application) activity between **Excel Application Scope** and the Recording sequence.
2. Use **Indicate window on screen** to select the active **Notepad** window.
3. Place the Recording sequence inside the **Open Application** activity.
4. Add a [**Close Application**](https://activities.uipath.com/docs/close-application) activity after **Open Application**.
5. Use **Indicate window on screen** again to select the active **Notepad** window to be closed.
6. Make sure the **OffsetX** and **OffsetY** properties (**Cursor Position**) are empty.  
   What was added to the project should look as in the following screenshot.
7. Press F5. The automation is executed as expected.

### Note:

To ensure that the workflow execution ends, close the Save Notepad window.

# Example of Automatic Recording with Web

[**SUGGEST EDITS**](https://studio.uipath.com/docs/example-of-automatic-recording-with-web)

To exemplify how you can use the web recording and understand how it works, let’s create a project that enables you to go to Amazon and sign in to your account.

### Note:

It is recommended to run your web automations on Internet Explorer 11 and above, Mozilla Firefox 50 or above, or the latest version of Google Chrome.

1. Open an Internet Explorer instance.
2. In UiPath Studio, create a new sequence.
3. In the **Design** tab, in the **Wizards** group, select **Record > Web**. The **Web Recording** toolbar is displayed and the main view is minimized.
4. Click **Automatic Recording**. The automating recording process starts.
5. Navigate to [www.amazon.com](http://www.amazon.com/).
6. Go to the Sign In page and input your e-mail and password.

### Note:

When the **Type Into** pop-up is displayed for your password, make sure that you select the **Type Password** check box. Besides the [**Type Into**](https://activities.uipath.com/docs/type-into) activity, another activity, [**Get Password**](https://activities.uipath.com/docs/get-password), is created in the sequence. This activity hides the password behind asterisks (\*) and stores it in a string variable.

1. Click **Sign In** and press Esc two times. The recording is saved and displayed as a project in the **Designer** panel.
2. Close and open Internet Explorer manually.
3. In Studio, press F5. The automation is executed as expected.

[**Click here to download the example.**](https://www.uipath.com/hubfs/Documentation/WorkflowExamples/WebRecording.zip?t=1497011465543)

As you can see, the project requires you to close and open Internet Explorer manually. If you want to automate this part too, you need to manually add the [**Open Browser**](https://activities.uipath.com/docs/open-browser) and [**Close Tab**](https://activities.uipath.com/docs/close-tab) activities at the beginning and end of the project.

# Manual Recording

[**SUGGEST EDITS**](https://studio.uipath.com/docs/manual-recording)

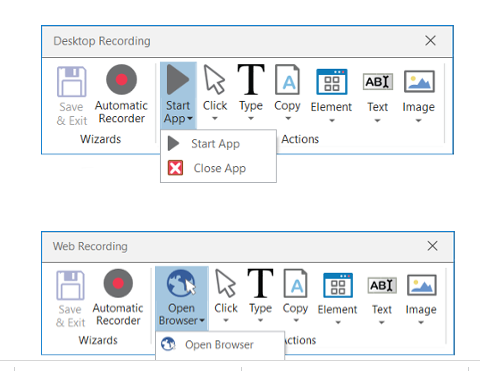
As explained in the [About Recording Types](https://studio.uipath.com/docs/about-recording-types) chapter, there are some actions that cannot be handled by the automatic recorder. We refer to these as single actions or manual recordings. You can use both manual and automatic recording in automations to achieve full automation of a task. Single actions can be found in the **Actions** group of any recording toolbar.

### Note:

It is recommended to run your web automations on Internet Explorer 11 and above, Mozilla Firefox 50 or above, or the latest version of Google Chrome.

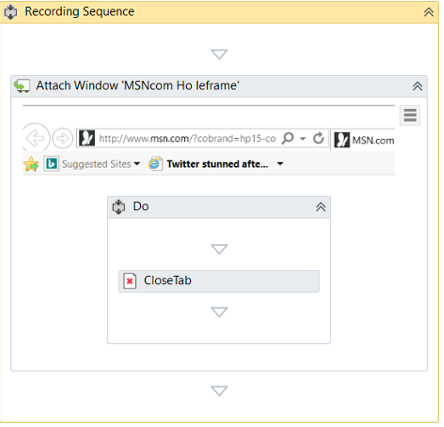
## **Types of Single Actions**

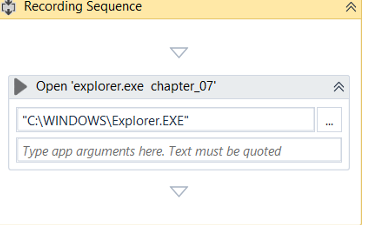
# Start and Stop an App or Browser



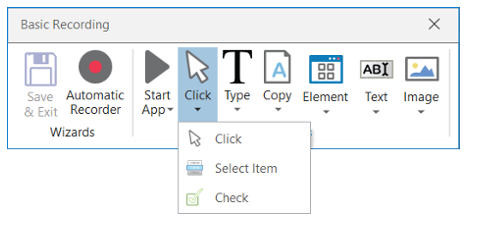
These single actions enable you to open an app or browser, as well as close them, by pointing and clicking them.

The activities generated using the desktop and web manual recorders contain partial selectors and containers (first screenshot), while the activities generated by the basic recorder contains a full selector and no container (second screenshot), just like with the automatic recording.



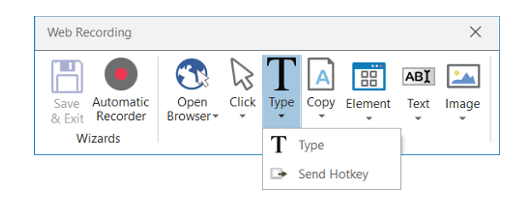


# Click

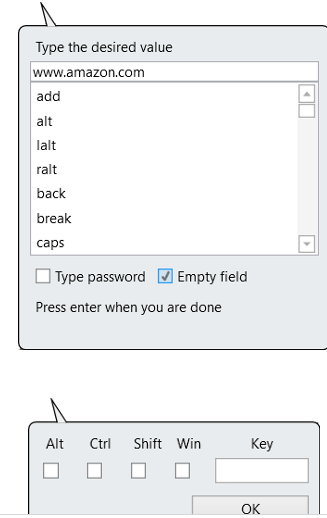


These types of actions enable you to record clicks on the desktop or a running application, select an option from a drop-down list or combo box, and select a check box or a radio button.

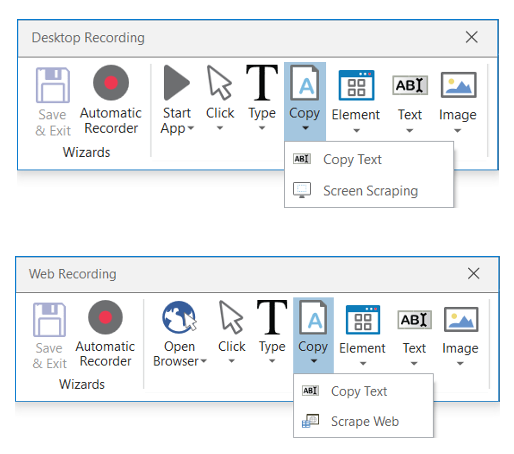
# Type



These single actions include those that require input from the keyboard, such as keyboard shortcuts and key presses. To achieve this, two pop-up windows are used to retrieve your keyboard input.



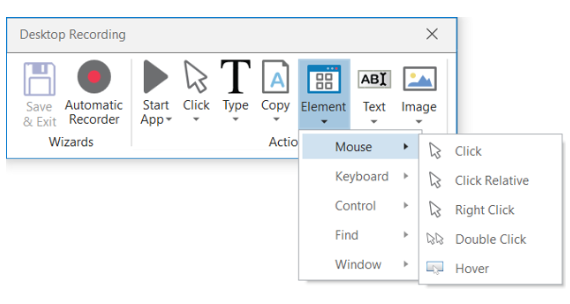
# Copy



These actions enable you to copy a selected text from an opened application or web browser, so that you can use it later in the project. Screen scraping is also available under the **Copy**menu, as it enables you to extract images and text from an app or browser.

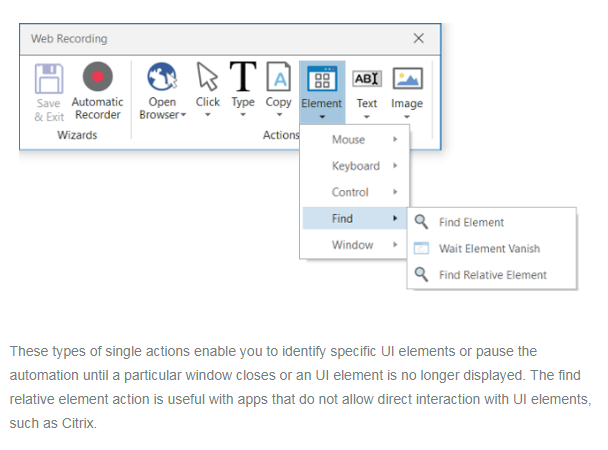
For more information, see [Screen Scraping](https://studio.uipath.com/docs/output-or-screen-scraping-methods).

# Mouse Element



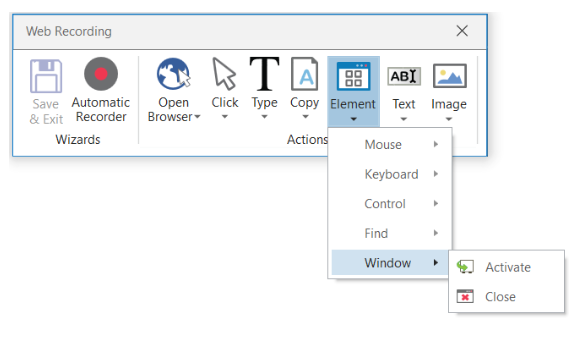
These types of actions enable you to simulate mouse movements that cannot be recorded but give you access to more functionalities, such as right-clicking, hovering or double-clicking.

# Find Element



These types of single actions enable you to identify specific UI elements or pause the automation until a particular window closes or an UI element is no longer displayed. The find relative element action is useful with apps that do not allow direct interaction with UI elements, such as Citrix.

# Window Element



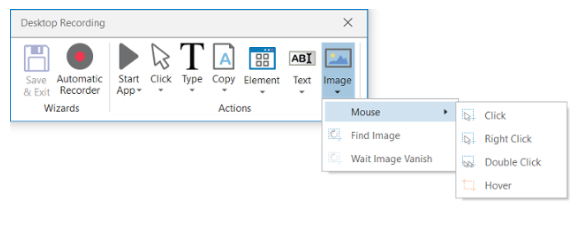
Window element actions enable you to shift the focus to a different window, as well as close windows. Studio does the latter by hooking in the operating system to make sure the application is closed.

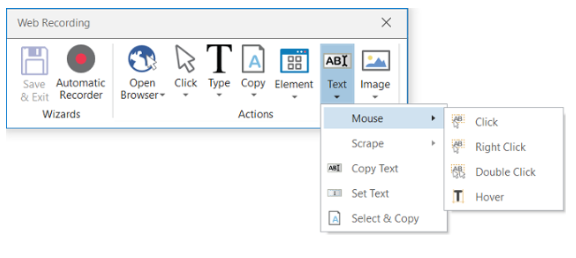
# Text

Text single actions enable you to select or hover over text to make tooltips visible for scraping, right-click to make the context menu visible, copy and paste text and many others.

# Image

Image single actions enable you to wait for a specified image to disappear, to find a specific image in an app or website, right-click or hover over an image and others. This type of manual recording can be useful with UI elements that are saved as graphics, as they cannot be highlighted as normal fields or text.





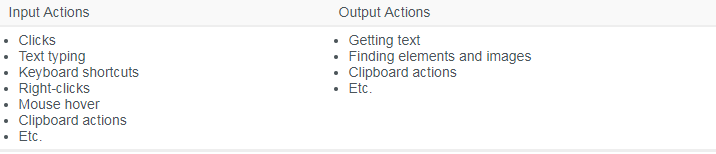
# About UI Elements

[**SUGGEST EDITS**](https://studio.uipath.com/docs/about-ui-elements)

UI elements refer to all graphical user interface pieces that construct an application, be they windows, check boxes, text fields or drop-down lists, and so on. Knowing how to interact with them enables you to implement UI automation much faster and easier.

It is possible to create automations with UI elements from most applications, including Universal Windows Platform apps.

All interactions with the UI can be split into input and output. This categorization helps you better understand which actions to use in different scenarios, when to use them, and the technology behind them. These are also going to be useful when dealing with scraping.



# UI Activities Properties

[**SUGGEST EDITS**](https://studio.uipath.com/docs/ui-activities-properties)

There are multiple activities that can be used to automate apps or web-apps and you can find them in the **Activities** panel, under the **UI Automation** category.

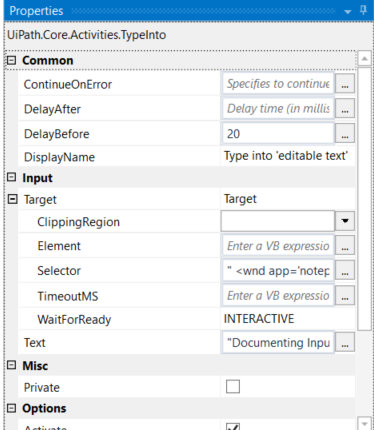
All of these activities have multiple properties in common:

* **ContinueOnError** – specifies if the automation should continue, even if the activity throws an error. This field only supports boolean values (True, False). The default value in this field is False. As a result, if this 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.

### Note:

If the **ContinueOnError** field of an activity inside a [**Try Catch**](https://activities.uipath.com/docs/try-catch) is set to True, no error is caught when the project is executed.

* **DelayAfter** – adds a pause after the activity, in milliseconds.
* **DelayBefore** – adds a pause before the activity, in milliseconds.
* **TimeoutMS** – specifies the amount of time (in milliseconds) to wait for a specified element to be found before an error is thrown. The default value is 30000 milliseconds (30 seconds).
* **WaitForReady** - Before performing the actions, wait for the target to become ready. The following options are available:
  + **None** - Does not wait for anything except the target UI element to exist before executing the action. For example, you can use this option if you want to retrieve just text from a web page or click a particular button, without having to wait for all UI elements to load. Note that this may have unwanted consequences if the button relies on elements which are not yet loaded, such as scripts.
  + **Interactive/Complete** - Waits all of the UI elements in the target app to exist before actually executing the action.  
    To assess if an application is in the Interactive or Complete state, the following tags are verified:
  + **Desktop applications** - A wm\_null message is sent to check the existence of the <wnd>, <ctrl>, <java>, or <uia> tags. If they exist, the activity is executed.
  + **Web applications:**  
    a. **Internet Explorer** - The <webctrl> tag is used to check if the **Ready** state of the HTML document is set to **Complete**. Additionally, the **Busy** state has to be set to "False".  
    b. **Others** - The <webctrl> tag is used to check if the **Ready** state of the HTML document is **Complete**.
  + **SAP applications** - First the presence of the <wnd> tag verified, after which a SAP specific API is used to detect if the session is busy or not.
* **Target** – identifies the UI element the activity works with.



The target is composed of multiple pieces, namely the container, selector and clipping region, to ensure that you correctly identify a UI element.

A container gives you a little more context for the button or field you want to use, so that you can tell windows apart or different areas of the same app. They are automatically generated, but you can make changes to them in the **Properties** panel.

The following are containers:

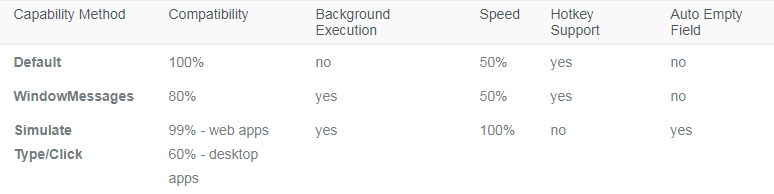
* [**Attach Window**](https://activities.uipath.com/docs/window-scope)
* [**Open Application**](https://activities.uipath.com/docs/open-application)
* [**Attach Browser**](https://activities.uipath.com/docs/browser-scope)
* [**Open Browser**](https://activities.uipath.com/docs/open-browser)
* [**Get Active Window**](https://activities.uipath.com/docs/get-active-window)

# Input Methods

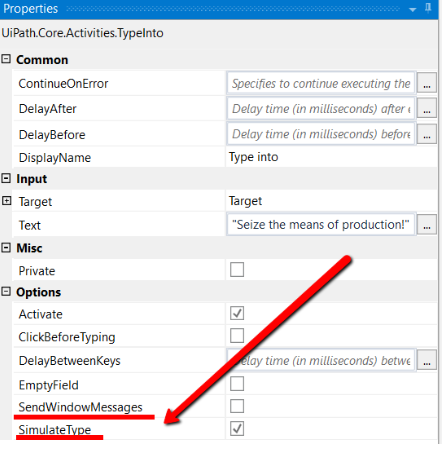
[**SUGGEST EDITS**](https://studio.uipath.com/docs/input-methods)

Input actions require you or the robot to directly interact with an opened application or web page. There are three types of input methods for click and type actions, that differ in terms of compatibility and capability.

We generally recommend the **Simulate Type/Click** method as it is the fastest of the three and works in the background, but only if you do not need to send special keyboard shortcuts. If this does not work for you, try the **Windows Messages** method and then the **Default** one, as it is the slowest.



The input method can be changed at any point from the **Properties** panel of the selected activity. If the **SimulateType** or **SendWindowMessages** check boxes are not selected, then the **Default** method is applied.



The **Default** application simulates a click or type with the help of the hardware driver, while the **Simulate Type/Click** method uses the technology of the target application. Lastly, the **SendWindowMessages** works by sending a specific message directly to the target application.

# Example of Using Input Methods

[**SUGGEST EDITS**](https://studio.uipath.com/docs/example-of-using-input-methods)

To get a clearer picture of how these three methods work, let’s create a simple project that writes something in a Notepad window and switch between the three input methods.

1. Open a Notepad window.
2. In Studio, from the **Basic Recording** toolbar, start the automatic recorder.
3. Type something into the Notepad window.
4. Minimize the window and restore it.
5. Type something else in the Notepad window.
6. Press Esc two times. The automation is saved and displayed in the **Designer** panel. It should look as in the following screenshot.
7. Move the activity that restores the Notepad window after the second [**Type Into**](https://activities.uipath.com/docs/type-into) one. We do this to check if the type of input method selected can also write to Notepad in background mode.
8. From the drop-down of the second **Type Into** activity, select **enter**. A special key string is displayed at the end of the previously-existing text.
9. Copy the special key string at the beginning of the sentence. This enables you to test special keys, such as Enter that adds a new line in a text editor.
10. Run the workflow once with every input method. Note the differences:
    * The **Default** method – it does not automatically erase previously written text, supports special keys, but writing in the background does not work;
    * The **Window Messages**method – works in the background, supports special keys, but it does not erase pre-existing text (you have to manually select the **Empty Field** check box in the **Properties** panel);

* The **Simulate Type/Click** method – works in the background, but it automatically erases pre-existing text, and does not support special keys.

Therefore, be careful to choose the method that best suits your needs. If special keys are a must, you might want to avoid the **Simulate Type/Click** method, or if speed is what matters most, then maybe **Simulate Type/Click** is the right one.

# Output or Screen Scraping Methods

[**SUGGEST EDITS**](https://studio.uipath.com/docs/output-or-screen-scraping-methods)

Output or screen scraping methods refer to those activities that enable you to extract data from a specified UI element or document, such as a .pdf file.

To understand which one is better for automating your business process, let’s see the differences between them.

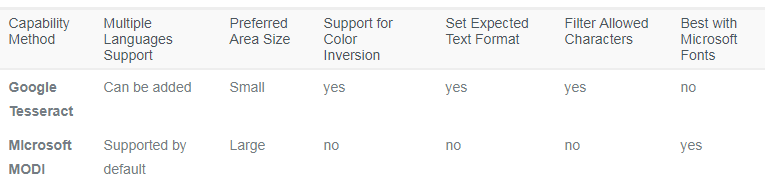


**FullText** is the default method, it is fast and accurate, yet unlike the **Native** method, it cannot extract the screen coordinates of the text.

Both these methods work only with desktop applications, but the **Native** method only works with apps that are built to render text with the Graphics Device Interface (GDI).

