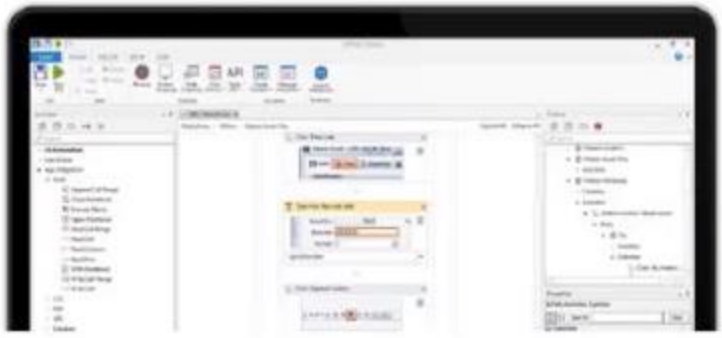
UiPath **Studio**

UiPath Studio is an advanced visual process modeling tool.

Straightforward drag-and-drop functionality and a built-in library of predefined activities greatly enhance the user's experience and speed up the learning curve.



UiPath Studio

Based on WWF and MS visio.

UiPath Robot: clerical robots programmed to execute processes modeled with UiPath Studio and orchestrated by the UiPath Server.

They can either run unattended in a VM on a datacentre (Back Office Robots) or share the same desktop with a human agent (Front Office Robots).

Robots are 100% accurate and reduce human error. They can refer the exceptions they find to a specialized human agent for processing. UiPath software robots perform fine detailed automation regardless of the data source.

Can be Human Triggered or automated robots.

UiPath **Orchestrator**

UiPath Server is the browser based server application that you can use to manage robots and processes.

Using the server web console you can deploy, start, stop and schedule processes and monitor their execution by the robots. UiPath Server facilitates human-robot collaboration and business exception handling using centralized work queues.

Management Platform , scheduling , deployment , centralized , reports dashboards.

Ui Path Studio :

Activities:

Actions that will use to automate applications. eg: clicking and typing .

Around 300 , excel , csv ,mail , Db types , Decision servers , amtp ,pop servers , email compose , send

Terminal: e.g. mainframe

4 recoding types:

1. Basic
2. Desktop
3. Citrix
4. Web

Attach window Activity: To select the application window

Type into: to type: indicate on screen

Click Activity

We can create variables: Function e.g. Trim () are available.

Get Text Activity– Indicates and store in variable.

Message box activity – Pass the above variable.

Flowchart

Start - >

read Csv activity : Reads the excel File

Get Row Item activity : column : excel column name

Output value : store in variable

Decision Block ->

In Get row item, we can use column name or simply index.

In ui path, we can copy anything anywhere.

Assign activity: increment counter

Q: How ui path identifies where to click and type it don't mess up ?

UI Explorer: shows specific ui element path within the application – its Ui path .This path is based on the unique name given the software developer of the application. Generally these names do not change in versions. So change in version or layout do not impact workflow as the names don't change.

All these principal applies to web automation also.

Clicking on any attribute on web application, will display all attributes down to html tree. We can select and specify and give more attributes to be more precise .Moreover we can give Regular Expression and wild card character allowing maximum precision.

Data scrapping wizard.

Create sequence -- > select Data Scrapping wizard... Data Spanning to amore pages. Yes/ No.

Write csv -- > send outlook web mail

Ui path can work with multiple platforms st once

Remote Desktop Automate:

Recording – Citrix – Image Based Automation.

Click Image and find the text of label in citrix . Then select indicate and type in the text box.

Use Scrape relative tool and get the transaction id located at the right side .

Scrapping is done via OCR – Optical character recognition. There are three scrapping engine integrated with ui path ;

1. Microsoft Modi and Google tesseract are included out of the box.
2. Abbyy OCR is available as a Plugin

In case the RDP location changes or the app inside the RDP moves, automation still goes through. Because the Robot is searching for the image and image relatives.

If the Resolution changes or the application aspects is altered somewhat, the robot can still identify the correct images by lowering the accuracy of the identification algorithm .Default Value is 0.8 , with 1 being a perfect 1-to-1 correspondence.

Choosing to go for a lower matching accuracy value say 0.6 will allow us to find the image if it has a different resolution, Useful when the quality of RDP drops or poor connection

UI Path Orchestrator

Used for managing deployed robots usually for large enterprises that use substantial number of robots.

To make sure robot can talk to orchestrator , we need to configure our robot with the same robot key and the orchestrator URL.

https://studio.uipath.com/v2016.2/docs/keyboard-shortcuts

Input Dialog: Empty Input Area

Select Folder: input dialog to select folder

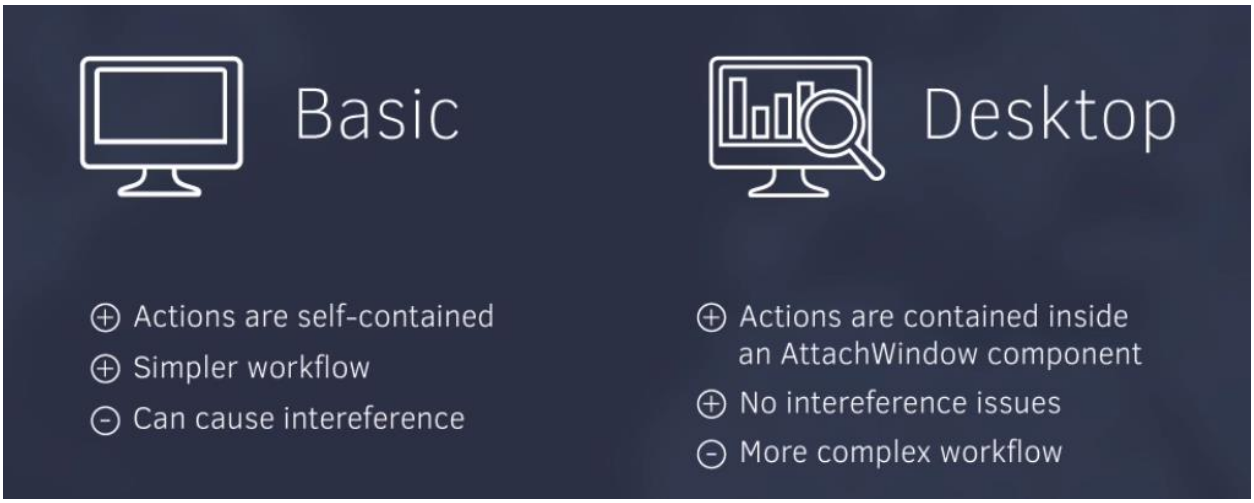
Directory.getfiles (“Folder path”) – returns array

writeLine

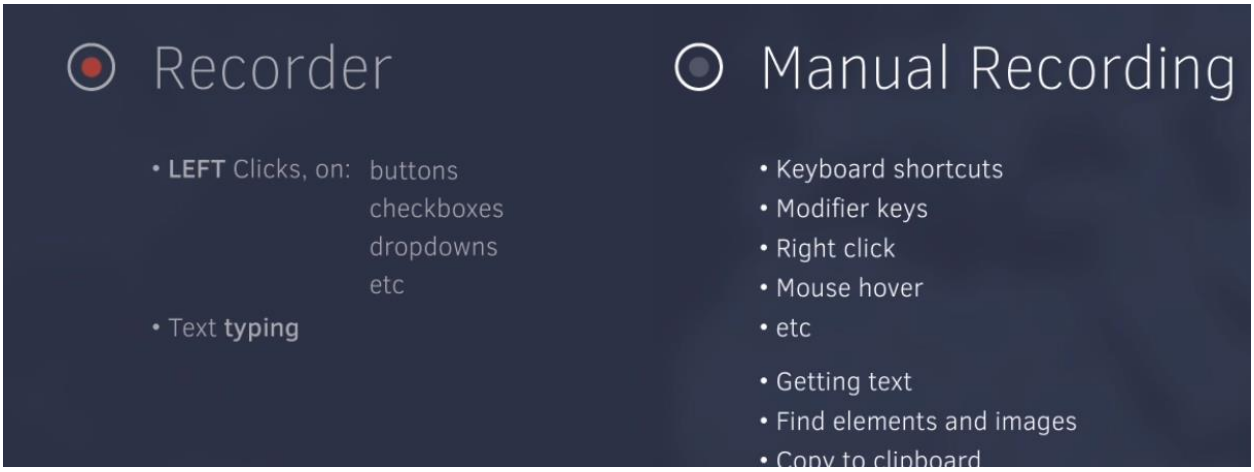
Practical Exercise

Log Message Activity :

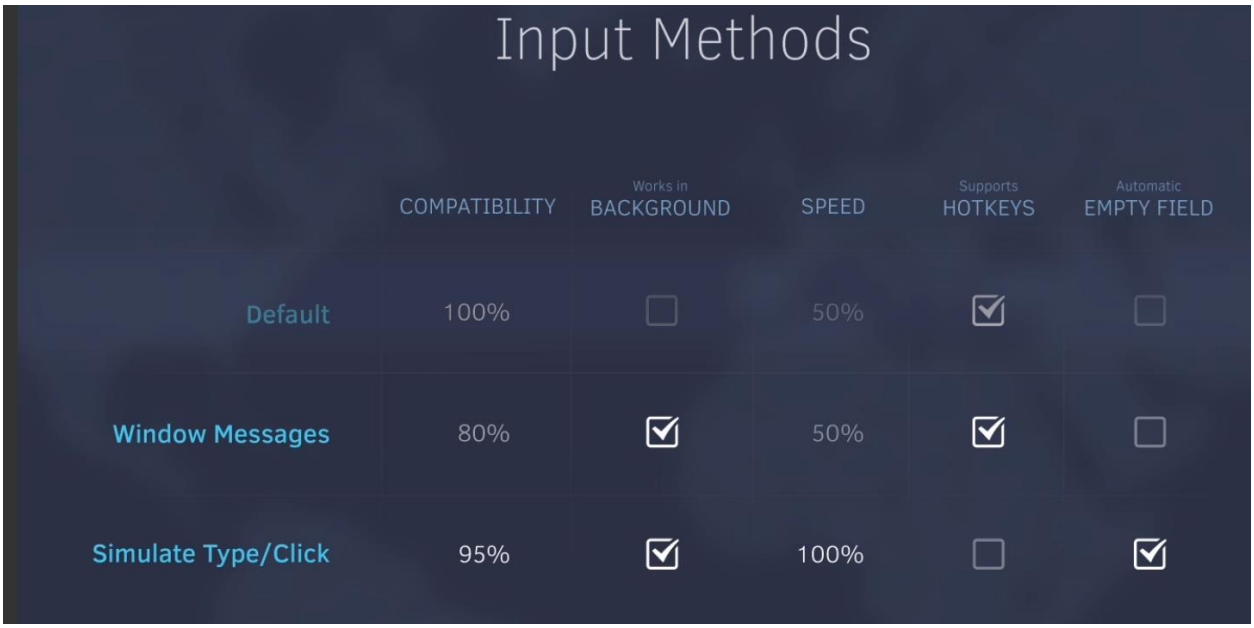
Invoke method – used with list

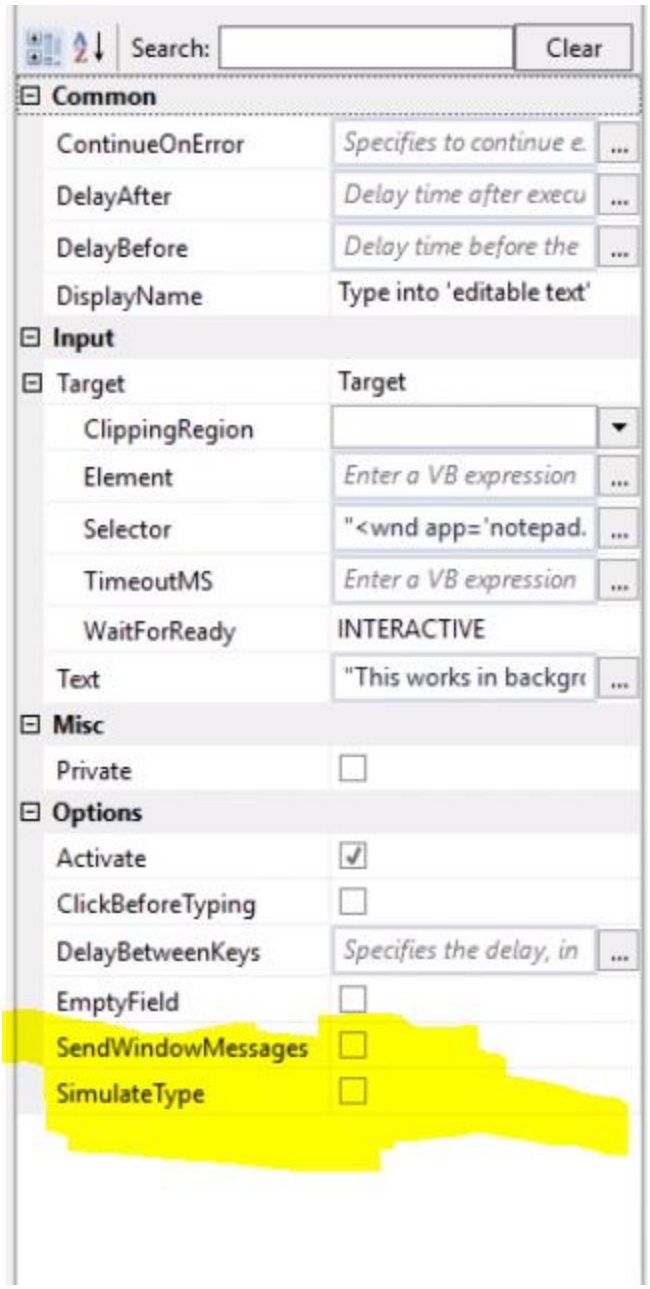


Use F2 to pause recordings, So that we can use this for opening sub menues.

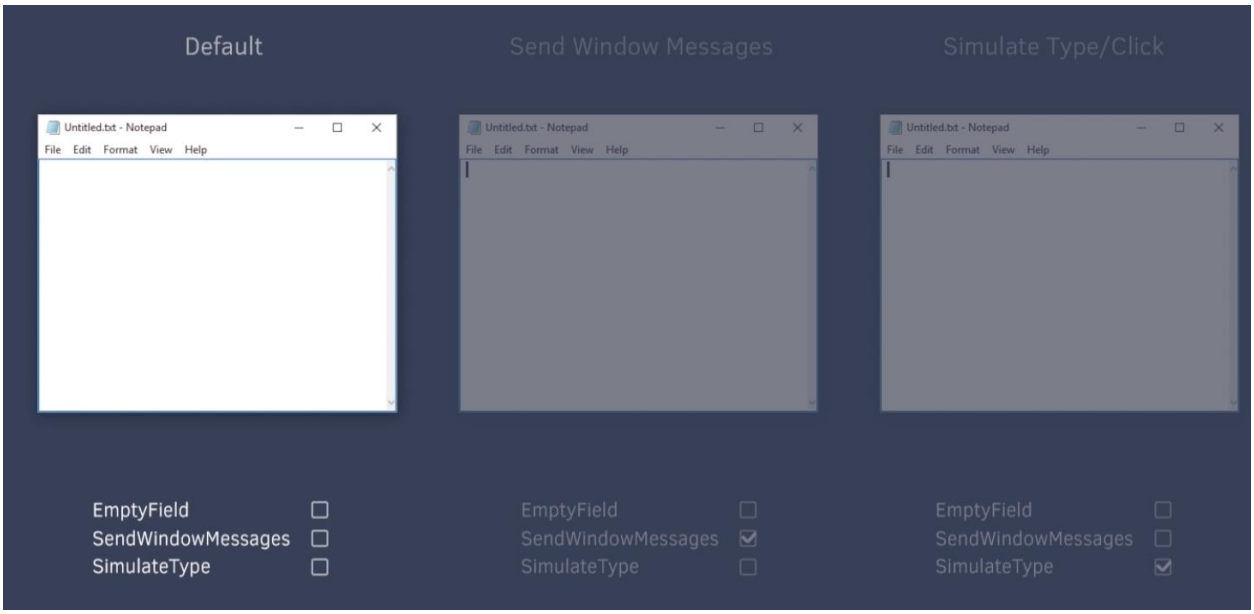


Open Browser




Search:
Clear
Common
ContinueOnError
Specifies to continue e.
DelayAfter
Delay time after execu
DelayBefore
Delay time before the
DisplayName
Type into 'editable text'
Input
Target
Target
ClippingRegion
Element
Enter a VB expression
Selector
"<wnd app='notepad.
TimeoutMS
Enter a VB expression
WaitForReady
INTERACTIVE
Text
"This works in backgr
Misc
Private
Options
Activate
ClickBeforeTyping
DelayBetweenKeys
Specifies the delay, in
EmptyField
SendWindowMessages
SimulateType

Muabbu t



Default gives the Expected result was not very fast

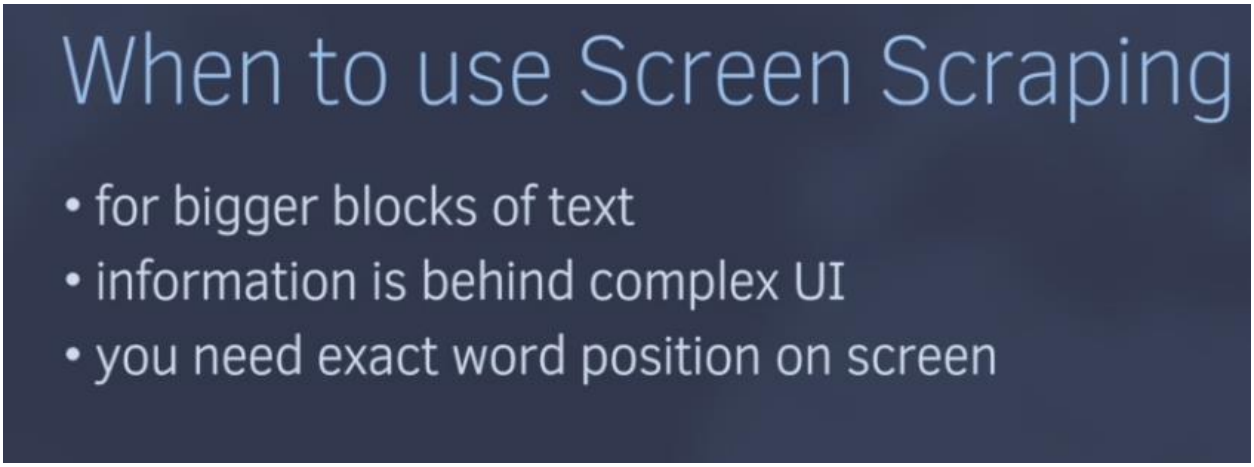
Send window messages was very similar but apparently converts all the texts to lower case

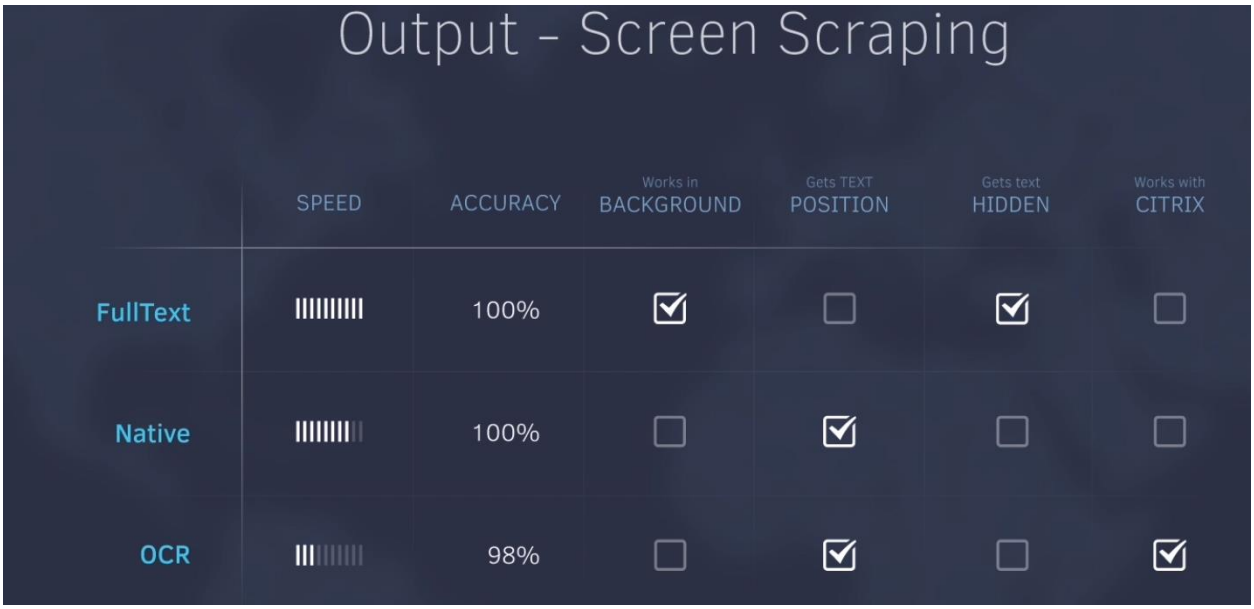
Simulate type is the fastest but the results were not as expected. Even though the empty field was checked , the second text completely replaced the first one. This is the limitation of simulate type and can be overcome by using the enter text before the second text.

Keyboard shortcuts appear on the text in case of the simulate type .

Default type needs to be open to enter the text .this does not work when the window is minimized , same like human behavior.

UIPath output methods.





Full text is the default method. Its fast accurate and it runs in the background.

Native method has the capability to extract screen coordinates .

OCR is not 100 % correct but is useful when none of the above seems to work.

Native gets only the editable Text.

Full text gets the editable texts with labels and hidden fields also that we can ignore by unchecking the ignore hidden check box.

Native : Retains Formatting. OCR doesn't

There are 2 Different OCR techniques available .

1. MS office OCR.
2. Google OCRsmsm

Each has dictionary with different languages.

MS Office OCR is suitable for larger images like scanned documents , receipts and so on.

Google OCR runs better with smaller , low resolution images like interface elements .It has an Invert option available with an option to upscale images

Web Scrapping is used to extract Structured Data unlike others that is used to scrape free form Data.

First click on the title then to give UI path a pattern select on the second link.

Is the data Spanning multiple pages -> Yes to indicate if we have same data with next button like amazon.

Write csv.

Max Number of results property : 100

Tabular Data can be Scrapped Directly

Full vs Partial Selectors

FULL SELECTOR

Full Path

```
<wnd app='applicationframehost.exe' cls='ApplicationWindow' title='Month View - Calendar' />
<wnd cls='Windows.UI.Core.CoreWindow' title='Calendar' />
<ctrl automationid='CalendarListNavPane' />
<ctrl automationid='MiniCalendarDay_2017-01-12T00:00:00Z' />
```

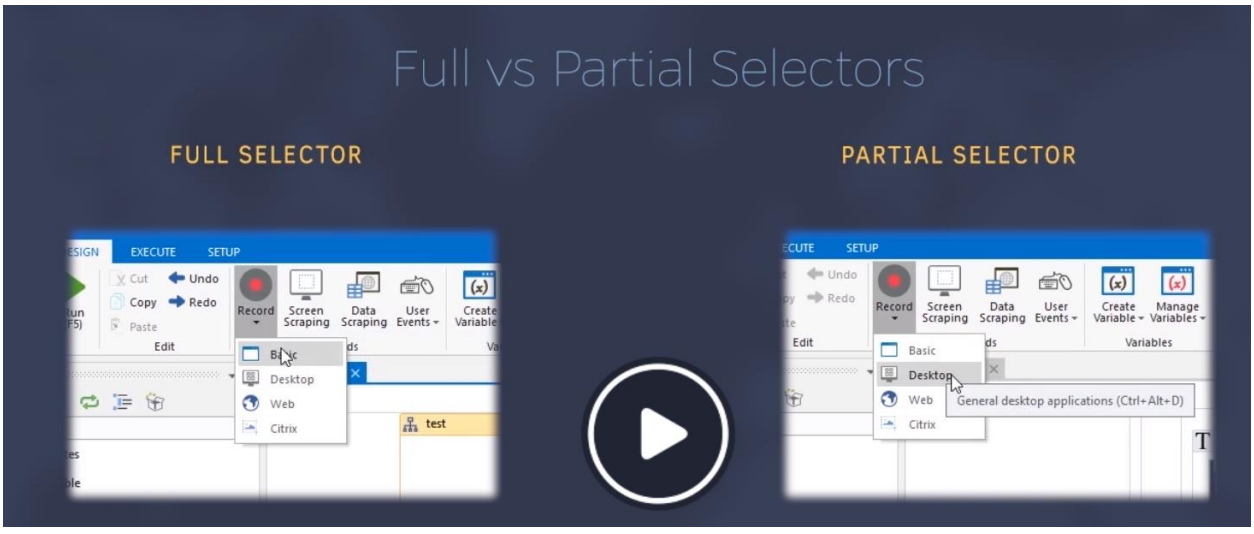
PARTIAL SELECTOR

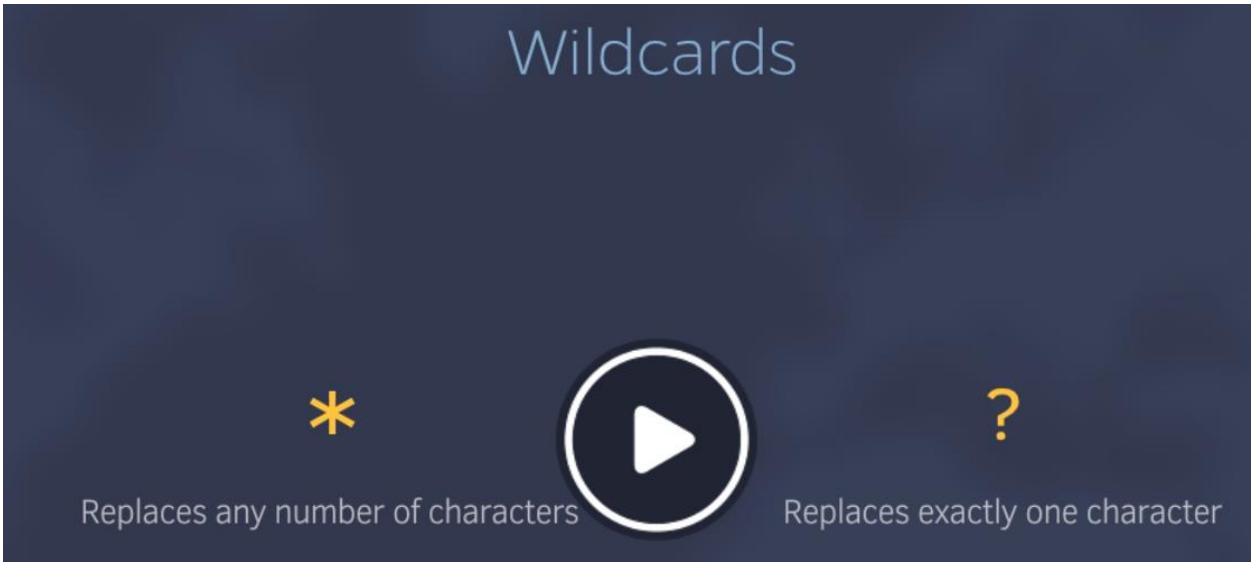
Top level element

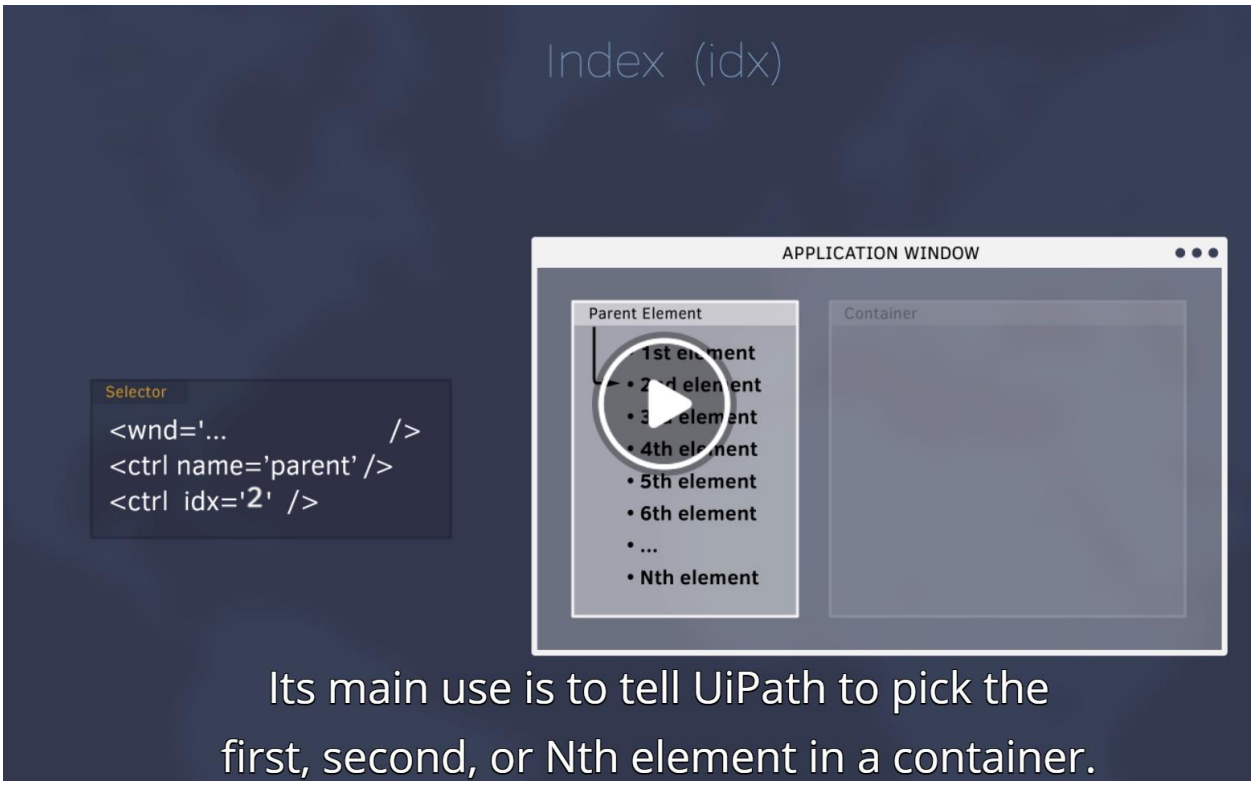
```
<wnd app='applicationframehost.exe' cls='ApplicationWindow' title='Month View - Calendar' />
```

Partial Selector

```
<wnd cls='Windows.UI.Core.CoreWindow' title='Calendar' />
<ctrl automationid='CalendarListNavPane' />
<ctrl automationid='MiniCalendarDay_2017-01-12T00:00:00Z' />
```







Click Text – Uses OCR

Click Image – Faster , Fails if color changes .