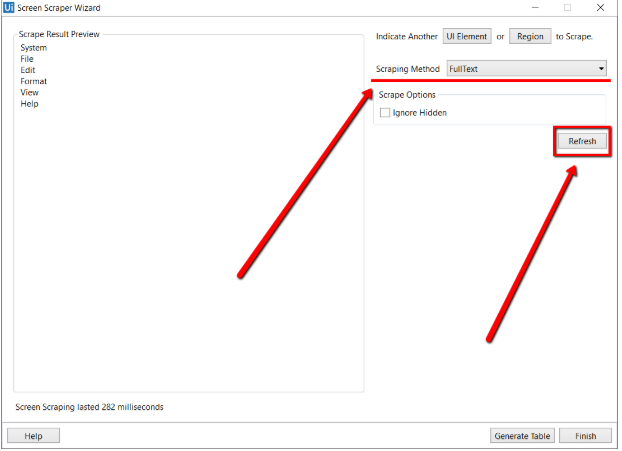
**OCR** is not 100% accurate, but can be useful to extract text that the other two methods could not, as it works with all applications including Citrix. Studio uses two OCR engines, by default: Google Tesseract and Microsoft Modi.

Languages can be changed for OCR engines and you can find out how to [Install OCR Languages here](https://studio.uipath.com/docs/installing-ocr-languages)



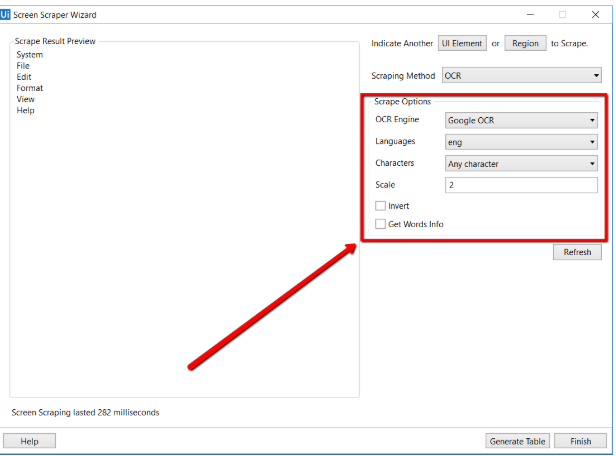
o start extracting text from various sources, click the **Screen Scraping** button, in the **Wizards** group, on the **Design** ribbon tab.

The screen scraping wizard enables you to point at a UI element and extract text from it, using one of the three output methods described above. Studio automatically choses a screen scraping method for you, and displays it at the top of the **Screen Scraper Wizard** window.



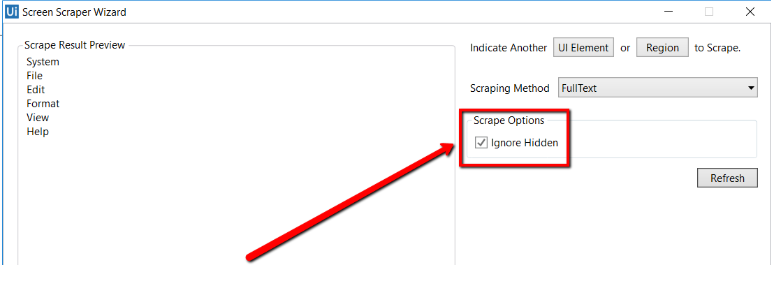
To change the method of screen scraping, select another one from the **Options** panel and then click **Refresh**.

When you are satisfied with the scraping results, click **Finish**. The latter option copies the extracted text to the Clipboard, while the former saves your information to the **Designer** panel. Just like [desktop recording](https://studio.uipath.com/docs/example-of-automatic-recording-with-basic-and-desktop), screen scraping generates a container (with the selector of the top level window) which contains activities, and partial selectors for each activity.

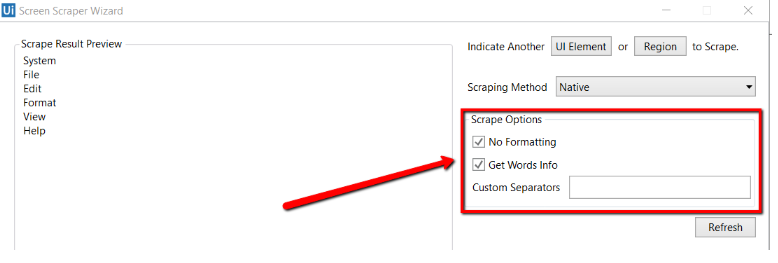


Each type of screen scraping comes with different features in the **Screen Scraper Wizard**, in the **Options** panel:

1. **FullText**

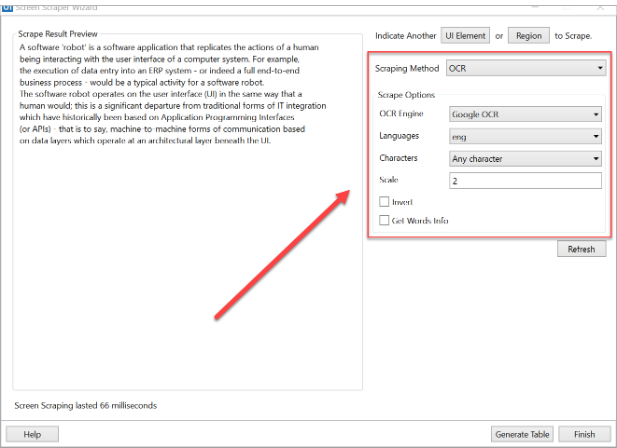
*   
  **Ignore Hidden** – when this check box is selected, the hidden text from the selected UI element is not copied.

1. **Native**



* **No Formatting** – when this check box is selected, the copied text does not extract formatting information from the text. Otherwise, the extracted text’s relative position is retained.
* **Get Words Info** – when this check box is selected, Studio also extracts the screen coordinates of each word. Additionally, the **Custom Separators** field is displayed, that enables you to specify the characters used as separators. If the field is empty, all known text separators are used.

1. **Google OCR**

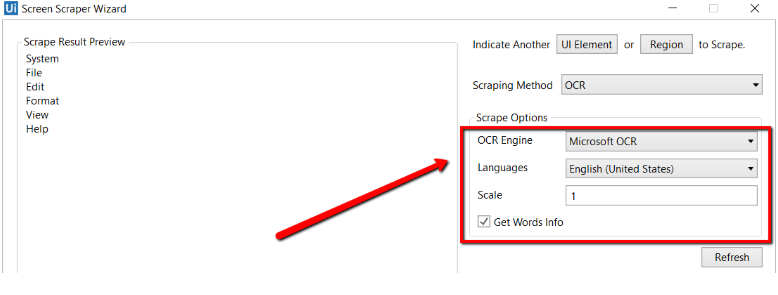


* **Languages** – only English is available by default.
* **Characters** – enables you to select which types of characters to be extracted. The following options are available: **Any character**, **Numbers only**, **Letters**, **Uppercase**, **Lowercase**, **Phone numbers**, **Currency**, **Date** and **Custom**. If you select **Custom**, two additional fields, **Allowed** and **Denied**, are displayed that enable you to create custom rules on which types of characters to scrape and which to avoid.
* **Invert** – when this check box is selected, the colors of the UI element are inverted before scraping. This is useful when the background is darker than the text color.
* **Scale** – the scaling factor of the selected UI element or image. The higher the number is, the more you enlarge the image. This can provide a better OCR read and it is recommended with small images.
* **Get Words Info** – gets the on-screen position of each scraped word.

### Note:

In some instances of UiPath Studio, the Google Tesseract engine may have training files (about training files: [Wikipedia](https://en.wikipedia.org/wiki/Test_set), [GitHub](https://github.com/tesseract-ocr/tesseract/wiki/Training-Tesseract#introduction)) that do not work for certain non-English languages. Running a project with these corrupted training files may lead to an exception being thrown. To fix this issue, download the training file for the language you wish to use from [here](https://github.com/tesseract-ocr/tessdata/tree/bf82613055ebc6e63d9e3b438a5c234bfd638c93) and copy it into the tessdata folder from the UiPath installation directory. To check if the training files you downloaded work, you can download this [test project](https://www.uipath.com/hubfs/Documentation/WorkflowExamples/TestOcrLanguages.zip).

1. **Microsoft OCR**



* **Languages** – enables you to change the language of the scraped text. By default, English is selected.
* **Scale** – the scaling factor of the selected UI element or image. The higher the number is, the more you enlarge the image. This can provide a better OCR read and it is recommended with small images.
* **Get Words Info** - gets the on-screen position of each scraped word.

Besides getting text out of an indicated UI element, you can also extract the value of multiple types of attributes, its exact screen position, and its ancestor.

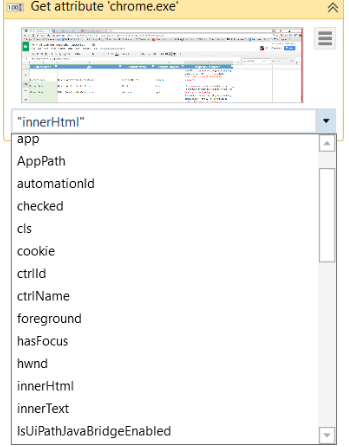
This type of information can be extracted through dedicated activities that are found in the **Activities** panel, under **UI Automation > Element > Find** and **UI Automation > Element > Attribute**.

These activities are:

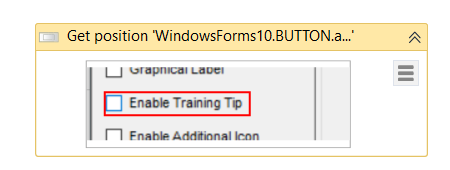
* [**Get Ancestor**](https://activities.uipath.com/docs/get-ancestor) – enables you to retrieve an ancestor from a specified UI element. You can indicate at which level of the UI hierarchy to find the ancestor, and store the results in a UiElement variable.



* [**Get Attribute**](https://activities.uipath.com/docs/get-attribute) – retrieves the value of a specified UI element attribute. Once you indicate the UI element on screen, a drop-down list with all available attributes is displayed.



* [**Get Position**](https://activities.uipath.com/docs/get-position) – retrieves the bounding rectangle of the specified UiElement, and supports only Rectangle variables.



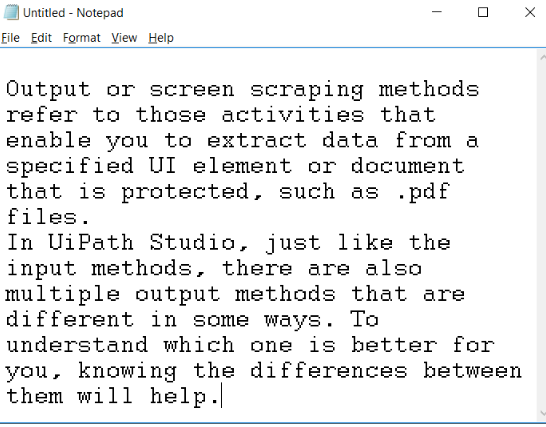
UiPath Studio also features **Relative Scraping**, a scraping method that identifies the location of the text to be retrieved relative to an anchor. You can find more about it [here](https://studio.uipath.com/docs/relative-scraping).

You can also generate tables from unstructured data and store the information in DataTable variables, by using the **Screen Scraping Wizard**. For more information, see [Generating Tables from Unstructured Data](https://studio.uipath.com/docs/generating-tables-from-unstructured-data).

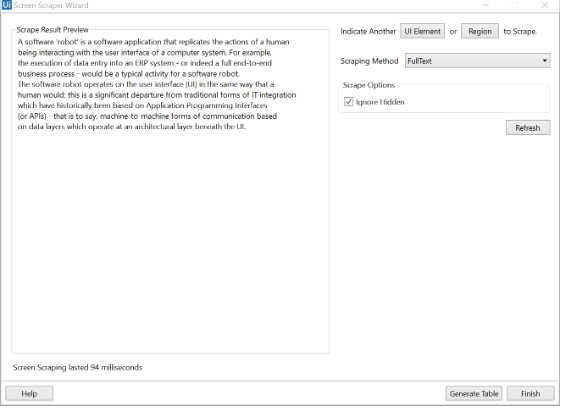
# Examples of Using Output or Screen Scraping Methods

[**SUGGEST EDITS**](https://studio.uipath.com/docs/examples-of-using-output-or-screen-scraping-methods)

To exemplify how to use the several screen scraping methods and the practical differences between them, let’s first scrape a Notepad window with some text and see what results we have. The following screenshot is what we used.

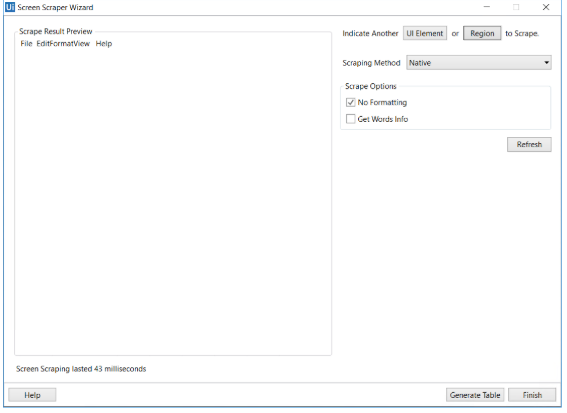


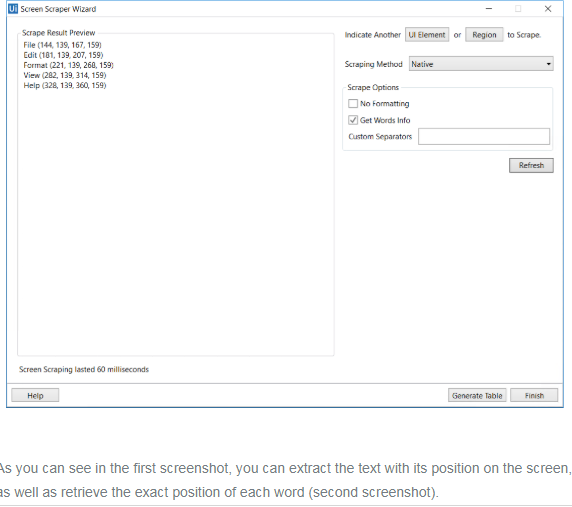
# The FullText method

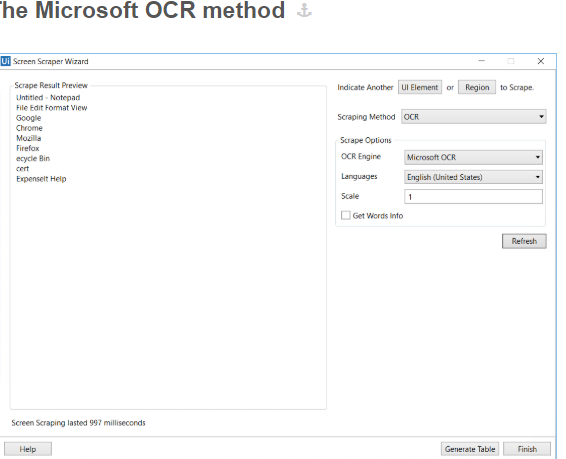


As you can see, no formatting is retained, but if you hide the Notepad window while scraping, the text is still retrieved. This is the fastest method.

# The Native method

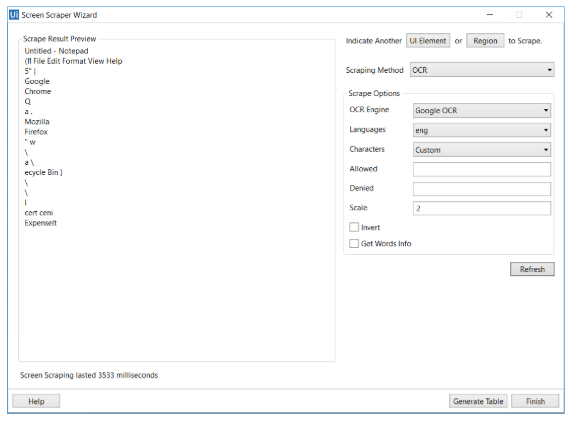






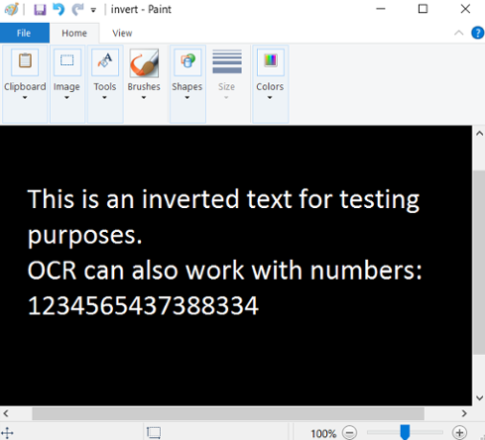
As you can see, the accuracy of this output method is not 100%, but it still manages to keep the position of the text. Getting the exact on-screen position, in pixels, is also available yet as you can see, it is not the fastest of the output methods.

# The Google OCR method

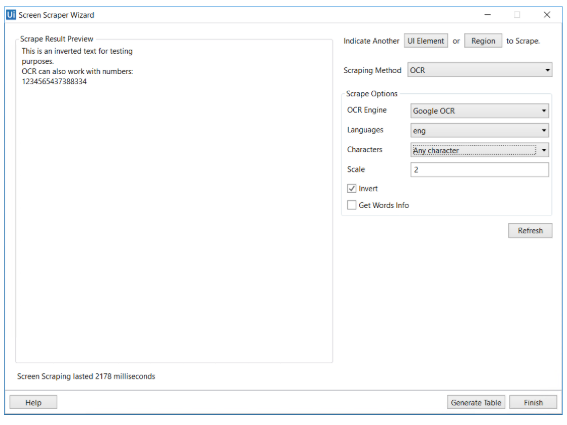


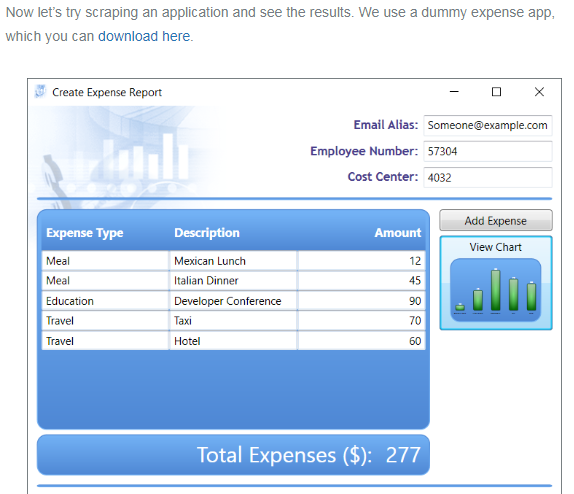
As with Microsoft’s Modi, the Google OCR method is not 100% accurate and takes longer when compared with the others. However, it retrieves the position within the window of the text.

Now, add some white text over a black page in Paint, for example, and try to scrape it.



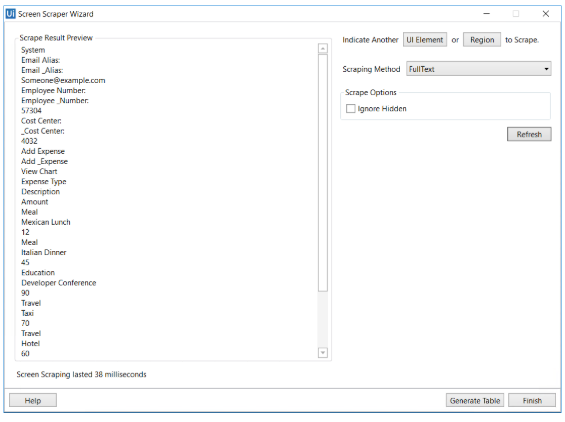
As you can see, only the OCR methods work in this scenario.