For Click Text and Image you can change the Click,double click , Right ,Left click ,Alt. Shift in the properties

Click Image has an property called accuracy , 1 means exact match .

Click text and Image has an drawback a, as it works on OCR , so may fails. Hack is to use Key board shortcuts eg: Tab instead of clicks .

To Get Information out of virtual machine , we have 2 methods.

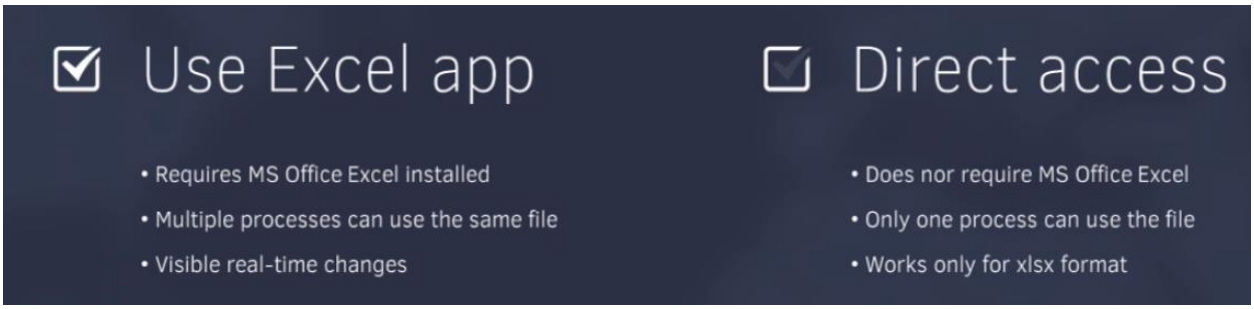
1. Select and Copy – Works only for the selectable Texts.
2. Screen Scrapping.- Screen relative . Google OCR – Scale 3

Find Image Activity : to find the Favicon Image in VDi to look whetheI page is loaded fully.

Highlight Action: To see what is detected. We can highlight clipping region , element or selector.

Send Hot Key combination : to create hotkey for opening a shortcut

Excel Application Scope – Container



Read Range : Reads a Portion of the excel and store in a data table

Output Data Table Action – Only convert to string .

Write Range : To Save to excel , Over rides

Append Range : No Over riding .

Build Data Table

Sort Table : required Header

Read cell

Write Cell

Get Row item

Build Data Table

Add Data Row

UI PATH PDF Packages

Read Pdf Text activity

Range : All or Range of Pages

Read PDF with OCR – Image inside PDF.

Screen Scrape Can be also used to read OCR.

Anchor base :

Find element – Anchor position : left

Find Image .

Find relative , Scrape Relative

- [ ] Pick an appropriate layout for each workflow
- [ ] Break the whole process in smaller workflows
- [ ] Use exception handling
- [ ] Make your workflows readable
- [ ] Keep it clean

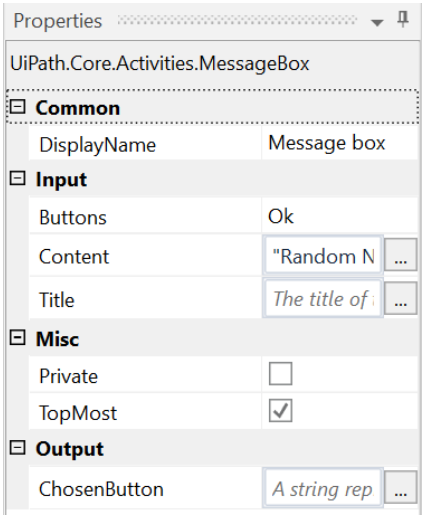
Activities in Details:

1. Activity Name: Message Box

Available at: System → Dialogue → Message Box

Function: Displays a message box with buttons options

Properties:

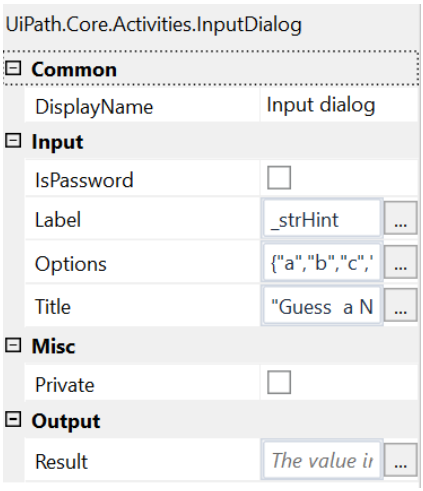


2. Activity Name: Input Dialog

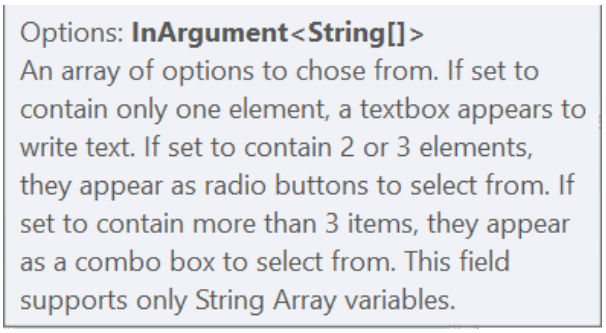
Available at: System → Dialogue → Input Dialog

Function: Prompt user with a label message and an input field.

Properties



Options:

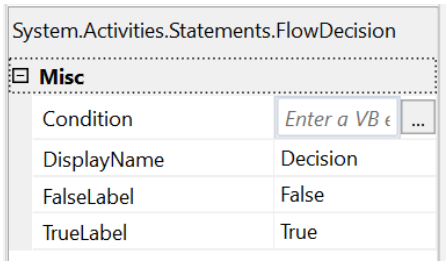


3. Activity Name: Flow Decision

Available at: Work Flow → FlowChart → Flow Decision

Function: True false branching.

Properties



Sequences

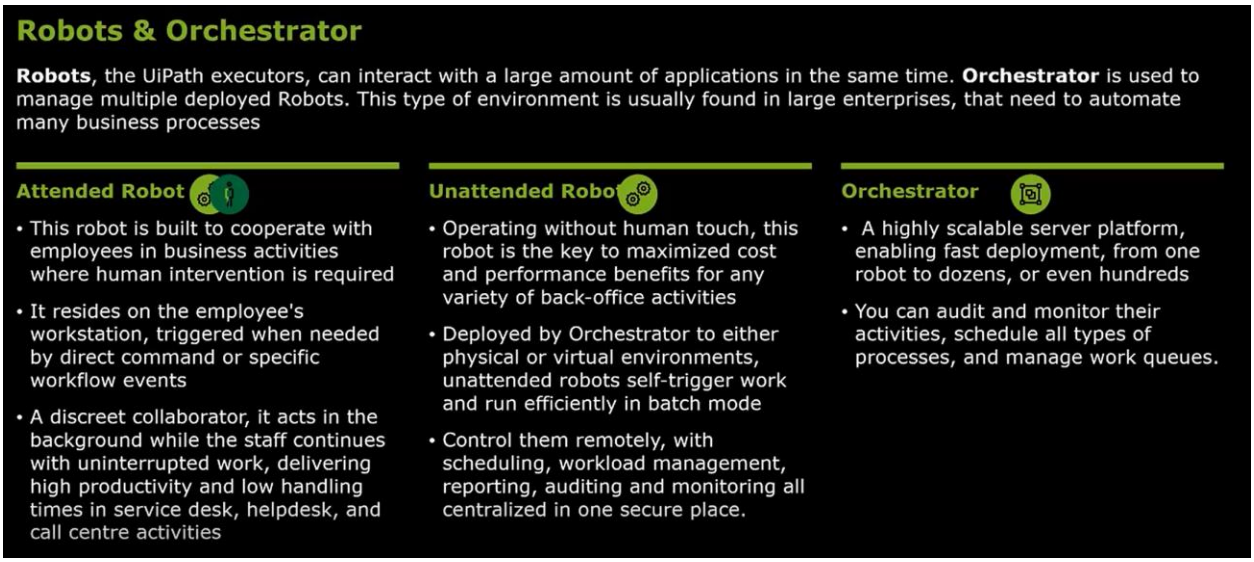
Sequences are the smallest type of project. They are suitable to linear processes as they enable you to go from one activity to another seamlessly, and act as a single block activity.

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.

Sequences do not use connectors.

Generic Datatypes are not available outside Uipath.



Orchestrator : machine Shutdown ,Bots stop . Orchestrator can monitor and perform post installation activities.

continueonError : Boolen

DelayBefore and DelayAfter : Takes Milliseconds

Target : Wait for Ready : NONE : Interactive : Complete – Eg: Chrome .

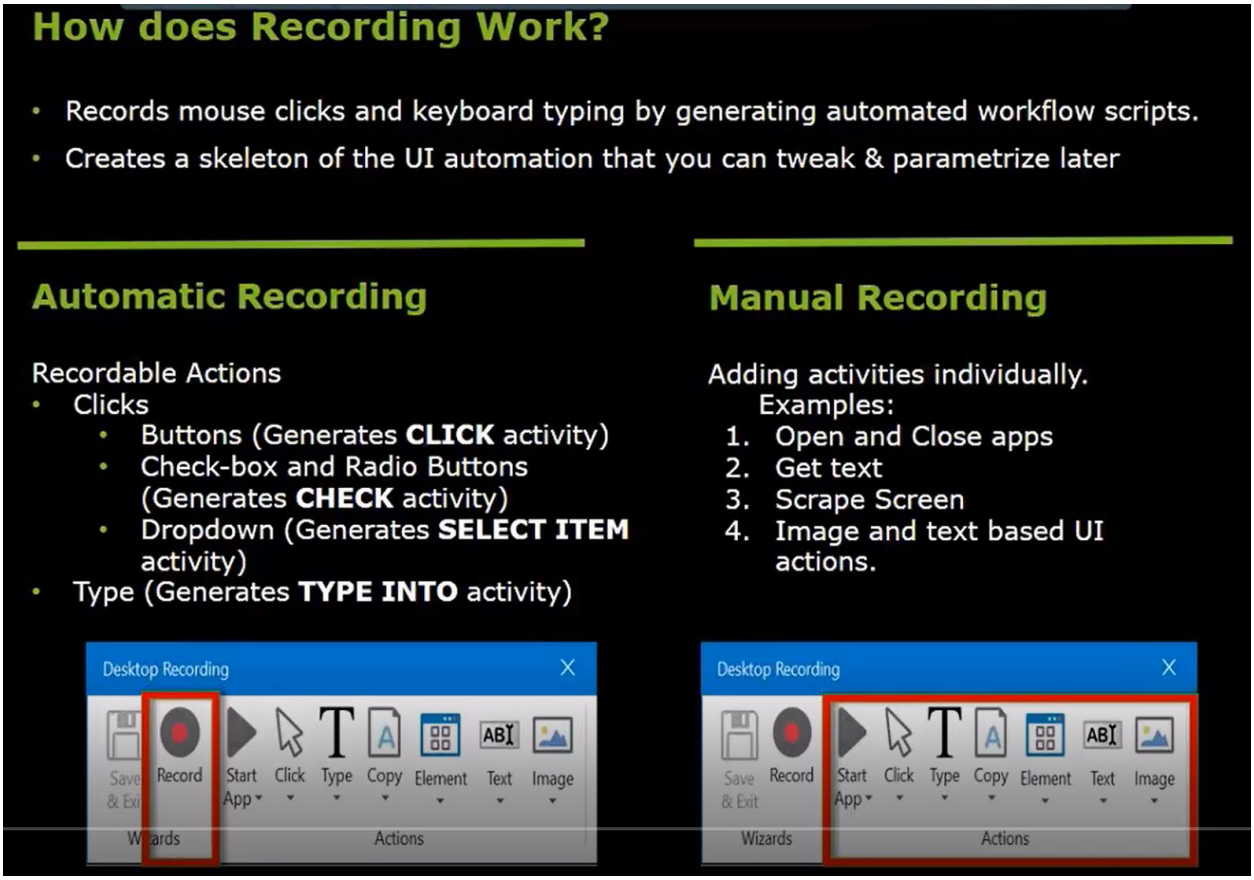
TimeOutMS : Error is going to happen .

Default TimeOut in Uipath is 30s , If webpage takes more than 30s, error is thrown. Specify TimeOutMS as 60s.

Directory.getFiles("Path ").

Recordings are saves as sequences, which can be manipulated later on.

Selectors : Notepad: Dialog is container, Files Tabs are other container , Editable Text is anaother Container. If container is not correct, correct the selectors.



Hot Key : Ctrl +s does not work with automatic recording.

Containers

- **What?**
  - Adds a context to the UI activities contained.
- **Which?**
  - Attach/Open Window
  - Attach/Open Browser
  - Active Window
- **When?**
  - When performing more actions o the same window.
  - When having multiple identical windows.

Basic Recording

- Generates a full selector for each activity.
- Generally slower than workflows which use attach containers.
- Recommended only when you have to use ONE activity per window.

Desktop Recording

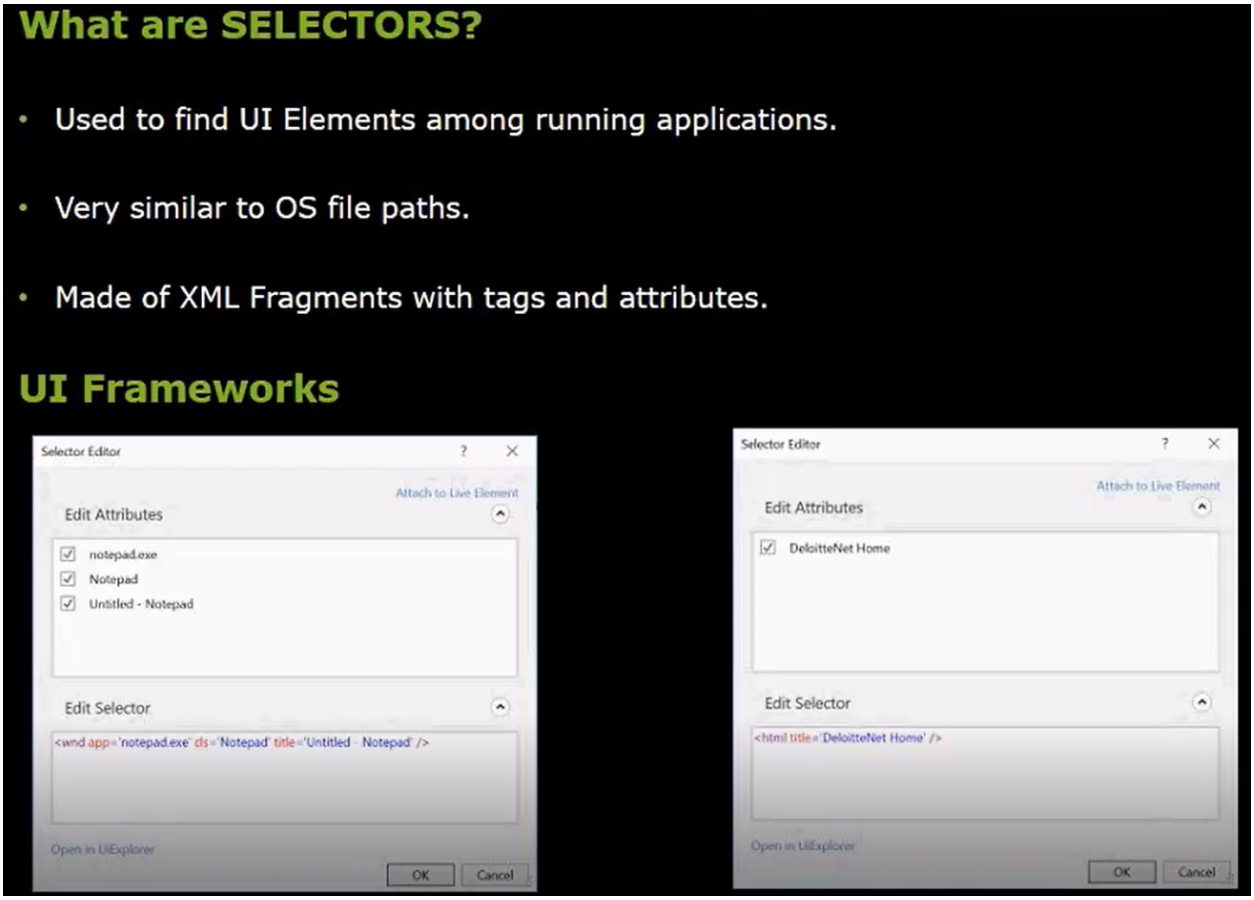
- Recorder generates an attach window container with the selector.
- Generates a partial selector for each activity inside attach.
- Optimal for doing more than one action.

Web Recording

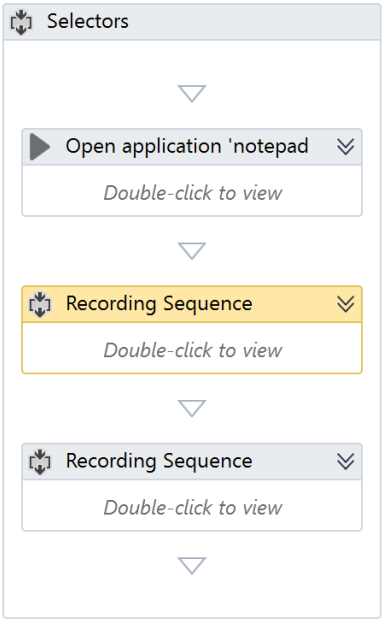
- Native support for **INTERNET EXPLORER**, install extensions for Chrome and Firefox.
- Actions taking place within a webpage are grouped under an Attach Browser container.

A Few Tricks

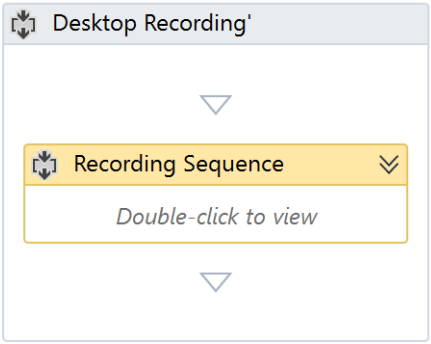
- For one action per container use Basic Recording Profile. For the rest use Desktop recording.
- Only make variables global (at top level scope) if they need to be used throughout the process. Otherwise put the variables in scope of where they are needed.



Basic recording

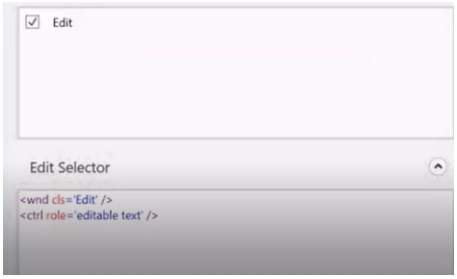


Desktop Recording

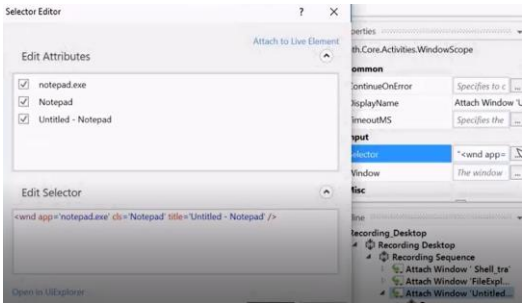


Everything under single Sequence.

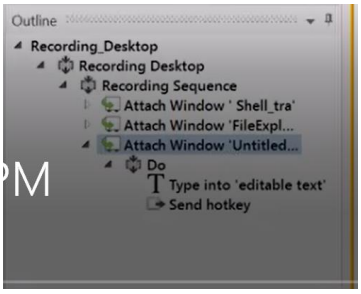
In Desktop Recording, Parent will have only have the complete selectors



Attach window will have the complete selector information.



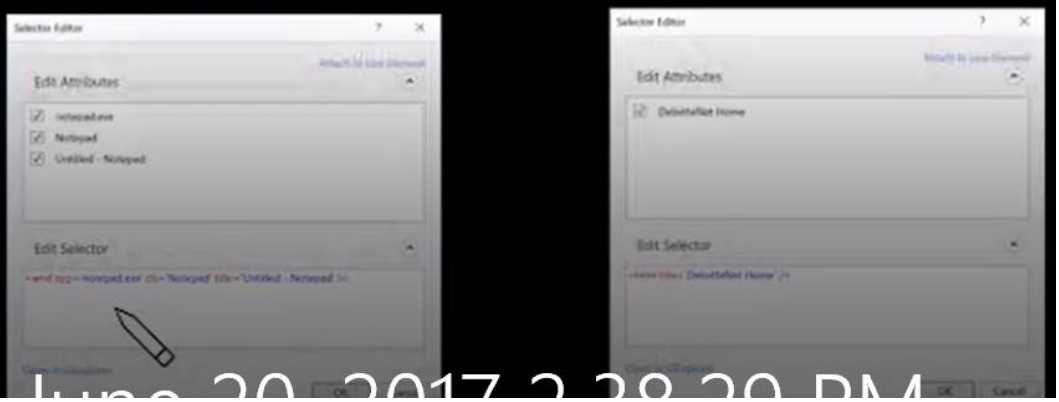
For each activity, selector is divided.



What are SELECTORS?

- Used to find UI Elements among running applications.
- Very similar to OS file paths.
- Made of XML Fragments with tags and attributes.

UI Frameworks



Basic Recording

- Generates a full selector for each activity.
- Generally slower than workflows which use attach containers.
- Recommended only when you have to use ONE activity per window.

Desktop Recording

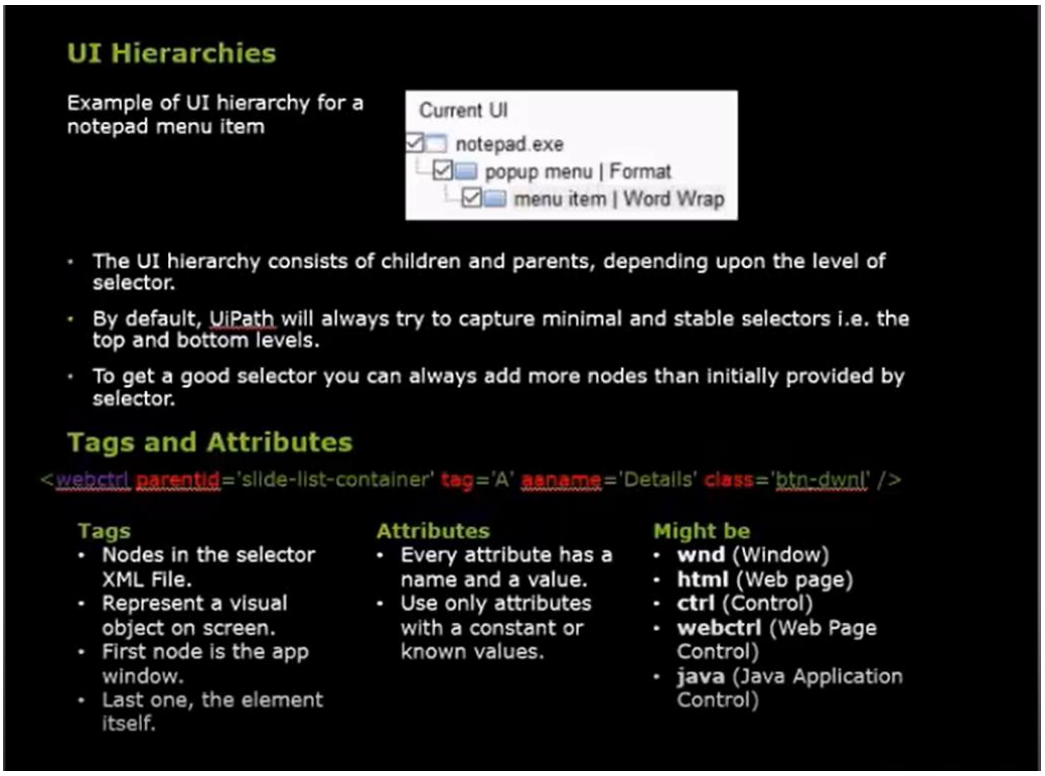
- Recorder generates an attach window container with the selector.
- Generates a partial selector for each activity inside attach.
- Optimal for doing more than one action.