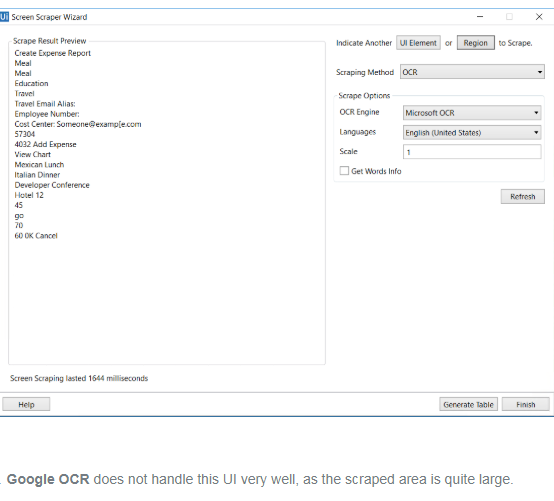


If we scrape this entire window, we receive the following results:

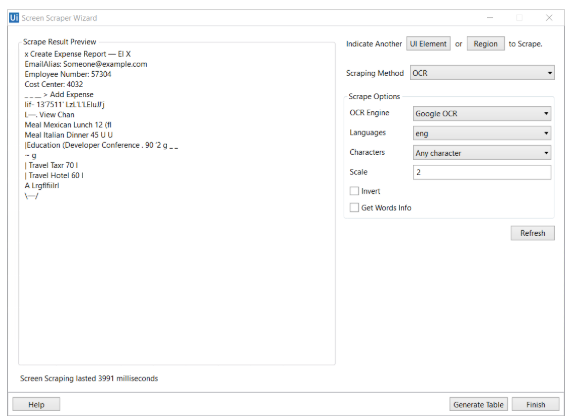
1. **FullText** with hidden text works really well, being able to read even the minimize and restore buttons.



1. **Native** does not work on this UI as it does not make use of the graphical device interface to render text. For more information on GDI, please see the official [Microsoft documentation](https://goo.gl/a3EGMc).
2. **Microsoft OCR** works pretty well, although accuracy is still not 100%



1. **Google OCR** does not handle this UI very well, as the scraped area is quite large.



# Generating Tables from Unstructured Data

[**SUGGEST EDITS**](https://studio.uipath.com/docs/generating-tables-from-unstructured-data)

You can also generate tables from unstructured data and store the information in DataTable variables, by using the **Screen Scraping Wizard**. Variables can then be used further in your automation projects to populate your database, an Excel sheet or many others.

### Note:

It is recommended to run your web automations on Internet Explorer 11 and above, Mozilla Firefox 50 or above, or the latest version of Google Chrome.

To create your table from unstructured data you can use the following whitespace characters as column and line separators:

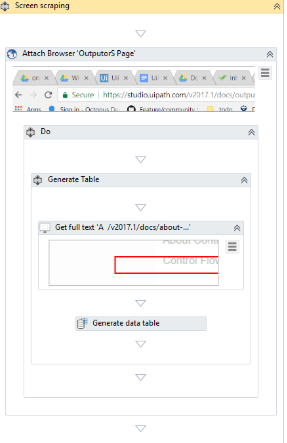
* space
* tab
* newline  
  Please note that you can use more than one option for each type of separator.

In addition, you can:

* automatically detect column types;
* use the first row as column headers;
* ignore the first column;
* use predefined columns (if they exist).

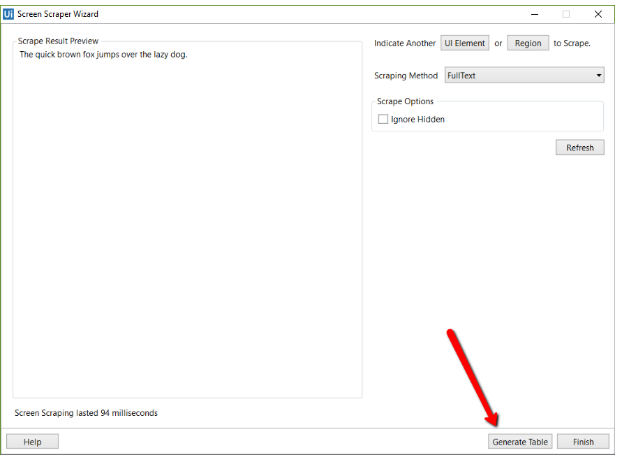
In the **Designer** panel, the result of the screen scraping contains the following, in sequential order:

* a **Screen Scraping** sequence
* an [**Attach Window**](https://activities.uipath.com/docs/window-scope) or [**Attach Browser**](https://activities.uipath.com/docs/browser-scope) activity
* a **Generate Table** container
* a [**Get Full Text**](https://activities.uipath.com/docs/get-full-text), [**Get Visible Text**](https://activities.uipath.com/docs/get-visible-text) or [**Get OCR Text**](https://activities.uipath.com/docs/get-ocr-text) activity, depending on the method used to scrape the unstructured data
* a [**Generate Data Table**](https://activities.uipath.com/v1.0/docs/generate-data-table) activity  
  The following screenshot is an example:

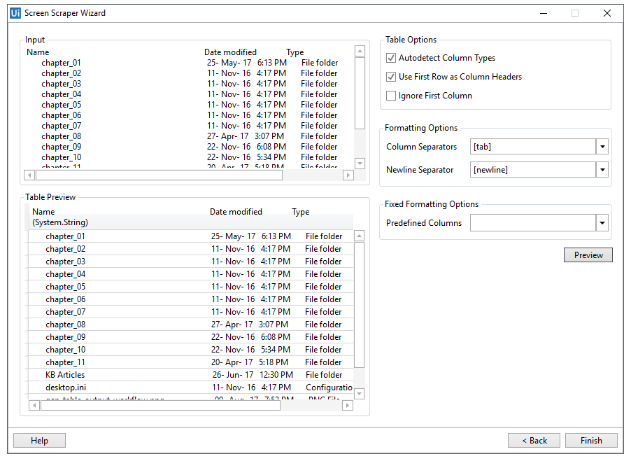


To take advantage of this feature, do the following:

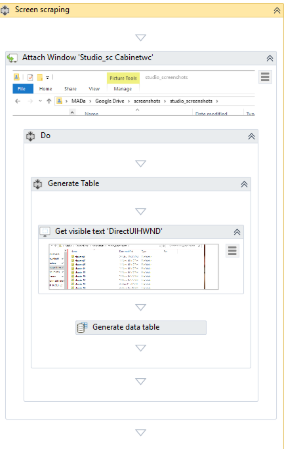
1. Open the **Screen Scraping Wizard**.
2. Point to the information you want to extract, such as a web page or a PDF. A preview of the scraped information is displayed in the wizard.



1. Click **Generate Table**. The next step of the wizard is displayed, where you can actually start tweaking your soon to be table.



1. Select the column and newline separators you want and tweak the rest of the parameters to your liking.
2. Click **Preview**. The **Table Preview** section is updated, displaying how your data could be arranged.
3. Perform steps 4 and 5 until you get the desired outcome.
4. Click **Finish**. The auto-generated automation project is displayed in the **Designer** panel.



# Relative Scraping

[**SUGGEST EDITS**](https://studio.uipath.com/docs/relative-scraping)

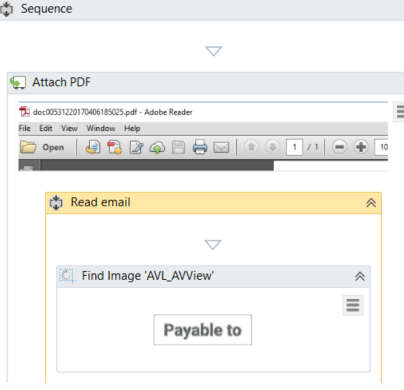
**Relative Scraping** is a technique that enables you to retrieve text from UI elements by using OCR technology. In situations where selectors cannot be found, the target UI objects are identified by using image recognition activities to look for adjacent labels or other elements.

This technique is useful in retrieving text from certain UI elements that are difficult to access by using normal means, such as applications in virtual environments. Using visual labels of UI elements makes up for the inability to find selectors.

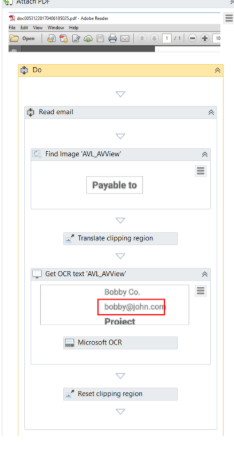
To use the **Scrape Relative** functionality, do the following:

1. Start the **Citrix Recording Wizard**.
2. Click **Screen Scraping > Scrape Relative**.
3. Select an **anchor**, which is the relative element used to identify the location of the target, such as the label of a text field.
4. 

An [**Attach Window**](https://activities.uipath.com/docs/window-scope) container is generated that sets focus to the app window and contains a [**Find Image**](https://activities.uipath.com/docs/wait-image-appear) activity that locates the position of the anchor on the screen.



1. Indicate the area where the target element is. A **Set Clipping Region** activity is generated, which translates the clipping region to where the target element can be found, relative to the anchor. Additionally, a [**Get OCR Text**](https://activities.uipath.com/docs/get-ocr-text) activity is generated that scrapes the target element. Since the clipping region is a shared resource, the recorder generates another **Set Clipping Region** activity which resets the clipping region, thus avoiding interference with other operations.



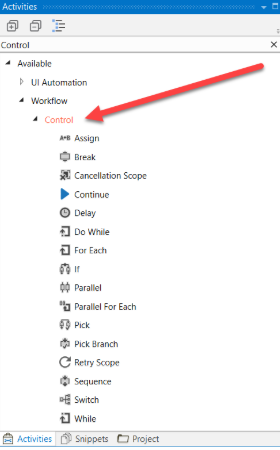
# About Control Flow

[**SUGGEST EDITS**](https://studio.uipath.com/docs/about-control-flow)

An important aspect of successfully working with UiPath Studio is understanding and knowing how to control your project. **Control Flow** is a concept borrowed from computer science that refers to the order in which actions are performed in an automation.

A proper **Control Flow** can be achieved through the intelligent use of variables and activities.

All of these activities can be found in the **Activities** panel, under **Workflow > Control**.



These activities enable you to define rules and automate conditional statements within the project, through **if/else** and **for each**statements or **loops**. **Delay** can also be added between actions, in order to perfectly time activities.

**Loops** represent an important part of automations as they enable you to easily check dependencies between variables, activities and conditions. Once created, they enable the user to run a segment of the automation for a specific number of times, until a condition is met, or indefinitely.

# Control Flow Activities



# The Assign Activity

[**SUGGEST EDITS**](https://studio.uipath.com/docs/the-assign-activity)

The [**Assign**](https://activities.uipath.com/docs/assign) activity is an important activity that is going to be used quite often, as it enables you to assign a value to a variable.

You can use an **Assign** activity to increment the value of a variable in a loop (see the example in the [The Do While Activity](https://studio.uipath.com/v2018.1/docs/the-do-while-activity)chapter), sum up the value of two or more variables and assign the result to another variable (see the example in the [Generic Value Variables](https://studio.uipath.com/docs/genericvalue-variables)), assign values to an array (see [Array Variables](https://studio.uipath.com/docs/array-variables)) and so on.

By default, this activity is also included in the **Favorites** group. To remove it, right-click it and select **Remove**.

# The Delay Activity

[**SUGGEST EDITS**](https://studio.uipath.com/docs/the-delay-activity)

The [**Delay**](https://activities.uipath.com/docs/delay) activity enables you to pause the automation for a custom period of time (in the hh:mm:ss format). This activity proves itself quite useful in projects that require good timing, such as waiting for a specific application to start or waiting for some information to be processed so that you can use it in another activity.

# Example of Using the Delay Activity

To exemplify how you can best use the **Delay** activity, let's create an automation that writes two messages to the **Output** panel, with a delay of 20 seconds between them.

1. Create a new flowchart.
2. Add a [**Write Line**](https://activities.uipath.com/docs/write-line) activity and connect it to the **Start** node.
3. Select the activity, and in the **Text** field, type "This is the start time."
4. Add a **Delay** activity and connect it to the previously added activity.
5. Select the activity, and in the **Properties** panel, in the **Duration** field, type 00:00:20. This is the 20 seconds delay that is going to be between the two logged messages.
6. Add another **Write Line** activity and connect it to the previously added activity.
7. In the **Text** field, type "Message delayed by 20 seconds.".  
   The final project should look as in the following screenshot.
8. Press F5. The automation is executed. Note that, in the **Output** panel, the two messages added in the **Write Line** activities are written twenty seconds apart.

# The Do While Activity

[**SUGGEST EDITS**](https://studio.uipath.com/docs/the-do-while-activity)

The [**Do While**](https://activities.uipath.com/docs/do-while) activity enables you to execute a specified part of your automation while a condition is met. When the specified condition is no longer met, the project exits the loop.

This type of activity can be useful to step through all the elements of an array, or execute a particular activity multiple times. You can increment counters to browse through array indices or step through a list of items.

### Note:

**Do While** activities are evaluated only after the body has been executed once.

# Example of Using a Do While Activity

To exemplify how to use a **Do While** activity, let’s create an automation that increments an integer variable from 0 to 10, and displays a message every time it is incremented.

1. Create a new sequence.
2. Create an integer variable, intCounter, with a default value of 0.
3. Add a **Do While** activity to the **Designer** panel.
4. In the **Body** section, add an [**Assign**](https://activities.uipath.com/docs/assign) activity.
5. Select the **Assign** activity, and in the **Properties** panel, in the **To** field, add the intCounter variable.
6. In the **Value** field, type intCounter + 1. This helps you increment the value of the variable with one.
7. Add a [**Write Line**](https://activities.uipath.com/docs/write-line) activity, under the **Assign** one.
8. In the **Text** field, type "The counter is " + intCounter.ToString. This writes the value of the counter in the **Output** panel each time it is incremented.
9. In the **Condition** section of the **Do While**activity, type intCounter < 10. The body of the **Do While** activity is repeated until the value of the intCounter variable is bigger than 10.  
   The project should look as in the following screenshot.
10. Press F5. The automation is executed. Note that the **Output** panel displays the message indicated in the **Write Line** activity.

# The If Activity

[**SUGGEST EDITS**](https://studio.uipath.com/docs/the-if-activity)

The [**If**](https://activities.uipath.com/docs/if) activity contains a statement and two conditions. The first condition (the activity in the **Then** section) is executed if the statement is true, while the second one (the activity in the **Else** section) is executed if the statement is false.

**If** activities can be useful to make decisions based on the value of variables.

### Note:

The **If** activity is almost identical to the [**Flow Decision**](https://activities.uipath.com/docs/flow-decision) one. However, the latter can only be used in flowcharts.

# Example of Using an If Activity

To exemplify how you can use the **If** activity, let’s create an automation that asks the user for two numbers, checks to see if one is divisible by the other, and depending on the result, displays a different message in the **Output** panel.

1. Create a new sequence.
2. Create two integer variables, intFirstNumber and intSecondNumber for example.
3. Add two [**Input Dialog**](https://activities.uipath.com/docs/input-dialog) activities to the **Designer** panel.
4. In the **Properties** panel, type labels and titles for both activities and, in the **Result** fields, add the intFirstNumber and intSecondNumber variables.
5. Add an **If** activity to the **Designer** panel, under the previously added **Input Dialog** ones.
6. In the **Condition** section, type intFirstNumber mod intSecondNumber = 0. This expression checks if the first number is divisible to the second one, using the [mod](https://msdn.microsoft.com/en-us/library/se0w9esz.aspx)operator.
7. In the **Then** section, add a [**Write Line**](https://activities.uipath.com/docs/write-line) activity.
8. In the **Text** field, type intFirstNumber.ToString + " is divisible by "+ intSecondNumber.ToString + ".". This is the message that is displayed if the first number is divisible by the second one.
9. In the **Else** section, add another **Write Line** activity.
10. In the **Text** field, type intFirstNumber.ToString + " is NOT divisible by "+ intSecondNumber.ToString + ".". This is the message that is displayed if the first number is not divisible with the second one.
11. Press F5. The automation is executed.
12. Add numbers when prompted. Note that the **Output** panel displays the result, depending on the values added in the **Input Dialog** windows.

# The Switch Activity

[**SUGGEST EDITS**](https://studio.uipath.com/docs/the-switch-activity)

The [**Switch**](https://activities.uipath.com/docs/switch) activity enables you to select one choice out of multiple, based on the value of a specified expression.

By default, the **Switch** activity uses the integer argument, but you can change it from the **Properties** panel, from the **TypeArgument** list.

The **Switch** activity can be useful to categorize data according to a custom number of cases. For example, you can use it to store data into multiple spreadsheets or sort through names of employees.

# Example of Using a Switch Activity

To exemplify how to use the **Switch** activity, we are going to create an automation that asks the user for a number, checks if is odd or even, and depending on that, a different message is written to the **Output** panel.

Since all odd numbers divided by two have a remainder equal to 1, this project needs only two cases (0 and 1) , yet keep in mind that this activity supports multiple cases.

1. Create a new sequence.
2. Create an integer variable, intNumber.
3. Add an [**Input Dialog**](https://activities.uipath.com/docs/input-dialog) activity to the **Designer** panel.
4. Add a **Title** and **Label** to prompt the user for a number.
5. In the **Result** field, add the intNumber variable.
6. Add a **Switch** activity, under the **Input Dialog**.
7. In the **Expression** field, type intNumber mod 2. This verifies if the user’s number is divisible by 2.
8. In the **Default** section, add a [**Write Line**](https://activities.uipath.com/docs/write-line) activity.
9. In the **Text** field, type intNumber.ToString + " is an even number.".
10. Click the **Add new case** line, and in the **Case Value** field, type 1.
11. Add a **Write Line** activity to this case.
12. In the **Text** activity, type intNumber.ToString + " is an odd number.".

The final project should look as in the following screenshot.

1. Press F5. The automation is executed. Note that the **Output** panel displays the data correctly.

# The While Activity

[**SUGGEST EDITS**](https://studio.uipath.com/docs/the-while-activity)

The [**While**](https://activities.uipath.com/docs/while) activity enables you to execute a specific process repeatedly, while a specific condition is met. The main difference between this and the [**Do While**](https://activities.uipath.com/docs/do-while) activity is that, in the first one, the condition is evaluated before the body of the loop is executed.

This type of activity can be useful to step through all the elements of an array, or execute a particular activity multiple times. You can increment counters to browse through array indices or step through a list of items.

# Example of Using a While Activity

To exemplify how to use a **While** activity, let’s create an automation that increments an integer variable from 10 to 100, and writes all the numbers to a Microsoft Word document.

1. Create a new sequence.
2. Create an integer variable, intCounter, with the default value of 10.
3. Add a **While** activity to **Designer** panel.
4. In the **Condition** field, type intCounter < 100. This means that the body of the loop is going to be repeated until the value of the intCounter variable is going to be bigger than 100.
5. In the **Body** section of the **While** activity, add an [**Assign**](https://activities.uipath.com/docs/assign) activity.
6. In the **Properties** panel, in the **To** field add the intCounter.
7. In the **Value** field, type intCounter + 1. This increments the value of the intCounter with one.
8. Add an [**Append Text**](https://activities.uipath.com/docs/word-append-text) activity under the **Assign** one.

### Note:

This activity is part of the Word activities package. If you do not have it, use the [package manager functionality](https://studio.uipath.com/docs/managing-activities-packages#section-managing-packages) to install it.

1. In the **FilePath** field, type the path of a Word document in between quotation marks.

### Note:

Make sure that the Word document is not used when running the automation, otherwise a message error is displayed and the execution is stopped.

1. In the **Text** field, type "The counter is now " + intCounter.ToString +".".  
   The final project should look as in the following screenshot.
2. Press F5. The automation is executed.
3. Double-click the Word document specified at step 9. Note that all the numbers between 10 and 100 are written, as expected.

# The For Each Activity

[**SUGGEST EDITS**](https://studio.uipath.com/docs/the-for-each-activity)

The [**For Each**](https://activities.uipath.com/docs/for-each) activity enables you to step through arrays, lists, data tables or other types of collections, so that you can iterate through the data and process each piece of information individually.

# Example of Using a For Each Activity

To exemplify how to use a **For Each** activity, let’s create an automation that goes through each element of an array of integers and writes the length of the array and each element to the **Output** panel.

1. Create a new sequence.
2. Create an array of integer variables, arrFibonacciNumbers.
3. In the **Default** field, type the Fibonacci sequence up to a desired value, such as {1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89}.
4. Add a **For Each** activity in the **Designer** panel.
5. Do not make any changes to the **Foreach** field.
6. In the **In** field, add the arrFibonacciNumbers variable. This activity looks at each individual item in the provided variable.
7. In the **Body** section of the **For Each** activity, add a [**Write Line**](https://activities.uipath.com/docs/write-line) activity.
8. In the **Text** field, type "The length of this Fibonacci sequence is " + arrFibonacciNumbers.Length.ToString + " and contains the " + item.ToString + " element.". This expression enables you to write the total number of array elements and each element of the array in the **Output** panel.

The final project should look as in the following screenshot.

1. Press F5. The automation is executed. Note that the **Output** panel displays the correct message for each element of the array.

# The Break Activity

[**SUGGEST EDITS**](https://studio.uipath.com/docs/the-break-activity)

The [**Break**](https://activities.uipath.com/docs/break) activity enables you to stop the loop at a chosen point, and then continues with the next activity.

### Note:

The **Break** activity can only be used within the [**For Each**](https://activities.uipath.com/docs/for-each) activity.

# Example of Using a Break Activity

To exemplify how to use the **Break** activity we are going to build upon the project created for the [The For Each Activity](https://studio.uipath.com/docs/the-for-each-activity). This new project writes only the first iteration of the loop and a few elements of the array to the **Output** panel.

1. In the **Body** of the **For Each** activity, under the [**Write Line**](https://activities.uipath.com/docs/write-line), add a **Break** activity.
2. Under the **For Each**, add a new **Write Line** activity.
3. In the **Text** field, type arrFibonacciNumbers(0).ToString + " " + arrFibonacciNumbers(2).ToString + " " + arrFibonacciNumbers(4).ToString + " " +arrFibonacciNumbers(6).ToString + " " + arrFibonacciNumbers(8).ToString + " " +arrFibonacciNumbers(10).ToString + " ". This means that only the indicated elements of the array are going to be written to the **Output** panel.  
   The final project should look as in the following screenshot.
4. Press F5. The automation is executed. Note that the **Output** panel only displays the first iteration of the loop and the specified array elements from the **Write Line**activity.

# Selectors

# About Selectors

[**SUGGEST EDITS**](https://studio.uipath.com/docs/about-selectors)

To automate specific actions in the user interface, you are required to interact with various windows, buttons, drop-down lists and many others. Most RPA products do this by relying on the screen position of UI elements, a method that is not at all dependable.

To overcome this problem, UiPath Studio uses what we call selectors. These store the attributes of a graphical user interface element and its parents, in the shape of an XML fragment.

Most of the times, selectors are automatically generated by Studio and do not require further input from you, especially if the apps you are trying to automate have a static user interface.

However, some software programs have changing layouts and attribute nodes with volatile values, such as some web-apps. UiPath Studio cannot predict these changes and, therefore, you might have to manually generate some selectors.

A selector has the following structure:

<node\_1/><node\_2/>...<node\_N/>

The last node represents the GUI element that interests you, and all the previous ones represent the parents of that element. <node\_1> is usually referred to as a root node, and represents the top window of the app.

Each node has one or more attributes that help you correctly identify a specific level of the selected application.

Each node has the following format:

<ui\_system attr\_name\_1='attr\_value\_1' ... attr\_name\_N='attr\_value\_N'/>

Every attribute has an assigned value. It is important to pick attributes with a constant value. If the value of an attribute changes each time the app is started, then the selector will not be able to correctly identify the element.

Selectors are stored in the **Properties** panel of activities, under **Input > Target > Selector**. All activities related to graphical elements have this property.

# 

# The **Selector Editor** window enables you to see the automatically-generated selector and edit it and its attributes. To access this window, in the **Workflow Designer** panel, click the Options  button in the body of an activity and select **Edit Selector**.

# This can also be done by using the Ellipsis “….” button next to the **Selector** field, in the **Properties** panel.

# 

# 

In some situations, when Studio and the targeted application are opened with different elevated privileges, the selector might not work. We recommend to always open both applications with the same privileges.

# Selectors with Wildcards

[**SUGGEST EDITS**](https://studio.uipath.com/docs/selectors-with-wildcards)

Wildcards are symbols that enable you to replace zero or multiple characters in a string. These can be quite useful when dealing with dynamically-changing attributes in a selector.

* Asterisk (\*) – replaces zero or more characters
* Question mark (?) – replaces a single character

## **Example of Generating a Selector with Wildcards in the Selector Editor Window**

Part of the name of a Notepad window changes according to the .txt file you open with it. This is where a well-placed wildcard can really help. Do the following to generate it:

1. Open an empty Notepad window. Note that the window title is Untitled – Notepad.
2. In Studio, create a new sequence.
3. Drag a [**Type Into**](https://activities.uipath.com/docs/type-into) activity to the **Designer** panel.
4. Click **Indicate on Screen** and indicate the editable text field in Notepad. A selector is automatically generated and stored in the **Selector** field.
5. In the **Properties** panel, click the Ellipsis  button next to the **Selector** field. The **Selector Editor** window is displayed.



1. Open any .txt file with Notepad. Note that the window title is partially different than the one at step 1.
2. In Studio, in the **Selector Editor** window, click **Repair** and indicate the editable text field in Notepad window opened at step 6. A dialog box indicating that the selector was updated is displayed.
3. Click **OK**. The **Selector Editor** window and the selector are updated with a wildcard.

# 

# Full versus Partial Selectors

[**SUGGEST EDITS**](https://studio.uipath.com/docs/full-versus-partial-selectors)

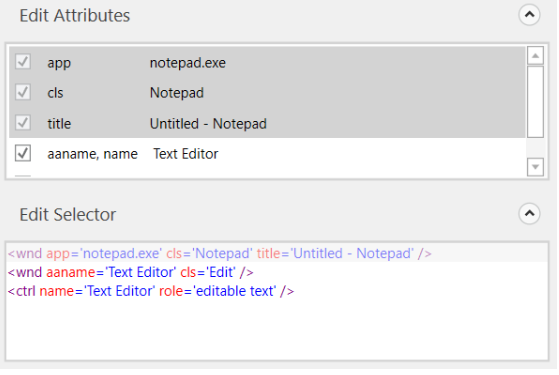
# Full selectors:

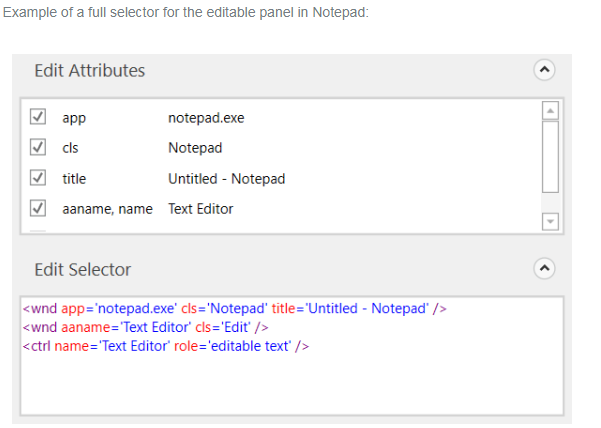
* Contain all the elements needed to identify a UI element, including the top-level window
* Generated by the [Basic recorder](https://studio.uipath.com/docs/about-recording-types)
* Recommended when switching between multiple windows

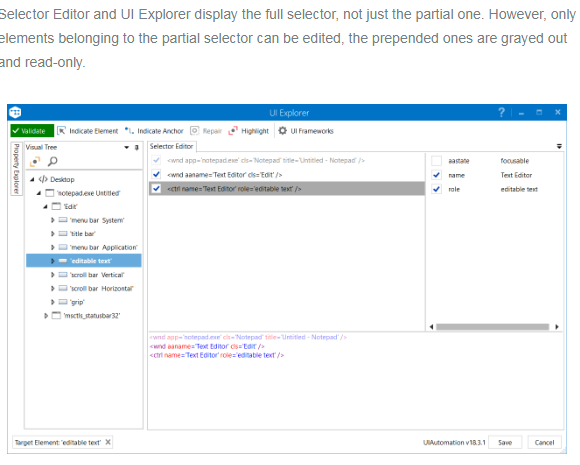
# Partial selectors:

* Generated by the [Desktop recorder](https://studio.uipath.com/docs/about-recording-types)
* Do not contain information about the top-level window
* Activities containing partial selectors are enclosed in a container ([**Attach Browser**](https://activities.uipath.com/docs/browser-scope) or [**Attach Window**](https://activities.uipath.com/docs/window-scope)) that contains a full selector of the top-level window
* Recommended when performing multiple actions in the same window

Example of a partial selector for the editable panel in Notepad:







# UI Explorer

[**SUGGEST EDITS**](https://studio.uipath.com/docs/uipath-explorer)

**UI Explorer** is an advanced tool that enables you to create a custom selector for a specific UI element. It is available only if the UiPath.UIAutomation.Activities package is installed as a dependency to the project.

To open the **UI Explorer** window, click the button in the **Selectors** section, in the **Design** tab.

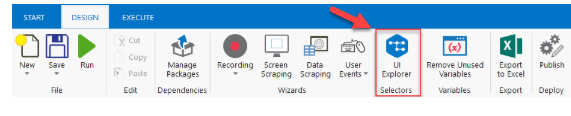
### Note:

If you do not have the **UiPath.UIAutomation.Activities** pack installed as a dependency for the current project, the **UI Explorer** button does not appear in the **Ribbon**.

Alternatively, the **UI Explorer** can be launched from the **Tools** page in the Studio backstage view. UI Explorer from the context menu uses the UI automation libraries shipped with the current version of Studio.

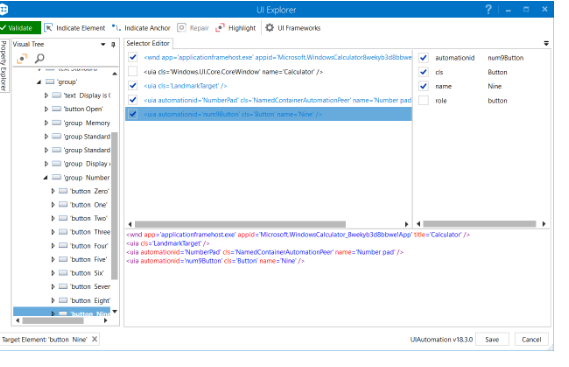
### Note:

The version of the UIAutomation package that is currently used is displayed in the lower right corner in the **UI Explorer** you have opened. This version varies, as launching the **UI Explorer** from the **Tools** page uses the default **UI Automation** version shipped with the Studio version you are using, while opening **UI Explorer** from the **Ribbon** uses the version you have installed as a dependency for the current project.



To be sure that you choose the best selector, remember to:

* Add or remove attributes
* Add parent or children tags
* Use wildcards to replace changing values



# Field Descriptions for the UiPath Explorer Window

**Validate**

The button shows the status of the selector by checking the validity of the selector definition and the visibility of the target element on the screen.  
The **Validate** button has three states:

* Selector is being validated
* Valid selector
* Invalid selector
* Modified selector, revalidate

**Indicate Element**

Indicates a new UI element to replace the previous one.

**Indicate Anchor**

Enables you to choose an anchor relative to the target UI element.

**Repair**

Enables you to re-indicate the same target UI element and repair the selector. This operation does not completely replace the previous selector. The button is available only when the selector is invalid.

**Highlight**

Brings the target element in the foreground. The highlight stays on until it's switched off. The button is enabled only if the selector is valid.

**UI Frameworks**

Changes the technology used to determine UI elements and their selectors. The following options are available:

* Default – UiPath proprietary method. Usually works fine with all types of user interfaces.
* Active Accessibility – an eariler solution from Microsoft for making apps accessible. It is recommended that you use this option with legacy software, when the Default one does not work.
* UI Automation – the improved accessibility model from Microsoft. It is recommended that you use this option with newer apps, when the Default one does not work.

# The Visual Tree Panel

Displays a tree of the UI hierarchy and enables you to navigate through it, by clicking the arrows in front of each node.

By default, the first time when you open **UI Explorer** or after you click the **Reset** button, this panel displays all opened applications, in alphabetical order.

Double-clicking a UI element (or right-clicking and selecting **Set as Target Element**) from the tree, populates the **Selector Editor**, **Selector Attributes** and **Property Explorer** panels.

**Highlight**

Highlights the selected element from the Visual Tree in real time. The highlight stays on until it's switched off.

**Show Search Options**

Displays the search box and search filter options.

**Search Box**

Enables you to look for a specific string. If an exact match is not found, nodes containing the nearest match are displayed.  
Wildcards are supported.  
Depending on the attribute selected from the Search by drop-down list, the search can be case sensitive.  
**Note:** The search only looks for matches in the tree structure under the selected UI object.

**Search by**

Filters your search to a selected attribute or a selector. The contents of this drop-down list change according to the selected UI element.  
**Note:** If Search by is set to Selector, you can only input one node in the <attribute name1='value1' ... /> format.

**Children Only**

Limit your search to the first level children of the selected node. By default, this check box is not selected.

# The Selector Editor Panel

Displays the selector for the specified UI object and enables you to customize it.

The bottom part of the panel displays the actual XML fragment that you have to use in a project. Once you find the selector you want, you can copy it from here and paste it in the **Properties** panel of an activity, in the **Selector** field.

The top part of this panel enables you to view all the nodes in a selector and eliminate the ones that are not necessary by clearing the check box in front of them. An element in the list of selector nodes becomes active when you enable or disable an attribute, or when editing a selector in the bottom panel. Only one node is active at a time.

Selecting a node here displays its attributes in the **Selector Attributes** and **Property Explorer** panels.

# The Selector Attributes Panel

Displays all the available attributes of a selected node (from the **Selector Editor** panel).

You can add or eliminate some of the node attributes by selecting or clearing the check box in front of each attribute.

Additionally, you can change the value of each attribute yet this modification is retained only if the new selector points at the originally selected UI object.

# The Property Explorer Panel

Displays all the attributes that a specified UI object can have, including the ones that do not appear in the selector. They cannot be changed.

# About Data Scraping

[**SUGGEST EDITS**](https://studio.uipath.com/docs/about-data-scraping)

Data scraping enables you to extract structured data from your browser, application or document to a database, .csv file or even Excel spreadsheet.

### Note:

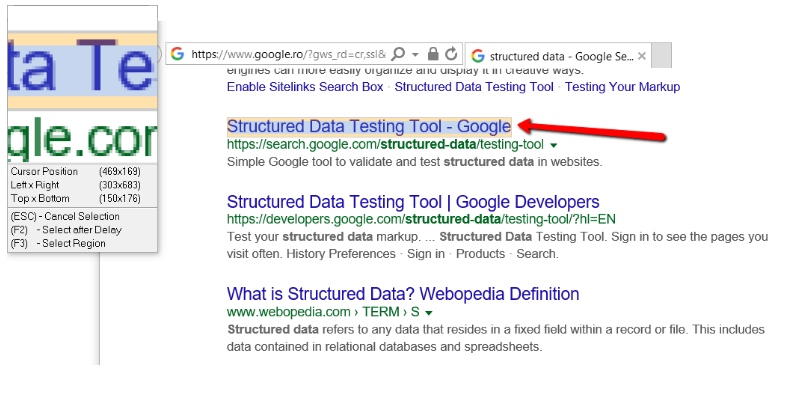
It is recommended to run your web automations on Internet Explorer 11 and above, Mozilla Firefox 50 or above, or the latest version of Google Chrome.

Structured data is a specific kind of information that is highly organized and is presented in a predictable pattern. For example, all Google search results have the same structure: a link at the top, a string of the URL and a description of the web page. This structure enables Studio to easily extract the information, as it always knows where to find it.

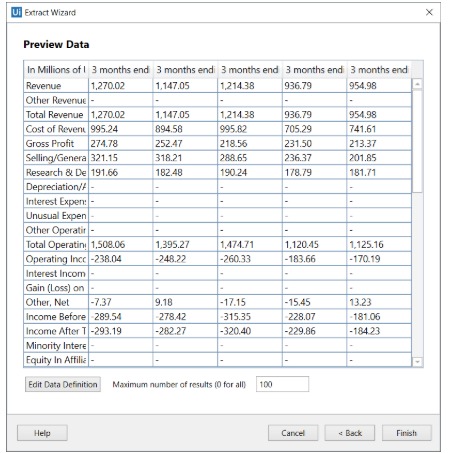
The scraping wizard can be opened from the **Design** tab, by clicking the **Data Scraping** button.

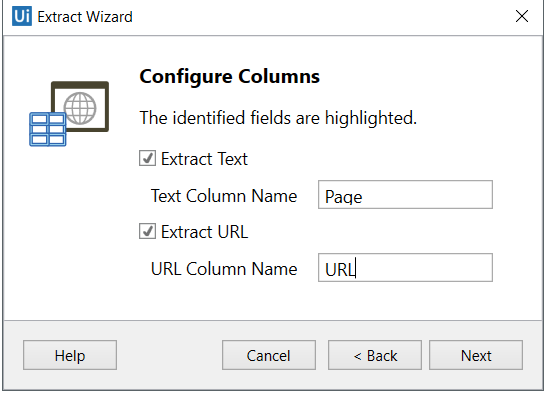
The main steps of the data scraping wizard are:

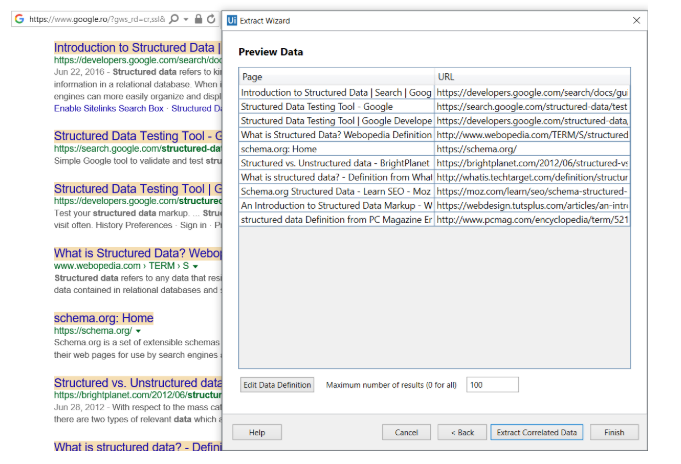
1. Select the first and last fields in the web page, document or application that you want to extract data from, so that Studio can deduce the pattern of the information.



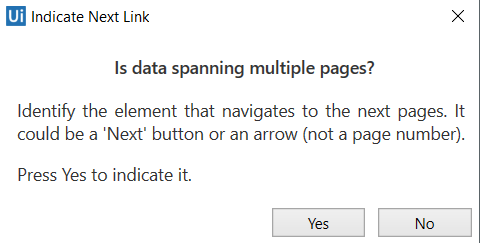
Studio automatically detects if you indicated a table cell, and asks you if you want to extract the entire table. If you click **Yes**, the **Extract Wizard** displays a preview of the selected table data.



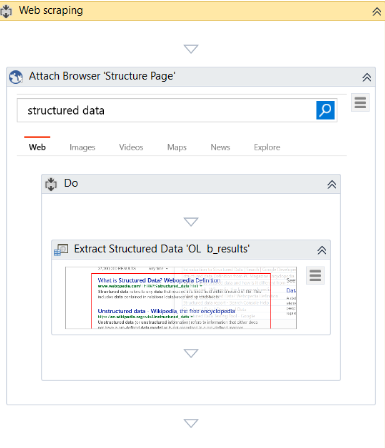
1. Customize column headers and choose whether or not to extract URLs.
2.   
   Preview the data, edit the number of maximum results to be extracted and change the order of the columns.



1. **Optionally** click **Extract Correlated Data**. This enables you to go through the **Extract Wizard** again, to extract additional info and add it as a new column in the same table.
2. Indicate the **Next** button in the web page, application or document (if the information you want to extract spans multiple pages).



After you are finished with the wizard, a project is generated in Studio.



Data scraping always generates a container ([**Attach Browser**](https://activities.uipath.com/docs/browser-scope) or [**Attach Window**](https://activities.uipath.com/docs/window-scope)) with a selector for the top-level window and an [**Extract Structured Data**](https://activities.uipath.com/docs/extract-data) activity with a partial selector, thus ensuring a correct identification of the app to be scraped.

Additionally, the **Extract Structured Data** activity also comes with an automatically generated XML string (in the **ExtractMetadata** property) that indicates the data to be extracted.

Lastly, all the scraped information is stored in a **DataTable** variable, that you can later use to populate a database, a .csv file or an Excel spreadsheet.