Web Recording

- Native support for **INTERNET EXPLORER**, install extensions for Chrome and Firefox.
- Actions taking place within a webpage are grouped under an Attach Browser container.

A Few Tricks

- For one action per container use Basic Recording Profile. For the rest use Desktop recording.
- Only make variables global (at top level scope) if they need to be used throughout the process. Otherwise put the variables in scope of where they are needed.



Desktop Recording takes much less time than basic recording.

In Case of Basic recording, every time a new container is to be created. First will go to the parent selectors and then the child selectors.

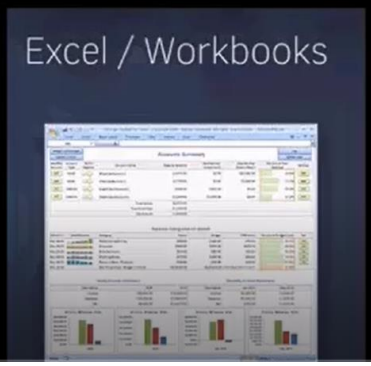
In Desktop parent will be created once, then all other inside it.

Excel Automation

- Excel Application Scope works as a container for all Excel Activities.
- We can use workbook or data table for excel automation

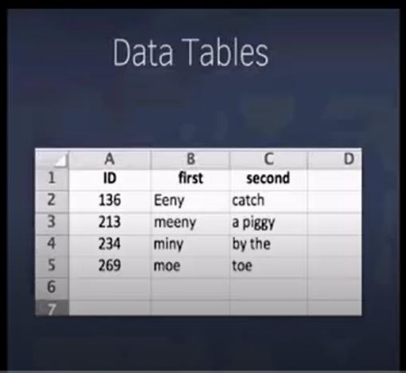
Work Book

Work Books are reference to Excel regardless of data, layout and format.



Data Table

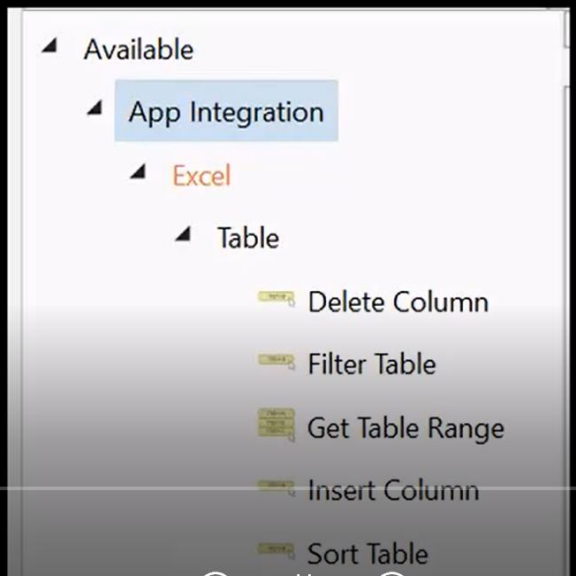
Data tables are the simplest types of spreadsheet data with rows and columns, with or without headers.

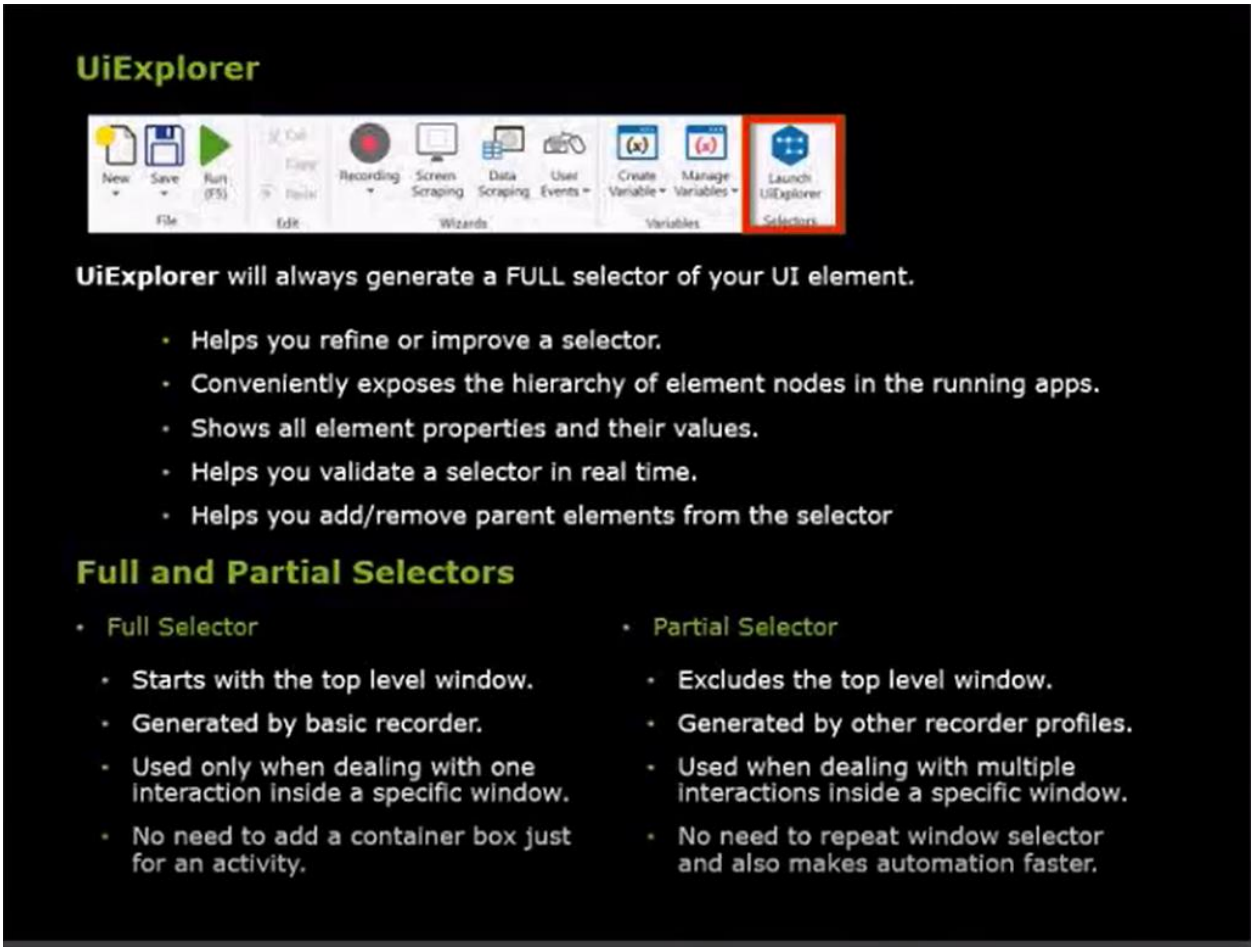


Basic Excel Activities

- **Sort Table**

Sorts a table from a spreadsheet based on the values of a column. The following options are available: Ascending - order from smallest to biggest (numbers) or from A to Z (alphabetic characters). Descending - order from biggest to smallest (numbers) or from Z to A (alphabetic characters).





If let say you rename Untiled -Notepad to something else, then the automation will fail.

There comes in UI explorer – we can use wildcards (*,?)

```
<wnd app='notepad.exe' cls='Notepad' title='Untitled?- Notepad' />
<wnd cls='Edit' />
<ctrl role='editable text' />
```

REVAMPED

Introduction

This page will help you get started with Studio. You'll be up and running in a jiffy!

SUGGEST EDITS

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** - suitable to linear processes, enabling you to smoothly go from one activity to another, without cluttering your project.
- **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** – suitable for very large projects; they use a finite number of states in their execution which are triggered by a condition (transition) or activity.

Note:

Administrator rights are not required to work with UiPath Studio, but only to install it.

Keyboard Shortcuts

SUGGEST EDITS

The complete list of keyboard shortcuts for UiPath Studio:

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.

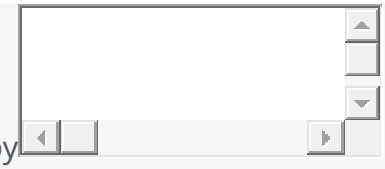
Note:

Modifier keys need to be written as full words, except for “Alt”. For example, write “Control” not “Ctrl”. Multiple modifiers should be separated through an empty space.

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**:

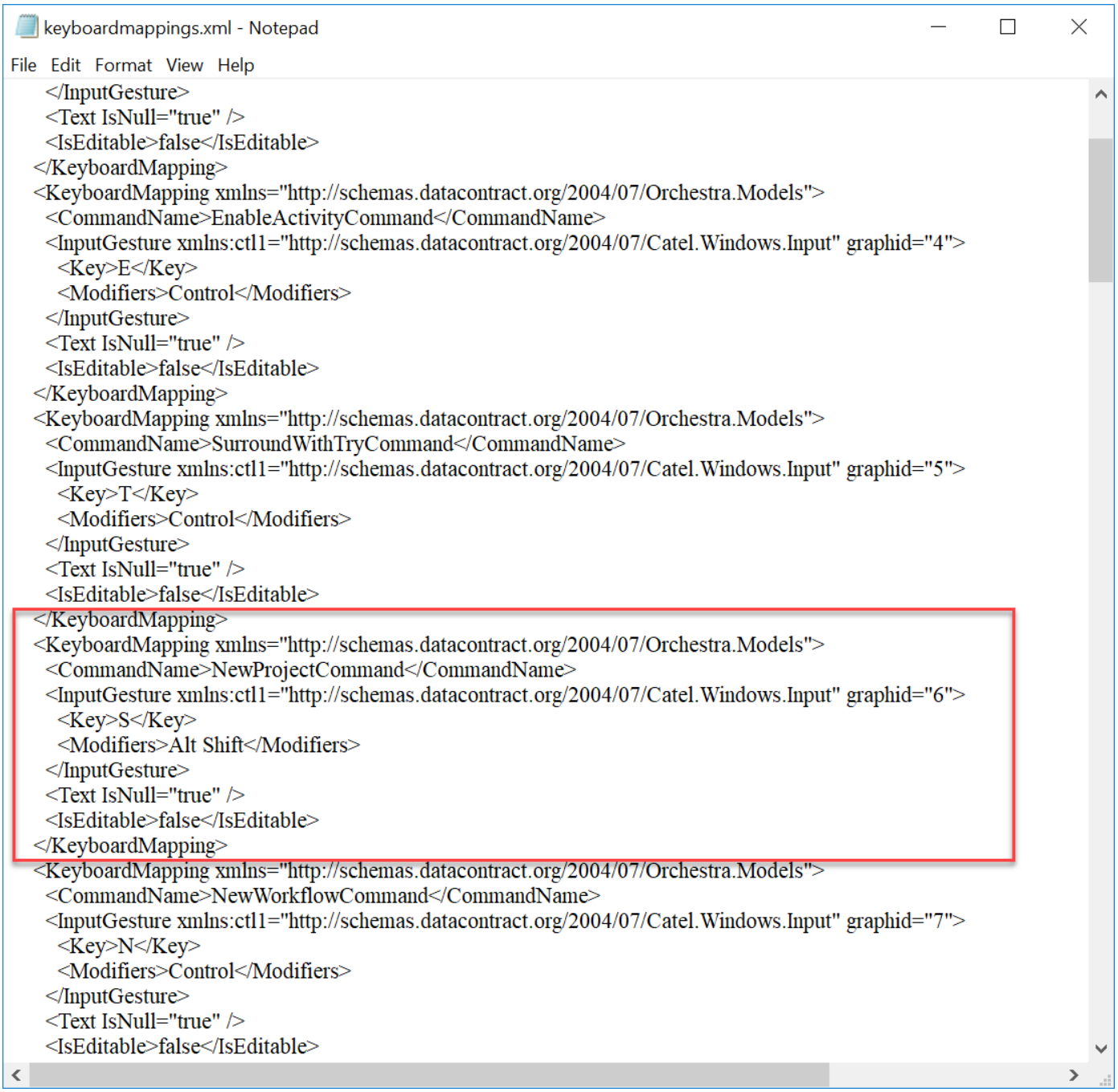
- **XML**




```
<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>
```

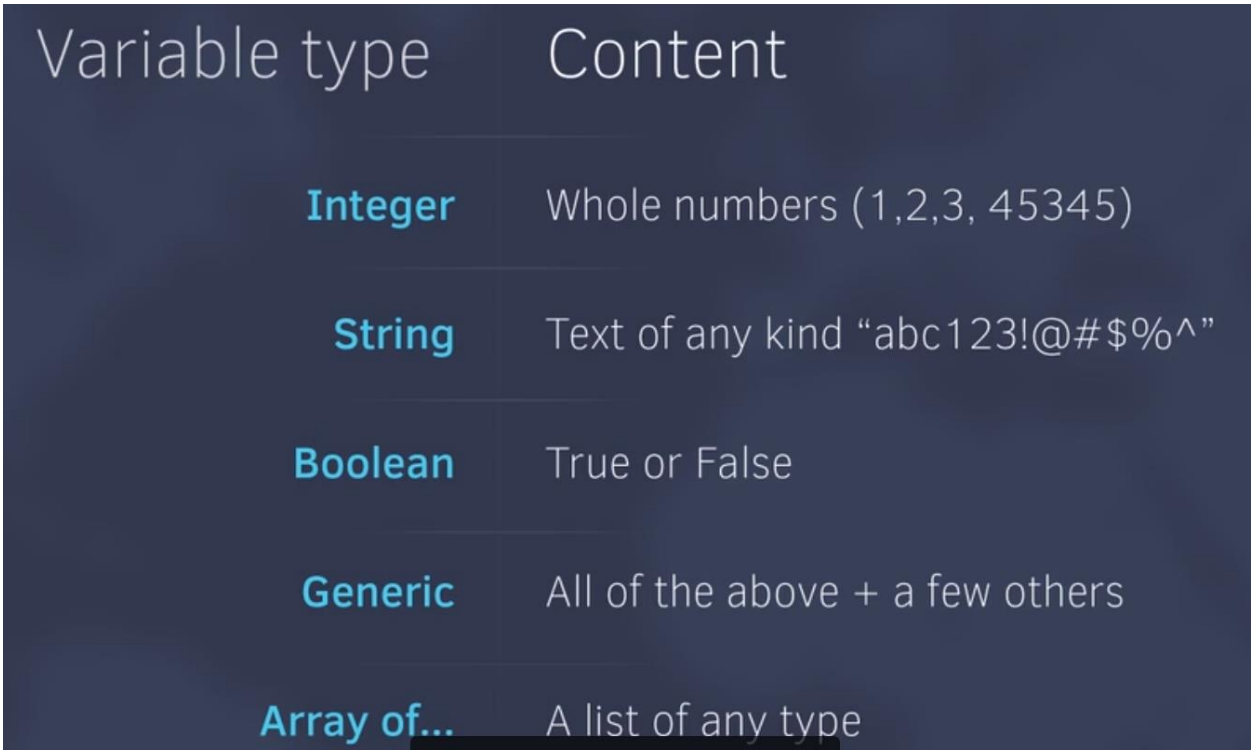
5. Replace the values between `<Key> </Key>` and `<Modifiers> </Modifiers>` with the desired keys. For example `<Key>S</Key> <Modifiers>Alt Shift</Modifiers>`. The keyboard shortcut should look as in the following screenshot.



6. Save and close the `keyboardmappings.xml` file.

Note:

The **F2, F3,** and **F4** keyboard shortcuts from Recording activities cannot be changed



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.

Write Range

SUGGEST EDITS

`UiPath.Excel.Activities.ExcelWriteRange`

Writes the data from a `DataTable` variable in a spreadsheet starting with the cell indicated in the **StartingCell** field. If the starting cell isn't specified, the data is written starting from the A1 cell. If the sheet does not exist, a new one is created with the value specified in the **SheetName** property. All cells within the specified range are overwritten. Changes are immediately saved. Can only be used in the **Excel Application Scope8**activity.

14. In the **DataTable** field, type the `datNamesList` variable.
15. 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.
16. Add another **Write Range** activity and place it under the first one.
17. In the **Properties** panel, fill in the **WorkbookPath** and **SheetName** fields as for the previous **Write Range** activity.
18. In the **Starting Cell** field, type "A7".
19. In the **DataTable** field, type the `datDate` variable.

We have everything 2 , one inside App Integration> Excel scope and other at system > workbook.

GenericValue Variables

SUGGEST EDITS

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.

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.

rguments 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.

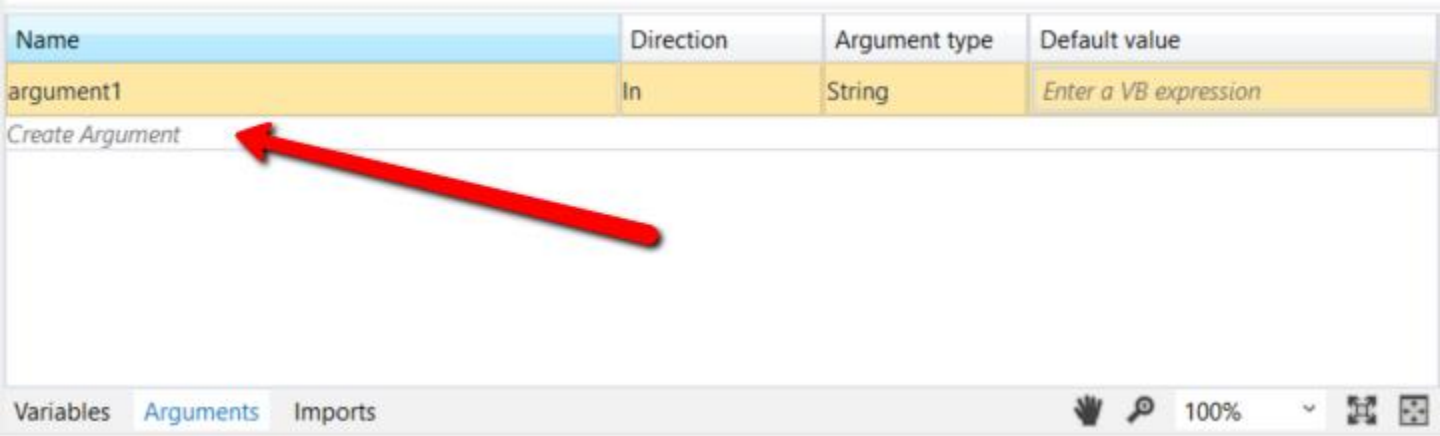
UiPath Studio supports a large number of argument types, which coincide with the types of variables. Therefore, you can create Generic Value, String, Boolean, Object, Array, or DataTable arguments and you can also browse for .Net types, just as you do in the case of variables.

Additionally, arguments have specific directions (**In**, **Out**, **In/Out**, **Property**) that tell the application where the information stored in them is supposed to go.

Creating Arguments

To create a new argument:

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

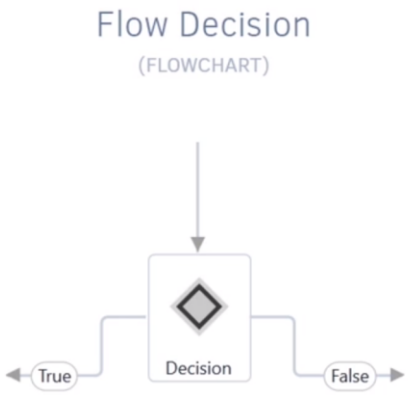


2. Click the **Create Argument** line. A new argument with the default values is displayed.

Note:

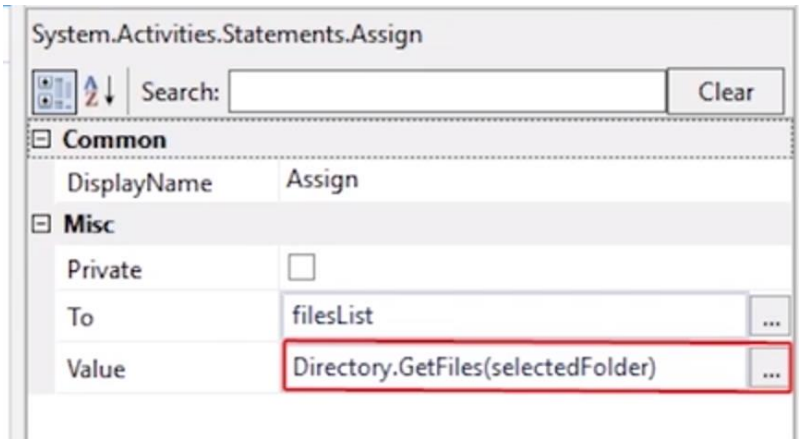
By default, all arguments are of **String** type and have an **In** direction.

Direction
Mandatory. Select a direction for your argument. The following options are available: In – the argument can only be used within the given project. Out – the argument can be used to pass data outside of a given project. In/Out – the arguments can be used both within and outside of a given project. Property – not currently used.



Get Names of all Files in a Folder.

Read Folder



Random Number Code : New Random().Next(0,100)

TYPES OF PROJECTS

1. 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.

2. 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.

3. 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.

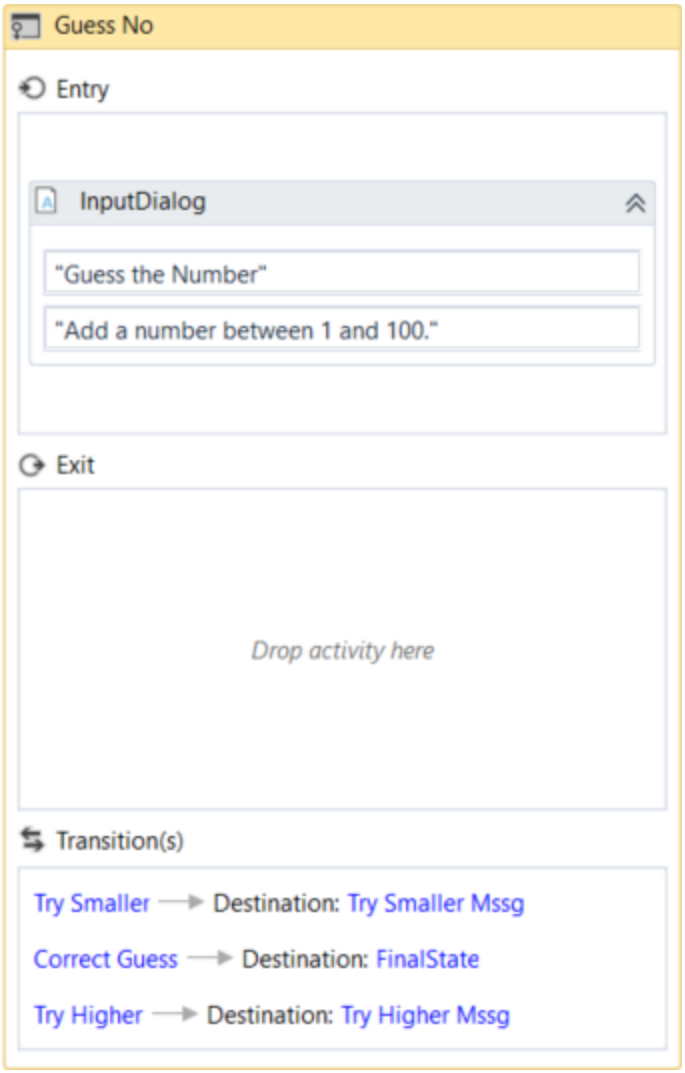
There are two activities that are specific to state machines, namely **State** and **Final State**, found under **Workflow > State Machine**.

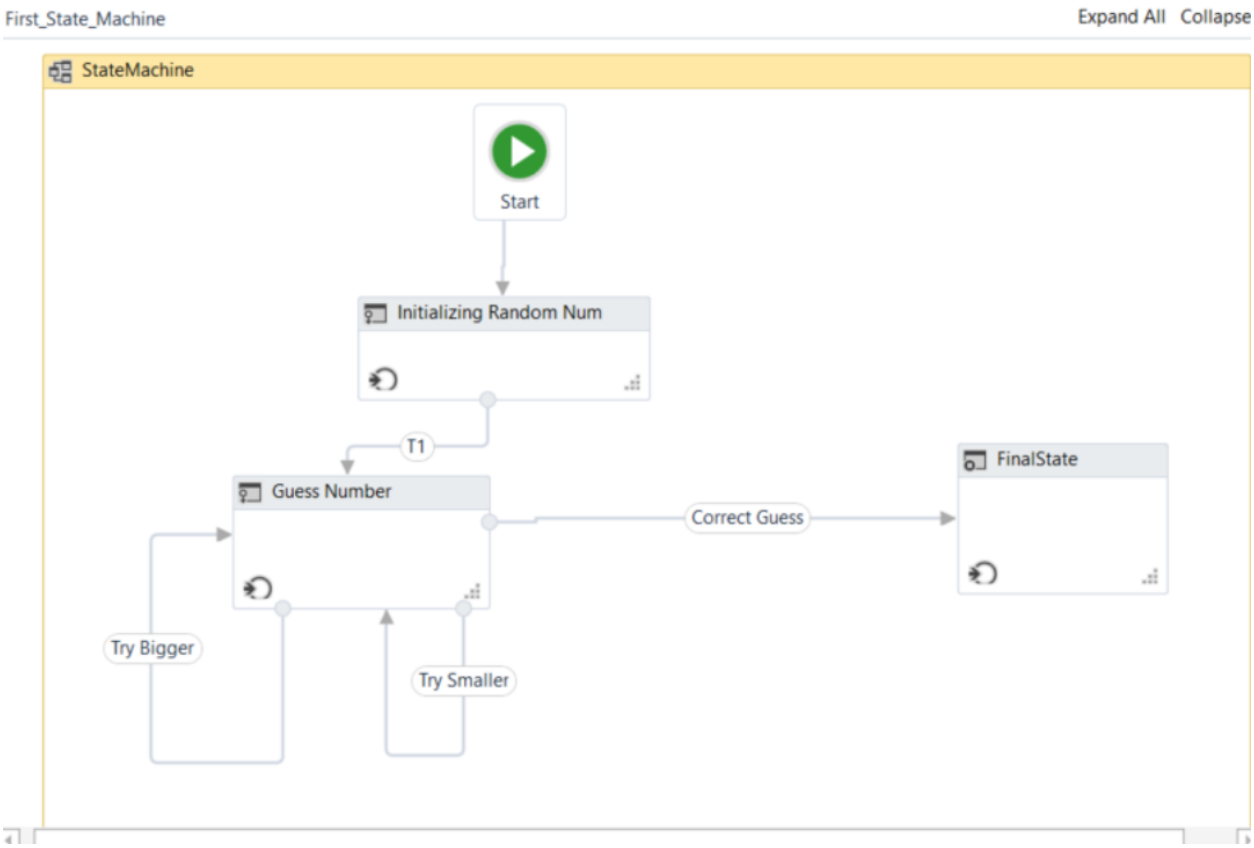
Note:

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.