# Example of Using Data Scraping

[**SUGGEST EDITS**](https://studio.uipath.com/docs/example-of-using-data-scraping)

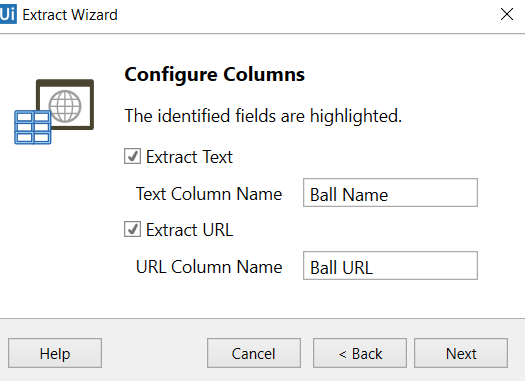
To better understand how you can take advantage of the data scraping functionality, let’s create an automation that extracts some specific information from Amazon.

### Note:

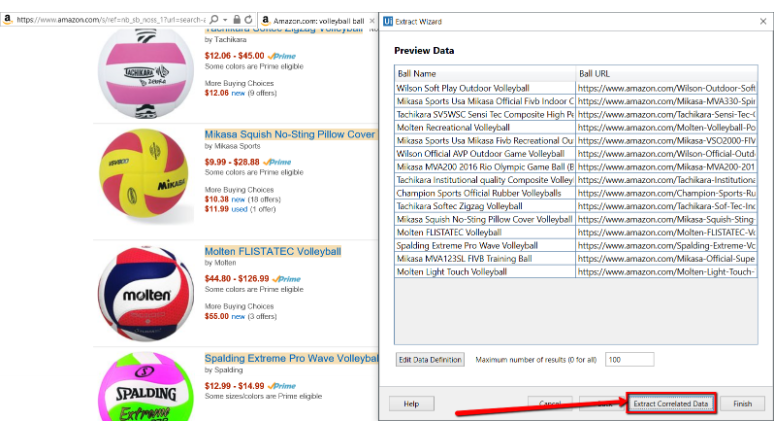
It is recommended to run your web automations on Internet Explorer 11 and above, Mozilla Firefox 50 or above, or the latest version of Google Chrome.

Let’s say you are a sports gear vendor and are interested in finding out the latest prices for volleyball balls on Amazon. You can do the following:

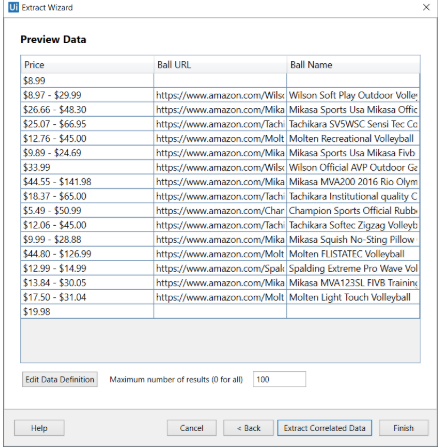
1. Open Internet Explorer and navigate to [www.amazon.com](http://www.amazon.com/).
2. In the search box type "volleyball ball" and press Enter. Results are displayed in the web page.
3. In Studio, on the **Design** tab, in the **Wizards** group, click **Data Scraping**. The **Extract Wizard** is displayed.
4. Following the wizard, select the first and last items in the web page. The **Configure Columns** wizard step is displayed.
5. Select the **Extract URL** check box.
6. Change the name of the column headers.



1. Click **Next**. A preview of the data is displayed and the fields you selected are highlighted in the web browser.



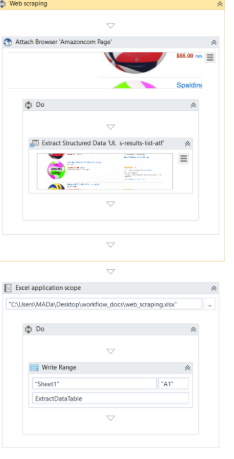
1. Click the **Extract Correlated Data** button. The **Extract Wizard** starts again.
2. Following the wizard again indicate the prices of the items. You get to the **Configure Columns** step.
3. Change the name of the new column, and click **Next**. The data preview is displayed.

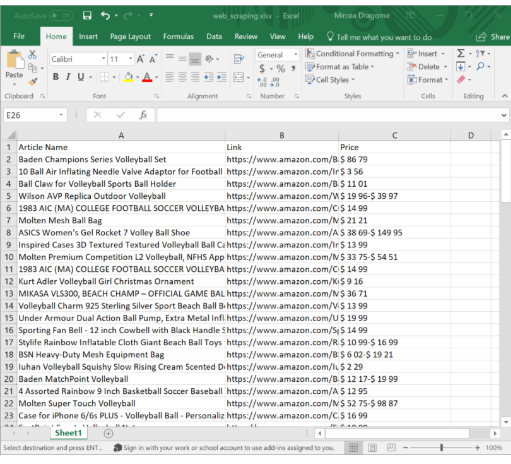


1. (Optionally) Change the order of the columns by dragging them in place.
2. Click **Finish**. The **Indicate Next Link** window is displayed prompting you to indicate the **Next** button if the spans more than one page.
3. Click **Yes** and select the **Next Page** button in Amazon. The project is saved and displayed in the **Designer** panel. Note that a data table variable, **ExtractDataTable**, has been automatically generated.
4. Drag an [**Excel Application Scope**](https://activities.uipath.com/docs/excel-application-scope) activity under the **Data Scraping** container.

### Note:

Install the Excel activities package using the **Manage Packager** to have access to these activities.

1. In the **Properties** panel, in the **WorkbookPath** field, type the file path of an existing Excel file to which you want to write the data.
2. In the **Variables** panel, change the scope of the automatically generated data table variable to **Sequence**.
3. In the **Excel Application Scope**, drag a [**Write Range**](https://activities.uipath.com/docs/write-range) activity.
4. In the **Properties** panel, in the **DataTable** field, add the **ExtractDataTable** variable. The final project should look as in the following screenshot.
5.  Press F5. The automation is executed.
6. Open the Excel file you used at step 15. Note that all columns are populated correctly.



# About Image and Text Automation

[**SUGGEST EDITS**](https://studio.uipath.com/docs/about-image-and-text-automation)

To enable image and text-based process automation, UiPath Studio features activities that simulate **keyboard and mouse** input, such as clicking, hovering or typing, **text recognition** and **OCR** activities that use screen scraping to identify UI elements, and **image recognition** activities that work directly with images to identify UI elements. Specialized recording wizards for Screen Scraping and Citrix recording can also automatically generate the activities required for each process, as explained [here](https://studio.uipath.com/docs/about-recording).

Image and Text automation is useful in situations when UI automation does not work, such as in virtual machine environments, where selectors cannot be found by using normal methods.

This chapter goes through the most important activities in each type of automation, explaining their behaviour and use cases, and then exemplifying three kinds of image and text-based process automation, using the activities described earlier.

# Mouse and Keyboard Activities

[**SUGGEST EDITS**](https://studio.uipath.com/docs/mouse-and-keyboard-activities)

UiPath Studio features activities that simulate any type of keyboard or mouse input that a human would use. Also, there are activities that can set focus to a certain window, minimize or maximize it, or perform any other kind of action on it. These activities are essential in creating an automation that simulates human behaviour. As explained [here](https://studio.uipath.com/docs/input-methods), there are several technologies that can be used for these activities, each with their own advantages in certain situations.

[**Double Click**](https://activities.uipath.com/docs/double-click), [**Click**](https://activities.uipath.com/docs/click), [**Hover**](https://activities.uipath.com/docs/hover) are activities that simulate the clicking or hovering of UI elements. These activities are very useful in situations when human behavior must be mimicked. As input, these activities receive a Target, which can be either a Region variable, a UIElement variable or a selector, that helps you identify what you want to automate and where the actions must be performed. The target can also be automatically generated by using the **Indicate on Screen** functionality, which tries to identify UI elements in the indicated region, and generates selectors for them. If this does not work for you, then manual intervention might be required.

[**Type Into**](https://activities.uipath.com/docs/type-into) sends keystrokes to a UI element. Special keys are supported and can be selected from the drop-down list. This is a basic text input activity that is widely used in automations and is also generated by the automatic recording wizards. As input, this activity receives a string or a string variable that contains the text to be written, and a Target, which can be either a Region variable, a UIElement variable or a selector, that helps you identify what you want to automate and where the actions must be performed. The target can also be automatically generated by using the **Indicate on Screen** functionality, which tries to identify UI elements in the indicated region, and generates selectors for them. If this does not work for you, then manual intervention might be required.

[**Type Secure Text**](https://activities.uipath.com/docs/type-secure-text) sends a secure string to a UI element. As input, this activity receives a SecureString variable that contains the text to be written, and a Target, which can be either a Region variable, a UIElement variable or a selector, that helps you identify what you want to automate and where the actions must be performed. The target can also be automatically generated by using the **Indicate on Screen** functionality, which tries to identify UI elements in the indicated region, and generates selectors for them. If this does not work for you, then manual intervention might be required. This activity is useful for secure automations, as it can use passwords that are stored in SecureString variables. Usually, the SecureString variable is supplied by a [**Get Secure Credential**](https://activities.uipath.com/docs/get-secure-credential) activity.

[**Send Hotkey**](https://activities.uipath.com/docs/send-hotkey) sends keyboard shortcuts to a UI element. This activity is useful for accessing shortcuts in applications and can help you simplify your automation project. For example, you can replace multiple activities that perform UI automation if there is a keyboard shortcut available for revealing certain UI elements or performing specific actions. As input, this activity receives a string or a string variable that contains the keys to be sent, and a Target, which can be either a Region variable, a UIElement variable or a selector, that helps you identify what you want to automate and where the actions must be performed. The target can also be automatically generated by using the **Indicate on Screen** functionality, which tries to identify UI elements in the indicated region, and generates selectors for them. If this does not work for you, then manual intervention might be required.

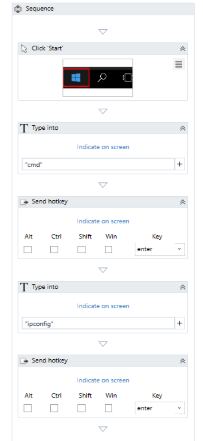
[**Set Focus**](https://activities.uipath.com/docs/set-focus) sets keyboard focus to a specified UI element. The ability to bring a certain window to the foreground is essential when performing image and text automation. As input, this activity receives a Target, which can be either a Region variable, a UIElement variable or a selector, that helps you identify what you want to automate and where the actions must be performed. The target can also be automatically generated by using the **Indicate on Screen** functionality, which tries to identify UI elements in the indicated region, and generates selectors for them. If this does not work for you, then manual intervention might be required.

# Example of using Mouse and Keyboard Automation

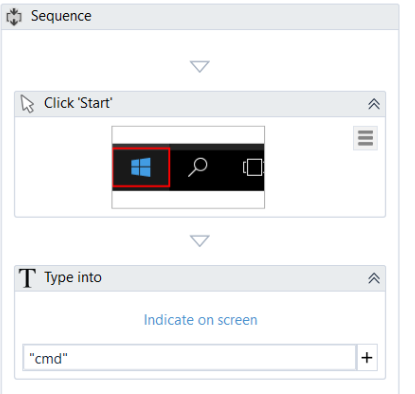
[**SUGGEST EDITS**](https://studio.uipath.com/docs/example-of-using-mouse-and-keyboard-automation)

To exemplify the automation of a process by using activities that simulate mouse and keyboard input, we created an automation that displays the IP address, subnet mask, and default gateway for all adapters from the Command Prompt, by using the ipconfig command and actions similar to human ones:

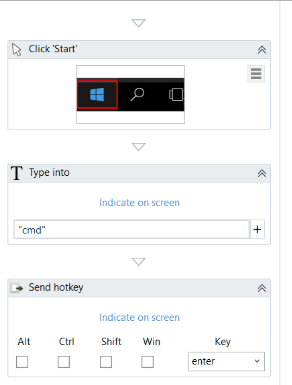
1. Create a new [**Sequence**](https://activities.uipath.com/docs/sequence).
2. Add a [**Click**](https://activities.uipath.com/docs/click) activity to the **Main** panel.
3. Select the activity, click the **Indicate on screen** button and click the **Start** button.



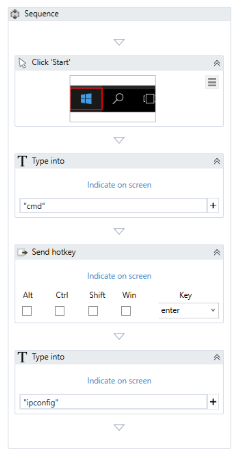
1. Add a [**Type Into**](https://activities.uipath.com/docs/type-into) activity under the previously added one.
2. Select the activity, and in the **Text** field, write cmd.



1. Add a [**Send Hotkey**](https://activities.uipath.com/docs/send-hotkey) activity under the previously added one.
2. Select the activity and, from the drop-down menu choose **enter**.



1. Add another **Type Into** activity under the previously added one.
2. Select the activity, and into the **Text** field, write ipconfig.



1. Add another **Send Hotkey** activity under the previously added one.
2. Select the activity, and from the drop-down menu select the **enter** key.
3. Press F5. The automation is executed. The IP address, subnet mask and default gateway for all adapters are displayed in the Command Prompt.

# Text Activities

[**SUGGEST EDITS**](https://studio.uipath.com/docs/text-activities)

Text recognition activities are useful in extracting text from UI elements on the screen, as well as extracting coordinates for UI elements relative to text on the screen. There are situations when UI elements cannot be identified through standard means, and the Text automation activities featured in UiPath Studio enable you to identify buttons, check boxes and other UI elements based on the text they contain. Text recognition activities share the **Occurrence** property, that enables the user to specify which instance of the text that is being scraped should be acted upon. For example, if the string that is being searched for appears 4 times on the screen, setting the **Occurrence** property to 3 selects the third occurrence of the word(s).

[**Double Click Text**](https://activities.uipath.com/docs/double-click-text), [**Click Text**](https://activities.uipath.com/docs/click-text) and [**Hover Text**](https://activities.uipath.com/docs/hover-text) are activities used to click the text inside a UI element or hover over it. After the user interface object and text are specified, the activity searches the UI for the text and clicks it or hovers over it. As input, these activities receive a Target, which can be either a string variable, a Region variable, a UIElement variable or a selector, which indicate the coordinates where the action must be performed. The target can also be automatically generated by using the **Indicate on Screen** functionality, which tries to identify UI elements in the indicated region, and generates selectors for them. If this does not work for you, then manual intervention might be required.

[**Find Text Position**](https://activities.uipath.com/docs/find-text) searches for a given string in a specified target, and returns a UIElement variable which has the clipping region set to the screen position of that string. This activity can be useful in locating UI elements relative to text on the screen when there is no other way of locating them, and using them further in your automation. As input, this activity receives a Target, which can be either a string variable, a Region variable, a UiElement variable or a selector, that helps you identify what you want to automate and where the actions must be performed. The target can also be automatically generated by using the **Indicate on Screen** functionality, which tries to identify UI elements in the indicated region, and generates selectors for them. If this does not work for you, then manual intervention might be required. The activity outputs the UiElement variable that contains the provided string.

[**Get Full Text**](https://activities.uipath.com/docs/get-full-text) extracts a string and its information from an indicated UI element using the [FullText screen scraping method](https://studio.uipath.com/docs/output-or-screen-scraping-methods). This activity can also be automatically generated when performing screen scraping, along with a container. This activity can be useful in retrieving text from desktop and web applications. As input, this activity receives a Target, which can be either a Region variable, a UiElement variable or a selector, that helps you identify what you want to automate and where the actions must be performed. The target can also be automatically generated by using the **Indicate on Screen** functionality, which tries to identify UI elements in the indicated region, and generates selectors for them. If this does not work for you, then manual intervention might be required. The activity outputs a string variable that contains the extracted text.

[**Get Visible Text**](https://activities.uipath.com/docs/get-visible-text) extracts a string and its information from an indicated UI element using the [Native screen scraping method](https://studio.uipath.com/docs/output-or-screen-scraping-methods). This activity can also be automatically generated when performing screen scraping, along with a container. This activity can be useful in retrieving text from desktop and web applications. As input, this activity receives a Target, which can be either a Region variable, a UiElement variable or a selector, that helps you identify what you want to automate and where the actions must be performed. The target can also be automatically generated by using the **Indicate on Screen** functionality, which tries to identify UI elements in the indicated region, and generates selectors for them. If this does not work for you, then manual intervention might be required. The activity outputs a string variable that contains the extracted text.

[**Extract Structured Data**](https://activities.uipath.com/docs/extract-data) extracts data from an indicated table. You can specify what information to extract by providing an XML string in the **ExtractMetadata** property. This can easily be generated with all the properties set by using the Data Scraping wizard. As input, this activity receives an XML string that defines what data is to be extracted from the indicated web page, and a Target, which can be either a Region variable, a UiElement variable or a selector, that helps you identify what you want to automate and where the actions must be performed. The target can also be automatically generated by using the **Indicate on Screen** functionality, which tries to identify UI elements in the indicated region, and generates selectors for them. If this does not work for you, then manual intervention might be required. The activity outputs a DataTable variable which contains the extracted data.

[**Text Exists**](https://activities.uipath.com/docs/text-exists) checks if a text is found in a given UI element and returns a boolean variable that is true if the text exists and false otherwise. This activity is useful in all types of text-based automation, as it enables you to make decisions based on whether or not a given string is displayed, or it can be used to perform certain actions on a loop, by using it as a Condition in the [**Retry Scope**](https://activities.uipath.com/docs/retry-scope) activity. As input, this activity receives a string variable which contains the text to be searched, and a Target, which can be either a Region variable, a UiElement variable or a selector, that helps you identify what you want to automate and where the actions must be performed. The target can also be automatically generated by using the **Indicate on Screen** functionality, which tries to identify UI elements in the indicated region, and generates selectors for them. If this does not work for you, then manual intervention might be required. The activity outputs a boolean variable that states whether the text was found.

# Example of using Text Automation

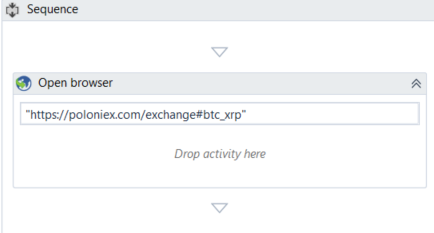
[**SUGGEST EDITS**](https://studio.uipath.com/docs/example-of-using-text-automation)

To exemplify the automation of a process by using text recognition activities, we created an automation that opens Internet Explorer and navigates to a cryptocurrency exchange market platform. From there, it extracts data from a table, displays it in the **Output** panel, double-clicks the third occurrence of the word **Amount** in the website, and checks if the word **Exchange** is found on the screen.

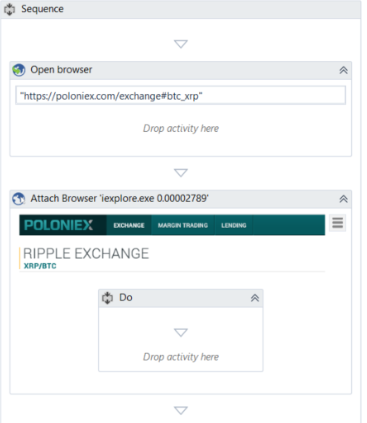
### Note:

It is recommended to run your web automations on Internet Explorer 11 and above, Mozilla Firefox 50 or above, or the latest version of Google Chrome.

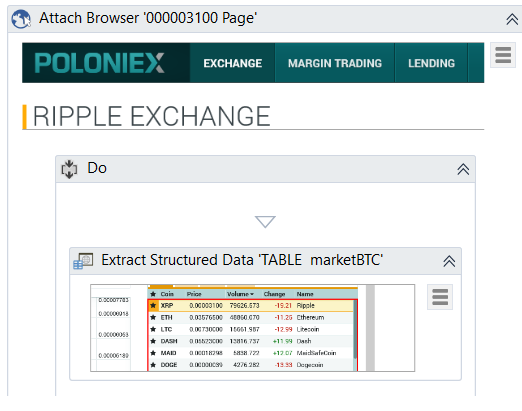
1. Create a new [**Sequence**](https://activities.uipath.com/docs/sequence).
2. Add an [**Open Browser**](https://activities.uipath.com/docs/open-browser) activity to the **Designer** panel.
3. Select the activity and, in the **Url** field, write <https://poloniex.com/exchange#btc_xrp>.



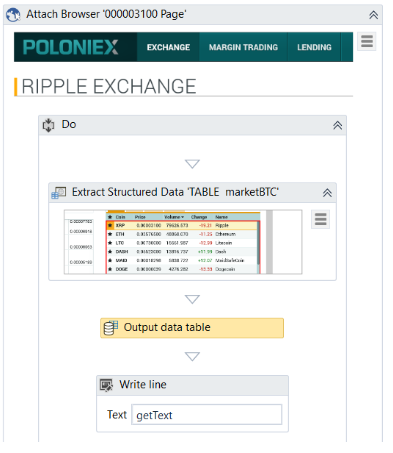
1. In the **Design** ribbon tab, click **Data Scraping**. The **Data Scraping** wizard is displayed.
2. As explained [here](https://studio.uipath.com/docs/example-of-using-data-scraping), scrape the **Markets** table. An **Attach Browser** activity is generated, containing an [**Extract Structured Data**](https://activities.uipath.com/docs/extract-data) activity set to retrieve the table contents.



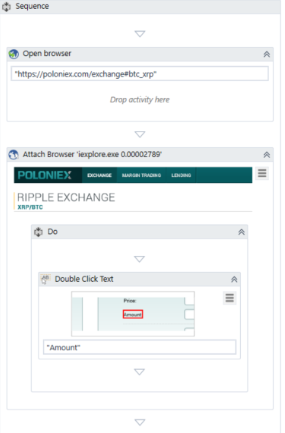
1. Add an [**Output Data Table**](https://activities.uipath.com/docs/output-data-table) activity in the **Do** container of the **Attach Browser** activity, after the **Extract Structured Data**.



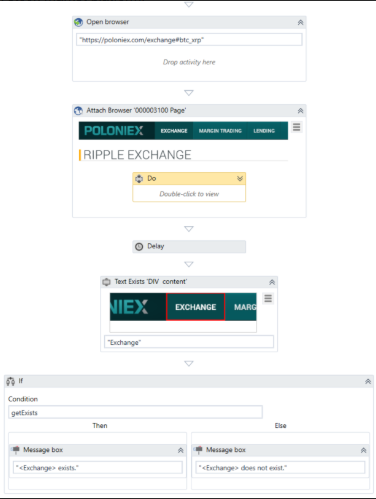
1. n the **Variables** panel, create a new variable, called ExtractDataTable, and set its type to **DataTable**.
2. Create a new **String** variable, called getText.
3. Select the **Extract Structured Data** activity and insert the ExtractDataTable variable in the **DataTable** field.
4. Select the **Output Data Table** activity, enter the ExtractDataTable variable in the **DataTable** field, and the getText variable in the **Text** field.
5. Add a [**Write Line**](https://activities.uipath.com/docs/write-line) activity after the **Output Data Table** one.
6. Select the **Write Line** activity and insert the getText variable in the **Text** field.



1. Add a [**Double Click Text**](https://activities.uipath.com/docs/double-click-text) activity after the previously added one.
2. Select the **Double Click Text** activity, and in the **Text** field, write **amount**. This helps you look for the word **amount** in the website specified at step 3.



1. In the **Occurrence** field, write 3. This helps you look for the third occurrence of the word **amount**.
2. Click the **Indicate on screen** button and click the third occurrence of the word **amount** in the previously opened browser page. This helps you indicate where to look for the specified word, by automatically extracting a selector.
3. Add a [**Delay**](https://activities.uipath.com/docs/delay) activity after the previously added activity. This delay has the purpose of allowing the webpage to become fully loaded.
4. Select the **Delay** activity and set the **Duration** property to 00:00:03 (3 seconds).
5. Add a [**Text Exists**](https://activities.uipath.com/docs/text-exists) activity after the **Delay**.
6. Select the **Text Exists** activity and, in the **Text** field, write Exchange.
7. Click the **Indicate on screen** button and click the word **Exchange** in the previously opened browser page. A selector is generated for the **Exchange** word.
8. n the **Variables** panel, create a new variable, called getExists, and set its type to Boolean.
9. Select the **Text Exists** activity and insert the getExists variable in the **Exists** field.
10. Add an [**If**](https://activities.uipath.com/docs/if) activity after the **Text Exists** activity.
11. Insert the getExists variable in the **Condition** field.
12. In the **Then** section of the **If** activity, add a [**Message Box**](https://activities.uipath.com/docs/message-box) activity.
13. Select the activity and write "Exchange exists" in the **Text** field. This message is displayed if the word Exchange is found in the Poloniex stock market platform.
14. In the **Else** section of the **If** activity, add a **Message Box** activity.
15. Select the activity and write "Exchange does not exist" in the **Text** field. This message is displayed if the word Exchange is not found in the Poloniex stock market platform.  
    Your workflow should look like this:



Press F5. The automation is executed. Note that the data from the Markets table is extracted and displayed in the **Output** panel, the third occurrence of **Amount** is double-clicked, and a message box is displayed, stating whether **Exchange** was found or not.

# OCR Activities

[**SUGGEST EDITS**](https://studio.uipath.com/docs/ocr-activities)

In some situations, certain applications are not compatible with the usage of normal scraping or UI automation technologies. Activities in UiPath Studio which use OCR technology scan the entire screen of the machine, finding all the characters that are displayed. This enables the user to create automations based on what can be seen on the screen, simplifying automation in virtual machine environments. Citrix and other remote desktop utilities are usually the target of OCR-based activities, as they only stream an image of the desktop to the user, which means normal UI selectors are impossible to find.

### Note:

A best practice in creating automations is using the Recording Wizard to create the project, automatically generating selectors, and then tweaking the activities to best fit your needs.

[**Double Click OCR Text**](https://activities.uipath.com/docs/double-click-ocr-text), [**Click OCR Text**](https://activities.uipath.com/docs/click-ocr-text) and [**Hover OCR Text**](https://activities.uipath.com/docs/hover-ocr-text) use OCR to scan the screen of the machine for text and perform actions relative to it. If graphic elements change, but the text does not, automations created using text recognition will usually still work. These are very useful activities in automating basic actions in virtual machine environments. As input, these activities receive a Target, which can be either a string variable, a Region variable, a UIElement variable or a selector, which indicate the coordinates where the action must be performed. The target can also be automatically generated by using the **Indicate on Screen** functionality, which tries to identify UI elements in the indicated region, and generates selectors for them. If this does not work for you, then manual intervention might be required.

[**Get OCR Text**](https://activities.uipath.com/docs/get-ocr-text) extracts a string and its information from an indicated UI element using the OCR screen scraping method. This activity can also be automatically generated when performing screen scraping, along with a container. By default, the Google OCR engine is used, but you can easily change it with Abbyy or Microsoft. There are some differences between these OCR engines, as explained [here](https://studio.uipath.com/docs/output-or-screen-scraping-methods), making them fit for different situations. As input, this activity receives a Target, which can be either a Region variable, a UiElement variable or a selector, that helps you identify what you want to automate and where the actions must be performed. The target can also be automatically generated by using the **Indicate on Screen** functionality, which tries to identify UI elements in the indicated region, and generates selectors for them. If this does not work for you, then manual intervention might be required. This activity returns a string variable containing the text found in the UI element, and a TextInfo variable that contains the screen coordinates of all the found words.

[**Find OCR Text Position**](https://activities.uipath.com/docs/find-ocr-text) searches for a given string in an UI element, and returns a UIElement variable which contains the said string. This activity can be useful in locating UI elements relative to text on the screen. As input, this activity receives a string which contains the text to be searched for, and a Target, which can be either a Region variable, a UiElement variable or a selector, that helps you identify what you want to automate and where the actions must be performed. The target can also be automatically generated by using the **Indicate on Screen** functionality, which tries to identify UI elements in the indicated region, and generates selectors for them. If this does not work for you, then manual intervention might be required. This activity returns a UiElement variable that contains the position where the text was found.

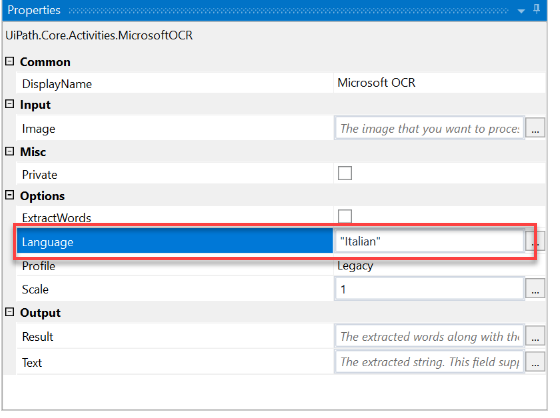
[**OCR Text Exists**](https://activities.uipath.com/docs/ocr-text-exists) checks if a text is found in a given UI element by using OCR technology and returns a boolean variable that is true if the text exists and false otherwise. This activity is useful in all types of text-based automation, as it enables you to make decisions based on whether or not a given string is displayed, or it can be used to perform certain actions in a loop, by using it as a Condition in the **Retry Scope** activity. As input, this activity receives a string which contains the text that is to be searched for, and a Target, which can be either a Region variable, a UiElement variable or a selector, that helps you identify what you want to automate and where the actions must be performed. The target can also be automatically generated by using the **Indicate on Screen** functionality, which tries to identify UI elements in the indicated region, and generates selectors for them. If this does not work for you, then manual intervention might be required. This activity returns a boolean variable that states whether the text was found or not.

**OCR Engines**, such as Google OCR, Google Cloud OCR, Microsoft OCR, Microsoft Cloud OCR and Abbyy Cloud OCR are also available as separate activities. These activities extract a string and its position from a provided image by using different OCR engines. These activities can be used with other OCR activities (Click OCR Text, Hover OCR Text, Double Click OCR Text, Get OCR Text, Find OCR Text Position). As input, these activities receive an Image variable that contains the image file to be scanned. As output, the activities return an IEnumerable<KeyValuePair<Rectangle,String>> variable, which contains the extracted text and their on-screen coordinates, and a string variable which contains the extracted text.

# nstalling OCR Languages

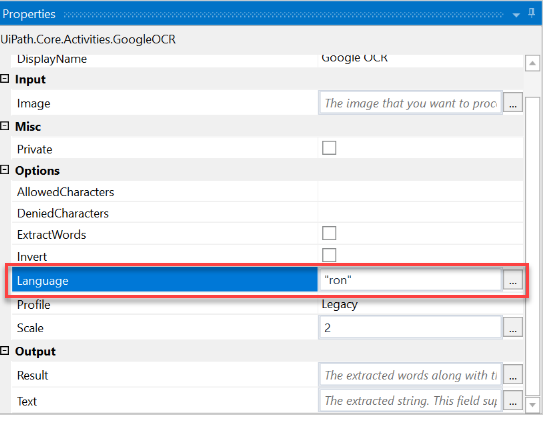
[**SUGGEST EDITS**](https://studio.uipath.com/docs/installing-ocr-languages)

The default language of an OCR engine is English. This can be changed for any of the built-in engines by accessing the **Properties** panel and adding the name of the language between quotation marks, as seen in the screenshots below:



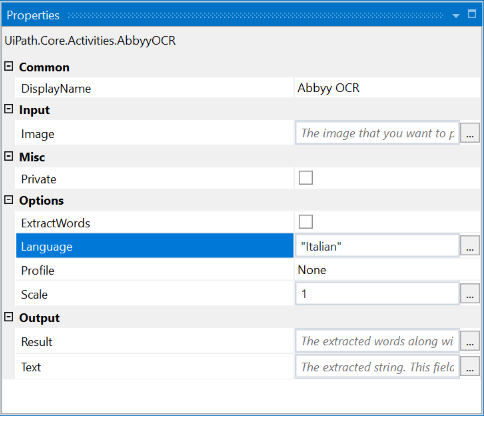
### Note:

For the Google OCR engine, the **Language** field needs to contain the language file prefix, such as “ron” for Romanian, “ita” for Italian, and “fra” for French.

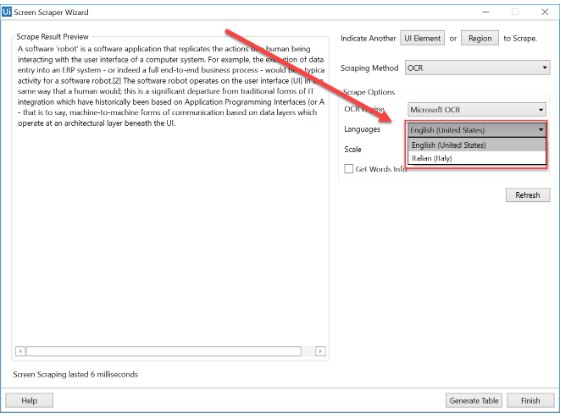


### Note:

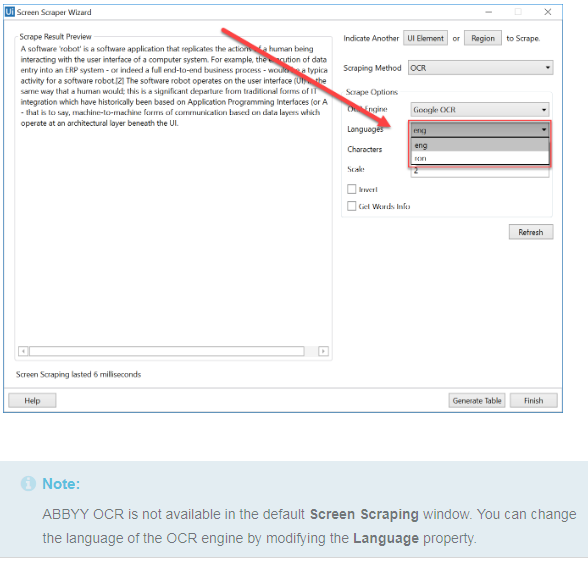
ABBYY FineReader Engine includes the majority of supported OCR languages by default. They can be used right after a successful installation of the engine.



The language for the Microsoft OCR engine can also be changed in a **Screen Scraping**activity when selecting “OCR” as the Scraping Method.



This can also be done for the Google OCR engine.



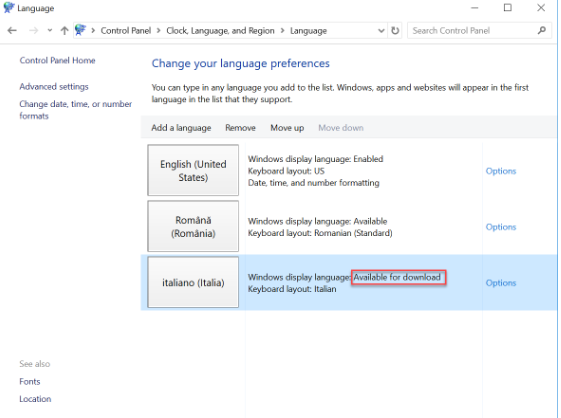
# Installing an OCR Engine and Changing the Language

## **Microsoft OCR**

### **Windows 10**

To add a language to your system and then use it in your workflow:

1. Open an Explorer window.
2. Paste “Control Panel\Clock, Language, and Region\Language” in the address field and press Enter.
3. Click **Add a language**.
4. Choose your preferred language and click **Add**. The language is added in the **Change your language preferences** panel.
5. Select the newly added language and click **Options**. The **Language Options** window is displayed.
6. Click **Download and install language pack** and wait for the installation to finish.
7. Restart UiPath Studio for new languages to become available. The language can now be used in Studio by adding its name between quotation marks (“Italian”).



### Note:

If a language is simply added and not installed it cannot be used by the Microsoft OCR engine. Not all system languages are supported.

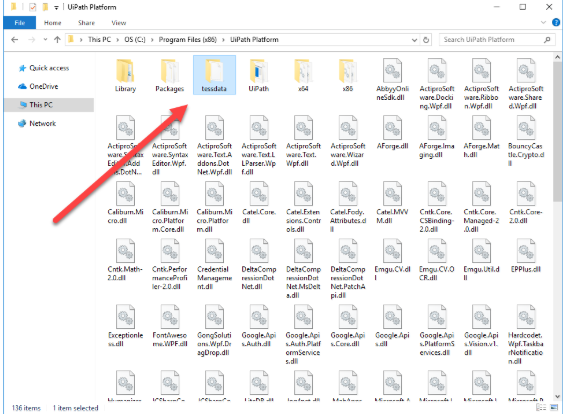
### **Windows 7 and Windows 8.1**

The Microsoft OCR engine needs to be manually installed.

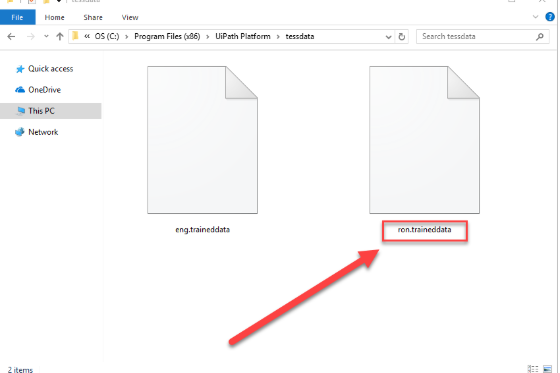
1. Download and install [Microsoft SharePoint Designer 2007](https://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=21581) .
2. Choose your Office version and language [here](https://support.office.com/en-us/article/Language-Accessory-Pack-for-Office-82ee1236-0f9a-45ee-9c72-05b026ee809f#ID0EAACAAA=2010), and follow the instructions to set up the desired language.

## **Google OCR**

1. Search for the desired language file [on this page](https://github.com/tesseract-ocr/tessdata/tree/bf82613055ebc6e63d9e3b438a5c234bfd638c93)
2. Save the file in the **tessdata** folder of the UiPath installation directory (C:\Program Files (x86)\UiPath\Studio\tessdata).



1. Restart UiPath Studio for the new languages to become available. The language can now be used in Studio by adding its name between quotation marks (“ron”).



## **ABBYY OCR**

### Important:

ABBYY FineReader Engine SDK is required. The engine only works with a license distributed by the UiPath sales department.

1. Contact our sales department to obtain a functional ABBYY FineReader Engine SDK License.  
   1.1. Access the [Contact us](https://www.uipath.com/contact-us) page.  
   1.2. Go to [Technical Support & Activations](https://www.uipath.com/contact-technical-and-activations).  
   1.3. Request a license by filling up the form.  
   1.4. Choose “Service Request” after providing a Name and Email.
2. Press **Win + S** to open up Search.
3. Type CMD and then press **Ctrl + Shift + Enter**. This opens Command Prompt with Administrator Privileges.
4. Navigate to the download directory.

### Note:

Use cd.. to go up one folder and cd folder\_name to access a specific folder in Command Prompt.

1. Type Setup.exe /qb /v INSTALLDIR="C:\Abbyy\FR11" SN=serialkey ARCH=x86 LICENSESRV=Yes.
   * The /qb and /v switches handle the interface and caching options.
   * INSTALLDIR is the installation path.
   * SN is the serial number obtained at step 1.
   * ARCH represents the installation architecture which needs to match that of UiPath Studio.
2. Navigate to the Bin folder in the installation directory. It should look something like C:\abbyy\fr11\Bin.
3. Type LicenseManager.exe /SilentActivation /SN:serialkey to activate the license key.
   * The /SilentActivation switch disables user prompts.
   * SN is the serial number obtained at step 1.
4. Restart UiPath Studio to use ABBYY OCR.

# Image Activities

[**SUGGEST EDITS**](https://studio.uipath.com/docs/image-activities)

Image recognition activities can also simulate human behaviour, using images as means of identifying UI elements. These activities enable you to make decisions based on whether or not a given image is displayed, or they can be used to perform certain actions in a loop, by using them as Conditions in the **Retry Scope** activity. They can also scan the screen of the machine for UI elements which appear at random positions and return UiElement variables that have the clipping region set to the found element. They also enable the upload and download of images. Image recognition activities have an **Accuracy** parameter, which states whether the images must match 100% or less to register as found which can compensate for possible changes. This feature is useful if the graphical elements you are searching for may be slightly different.

[**Click Image**](https://activities.uipath.com/docs/click-image), [**Double Click Image**](https://activities.uipath.com/docs/double-click-image) and [**Hover Image**](https://activities.uipath.com/docs/hover-image) are activities used to identify UI elements based on their image. After an image is specified, the activity scans the screen for a given element and either clicks or hovers it. These activities are fast and reliable, but sensitive to graphical variations, as they can fail if colors or background details change. These activities are useful in automating processes that imply simulating human behaviour, using UI elements such as buttons or check boxes. These activities are also important when automating processes in virtual machine environments, such as Citrix, as they make interaction with UI elements possible. As input, these activities receive an image variable which contains the image to be clicked, and a Target, which can be either a string variable, a Region variable, a UIElement variable or a selector, which indicate the coordinates where the action must be performed. The target can also be automatically generated by using the **Indicate on Screen** functionality, which tries to identify UI elements in the indicated region, and generates selectors for them. If this does not work for you, then manual intervention might be required.