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**.

The Assign Activity

SUGGEST EDITS

The **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 Activitychapter), sum up the value of two or more variables and assign the result to another variable (see the example in the Generic Value Variables), assign values to an array (see 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

The **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.

The Do While Activity

The **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.

The If Activity

The **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** one. However, the latter can only be used in flowcharts.

The Switch Activity

The **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.

The While Activity

The **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** 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.

The For Each Activity

The **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.

The Break Activity

The **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** activity.

Attach Window

SUGGEST EDITS

`UiPath.Core.Activities.WindowScope`

A container that enables you to attach to an already opened window and perform multiple actions within it. This activity is also automatically generated when using the Desktop recorder.

Properties

Input

- **Selector** - Text property used to find a particular UI element when the activity is executed. It is actually a XML fragment specifying attributes of the GUI element you are looking for and of some of its parents.
- **Window** - The window to attach to. This field accepts only Window variables.

Options

- **SearchScope** - The application window in which to search for the UI element defined by the Selector property.

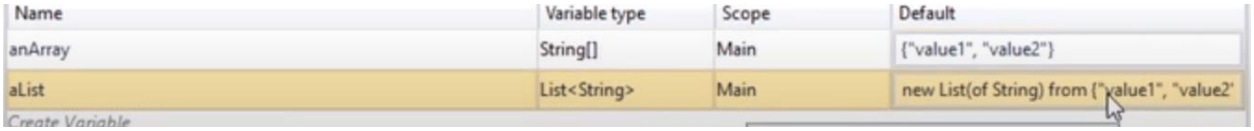
Common

- **DisplayName** - The display name of the activity.
- **TimeoutMS** - Specifies the amount of time (in milliseconds) to wait for the activity to run before an error is thrown. The default value is 30000 milliseconds (30 seconds).
- **ContinueOnError** - Specifies if the automation should continue even when the activity throws an error. This field only supports Boolean values (True, False). The default value is False. As a result, if the field is blank and an error is thrown, the execution of the project stops. If the value is set to True, the execution of the project continues regardless of any error.

Note:

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

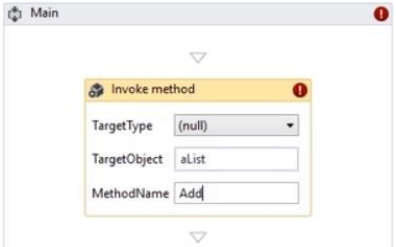
Initializing a list and Array



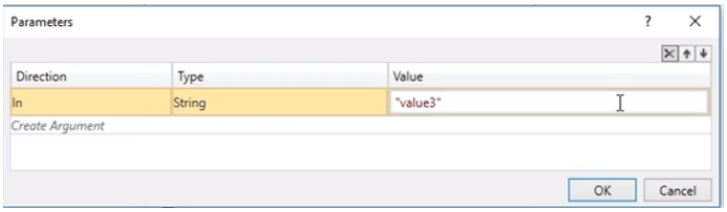
Array size is fixed.

List values can be added or Deleted.

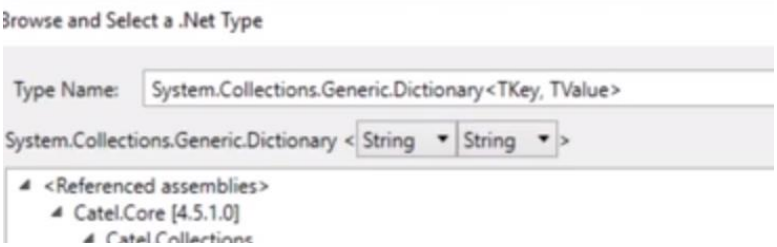
To add new element in a list we need to use a Invoke method activity.



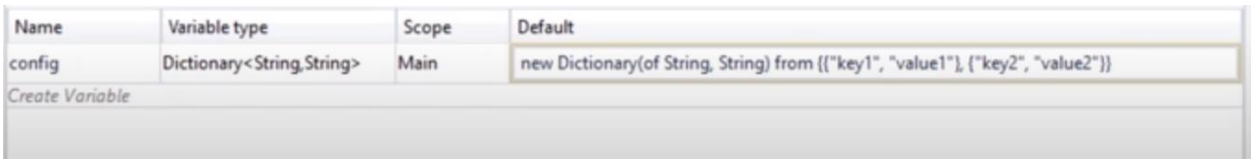
Add New value in the parameter as shown below.



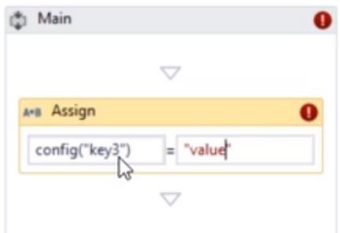
Dictionary



Assigning Default Value

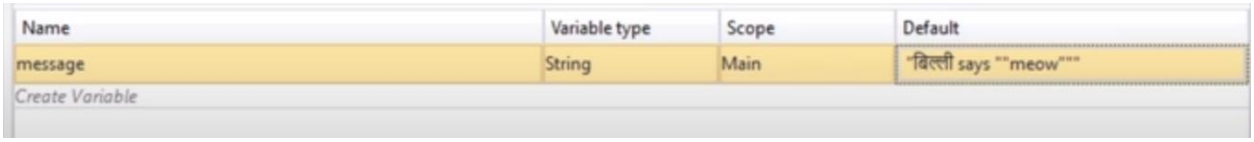


Setting variable value



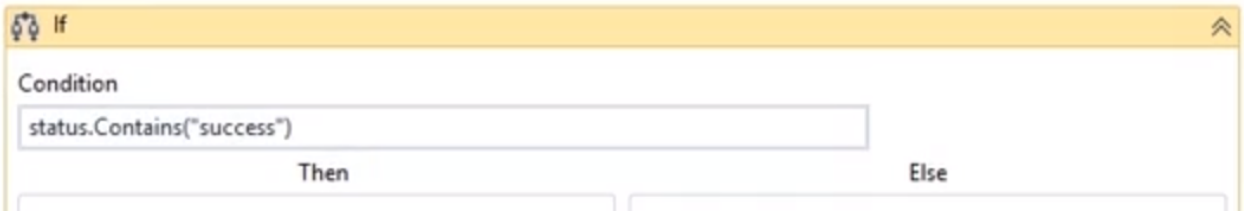
Escape Sequence:

To print Meow in “” . type ““meow””

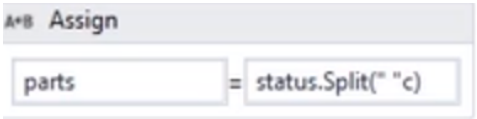


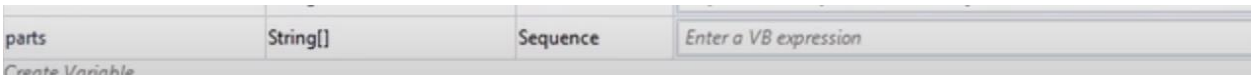
String Manipulation:

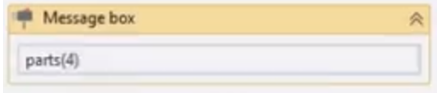
1.Contains: Returns Boolean



2.Split

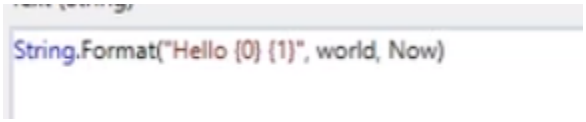


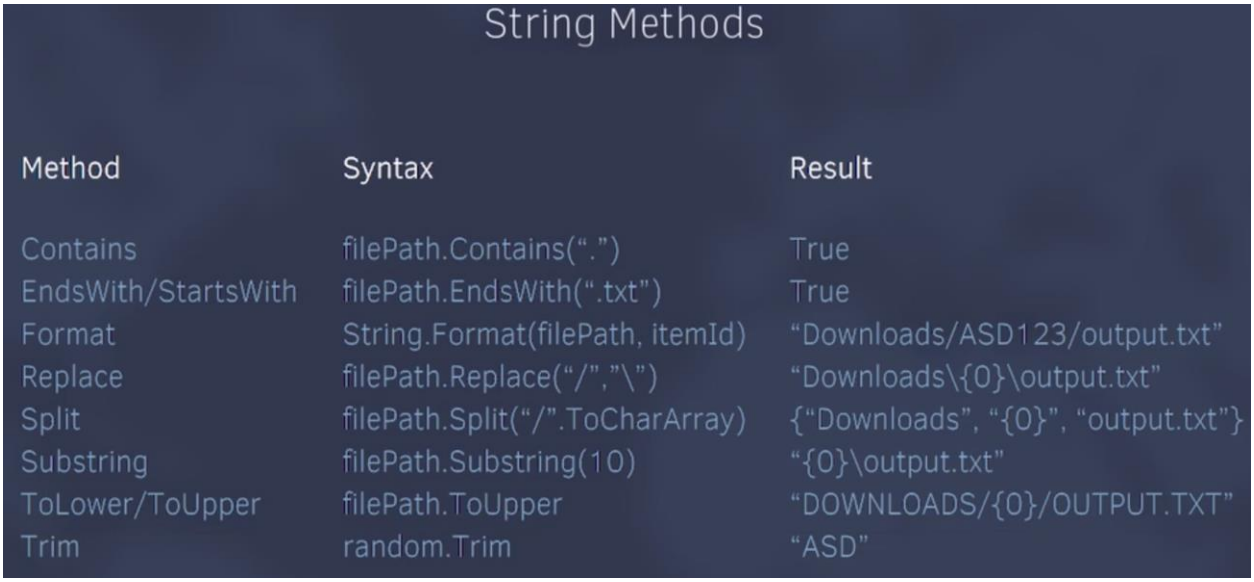




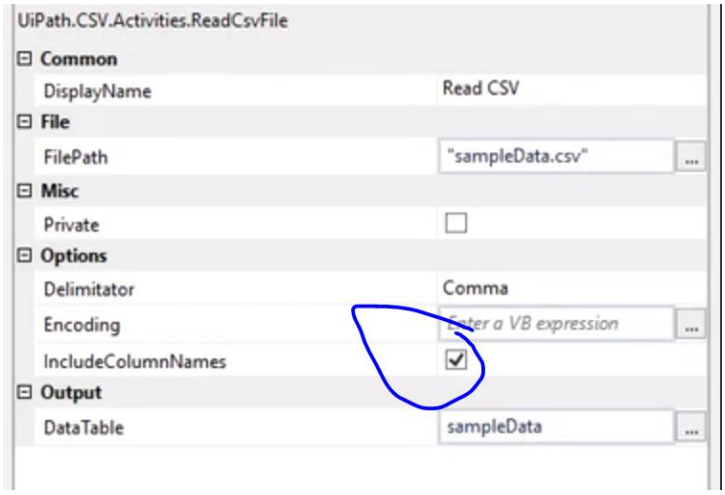
StringSplitOptions.None : Means Do not Retuen Null or empty

String.Format

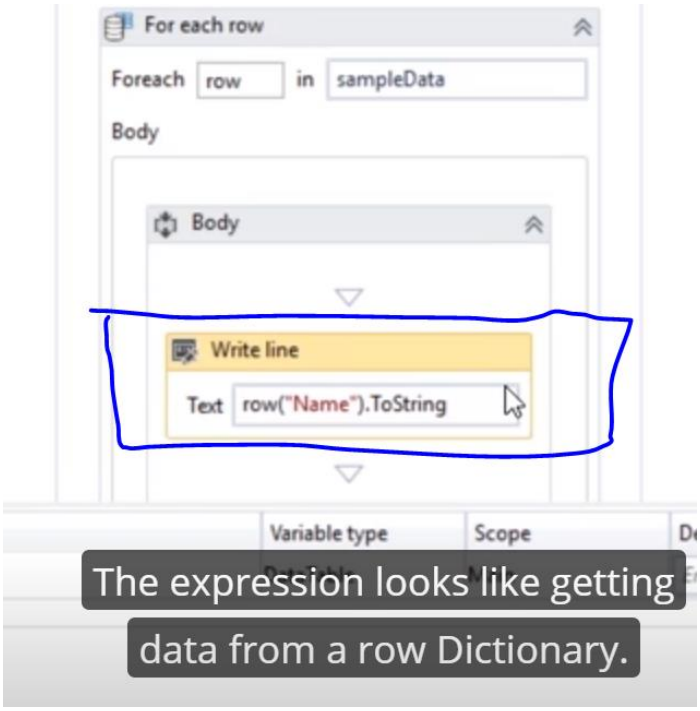




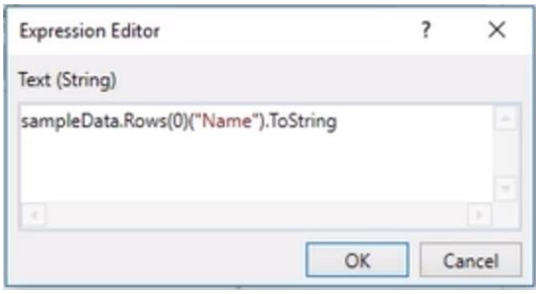
DataTables



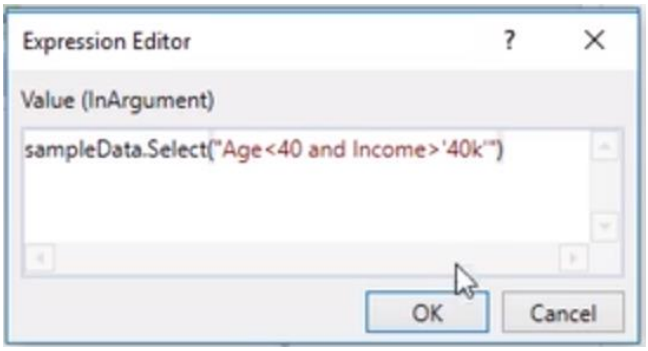
To interpret First Row as Header



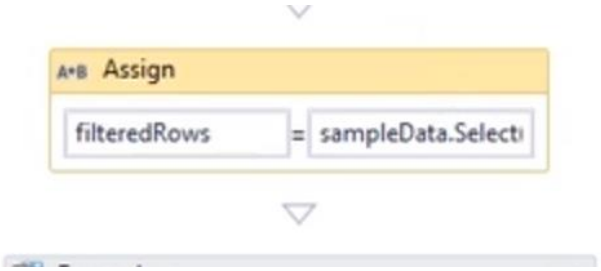
To Get value of a particular column



To Get selected values frim DataTable



Assign it to an array of Data Rows



We must use [square brackets] with the **Select** method when the column name contains spaces.

clubMembers = In the **Value** field, type **names.Select("[Club Member]= 'Yes' ")**

That extracts all the rows where the Club Member field matches the "Yes" string and stores them into the **clubMembers** array variable.

firstName.Substring(0,3).ToUpper()+lastName.Substring(0,3).ToLower()

.

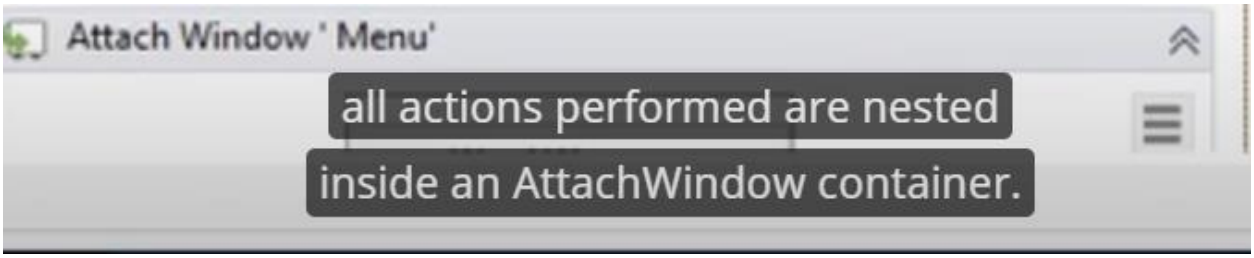
Most of the variables can be split in these categories: scalar, collections and tables.

When using Read CSV make sure you check the property IncludeColumnNames if you want the first row to be treated as column names. You can filter table rows by using the Select method.

names.Select("[Club Member]='Yes'")

Recordings

Desktop recording



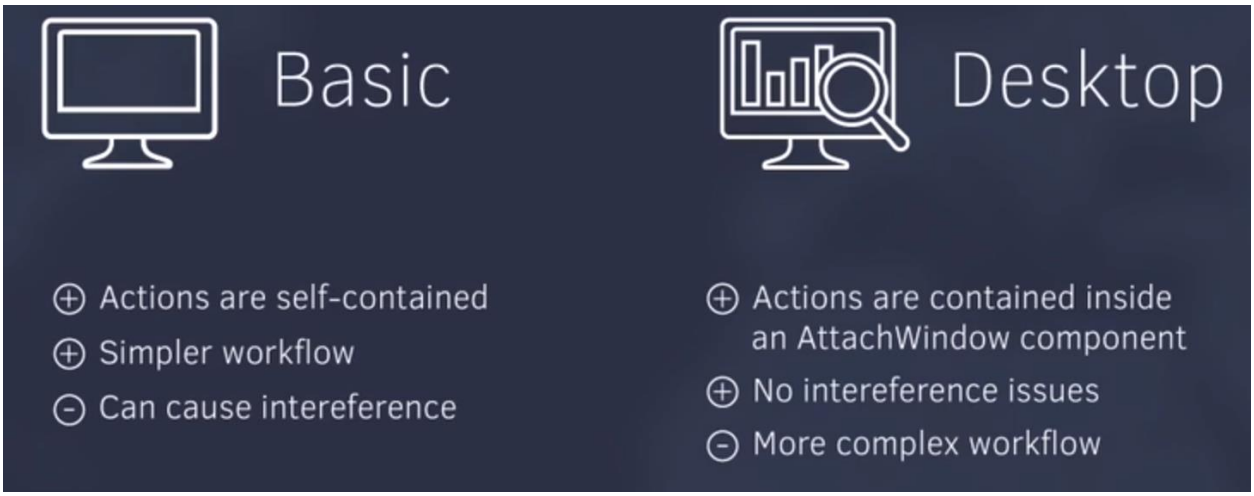
Desktop recording creates Three different containers nested inside attach window.

1. Actual notepad window
2. Menu Pop up
3. Font Dialog

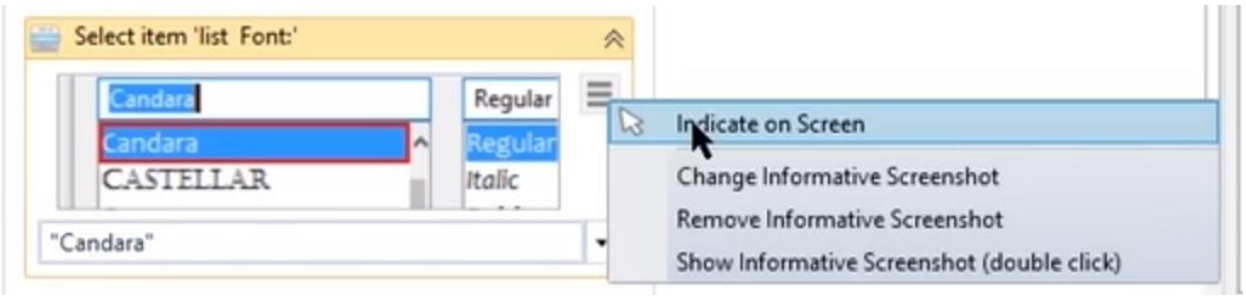
The Container are for : when you want to ensure that other window of the same application don't interfere with each other.

UI path used the name of the application , the title of the windows , the current open file and other such details to ensure that it correctly identifies the right window.

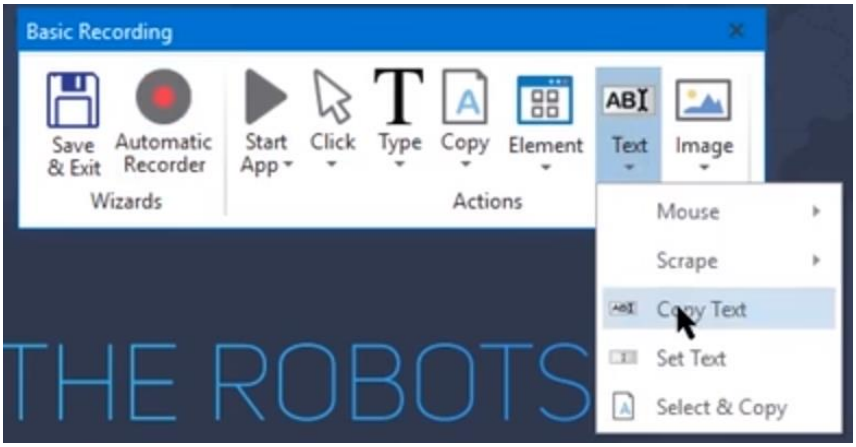
If we open two instances of the same window , having same name .Ui Path will run the automation on the topmost open instance running into problems.



To Update the Informative screenshot you have to open the instance in the instance window in notepad.

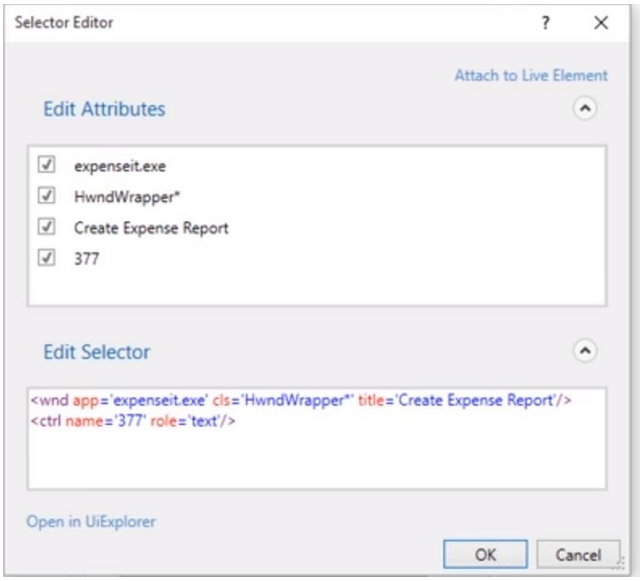


To get the text from screen



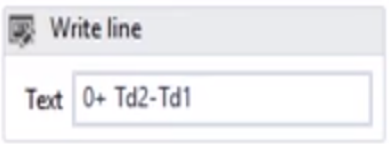
Selector:

When Ui path have trouble finding an element, Look for the selectors.



Click on Attach to live element to update the selectors.

Convert string to int



Open Browser

Close Window: Put it inside attach window , uipath will know which application to close.

Note:

If you make changes to your display settings without restarting the computer, elements are not going to be properly identified.

Recording Types

SUGGEST EDITS

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.

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

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

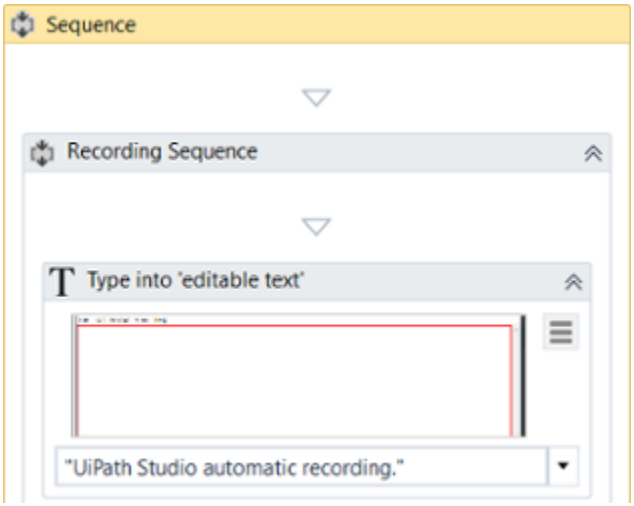
This is the improved accessibility model from Microsoft. It is recommended that you use this option with newer apps, when the Default one does not work.

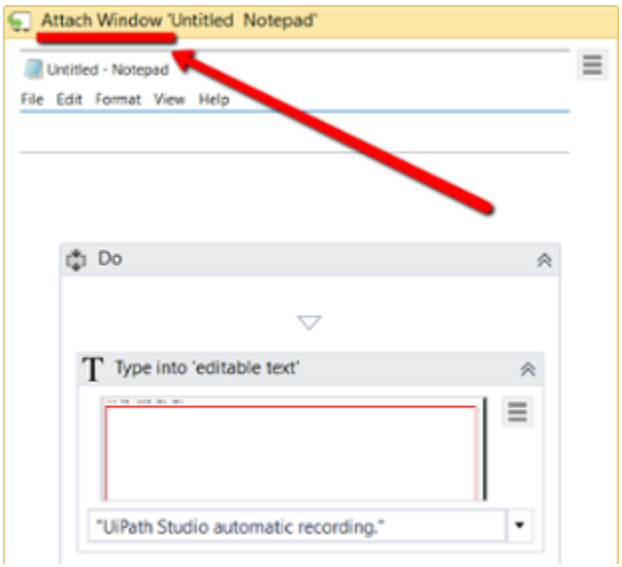
Note:

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

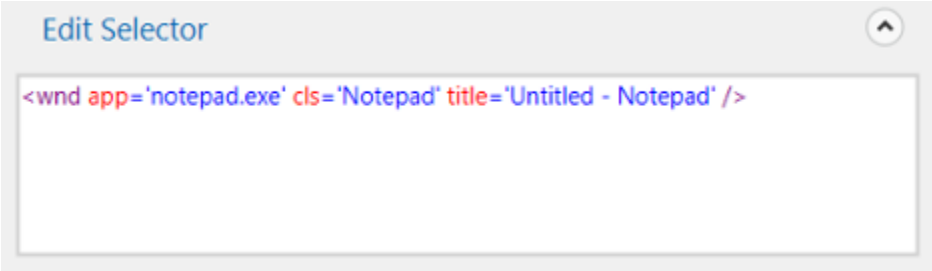
Example of Automatic Recording with Basic and Desktop

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** container, while the **Basic** one does not.

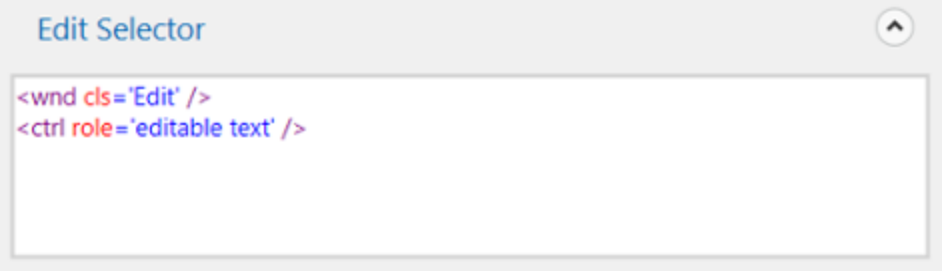




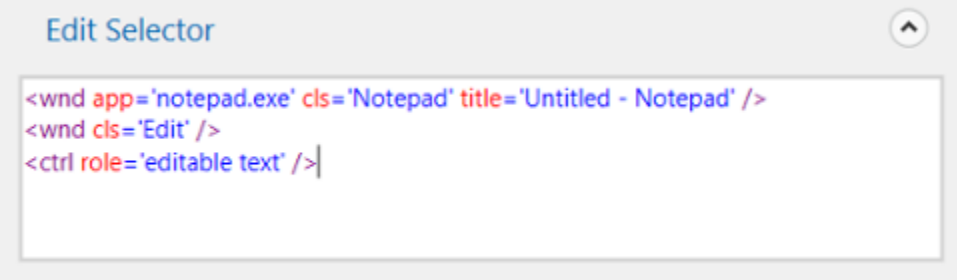
Desktop recorder - The top level window selector from the **Attach Window** container:



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



Basic recorder – The full selector for the **Type Into** activity:



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** and **Close Tab** activities at the beginning and end of the project.

Manual Recording

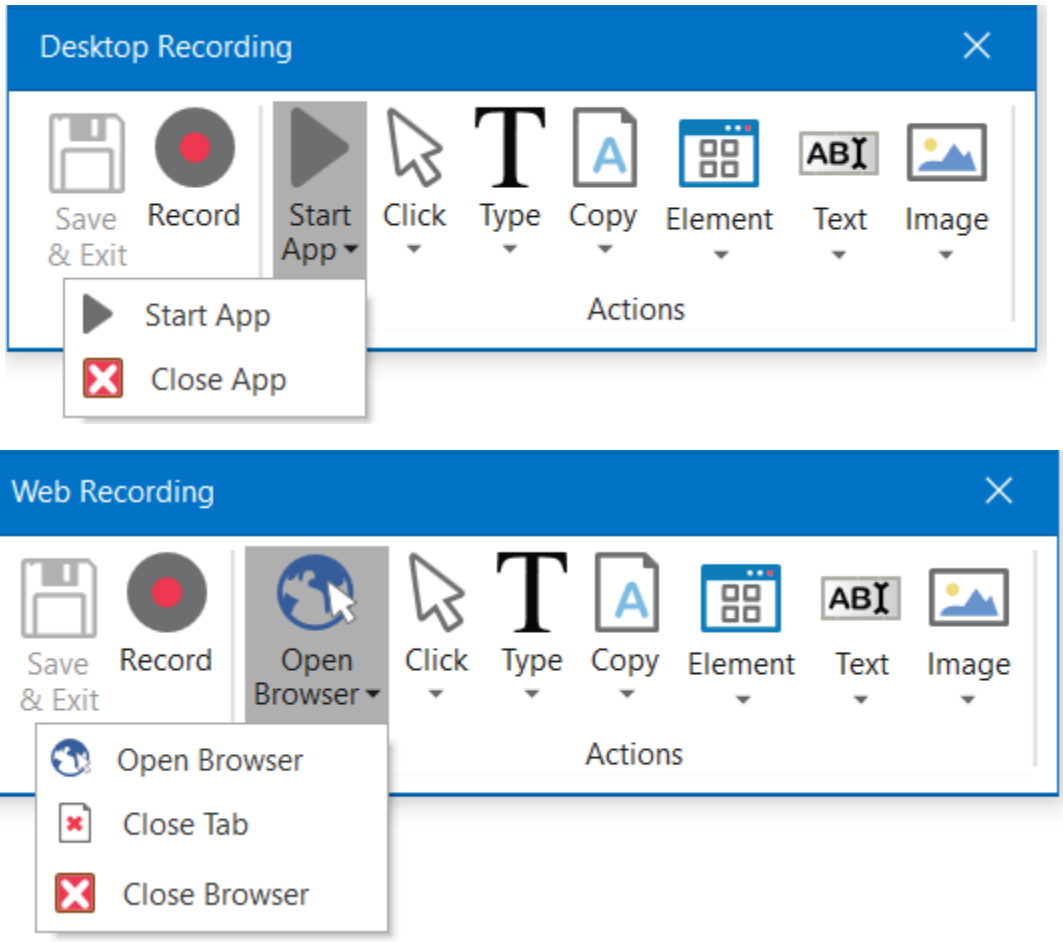
As explained in the About Recording Types chapter, there are some actions that cannot be handled by the automatic recorder. We refer to these as single actions or manual recordings. You can use both manual and automatic recording in automations to achieve full automation of a task. Single actions can 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.

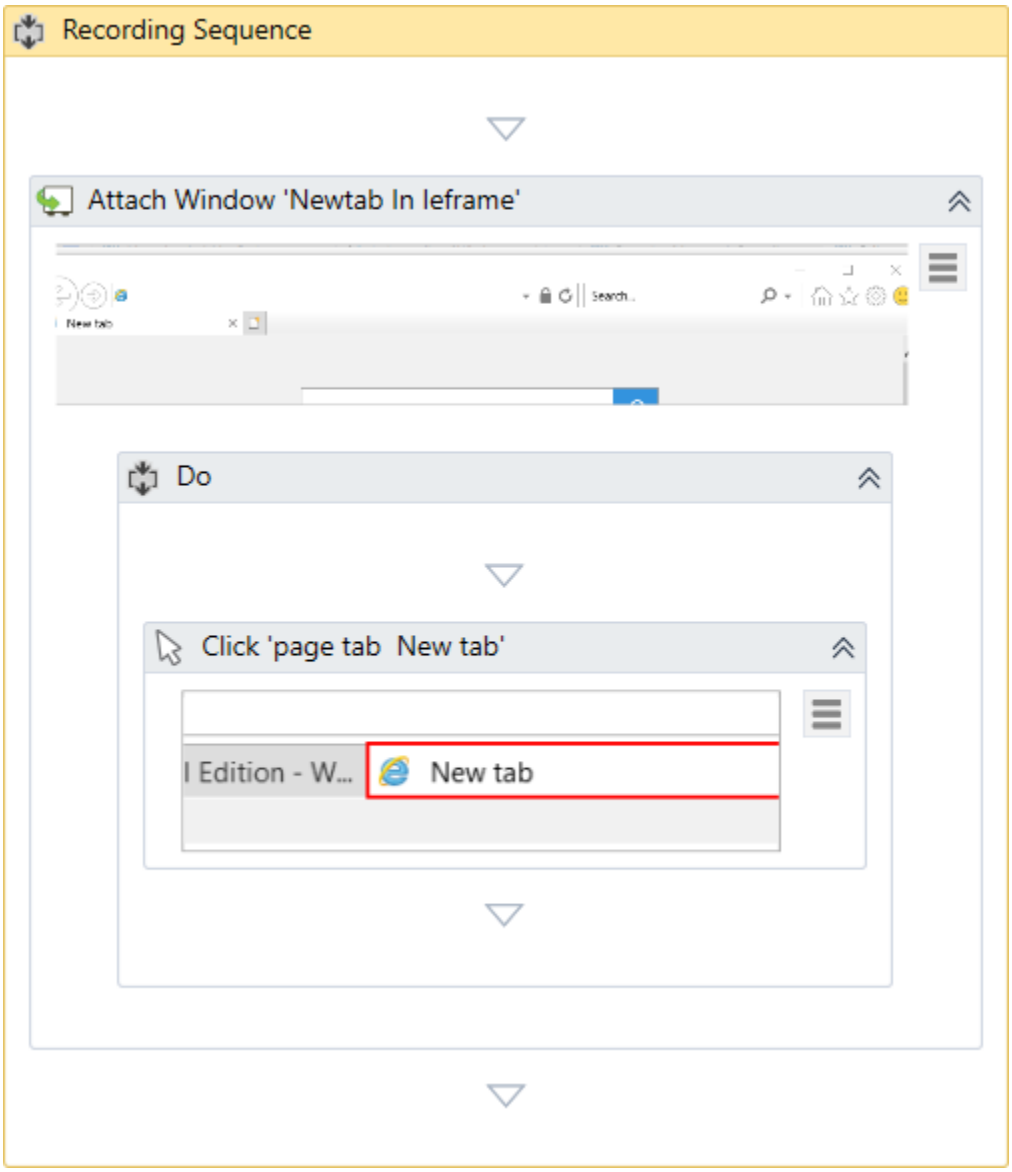
Types of Single Actions

Start and Stop an App or Browser

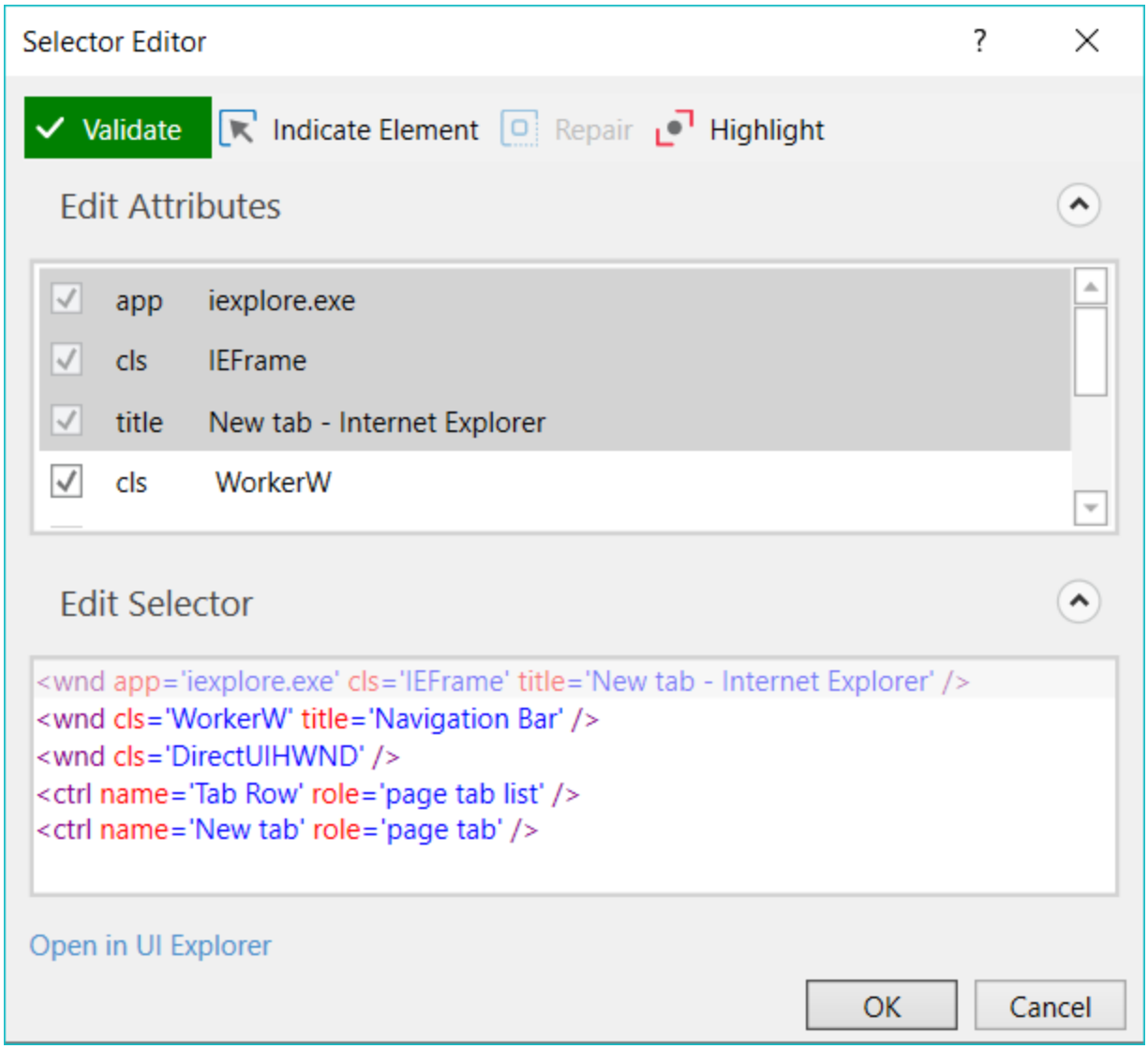


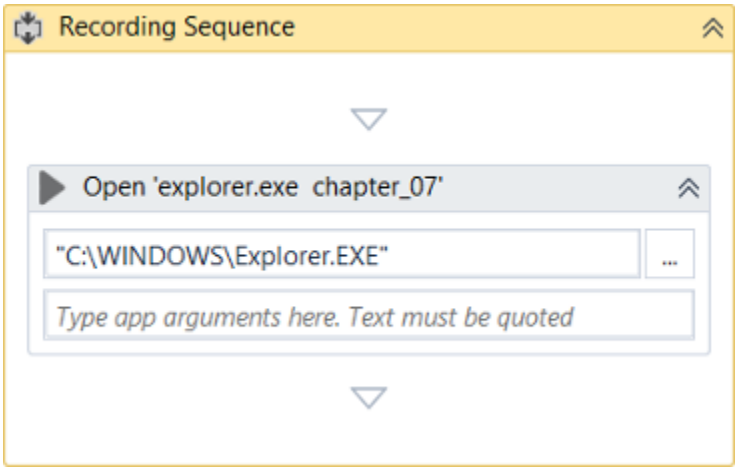
These single actions enable you to open an app or browser, as well as close them, by pointing and clicking them.

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 (third screenshot), just like with the automatic recording. More information about selectors is available in the [About Selectors]page.

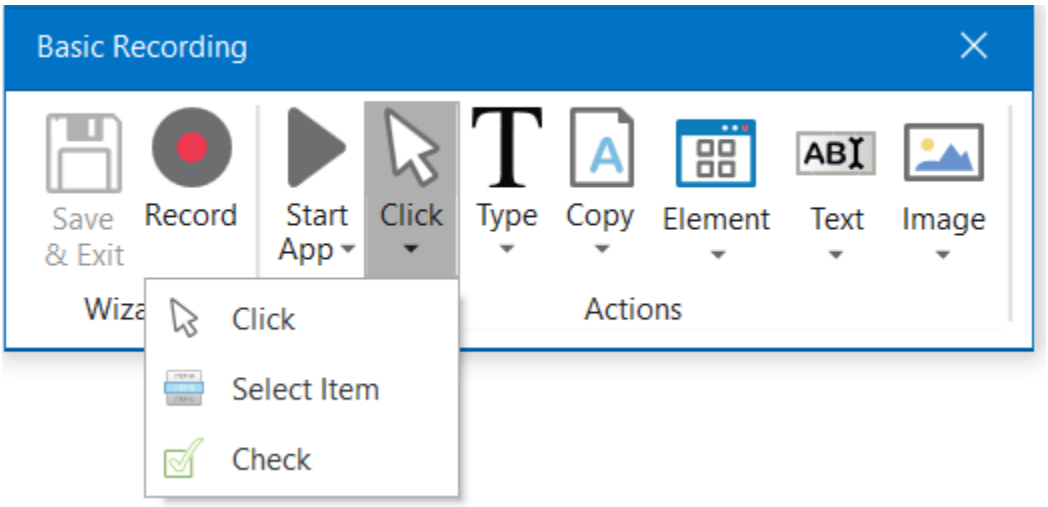


The partial selector for the example's **Click** activity:


Selector Editor
Validate
Indicate Element
Repair
Highlight
Edit Attributes
app iexplore.exe
cls IEFrame
title New tab - Internet Explorer
cls WorkerW
Edit Selector
<wnd app='iexplore.exe' cls='IEFrame' title='New tab - Internet Explorer' />
<wnd cls='WorkerW' title='Navigation Bar' />
<wnd cls='DirectUIHWND' />
<ctrl name='Tab Row' role='page tab list' />
<ctrl name='New tab' role='page tab' />
Open in UI Explorer
OK
Cancel

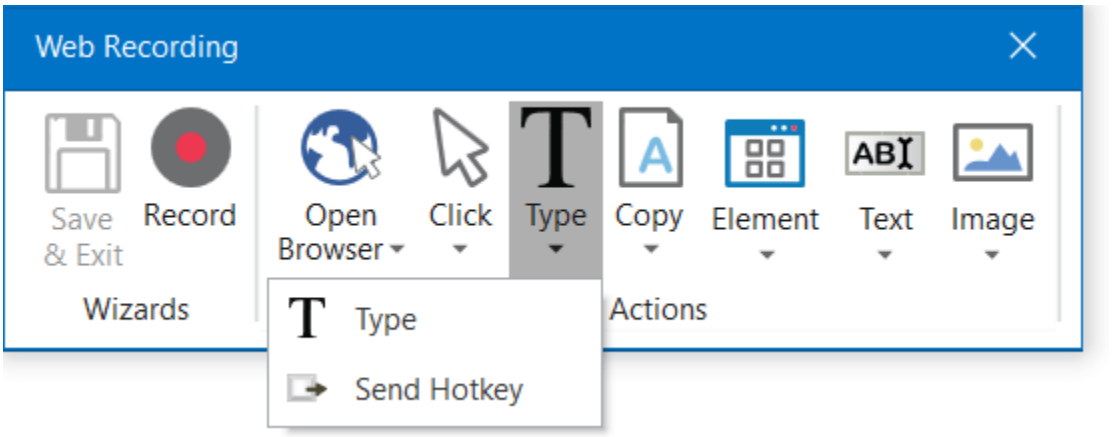

Recording Sequence
Open 'explorer.exe chapter_07'
"C:\WINDOWS\Explorer.EXE"
...
Type app arguments here. Text must be quoted

Click

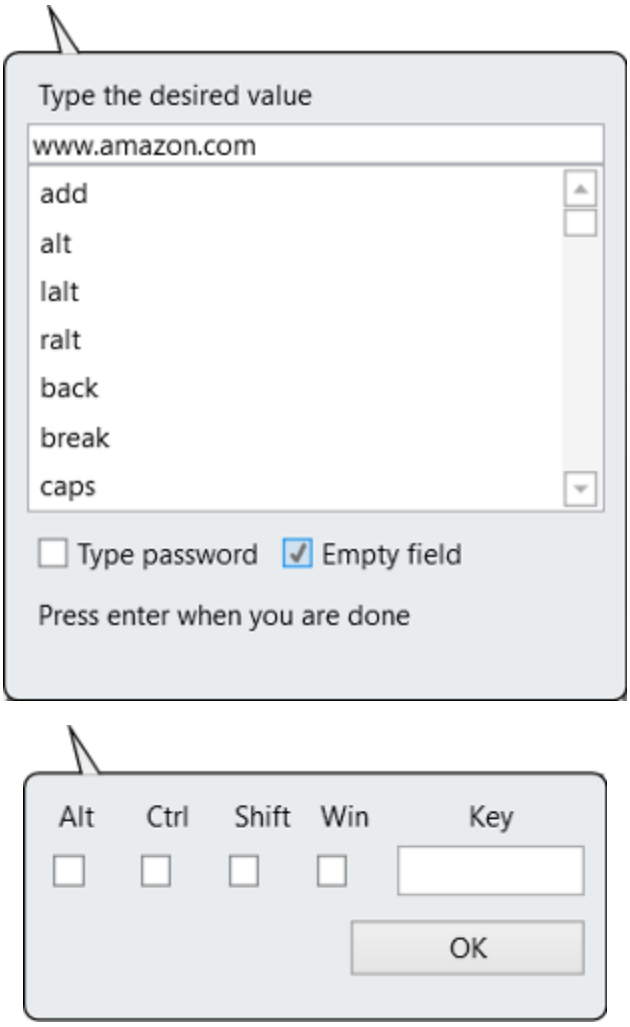


These types of actions enable you to record clicks on the desktop or a running application, select an option from a drop-down list or combo box, and select a check box or a radio button.

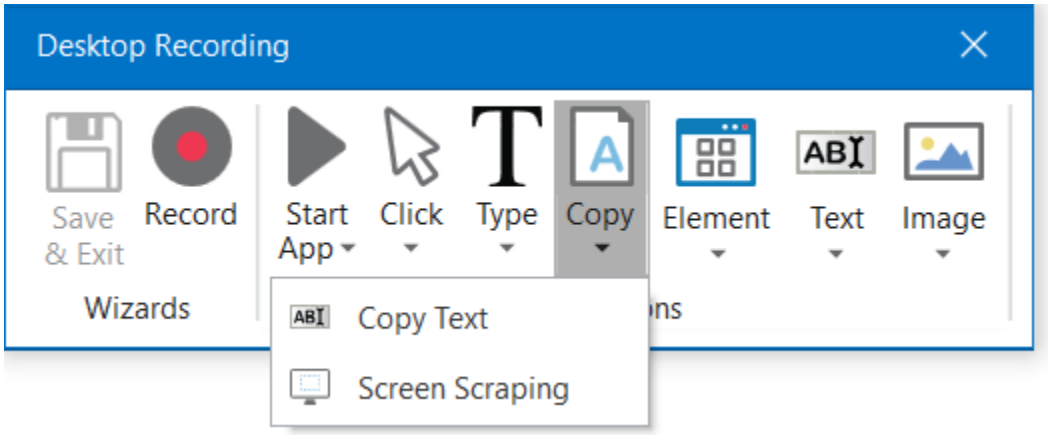
Type

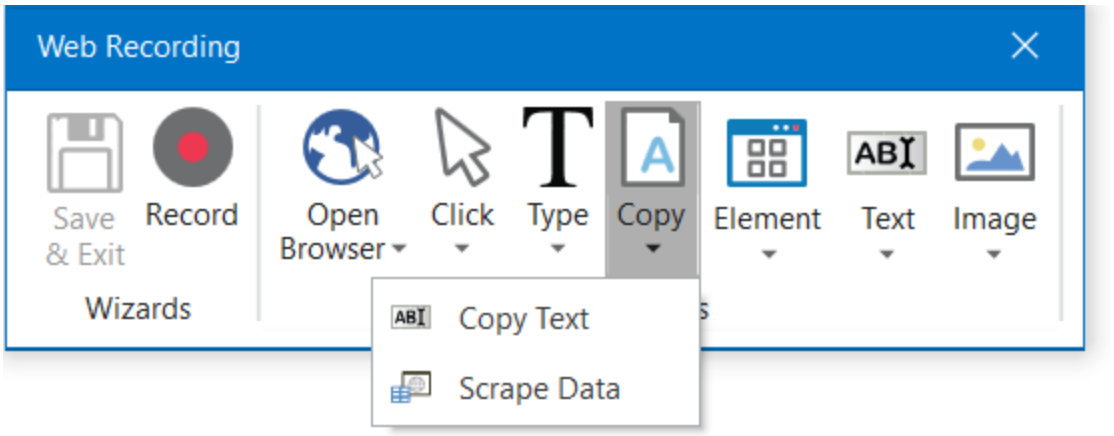


These single actions include those that require input from the keyboard, such as keyboard shortcuts and key presses. To achieve this, two pop-up windows are used to retrieve your keyboard input.


Type the desired value
www.amazon.com
add
alt
lalt
ralt
back
break
caps
Type password
Empty field
Press enter when you are done
Alt
Ctrl
Shift
Win
Key
OK

Copy

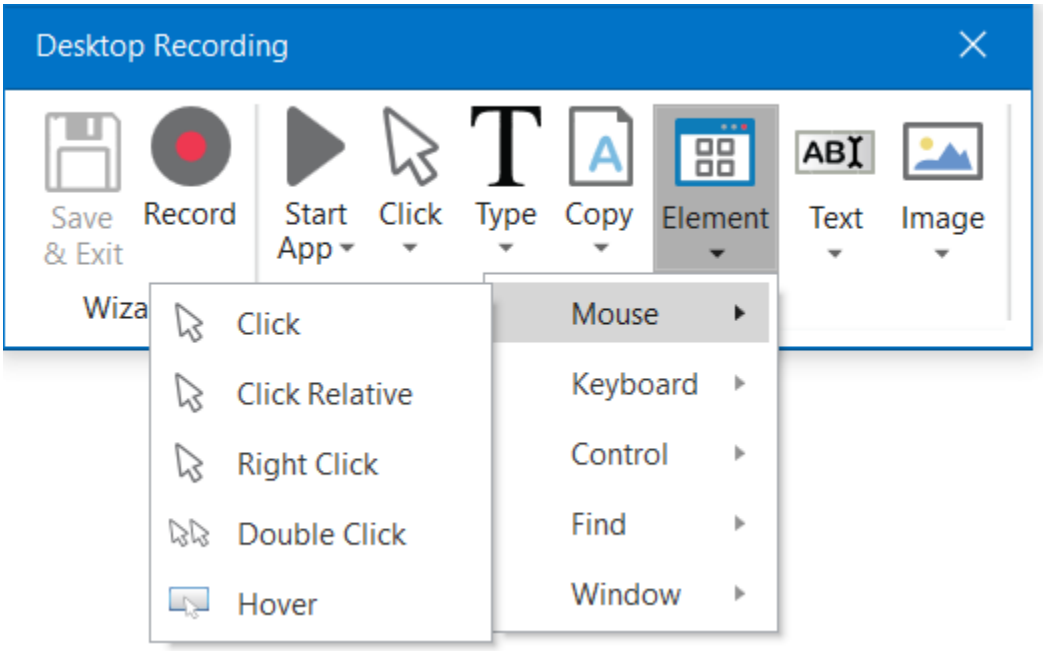

Desktop Recording
Save & Exit
Record
Wizards
Start App
Click
Type
Copy
Element
Text
Image
Copy Text
Screen Scraping



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.

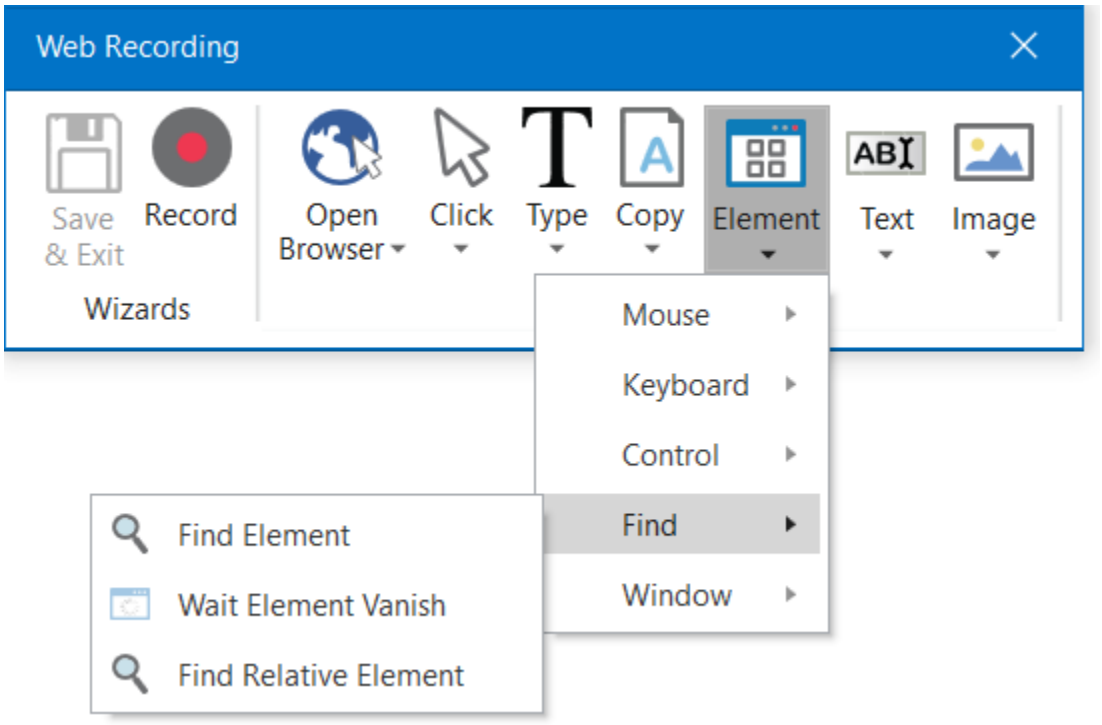
For more information, see Screen Scraping.