[**Find Image**](https://activities.uipath.com/docs/wait-image-appear) is an activity that waits for a certain UI element to appear. To do this, an image of the UI element is provided by the user as a model of the image to be searched. Once the element appears, the activity returns a UiElement variable with the clipping region set to the found image. This activity can be a useful tool in identifying UI elements in virtual machines and performing different actions on them. **Find Image** also enables you to make decisions based on whether or not a given image is displayed, or it can be used to perform certain actions in a loop, by using it as a Condition in the **Retry Scope** activity. As input, this activity receives an image variable which contains the image to be searched for, and a Target, which can be either a Region variable, a UiElement variable or a selector, that helps you identify what you want to automate and where the actions must be performed. The target can also be automatically generated by using the **Indicate on Screen** functionality, which tries to identify UI elements in the indicated region, and generates selectors for them. If this does not work for you, then manual intervention might be required.

[**Image Exists**](https://activities.uipath.com/docs/image-found) is an activity that is used to verify if a certain image exists on the screen. It returns a boolean variable which states whether the image was found or not. This activity can be useful as it enables you to make decisions based on whether or not a given image is displayed, or it can be used to perform certain actions in a loop, by using it as a Condition in the **Retry Scope**activity. As input, this activity receives an image variable which contains the image to be searched for, and a Target, which can be either a Region variable, a UiElement variable or a selector, that helps you identify what you want to automate and where the actions must be performed. The target can also be automatically generated by using the **Indicate on Screen** functionality, which tries to identify UI elements in the indicated region, and generates selectors for them. If this does not work for you, then manual intervention might be required.

[**On Image Appear**](https://activities.uipath.com/docs/on-image-appear) waits for an image to appear on screen for a set amount of time. This activity is a container, which means that multiple actions can be inserted in it and performed on the found image. This is a very useful activity in virtual machine environments, as it can monitor when a UI element appears and then perform a suite of actions. On Image Appear can also be used as a trigger for other activities. As input, this activity receives an image variable which contains the image to be searched for, and a Target, which can be either a Region variable, a UiElement variable or a selector, that helps you identify what you want to automate and where the actions must be performed. The target can also be automatically generated by using the **Indicate on Screen** functionality, which tries to identify UI elements in the indicated region, and generates selectors for them. If this does not work for you, then manual intervention might be required.

[**On Image Vanish**](https://activities.uipath.com/docs/on-image-vanish) waits for an image to vanish from the screen for a set amount of time. This activity is a container, which means multiple actions can be inserted in it and performed after the image disappears. This is a very useful activity in virtual machine environments, as it can monitor when a UI element disappears and then perform a suite of actions. On Image Vanish can also be used as a trigger for other activities. As input, this activity receives an image variable which contains the image to be searched for, and a Target, which can be either a Region variable, a UiElement variable or a selector, that helps you identify what you want to automate and where the actions must be performed. The target can also be automatically generated by using the **Indicate on Screen** functionality, which tries to identify UI elements in the indicated region, and generates selectors for them. If this does not work for you, then manual intervention might be required.

# Example of using OCR and Image Automation

[**SUGGEST EDITS**](https://studio.uipath.com/docs/example-of-using-ocr-and-image-automation)

Since OCR and Image automation usually go hand in hand due to the difficulty of automating in virtual environments, we created an automation that retrieves an employee’s email and the invoice number from a scanned invoice. Afterwards, it inputs the information into a dummy expense app (ExpenseIt) on a virtual machine, and compares the total from the invoice with the one in the app:

1. Create a new [**Sequence**](https://activities.uipath.com/docs/sequence).
2. Open the scanned invoice. We recommend using Adobe Acrobat Reader for compatibility reasons.
3. As explained [here](https://studio.uipath.com/docs/relative-scraping), scrape the employee’s email by using OCR technology. A container, **Attach PDF**, that holds the selector and lets all the other activities know where to perform actions is generated. In it, there are a [**Find Image**](https://activities.uipath.com/docs/wait-image-appear), that selects the anchor for relative scraping, a [**Get OCR Text**](https://activities.uipath.com/docs/get-ocr-text) that retrieves the email address of the employee, and two [**Set Clipping Region**](https://activities.uipath.com/docs/set-clipping-region)activities: one to translate the first clipping region to the second one, and the other to reset the clipping region.
4. In the **Variables Panel**, create a new **GenericValue** variable, called email.
5. Enter the email variable into the **Text** property of the previously generated **Get OCR Text** activity.
6. Name the above sequence **Read Email**.

### Note:

The OCR engines featured by UiPath Studio have their pros and cons, using them depends on the circumstances, and testing which one does the best job in each situation is key in deciding which one to use. Changing the OCR engine for different tasks can make your results better.

1. As explained [here](https://studio.uipath.com/docs/relative-scraping), scrape the invoice number by using OCR technology. The recorder generates a container, **Attach PDF**, that holds the selector and lets all the other activities know where to perform actions. Inside the container, there are a **Find Image**, that selects the anchor for relative scraping, a **Get OCR Text** that retrieves the invoice number of the employee, and two **Set Clipping Region** activities, one to translate the first clipping region to the second one, and one to reset the clipping region.
2. In the **Variables Panel**, create a new **GenericValue** variable called invoiceNo.
3. Enter the invoiceNo variable into the **Text** property of the previously generated **Get OCR Text** activity.
4. Name the above sequence **Read Invoice Number**
5. As explained [here](https://studio.uipath.com/docs/relative-scraping), scrape the employee’s total expenses by using OCR technology. The recorder generates a container, **Attach PDF**, that holds the selector and lets all the other activities know where to perform actions. Inside it, there are a **Find Image**, that selects the anchor for relative scraping, a **Get OCR Text** that retrieves the total expenses of the employee, and two **Set Clipping Region** activities, one to translate the first clipping region to the second one, and one to reset the clipping region.
6. In the **Variables Panel** create a new **GenericValue** variable called totalValue.
7. Enter the totalValue variable into the **Text** property of the previously generated **Get OCR Text** activity.
8. Name the above sequence **Read Total Expenses**.

### Note:

A best practice in opening applications in virtual machine environments is creating a shortcut for the application that is to be opened on the desktop of the machine and assigning it a hotkey. Trying to click the application’s icon by using Click OCR Text can sometimes fail due to changes in the background colour or to the icon being selected.

1. Create a shortcut for the **ExpenseIt** application on the desktop of the virtual machine.
2. Right-click the shortcut and select **Properties** from the context menu. The **Shortcut Properties** window is displayed.
3. On the **Shortcut** tab, in the **Shortcut Key** field, assign a hotkey to the app by pressing the keys you want to use, for example Alt + Ctrl + T.
4. Start a new recording session by opening the **Citrix Recording Wizard**.
5. Record sending the virtual machine window the hotkey that was previously assigned to the application’s shortcut.
6. Use [**Relative Scraping**](https://studio.uipath.com/docs/relative-scraping) to click the **Email** field in the **ExpenseIt** application based on its label’s location.
7. In the **Email** field of the **ExpenseIt** application, enter the email variable.
8. Use **Relative Scraping** to click the **Employee Number** field in the **ExpenseIt** application based on its label’s location.
9. Add the invoiceNo variable in the **Employee Number** field of the **ExpenseIt** application.
10. Click **Save & Exit** in the **Recording Wizard**. The recorder generates an **Attach Window** container that holds the selector and lets all the other activities know where to perform actions. It contains a **Send Hotkey** activity, a **Click Image** activity that clicks the field to the right of the **Email** label, a **Type Into** activity that types the email variable into the field, a **Click Image**activity that clicks the field to the right of the **Employee Number** label and a **Type Into** activity that types the invoiceNovariable into the field.
11. Start a new recording session by opening the **Citrix Recording Wizard**.
12. Record clicking the **Create Expense Report** button in the **ExpenseIt** application.
13. Use **Relative Scraping** to scrape the value that corresponds to the **Total Expenses ($)** label.
14. Click **Save & Exit** in the **Recording Wizard**. The recorder generates an **Attach Window** container that holds the selector and lets all the other activities know where to perform actions. It contains a **Click OCR Text** that clicks the **Create Expense Report** button, a **Find Image** that sets the anchor for the relative scrape, a **Get OCR Text** that retrieves the total expense value and two **Set Clipping Region** activities, one to translate the first clipping region to the second one, and one to reset the clipping region.
15. In the **Variables Panel**, create a new **GenericValue** variable called totalExpense.
16. Enter the totalExpense variable in the **Text** property field of the previously generated **Get OCR Text** activity.
17. Drag an **If** activity after the last generated recording sequence.
18. Set the **Condition** property of the **If** activity to totalExpense = totalValue. This means that the automation checks if the total value of the invoice equals the value in the expense application.
19. Drag a **Message Box** activity to the **Then** section of the **If** activity.
20. In the **Content** property field of the **Message Box**, write a message that states the values are equal, hence the invoice is correct.
21. Drag a **Message Box** activity to the **Else** section of the **If** activity.
22. In the **Content** property field of the **Message Box**, write a message that states the values are not equal, hence the invoice is incorrect.
23. Press F5 to run the automation. Note that the automation inputs the employee data from the scanned invoice into the **ExpenseIt**application, compares the total value of the invoice with the total expenses registered, and informs the user if the values are equal or not.

Since this automation involves connecting to a virtual machine, uploading the original project is redundant, as it would not work on another machine. We encourage you to build your own project by following the above steps. The scanned invoice used in the example can be downloaded [here](https://www.uipath.com/hubfs/Documentation/WorkflowExamples/Scanned%20Invoice.pdf).

# Introduction to Citrix Automation

[**SUGGEST EDITS**](https://studio.uipath.com/docs/introduction-to-citrix-automation)

A virtual machine (VM) is an emulation of a computer system. [Virtual machines](https://en.wikipedia.org/wiki/Virtual_machine) are based on computer architectures and provide functionality of a physical computer. Examples of such virtual environment providers are Citrix, RDP, Microsoft Azure, etc.. However, even if a user can work with this virtual PC as it would with a normal one, only the **image** of the interface is actually streamed to the user. Therefore, UiPath Studio cannot address the interface through the operating system, as it would on a standard machine. To efficiently automate processes in Citrix machines, special tools and techniques must be used.

The **Citrix Recording Wizard** is one of the most important tools in automating virtual machines, as it facilitates the automation of any action, generating activities which are already configured.

Opening applications must be done according to certain rules, as the [**Click**](https://activities.uipath.com/docs/click) activity is not very reliable in virtualized environments. There are usually other, safer methods through which applications can be opened, as they do not rely on communicating with the virtualized environment by directly clicking the window.

Understanding how **OCR** and **Image** activities work is also essential in automating in virtual machines, as they enable the identification of UI elements and the inputting and retrieving of data from applications.

# Citrix Specific Automation Techniques and Tools

[**SUGGEST EDITS**](https://studio.uipath.com/docs/citrix-specific-automation-techniques-and-tools)

# Citrix Recording Wizard

The easiest way to automate in virtualized environments is using the specialized **Citrix Recorder**, which automatically generates fully-configured activities based on the user’s actions. It also facilitates the use of techniques such as **Relative Scraping**. The Recording Wizard is designed to simulate human behaviour and is specialized in using activities and technologies specific to virtual environment automation, such as OCR and Image Recognition activities.

Sometimes, the automatically generated [selectors](https://studio.uipath.com/docs/about-selectors) propose volatile attribute values to identify elements. This means activities might not work properly in all circumstances, and manual intervention is required to calibrate the selectors. A reliable selector should successfully identify the same element every time in all conditions, regardless of external changes in resolution or UI element position.

# Opening Applications in Citrix

Usually, apps are opened by clicking their shortcut or executable file. The location of these files can normally be identified by several means, such as screen coordinates or selectors.

In virtualized environments, these ways of identifying the shortcut’s location are unavailable, so, for clicking, **Image**and **OCR** activities must be used to identify the location of the shortcut or executable file. Since these activities are based on image and text recognition, slight graphical differences, such as changes in resolution or highlighting the icon, can cause the identification of the shortcut to fail. A solution for this issue is selecting an area of the icon that does not include any portion of the background image, such as the center area of the icon.

A best practice in opening applications in virtual machine environments is creating a **shortcut**for the application on the desktop of the machine, assigning it a **hotkey**, and then sending that hotkey to the remote desktop connection window by using a [**Send Hotkey**](https://activities.uipath.com/docs/send-hotkey) activity. It is recommended to use a more complex key combination for the shortcut, to avoid interfering with existing ones.

Another safe way to start apps in virtual environments is by using the **Command Prompt**. For example, you can send the path of the application to the **Command Prompt** terminal with the **Send Hotkey** and [**Type Into**](https://activities.uipath.com/docs/type-into) activities. This method also enables you to input arguments for the app to be opened.

# Waiting for Certain States of Applications

There are situations when waiting for a certain state of an application is essential to creating an optimal automation. In desktop environments, UiPath activities are configured to wait for certain states before acting, as Studio has direct access to the operating system and can understand applications on a logical level.

In virtual environments, Studio does not have access to the underlying elements of the operating system, so other methods must be employed to identify application states.

To make sure an application is fully loaded before interacting with it, you must identify visual elements that show the page or app is done loading, such as specific pictures or buttons. In this regard, you can use the [**On Image Appear**](https://activities.uipath.com/docs/on-image-appear) and [**Find Image**](https://activities.uipath.com/docs/wait-image-appear) activities to monitor the virtual environment, and allow the project to continue the execution only when a certain UI element appears. A better and more general solution is to wait for the application’s loading icon to disappear, in case it exists. An [**On Image Vanish**](https://activities.uipath.com/docs/on-image-vanish) activity can be used for this purpose, allowing the automation to continue only when the loading icon vanishes.

A bad practice in waiting for an application to load is adding a [**Delay**](https://activities.uipath.com/docs/delay) activity to your project. This method is prone to failure because loading times for software programs can vary due to many factors.

# Identifying UI Elements

Since virtual environments offer no way to identify UI elements via standard means, visual anchors are the only remaining option. UiPath Studio features activities that use OCR or Image Recognition technologies that are meant to be used in such situations.

There are several OCR engines that can be used with UiPath Studio: Google Tesseract, Microsoft MODI and Abbyy. The Google Tesseract engine works better for scraping smaller areas, while Microsoft MODI is more suitable for larger ones.

# Inserting Data in Citrix

As explained earlier, clicking UI elements in virtual environments can be tricky, due to changes in resolution or background colors. Thus, inserting data in Citrix in an optimal fashion implies using methods that are not prone to failure, such as making use of keyboard shortcuts and sending hotkeys to the virtual machine window in order to avoid clicking.

**Relative Click** is a technique that enables you to click UI elements by using other buttons or labels around them as anchors. In situations where selectors cannot be found, the target UI objects are identified by using image recognition activities to look for adjacent visual labels or other such elements.

A good way to insert data from a machine into a virtual environment is using the shared clipboard. This method has the advantage that it can easily paste data into the virtual machine by first clicking the app to be automated and sending it the Ctrl + V hotkey.

To avoid having to identify the UI elements’ location in order to click them, it is recommended to switch between buttons and text fields by using Tab, Enter and the navigation keys. Another very useful activity for typing text in virtual machines is **Type Into**, because it interacts with the application by sending keystrokes, just like a human user would.

### Note:

Using Tab to switch between UI elements can sometimes be unreliable, as updates that change the UI layout can cause automations to no longer function correctly. It is recommended to keep an eye out for such changes in the layout and to update the projects accordingly. Also, sending the Tab keys too fast may cause some of them to not be received by the target app, in which case it is recommended to use the **Delay** activity to increase the duration between sending the keys.

If using keyboard commands to navigate through UI elements is not an option, then Image and Text recognition is the alternative to automating in virtual environments. Image recognition has its own weaknesses, being sensitive to environment variations like changes in desktop theme or screen resolution. When the application runs in Citrix, the resolution should be kept greater or equal than when recording the automations. Otherwise, small image distortions can be compensated by slightly lowering the **Accuracy** property of the image activities. Check how the application layout adjusts itself to different resolutions to ensure visual elements’ proximity, especially in the case of coordinate based techniques like **Relative Clicking** and **Relative Scraping**. To enable the automation to support different resolutions, parallel recordings can be placed inside a **Pick Branch** activity and the suitable one can be automatically chosen for the optimal resolution.

# Retrieving Data from Citrix

Retrieving data from a virtual environment has its own limitations, as the Native and FullText scraping methods that retrieve text directly from the operating system do not work in virtual environments. Thus, OCR-based activities are essential in scraping the screen of the Citrix machine.

Just like in the case of data input, the shared clipboard is a useful and reliable tool in retrieving text from Citrix, as it can be easily accessed by sending the Ctrl + C hotkey to the window via the **Send Hotkey** activity.

The [**Copy Selected Text**](https://activities.uipath.com/docs/copy-selected-text) activity is another activity that can copy text from the virtual machine environment, that has a very similar behaviour to using the shared clipboard.

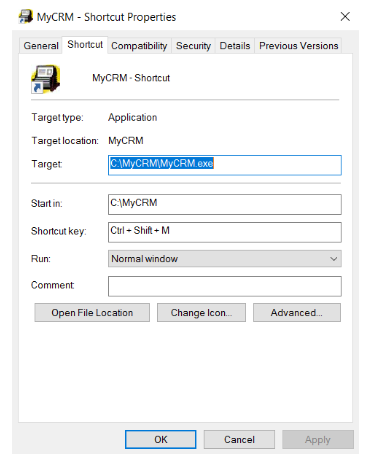
Additionally, relative scraping is a useful technique that enables you to retrieve text from UI elements by using OCR technology, relative to anchors in the window, such as text box labels or buttons.

# Example of Citrix Automation

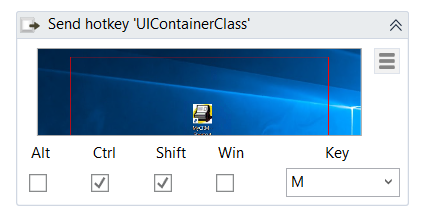
[**SUGGEST EDITS**](https://studio.uipath.com/docs/example-of-citrix-automation)

To exemplify the specific techniques of Citrix automation, we have built an example that opens a dummy app in a virtual environment, retrieves the first and last name of an employee from the said app, and stores them in variables. After that, it navigates to another window, inserts the above variables in text fields and clicks a button, searching for the employee in the app’s database.

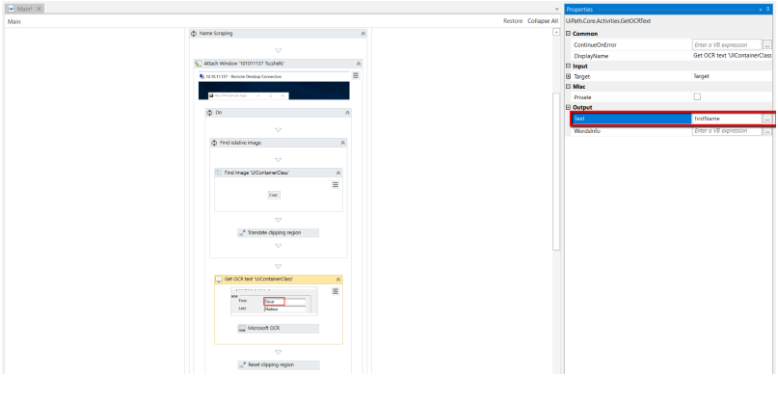
1. Open the connection to the Virtual Machine you want to automate in.
2. Create a shortcut for the **MyCRM** application on the desktop of the virtual machine.
3. Right-click the shortcut and select **Properties** from the context menu. The **Shortcut Properties** window is displayed.
4. On the **Shortcut** tab, in the **Shortcut Key** field, assign a hotkey to the app by pressing the keys you want to use, for example Ctrl + Shift + M.



1. In **UiPath Studio**, create a new [**Sequence**](https://activities.uipath.com/docs/sequence).
2. Add a [**Send Hotkey**](https://activities.uipath.com/docs/send-hotkey) activity and connect it to the **Start** node.
3. Click the **Indicate on Screen** button and select the virtual machine window.
4. Set the **Send Hotkey** activity to send the shortcut designated for the **MyCRM** application to the virtual machine window.