Mouse Element



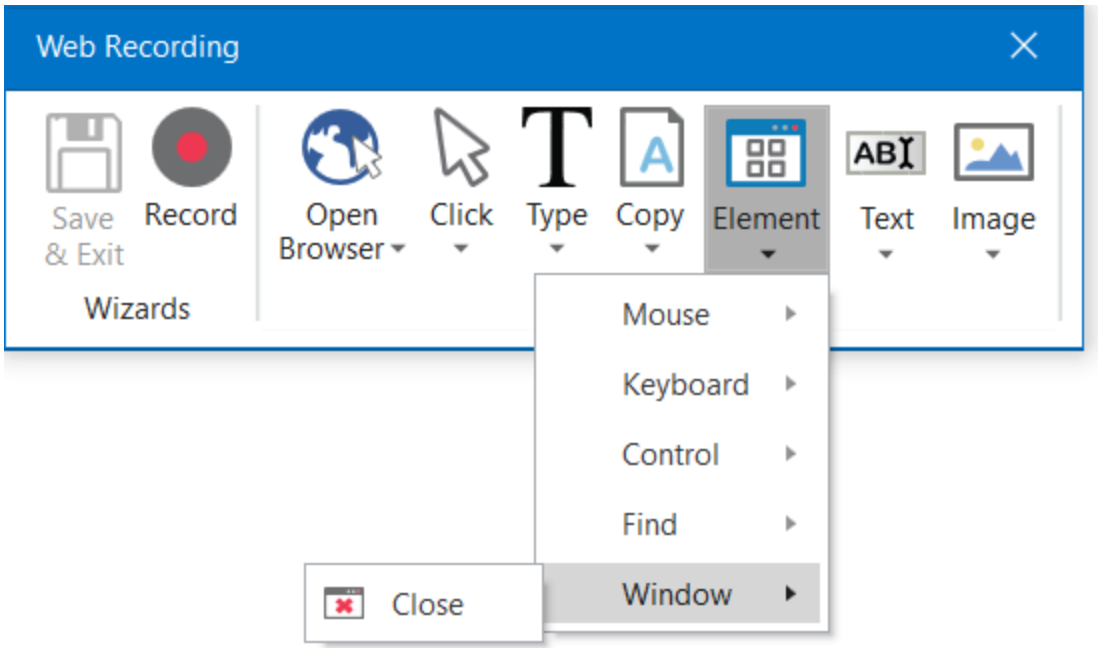
These types of actions enable you to simulate mouse movements that cannot be recorded but give you access to more functionalities, such as right-clicking, hovering or double-clicking.

Find Element



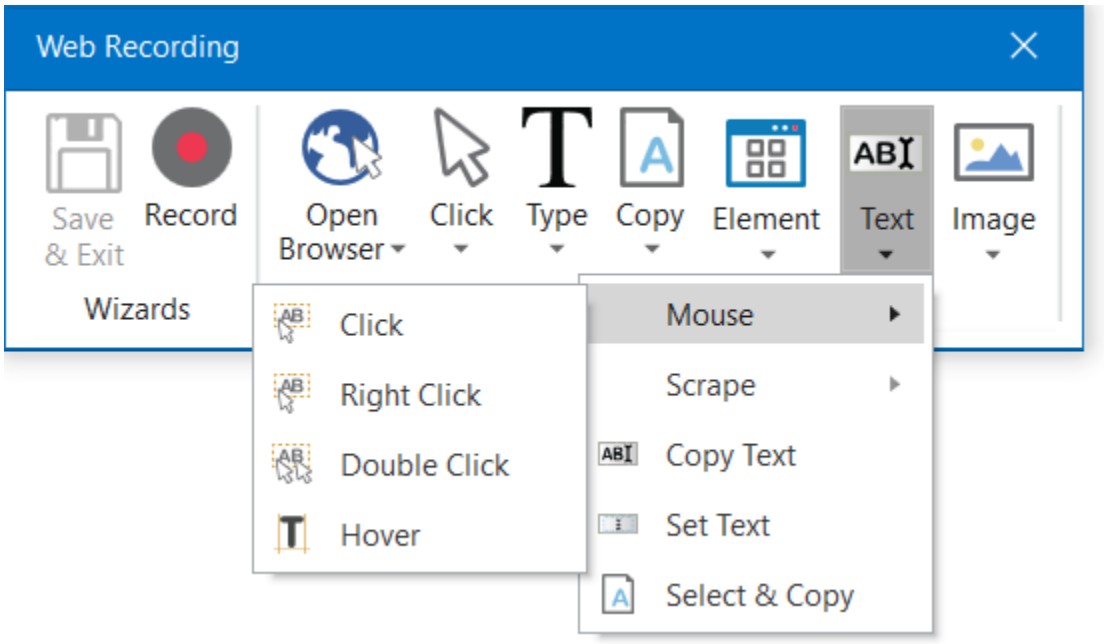
These types of single actions enable you to identify specific UI elements or pause the automation until a particular window closes or an UI element is no longer displayed. The find relative element action is useful with apps that do not allow direct interaction with UI elements, such as Citrix.

Window Element



Window element actions enable you to close windows. Studio does this 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

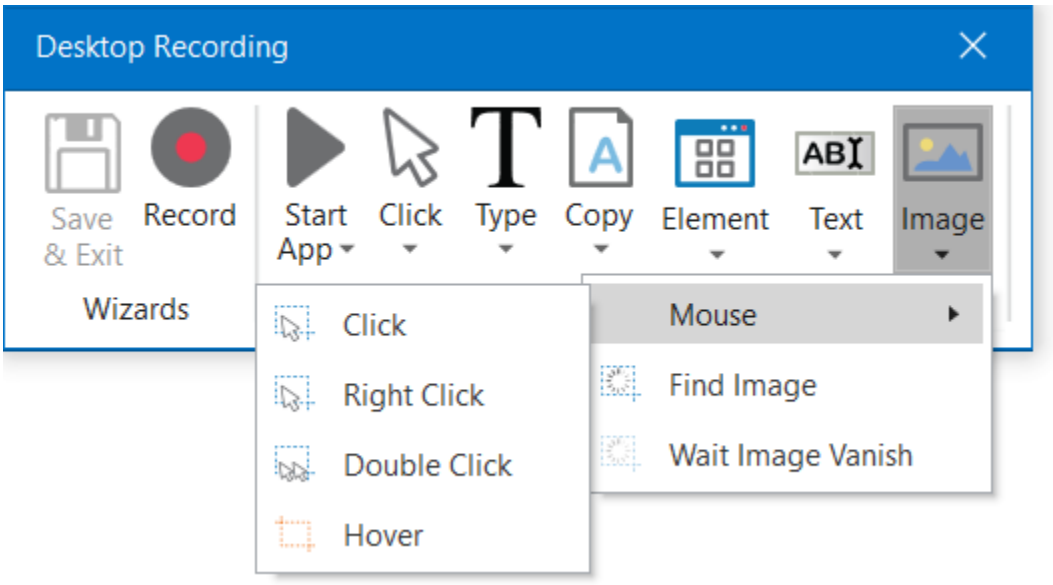


Image single actions enable you to wait for a specified image to disappear, to find a specific image in an app or website, right-click or hover over an image and others. This type of manual recording can be useful with UI elements that are saved as graphics, as they cannot be highlighted as normal fields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

There are 4 types of recording:

Basic, Desktop, Web and Citrix.

You can record by clicking the appropriate button in UiPath Studio and stop or save the recording by hitting the “Escape” key The images associated with an action are purely informative and do not affect the workflow in any way.

Use the Empty Field property of the Type Into activity if you need to clear the existing text before writing.

Some things are getting recorded, some are not: - Recorded: left clicks on buttons, checkboxes and other clickable elements, typing into editable fields - Not recorded: keyboard shortcuts, modifier keys, right clicks, mouse hover

Takeaways

When attempting to record actions that involve elements that automatically hide use “F2” to temporarily pause the recording for 3 seconds.

Some steps cannot be automatically added when recording – pause the recording and manually add them from the recording controller.

Similar to Desktop recording, Web recording generates all of its actions inside an Attach Browser container.

When using expressions with variables of the type Generic Value make sure you start with an element that is of your desired type.

Best practices UI automation works best when the Robot and the applications that it controls are on the same machine, thus allowing direct integration with the technologies behind those applications.

About UI Elements

SUGGEST EDITS

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.

Input Actions	Output Actions
• Clicks • Text typing • Keyboard shortcuts • Right-clicks • Mouse hover • Clipboard actions • Etc.	• Getting text • Finding elements and images • Clipboard actions • Etc.

UI Activities Properties

SUGGEST EDITS

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:

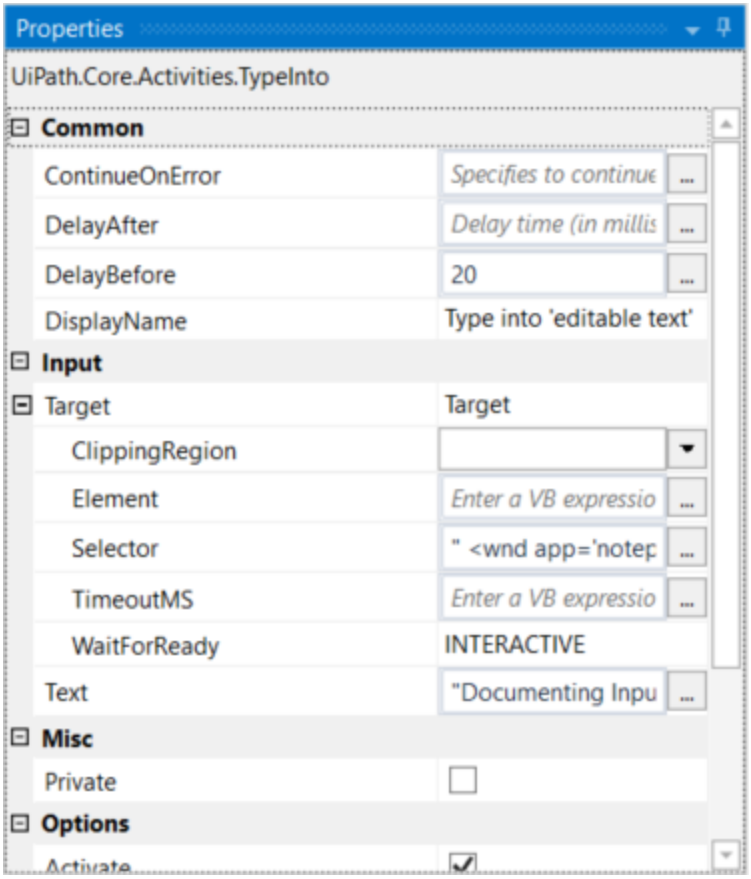
If the **ContinueOnError** field of an activity inside a **Try Catch** is set to True, no error is caught when the project is executed.

- **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.

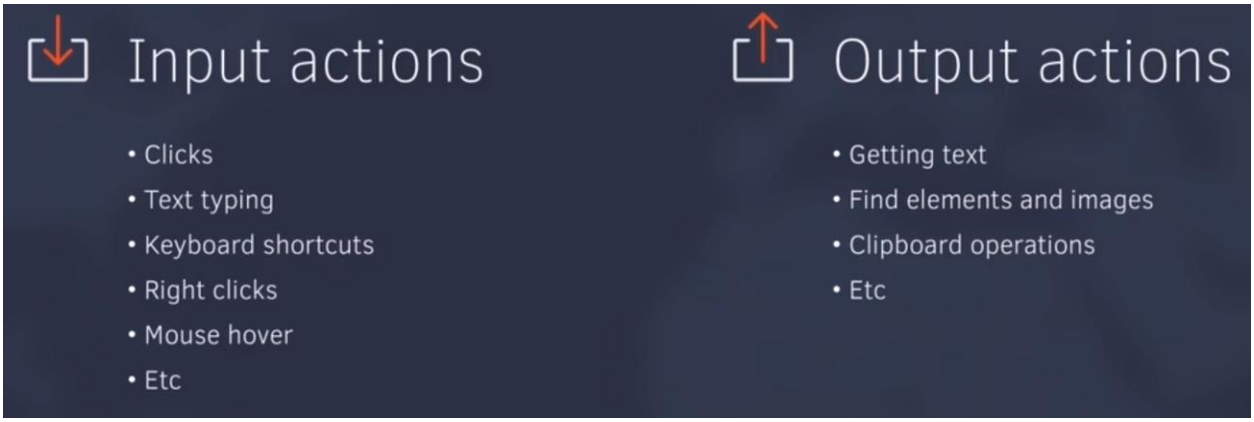


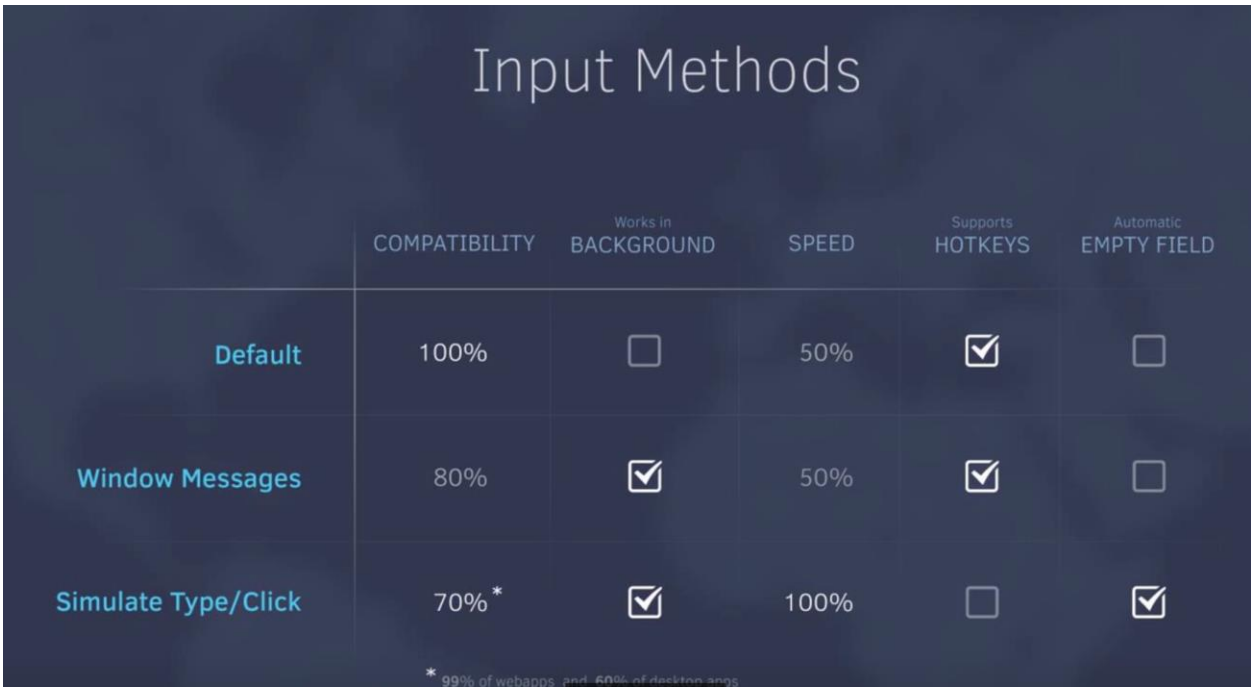
The target is composed of multiple pieces, namely the container, selector and clipping region, to ensure that you correctly identify a UI element.

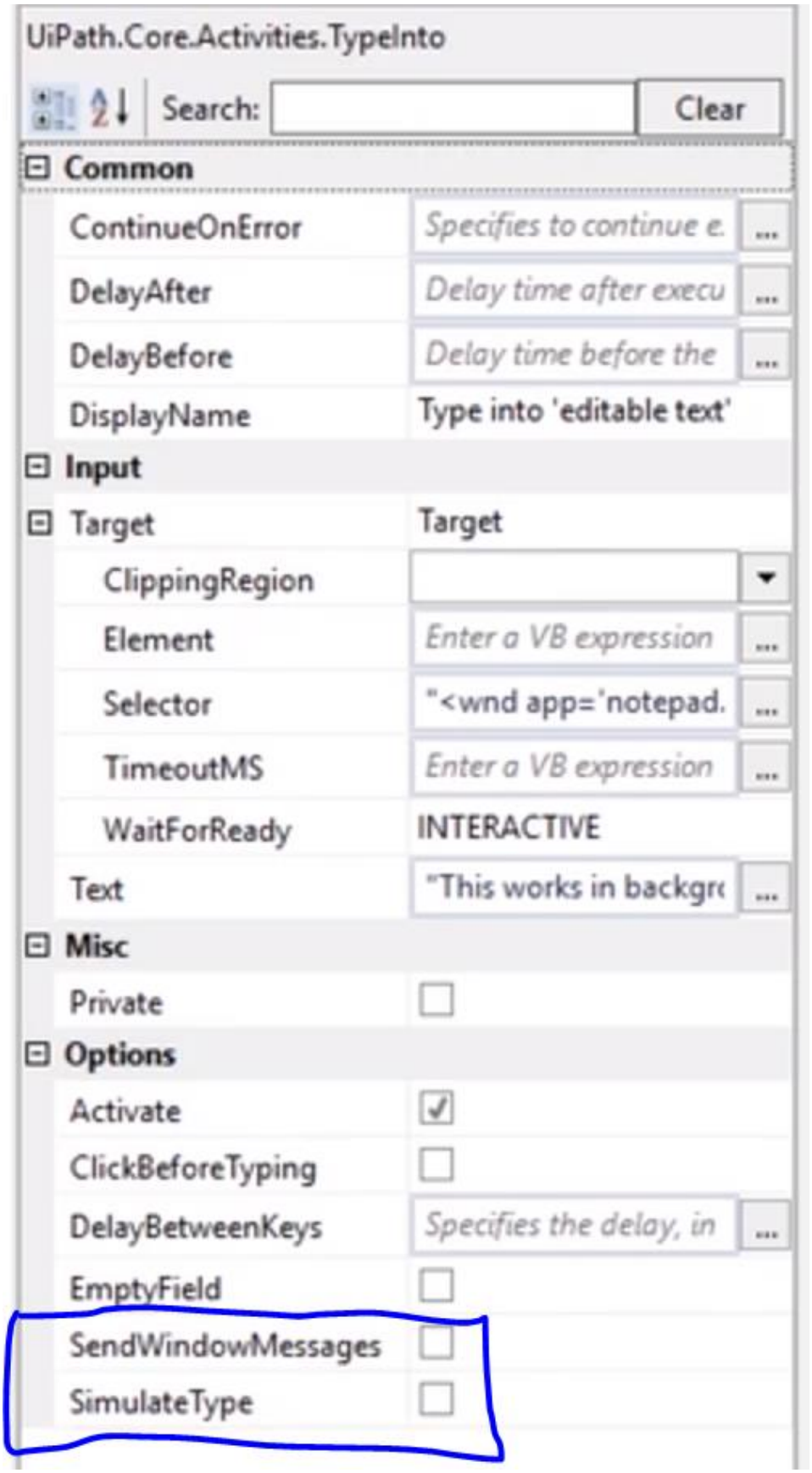
A container gives you a little more context for the button or field you want to use, so that you can tell windows apart or different areas of the same app. They are automatically generated, but you can make changes to them in the **Properties**panel.

The following are containers:

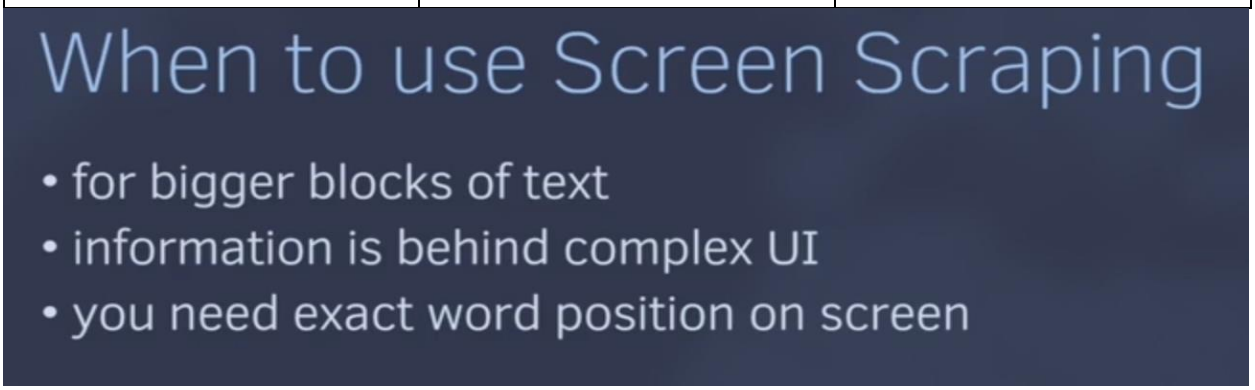
- **Attach Window**
- **Open Application**
- **Attach Browser**
- **Open Browser**
- **Get Active Window**

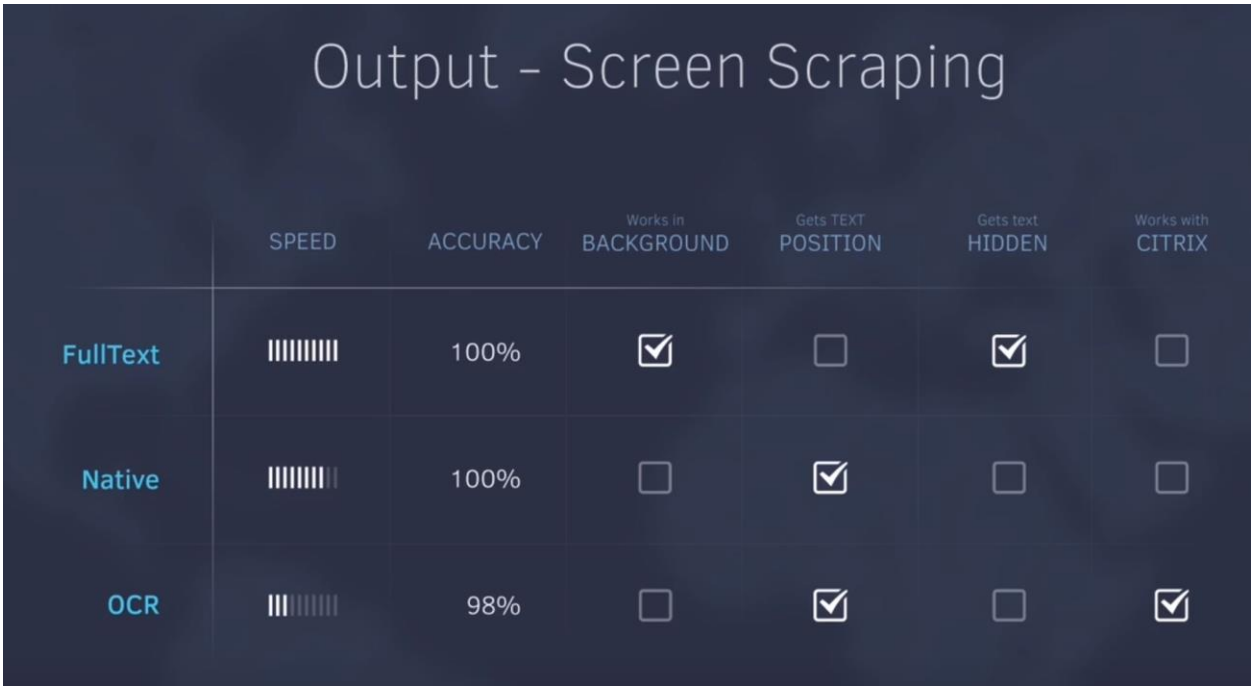



Input Methods
COMPATIBILITY
Works in BACKGROUND
SPEED
Supports HOTKEYS
Automatic EMPTY FIELD
Default
100%
50%
Window Messages
80%
50%
Simulate Type/Click
70% *
100%
* 99% of webapps, and 60% of desktop apps


UiPath.Core.Activities.TypeInto
Search:
Clear
Common
ContinueOnError
Specifies to continue e.
DelayAfter
Delay time after execu
DelayBefore
Delay time before the
DisplayName
Type into 'editable text'
Input
Target
Target
ClippingRegion
Element
Enter a VB expression
Selector
"<wnd app='notepad.
TimeoutMS
Enter a VB expression
WaitForReady
INTERACTIVE
Text
"This works in backgr
Misc
Private
Options
Activate
ClickBeforeTyping
DelayBetweenKeys
Specifies the delay, in
EmptyField
SendWindowMessages
SimulateType

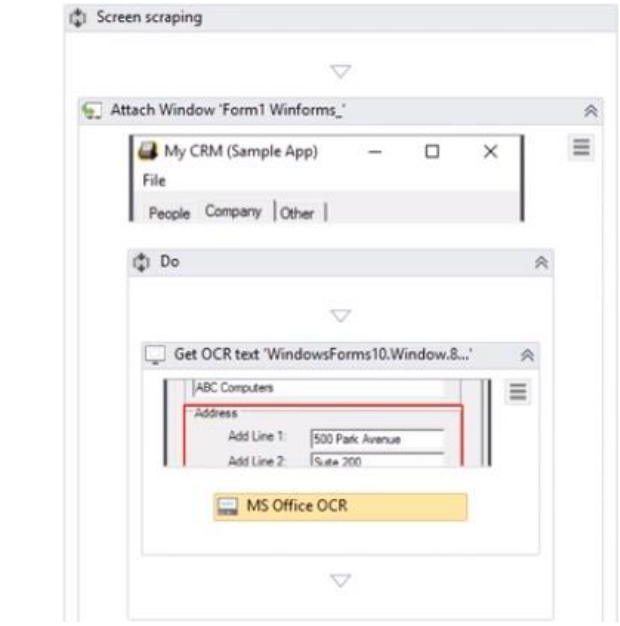
Default	Send Window Message	Simulate Type/click
gives the expected results but wasn't very Fast.	was slowest of all.	Fastest. Results were not as expected.
Does not work on background , Needs Focus to Run On same like humans		**Limitation :** Cannot handle Keyboard shortcuts , that can be solved by using get text activity before the typeInto and appending the new text to it.

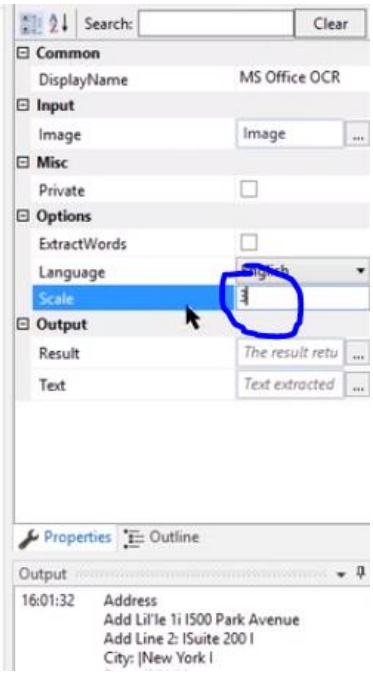




FullText	Native	OCR
Default Mode	Advantage of having to extract word information, like screen coordinates of each word and character.	Looks indicated area as an image. Suitable where accuracy is not paramount.
Fast Accurate and runs in background.		MS Office OCR GOOGLE OCR It has an option of language for better screpping.
Retrieves the • visible Texts, • editable boxes and • hidden information too. We can ignore by clicking Ignore Hidden	Only Retrieves the editable texts.	MS OCR • is suitable for larger images like scanned documents, receipts and so on. Google OCR • works best with smaller low resolutions images like interface elements and such. • It has an invert option for white over black ground. • An Scale option available , an option to unsample images for better results.

If by any chance , the output generated by the OCR is not working properly. We can drag and drop the Microsoft Office OCR inside the get OCR Text window and update the scale to 3 because it works better with large images as shown below.





There are 4 Main UI output Methods:

Environment.NewLine : To Add a New Line.

Data Scrapping:

-Used to extract structured Data

- Web Scrapping.

Properties : MaxNumberofresults : Default -100

If data is in tabular Form on screen. Web Scrapping has the below option to extract the complete Data as a Table.

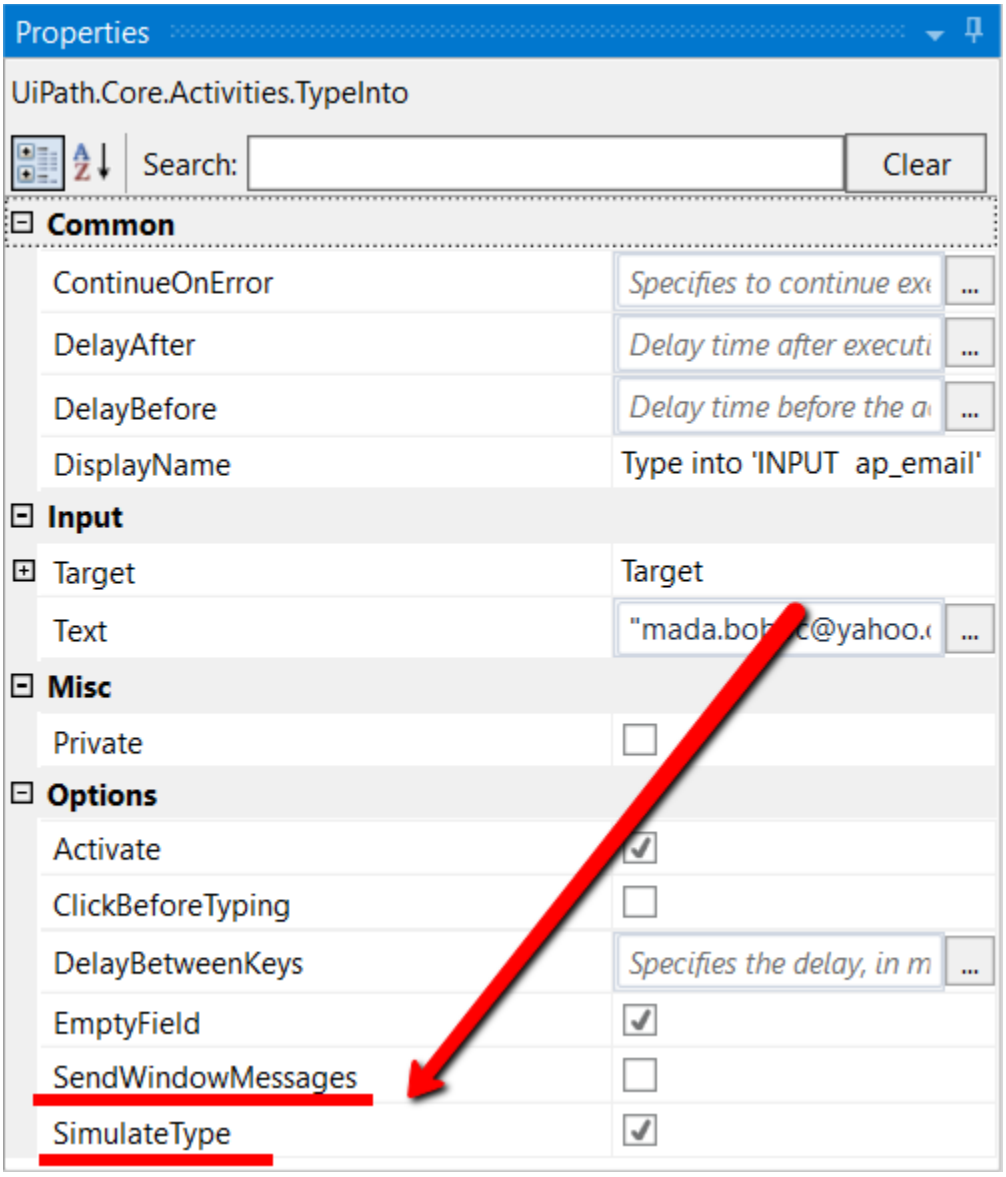
Input Methods

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 **SendWindowMessages** method and then the **Default** one, as it is the slowest.

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

The input method can be changed at any point from the **Properties** panel of the selected activity. If the **Simulate Type/Click** 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.

Output or Screen Scraping Methods

SUGGEST EDITS

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.

Capability Method	Speed	Accuracy	Background Execution	Extract Text Position	Extract Hidden Text	Support for Citrix
FullText	10/10	100%	yes	no	yes	no
Native	8/10	100%	no	yes	no	no
OCR	3/10	98%	no	yes	no	yes

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

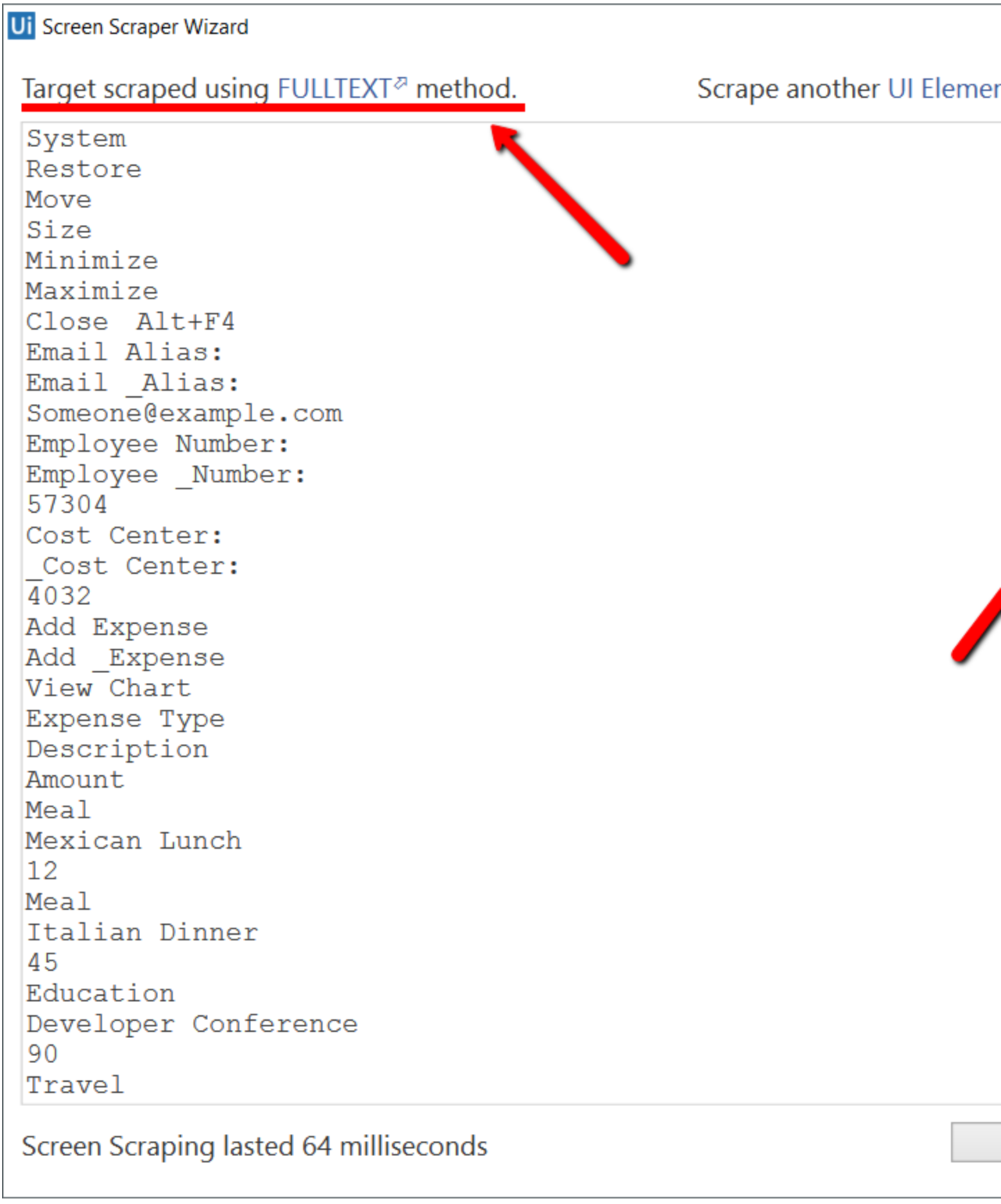
Both these methods work only with desktop applications, but the **Native** method only works with apps that are built to render text with the Graphics Device Interface (GDI).

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.

Capability Method	Multiple Languages Support	Preferred Area Size	Support for Color Inversion	Set Expected Text Format	Filter Allowed Characters	Best with Microsoft Fonts
Google Tesseract	Can be added	Small	yes	yes	yes	no
MIcrosoft MODI	Supported by default	Large	no	no	no	yes

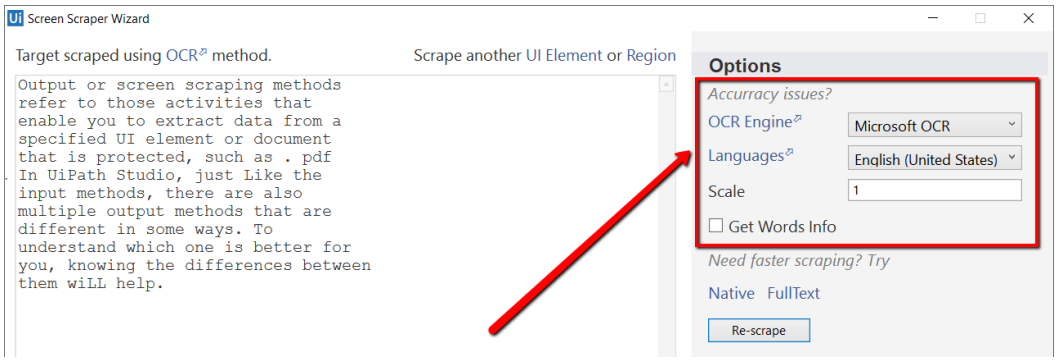
To start extracting text from various sources, click the **Screen Scraping** button, in the **Wizards** group, on the **Design**ribbon tab.

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.


Screen Scraper Wizard
Target scraped using FULLTEXT method.
Scrape another UI Elemer
System
Restore
Move
Size
Minimize
Maximize
Close Alt+F4
Email Alias:
Email _Alias:
Someone@example.com
Employee Number:
Employee _Number:
57304
Cost Center:
_Cost Center:
4032
Add Expense
Add _Expense
View Chart
Expense Type
Description
Amount
Meal
Mexican Lunch
12
Meal
Italian Dinner
45
Education
Developer Conference
90
Travel
Screen Scraping lasted 64 milliseconds

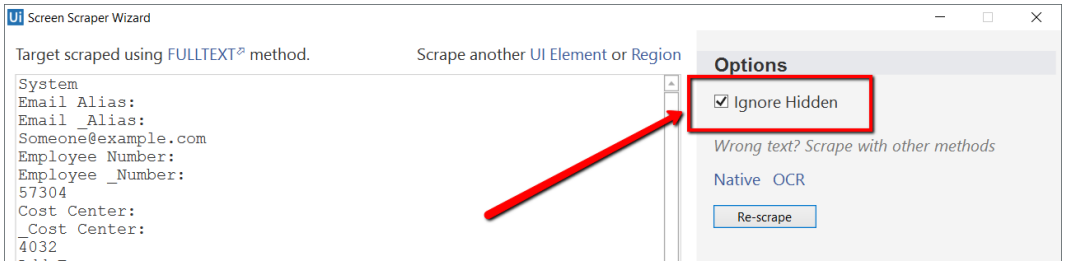
To change the method of screen scraping, select another one from the **Options** panel and then click **Re-scrape**.

When you are satisfied with the scraping results, click **Continue** or **Copy**. The latter option copies the extracted text to the Clipboard, while the former saves your information to the **Designer** panel. Just like desktop recording, screen scraping generates a container (with the selector of the top level window) which contains activities, and partial selectors for each activity.



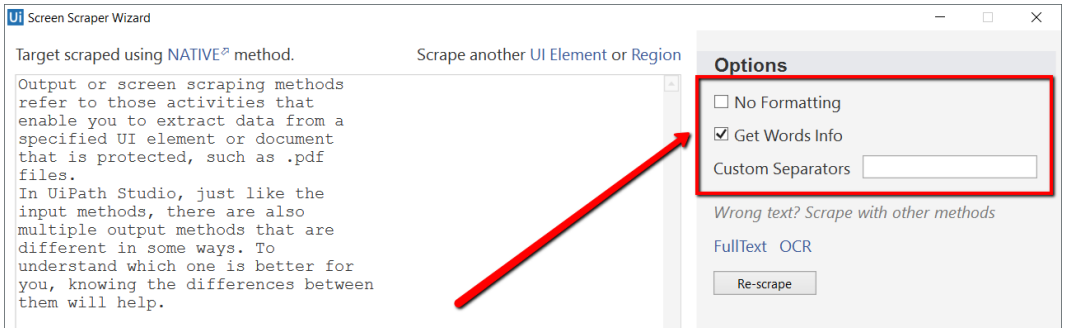
Each type of screen scraping comes with different features in the **Screen Scraper Wizard**, in the **Options** panel:

1. **FullText**



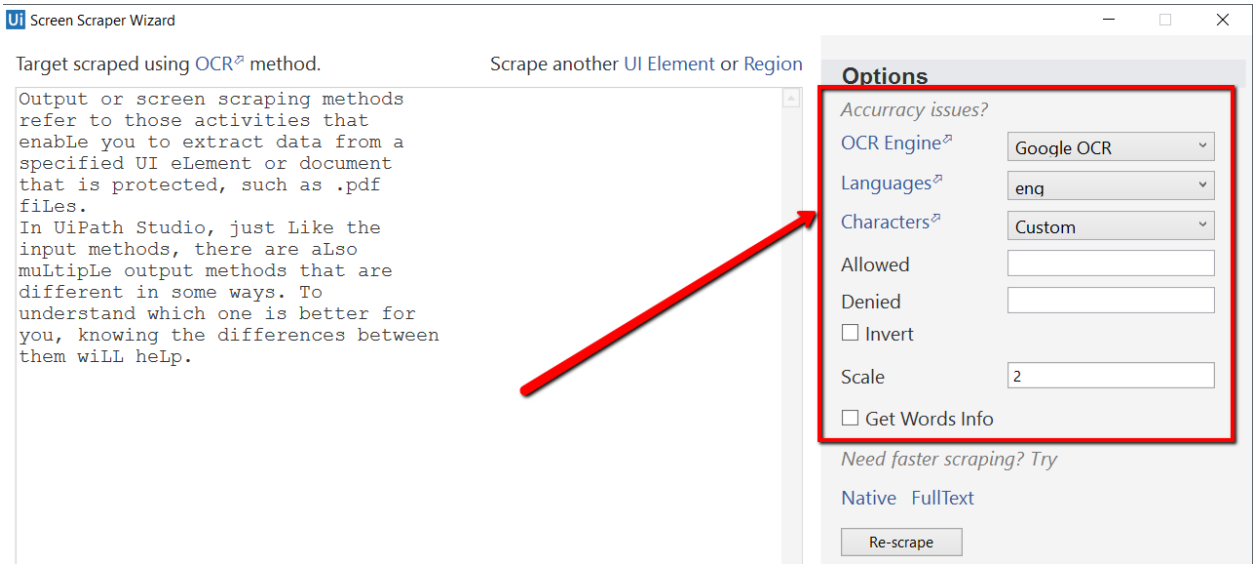
- **Ignore Hidden** – when this check box is selected, the hidden text from the selected UI element is not copied.

2. **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.

3. **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

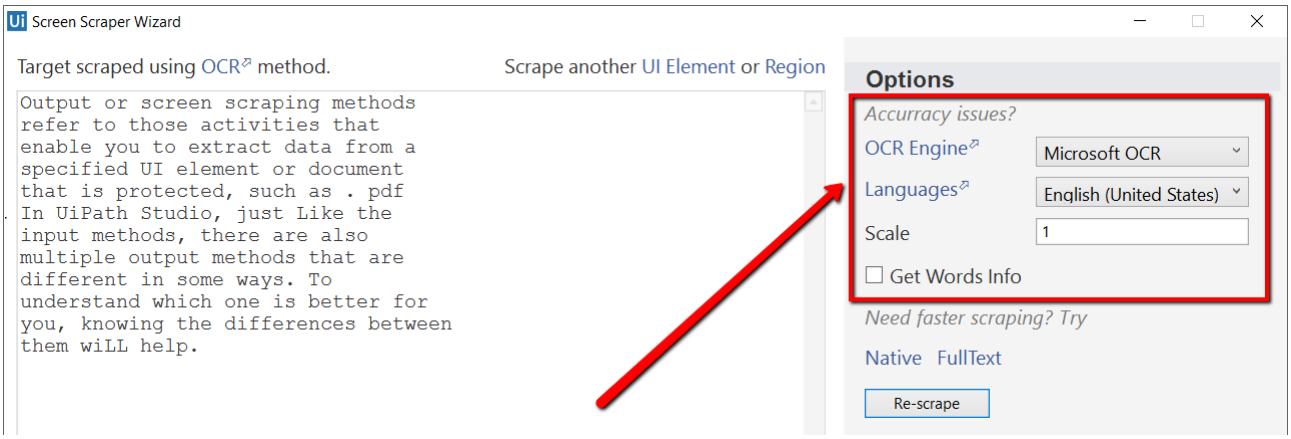
you select **Custom**, two additional fields, **Allowed** and **Denied**, are displayed that enable you to create custom rules on which types of characters to scrape and which to avoid.

- **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, GitHub) that do not work for certain non-English languages. Running an automation 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 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.

4. **Microsoft OCR**



- **Languages** – enables you to change the language of the scraped text. By default, English is selected.

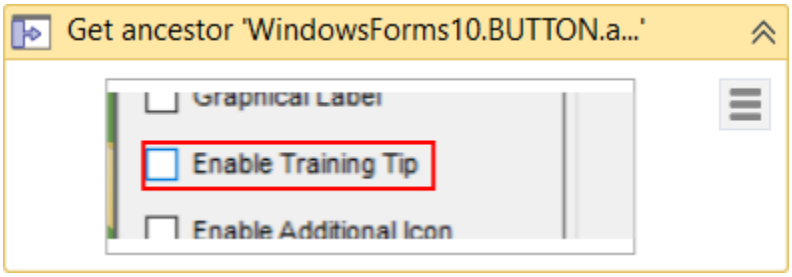
- **Scale** – the scaling factor of the selected UI element or image. The higher the number is, the more you enlarge the image. This can provide a better OCR read and it is recommended with small images.
- **Get Words Info** - gets the on-screen position of each scraped word.

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.

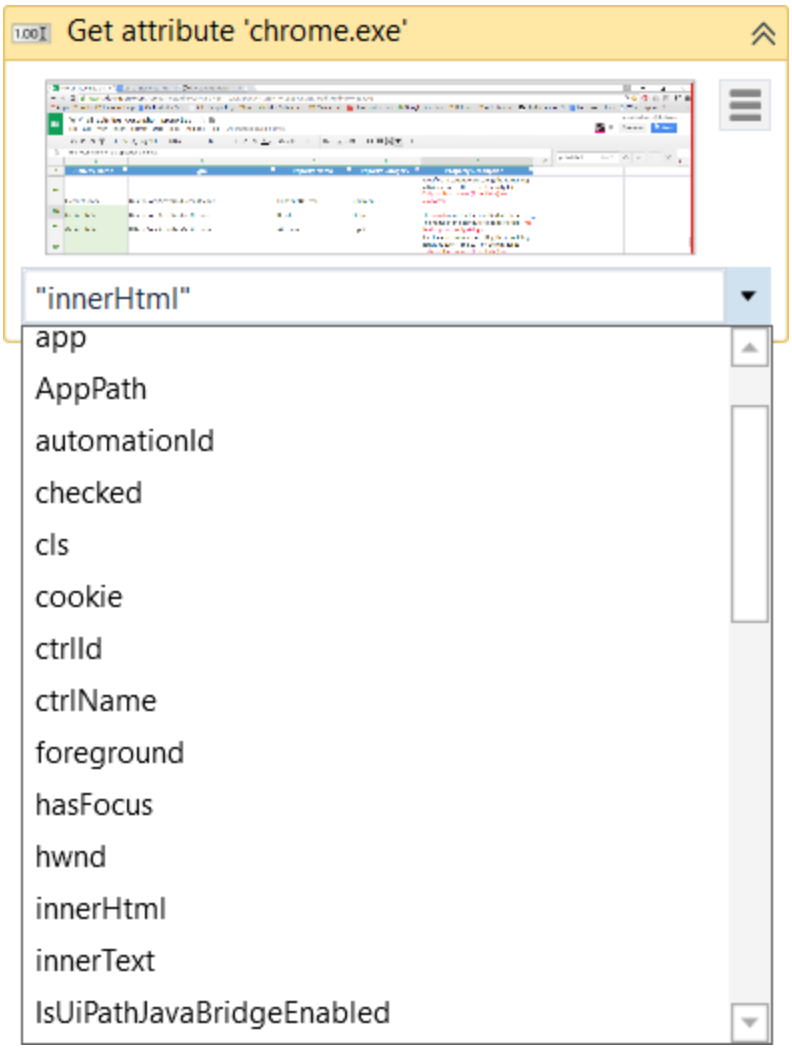
This type of information can be extracted through dedicated activities that are found in the **Activities** panel, under **UI Automation > Element > Find** and **UI Automation > Element > Attribute**.

These activities are:

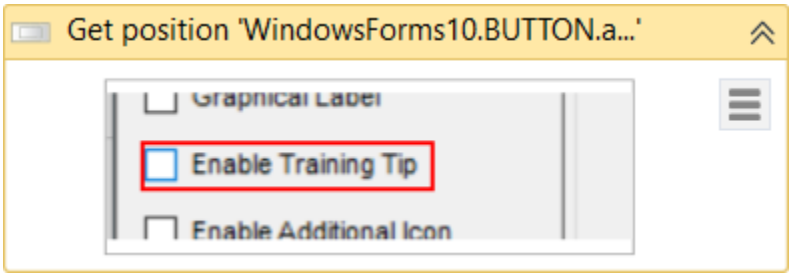
- **Get Ancestor** – enables you to retrieve an ancestor from a specified UI element. You can indicate at which level of the UI hierarchy to find the ancestor, and store the results in a UiElement variable.



- **Get Attribute** – 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** – retrieves the bounding rectangle of the specified UiElement, and supports only Rectangle variables.

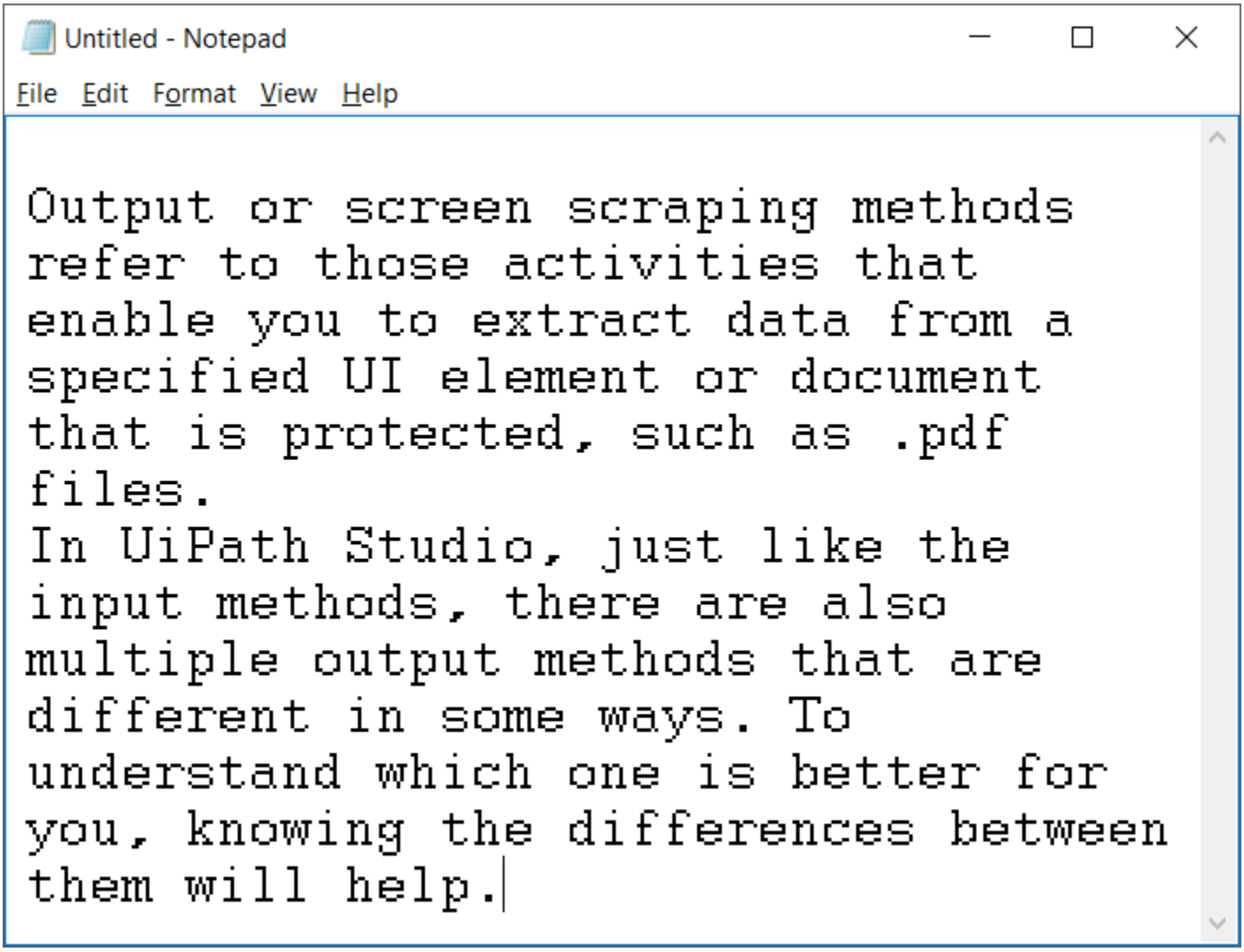


UiPath Studio also features **Relative Scraping**, a scraping method that identifies the location of the text to be retrieved relative to an anchor. You can find more about it here.