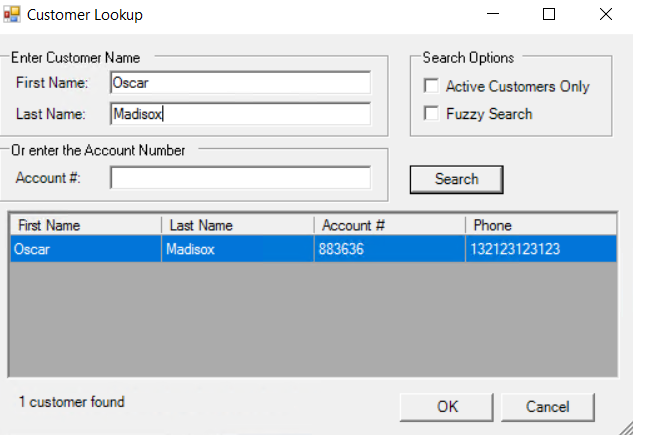


1. By using [**Relative Scraping**](https://studio.uipath.com/docs/relative-scraping), scrape the employee’s first name in the **MyCRM** window by using OCR technology. A container, [**Attach Window**](https://activities.uipath.com/docs/window-scope), is generated, holding the selector and letting all the other activities know where to perform the generated actions. In it, there are a [**Find Image**](https://activities.uipath.com/docs/wait-image-appear), that selects the anchor for relative scraping, a [**Get OCR Text**](https://activities.uipath.com/docs/get-ocr-text) that retrieves the first name of the employee, and two [**Set Clipping Region**](https://activities.uipath.com/docs/set-clipping-region) activities: one to translate the first clipping region to the second one, and the other to reset the clipping region.
2. In the **Variables Panel**, create a new **GenericValue** variable called firstName.
3. In the **Text** property of the **Get OCR Text** activity, insert the firstName variable.



1. By using [**Relative Scraping**](https://studio.uipath.com/docs/relative-scraping), scrape the employee’s last name in the **MyCRM** application by using OCR technology. A container, **Attach PDF**, is generated, holding the selector and letting all the other activities know where to perform the generated actions. In it, there are a **Find Image**, that selects the anchor for relative scraping, a **Get OCR Text** that retrieves the last name of the employee, and two **Set Clipping Region** activities: one to translate the first clipping region to the second one, and the other to reset the clipping region.
2. In the **Variables Panel**, create a new **GenericValue** variable called lastName.
3. In the **Text** property of the **Get OCR Text** activity, insert the lastName variable.
4. Drag another **Send Hotkey** activity to the project.
5. To open the **File** context menu of the **MyCRM** app, set the **Send Hotkey** activity to send the Alt + F hotkey to the **MyCRM**app.
6. Add another **Send Hotkey** activity to the project.
7. Set the second **Send Hotkey** activity to send the Enter hotkey to the **MyCRM** app, to open the **Customer Lookup** search menu.
8. Start a new recording session by using the **Citrix Recording Wizard**.
9. Record typing the firstName variable into the **Customer Lookup** window.
10. Record sending the **Tab** hotkey to the application in order to switch to the next text field.
11. Record typing the lastName variable into the **Customer Lookup** window.
12. Record sending the **Enter** hotkey to the application.

Press F5. The automation is executed.The customer in the **MyCRM** app is found in the **Customer Lookup** window.



**LOGGING**

# Studio Traces

The Studio Traces are diagnostic log messages generated by Studio, containing information related to its behavior.

The **logging level settings** and **log file path** can be modified by editing the <Installation Folder>\UiPath.Studio.exe.config file. If this file does not exist, it is automatically created when an event occurs. The section can be found under the <nlog> tag.

## **UiPath Studio Traces Logging Level**

The default selected logging level for Studio is **Information**, but more levels can be added or excluded in the same configuration file.

To only log **Error** and **Warning** levels, specify them in between the <rules> tag, like in the example below:

<rules> <logger name="\*" level="Error" writeTo="AsyncTarget" />  
<logger name="\*" level="Warning" writeTo="AsyncTarget" /> </rules>

To limit the logs to a specific size, set the archiveAboveSize="<value in bytes>" in the <target> section:

<target name="AsyncTarget" xsi:type="AsyncWrapper" minlevel = "Off"> <target name="logfile" xsi:type="File" fileName="${logDirectory}/${shortdate}\_Studio.log" layout="${time} =&gt; [${level:uppercase=true}] [${logger}] [${threadid}] ${message}" archiveAboveSize="<value in bytes>/> </target>

To disable logging, simply set the minlevel tag to ”Off”, as you can see below:  
<rules> <logger name="\*" minlevel="Off" writeTo="AsyncTarget" /> </rules>

## **Targets of the UiPath Studio Traces**

The default log file path for Studio is %localappdata%\UiPath\Logs\<shortdate>\_Studio.log. The format for <shortdate> is yyyy-mm-dd.

To change the log file path, simply change the value of the logDirectory parameter with the path you want to use.

For example, to move the log file path to the Desktop, modify the value="${specialfolder:folder=LocalApplicationData}/UiPath/Logs" parameter to value="${specialfolder:folder=DesktopDirectory".

# Setup Logs

[**SUGGEST EDITS**](https://studio.uipath.com/docs/setup-logs)

# Setup Logs

When running the Windows installer (**UiPathStudio.msi**), all errors are logged in the [**Event Viewer**](https://technet.microsoft.com/en-us/library/cc766042.aspx).

For more detailed logs, you can execute the installer from the command line with the following parameters:

msiexec.exe /i "path\_of\_the\_installer" /lv\* "path\_of\_the\_log\_file"

For example:

msiexec.exe /i "%UserProfile%\Downloads\UiPathStudio.msi" /lv\* "%UserProfile%\Desktop\Log.txt"

Executing this command line generates a file called Log.txt to the specified folder, which contains the installation log data.

If you are running the user mode installer (**.exe**), setup logs are automatically generated and can be found in %localappdata%\SquirrelTemp\SquirrelSetup.log

# OCR Diagnostic Logs

[**SUGGEST EDITS**](https://studio.uipath.com/docs/ocr-diagnostic-logs)

# UiPath.Vision.Host Diagnostic Logs

The UiPath.Vision.Host Diagnostic logs enable you to collect information regarding OCR engine errors.

## **Enabling UiPath.Vision.Host Diagnostic Logs**

To collect information regarding OCR errors, you can enable the generation of the diagnostic log files by adding the following lines to the <Installation Folder>\NLog.config file:

<target xsi:type="File" name="visionHostFile" fileName="${LogDirectory}/${shortdate}\_VisionHost.log" layout="${time} ${level} ${message}" concurrentWrites="true" />

<logger name="UiPath.Vision.Host.exe" minLevel="Trace" writeTo="visionHostFile" final="true" />

### Note:

Editing the NLog.config file requires administrator permissions.

## **Targets of the UiPath.Vision.Host Diagnostic Logs**

The log files are generated at the path specified in the <Installation Folder>\NLog.config, under the visionHostFile parameter. The default value is %localappdata%\UiPath\Logs\${shortdate}\_VisionHost.log.

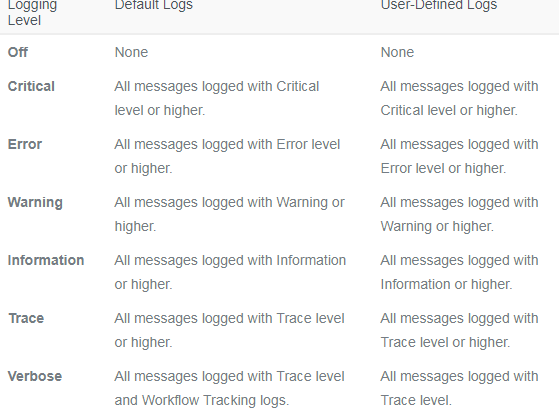
### Note:

Editing the NLog.config file requires administrator permissions.

# Logging Levels

[**SUGGEST EDITS**](https://studio.uipath.com/docs/logging-levels)

# Logging Levels in UiPath



The **Verbose** level logs a message for both the activity **start** and **end**, plus the values of the variables and arguments that are used.

By default, the Verbose level includes:

* **Execution Started** log entry - generated every time a process is started.
* **Execution Ended** log entry - generated every time a process is finalized.
* **Transaction Started** log entry - generated every time a transaction item is obtained by the robot from Orchestrator.
* **Transaction Ended** log entry - generated every time the robot sets the transaction status to either Success or Failed.
* **Activity Information** log entry - generated every time an activity is **started**, **faulted** or **finished** inside a workflow.

### Note:

The priority order of the log types is: **Verbose** < **Trace** < **Information** < **Warning** < **Error** < **Critical** < **Off**.

Log Level

Logged

Example / Comment

Log File

Output Panel

Orchestrator Log page

**Verbose**

**Activities**

Trace {"message":{"DisplayName":"Message box","State":"Executing","Activity":"UiPath.Dialog.Activities.MessageBox","Arguments":{"Caption":"","Text":"String in message BOX"}...  
Trace {"message":{"DisplayName":"Message box","State":"Closed","Activity":"UiPath.Dialog.Activities.MessageBox","Arguments":{"Caption":"","Text":"String in message BOX","ChosenButton":"Ok"}

Yes

No

Yes

**Verbose**

**Variables**

"Variables":{"NewTransaction":"False"}}

Yes

No

Yes

**Verbose**

**Arguments (properties)**

"Arguments":{"Caption":"","Text":"String in message BOX","ChosenButton":"Ok"}

Yes

No

Yes

**Trace**

**Activities**

Trace {"message":{"DisplayName":"Main","State":"Executing","Activity":"System.Activities.DynamicActivity"}  
Trace {"message":{"DisplayName":"Main","State":"Executing","Activity":"System.Activities.Statements.Flowchart"}

Yes

No

Yes

**Information**

**WriteLine**  
**Log Message**

Info {"message":"message from activity"  
**Note:** Except messages logged with Trace level set in activity.

Yes

Yes

Yes

**Warning**

**Warnings**

Warn {"message":"Warning from log message activity"

Yes

Yes

Yes

**Warning**

**Errors**

Error {"message":"Error from log message activity"

Yes

Yes

Yes

**Warning**

**Critical**

Critical Errors

Yes

Yes

Yes

**Error**

**Errors**

Error {"message":"Error from log message activity"

Yes

Yes

Yes

**Error**

**Critical**  
**Fatal**

Critical Errors

Yes

Yes

Yes

**Critical**

**Critical**  
**Fatal**

Critical Errors

Yes

Yes

Yes

**OFF**

n/a

n/a

No

No

No

# Log Message Types

There are several possible occurrences of log messages, depending on the event that is logged, as follows:

## **Default Logs**

Generated by default when the execution of a project starts and ends, when a system error occurs and the execution stops, or when the logging settings are configured to log the execution of every activity.

### Note:

These logs have the Default value in the logType field.

The events logged by this category are:

* **Execution Start** is generated every time a process is started. This is logged starting with the **Information** logging level.
* **Execution End** is generated every time a process is finalized. This is logged starting with the **Information** logging level.
* **Transaction Start** is generated every time a transaction within a process is started. This is logged starting with the **Information** logging level.
* **Transaction End** is generated every time a transaction within a process is finalized. This is logged starting with the **Information** logging level.
* **Error Log** is generated every time the execution encounters an error and stops. This is logged starting with the **Error** logging level.
* **Debugging Log** is generated if the Robot Logging Setting is set to Verbose and contains, activity names, types, variable values, arguments etc. This is logged starting with the **Trace**logging level.

## **User-Defined Logs**

Generated according to the process designed by the user in Studio, when using the **Log Message** activity or the **Write Line** activity.

### Note:

These logs have the User value in the logType field.

# Log Fields

There are multiple types of log fields that can be found throughout the above log message types. These can be classified as follows:

## **Default fields**

These log fields are present in all execution type logs, such as SQL (if configured), Elasticsearch (if configured), and the default EventViewer Logs:

* Message - The log message.
* Level - Defines the log severity.
* Timestamp - The exact date and time the action was performed.
* FileName - The name of the .xaml file being executed.
* jobId - The key of the job running the process.
* processName - The name of the process that triggered the logging.
* processVersion - The version number of the process.
* windowsIdentity - The name of the user that performed the action that was logged.
* robotName - The name of the robot (as defined in Orchestrator).

### Note:

The processName and processVersion fields do not appear in logs if the process is run locally, without being connected to Orchestrator.

## **Type-specific fields**

These logs are present depending on the log type:

### **Execution End**

* totalExecutionTimeInSeconds
* totalExecutionTime

### **Transaction Start**

* queueName
* transactionID
* transactionState

### **Transaction End**

* queueName
* transactionID
* transactionState
* transactionStatus
* transactionExecutionTime
* processingExceptionType
* processingExceptionReason
* queueItemReviewStatus
* queueItemPriority

### **Debugging Log**

* activityInfo, which is a JSON message with the following fields:
  + DisplayName
  + State (Faulted, Closed, Executing)
  + Activity
  + Variables
  + Arguments

### Note:

Only totalExecutionTimeInSeconds, totalExecutionTime and queueName are always present in the log messages. Variables and Arguments usually have sub-fields.

## **User-defined fields**

These fields are defined in Studio by using the **Add Log Fields** activity and appear in all subsequent logs after the activity is generated, unless they are removed by the **Remove Log Fields** activity.

### Important!

Creating custom log fields that have the same name as a default log field causes the logging process to become corrupted and may cause issues in the workflow you are running.  
For example, creating a custom log field called jobId causes this issue, as jobId is a log field that is generated by default.

* **Timestamp**: The exact date and time the action was performed
* **FileName**: the name of the .xaml file being “executed”
* **jobId**: The key of the job running the process
* **processName**: The name of the process that triggered the logging
* **processVersion**: The version number of the process
* **windowsIdentity**: The name of the user that performed the action that was logged
* **robotName**: The name of the robot (defined in Orchestrator)

### Note:

The **processName** and processVersion (**i'm guessing JobID and RobotName might also not appear**) might not appear if the process is run locally without connection to the Orchestrator.

## **Type-specific fields**

Thse log fields are present depending on the log type:

* **totalExecutionTimeInSeconds** for Execution End
* **totalExecutionTime** for Execution End
* **queueName** for Transaction Start and Transaction End
* **transactionID** for Transaction Start and Transaction End
* **transactionState** for Transaction Start and Transaction End
* **transactionStatus** for Transaction End
* **transactionExecutionTime** for Transaction End
* **activityInfo** for Debugging Log. It is a JSON message with the following fields
  + **DisplayName**
  + **State** (Faulted, Closed, Executing)
  + **Activity**
  + **Variables**
  + **Arguments**

### Note:

Only the first 3 are always present in the message. Variables and Arguments usually have sub-fields.

## **User-defined fields**

These fields are defined in Studio (by using the **Add Log Fields** activity) and appear in all subsequent logs after the activity is generated, unless they are (programmatically) removed by the activity **Remove Log Fields**

# About the ScreenScrapeJavaSupport Tool

[**SUGGEST EDITS**](https://studio.uipath.com/docs/about-the-screenscrapejavasupport-tool)

A machine can have different Java versions installed (such as 32-bit and 64-bit), and in different locations, so you need the Java extension to be able to help you build automations regardless of the version. This is where the **ScreenScrapeJavaSupport** tool comes in handy. It is meant to help you install and uninstall the Java extension, but also to give you info about the installed Java versions and what processes have the extension loaded. Moreover, you can especially benefit from this tool if you have Java installed in custom locations.

The **ScreenScrapeJavaSupport** tool is installed with Studio, and can be found in the C:\Program Files (x86)\UiPath\Studio\UiPath\JavaSupport folder.

### Note:

The **ScrenScrapeJavaSupport** tool requires administrator privileges.

# Using the ScreenScrapeJavaSupport tool

## **Installation**

The Java extension can be installed in the default location of 32-bit an 64-bit versions of Java, on all drives, or in a custom location.

* ScreenScrapeJavaSupport.exe /install /jrepath "<path>" - Installs the Java extension for the specified path. The path is the folder path containing the java.dll file. If the Java extension is already installed then it reinstalls it.
* ScreenScrapeJavaSupport.exe /install /default - Searches for the default Java installation folder, Program Files or Program Files (x86), and installs the Java extension there. This is because both Java x86 and x64 can be installed on the same machine.
* ScreenScrapeJavaSupport.exe /install /allDrives - Recursively searches all fixed drives for Java versions, and installs the Java extension for each one.
* ScreenScrapeJavaSupport.exe /install - Reinstalls the Java extension for all the Java versions that are currently installed. The ScreenScrapeJavaSupport tool keeps a record of each version it installs.

## **Removal**

Upon removal, the Java extension path is remembered, unless removed from the log with the /forget argument.

* ScreenScrapeJavaSupport.exe /uninstall /jrepath "<path>" - Uninstalls the Java extension from the specified path.
* ScreenScrapeJavaSupport.exe /uninstall - Uninstalls the Java extension from the system.
* ScreenScrapeJavaSupport.exe /uninstall /allDrives - Uninstalls the Java extension from all fixed drives.

# Argument Descriptions for the ScreenScrapeJavaSupport Tool

[**SUGGEST EDITS**](https://studio.uipath.com/docs/argument-descriptions-for-the-screenscrapejavasupport-tool)

The **ScreenScrapeJavaSupport** tool can be used to install or remove the Java extension from all fixed drives, default, or custom Java locations. Whenever you uninstall the Java extension with this tool, its path is kept track of, unless removed with the /forget argument.

To use the tool launch an elevated instance of the Command Prompt and navigate to the tool location by using the cd C:\Program Files (x86)\UiPath\Studio\UiPath\JavaSupport command. You can use it with the following arguments:

Argument

Description

/install

Reinstalls the Java extension from all the Java versions that are currently installed. Can be used with the /jrepath “<path>”,/allDrives,/default, and/silent` commands.  
**Examples:**

* ScreenScrapeJavaSupport.exe /install
* ScreenScrapeJavaSupport.exe /install /jrepath "C:\Business drivers\Java\jre1.8.0\_181\bin"
* ScreenScrapeJavaSupport.exe /install /default
* ScreenScrapeJavaSupport.exe /install /allDrives
* ScreenScrapeJavaSupport.exe /install /silent

/uninstall

Uninstalls the Java extension from all the Java versions that are currently installed. Can be used with the /jrepath “<path>”, and/allDrives` commands.  
**Examples:**

* ScreenScrapeJavaSupport.exe /uninstall
* ScreenScrapeJavaSupport.exe /uninstall /jrepath "C:\Business drivers\Java\jre1.8.0\_181\bin"
* ScreenScrapeJavaSupport.exe /uninstall /allDrives
* ScreenScrapeJavaSupport.exe /uninstall /silent

/jrepath “<path>”

Specifies the path of an installed Java version. Valid with the /install, /uninstall, /forget, and /status commands.  
**Examples:**

* ScreenScrapeJavaSupport.exe /install /jrepath "C:\Business drivers\Java\jre1.8.0\_181\bin"
* ScreenScrapeJavaSupport.exe /uninstall /jrepath "C:\Business drivers\Java\jre1.8.0\_181\bin"
* ScreenScrapeJavaSupport.exe /forget /jrepath "C:\Business drivers\Java\jre1.8.0\_181\bin"
* ScreenScrapeJavaSupport.exe /status /jrepath "C:\Business drivers\Java\jre1.8.0\_181\bin"

/allDrives

Autodetects Java installations on all fixed drives. Valid with the /install and /uninstall commands.

/default

Autodetects 32-bit and 64-bit Java installations in Program Files folders. Valid with the /install and /uninstall commands.

/silent

Turns off all message boxes. Valid with the /install and /uninstall commands.

/forget

Forgets all the location where the Java extension was installed. Valid with the /jrepath “<path>”command.

/status

Offers details about each installed Java version, as well as a cumulative status of all the installed Java versions. Displays the default 32-bit and 64-bit Java installations if present on the system. Shows the history of uninstalled Java versions (if the /forget parameter was not used for any of them). Valid with the /jrepath "<path>" command.

/scan

For the current Windows user and current session, scans all the processes with a corresponding UI and which have the java.dll library loaded. For each such process it checks if the Java extension is loaded. In the end, it prints detailed information about that process and a cumulative status for all processes.  
**Example:**

* ScreenScrapeJavaSupport.exe /scan

/help

Displays a list of all supported commands.  
**Example:**

* ScreenScrapeJavaSupport.exe /help