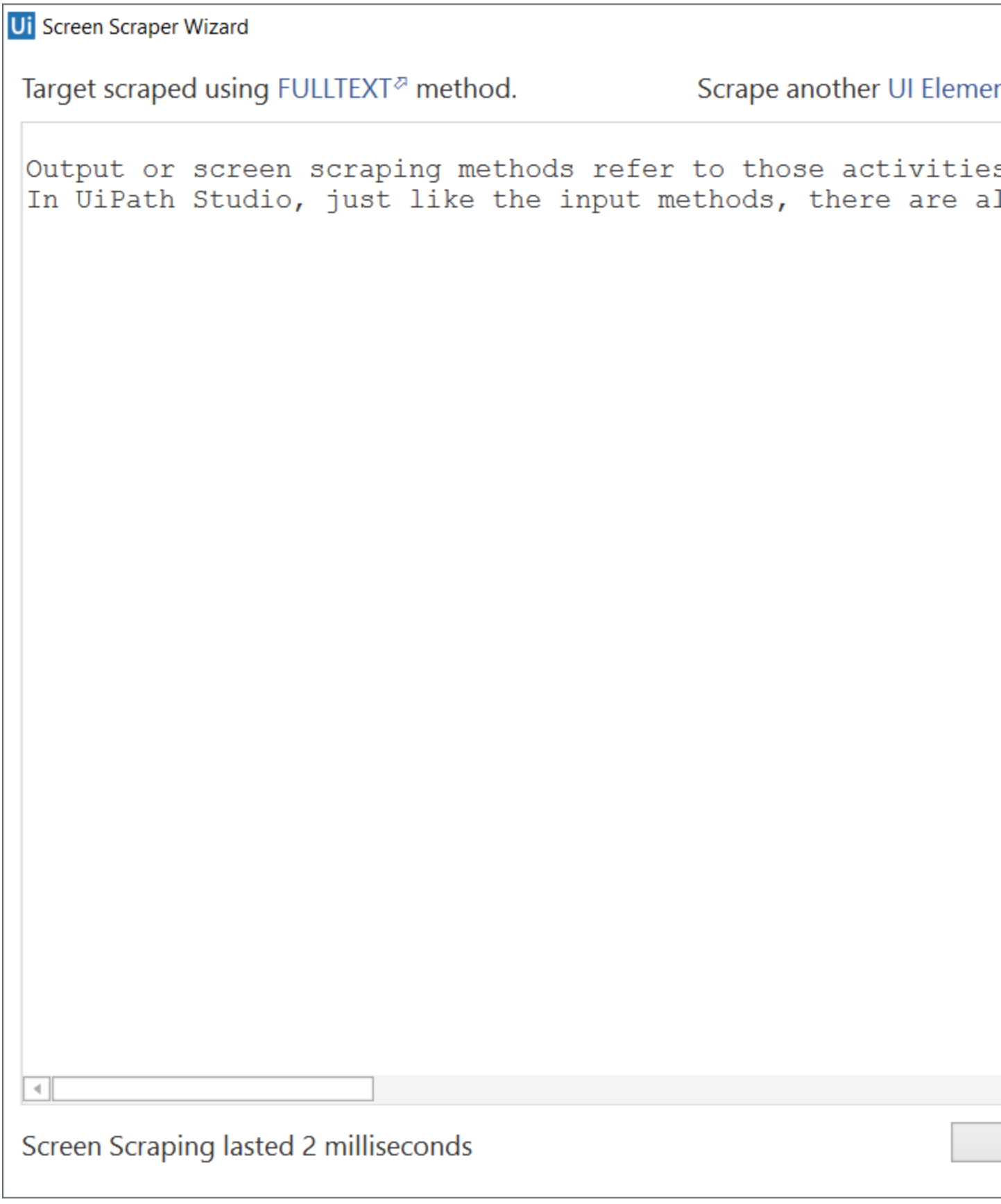
Examples of Using Output or Screen Scraping Methods

SUGGEST EDITS

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.

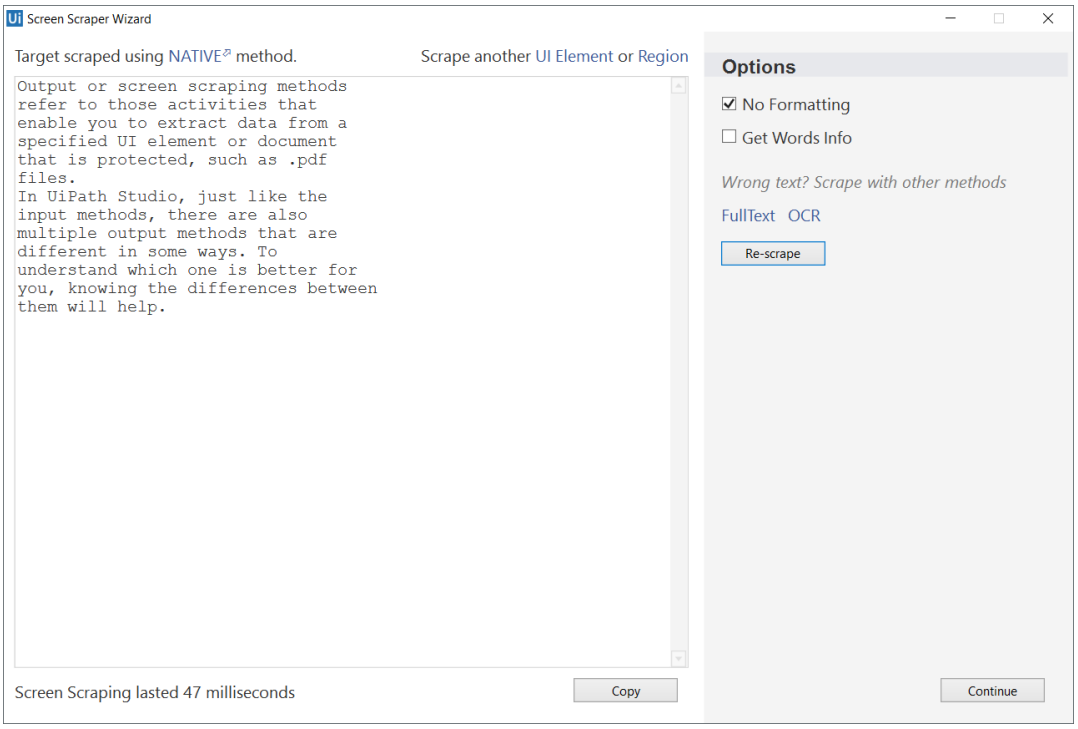


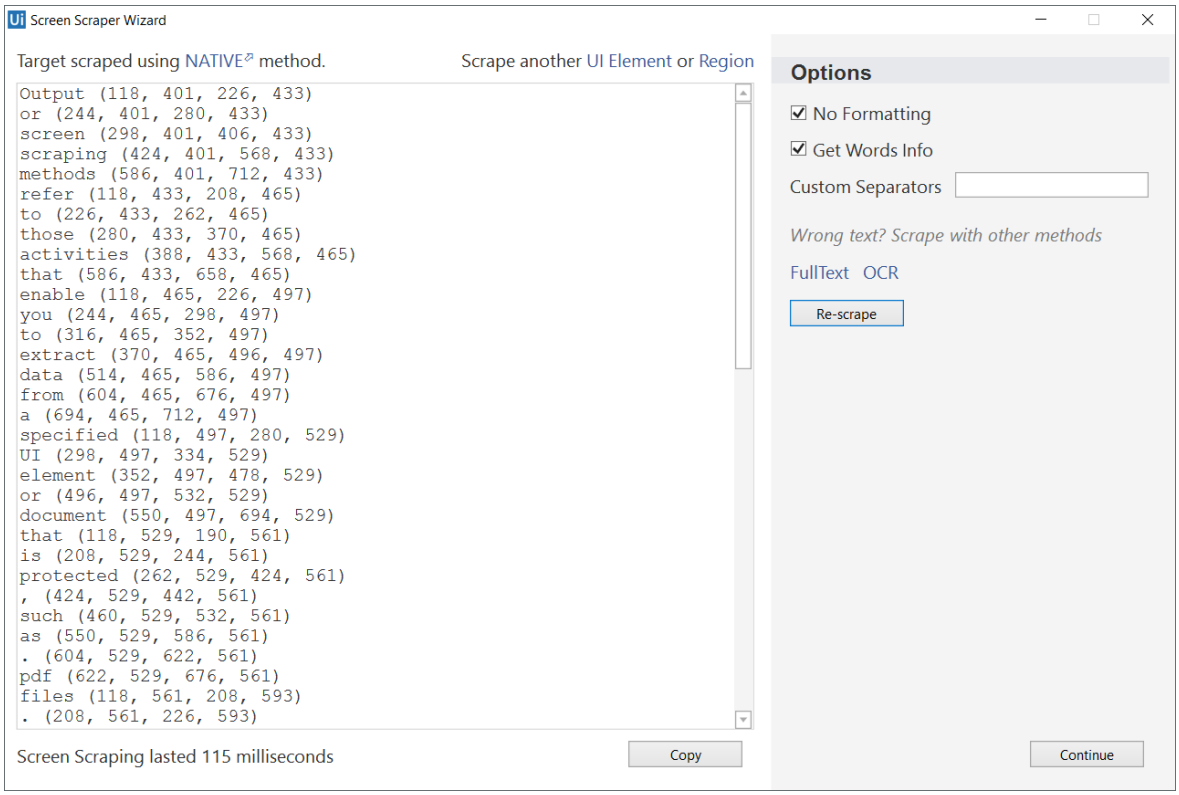
The FullText method


Screen Scraper Wizard
Target scraped using FULLTEXT method.
Scrape another UI Elemer
Output or screen scraping methods refer to those activities
In UiPath Studio, just like the input methods, there are al
Screen Scraping lasted 2 milliseconds

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.

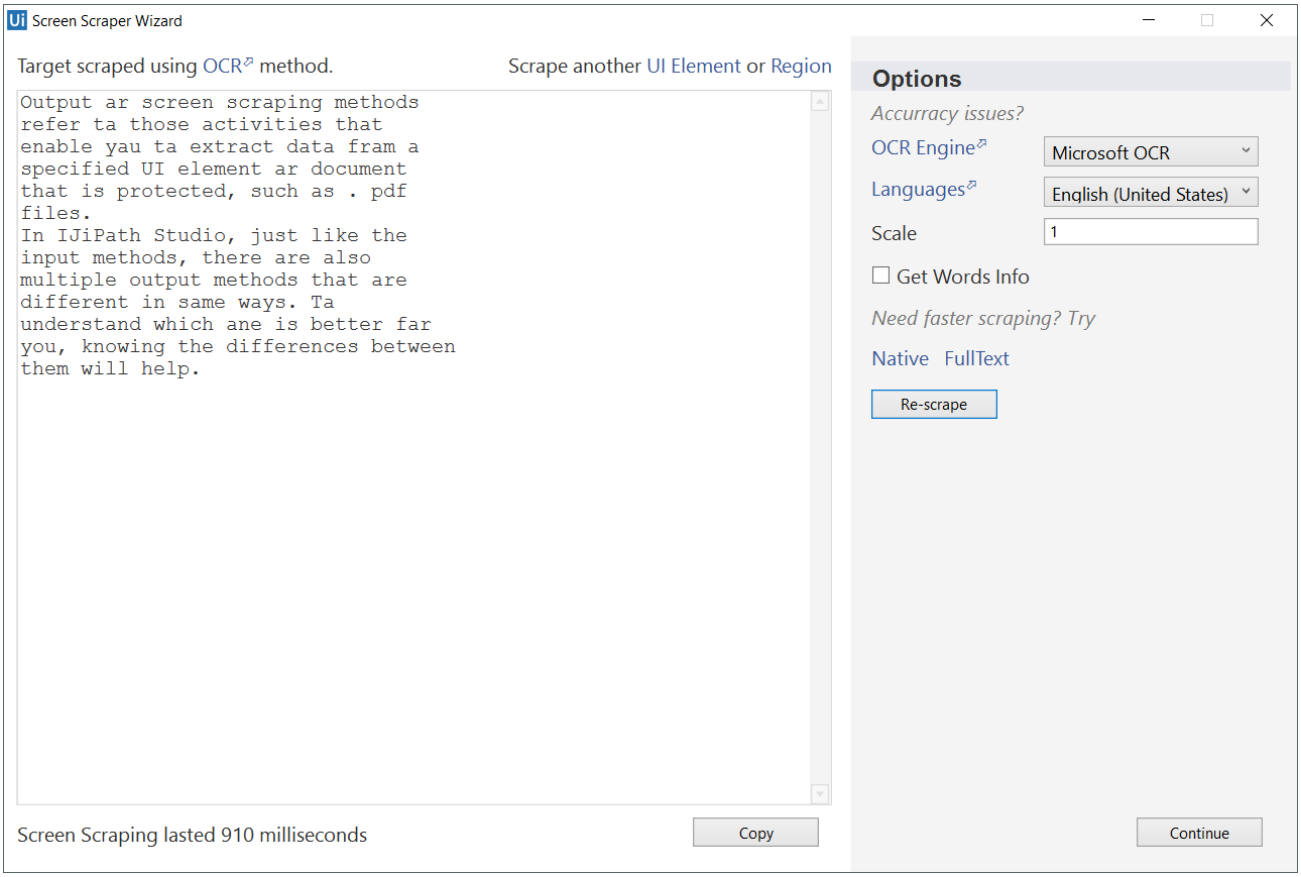
The Native method





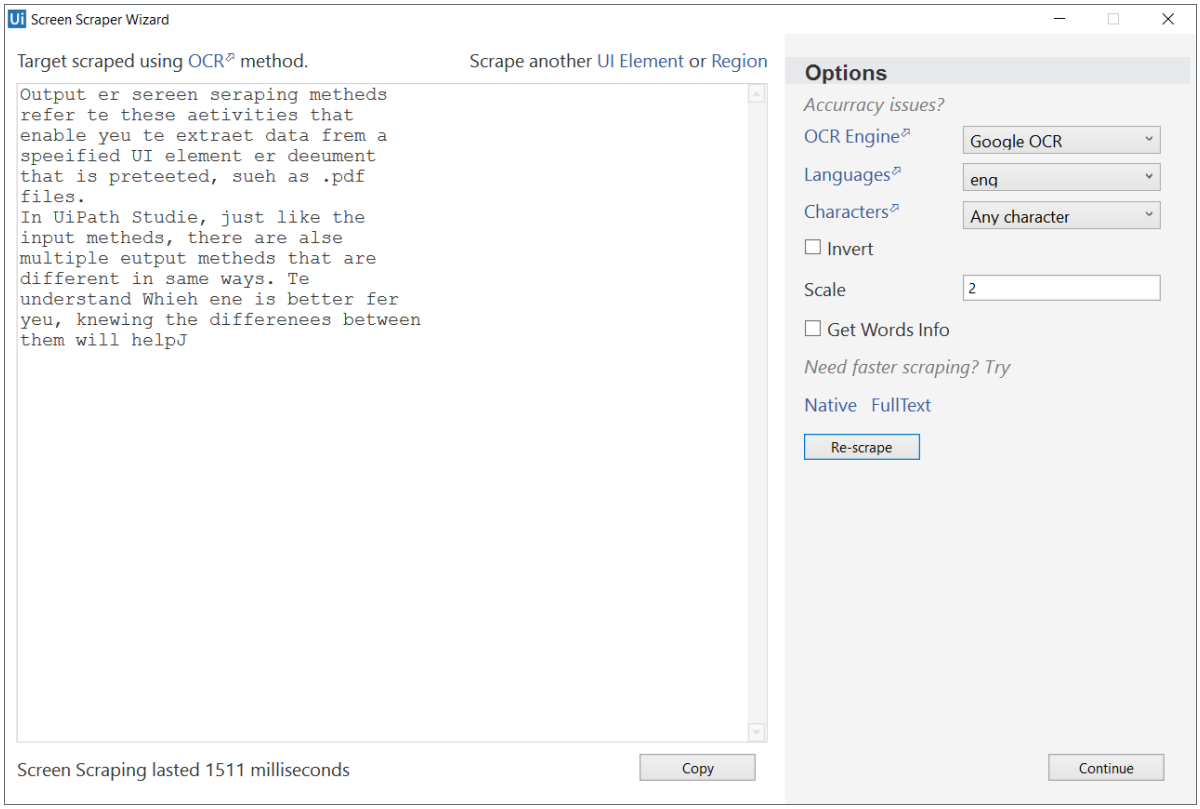
As you can see in the first screenshot, you can extract the text with its position on the screen, as well as retrieve the exact position of each word (second screenshot).

The Microsoft OCR method



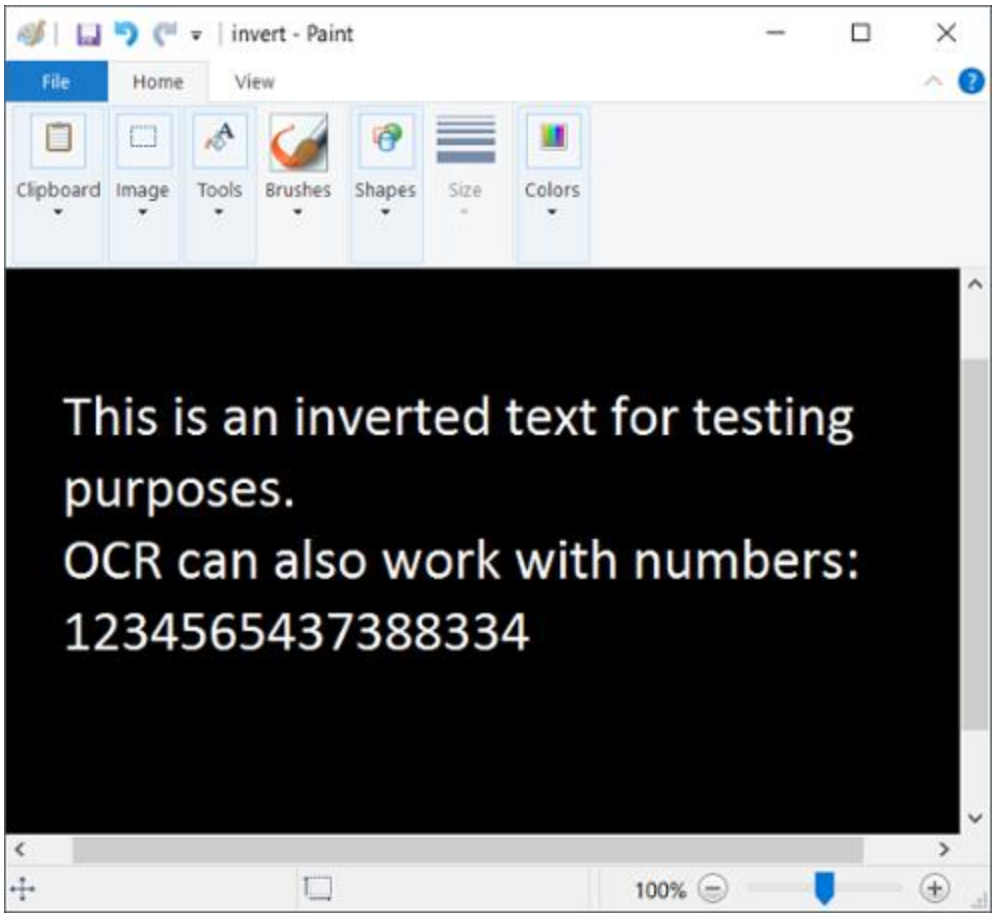
As you can see, the accuracy of this output method is not 100%, but it still manages to keep the position of the text. Getting the exact on-screen position, in pixels, is also available yet as you can see, it is not the fastest of the output methods.

The Google OCR method

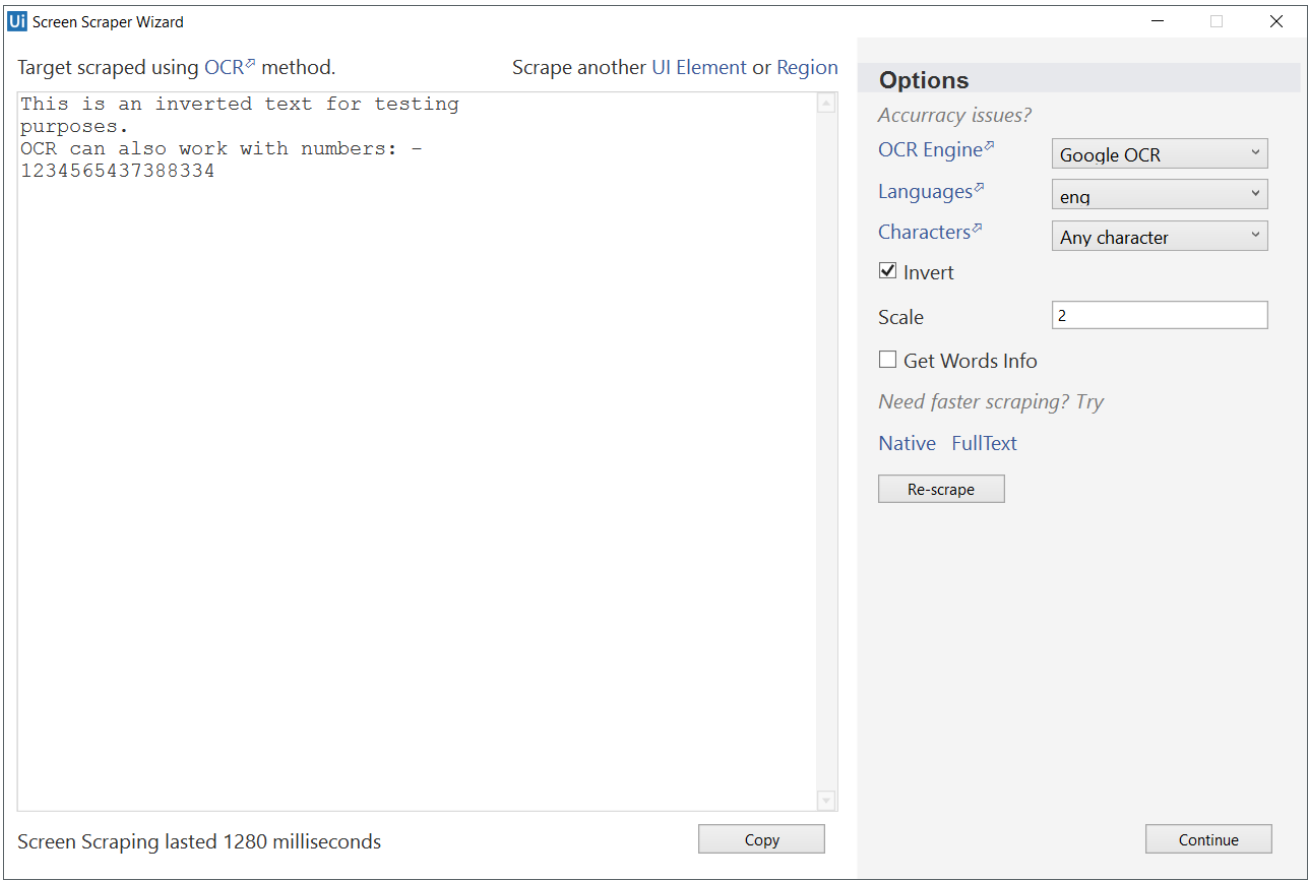


As with Microsoft's Modi, the Google OCR method is not 100% accurate and takes longer when compared with the others. However, it retrieves the position within the window of the text.

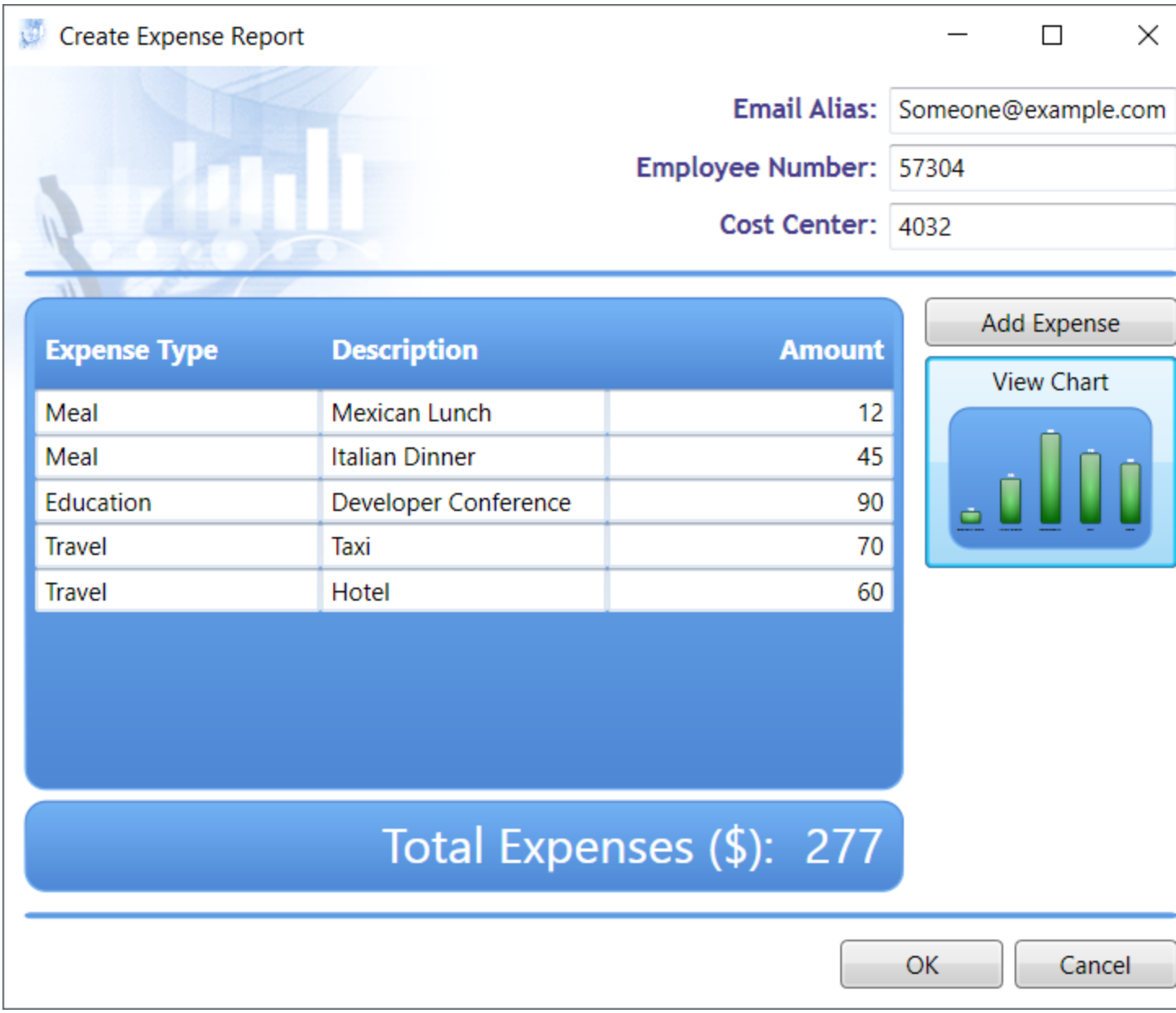
Now, add some white text over a black page in Paint, for example, and try to scrape it.



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

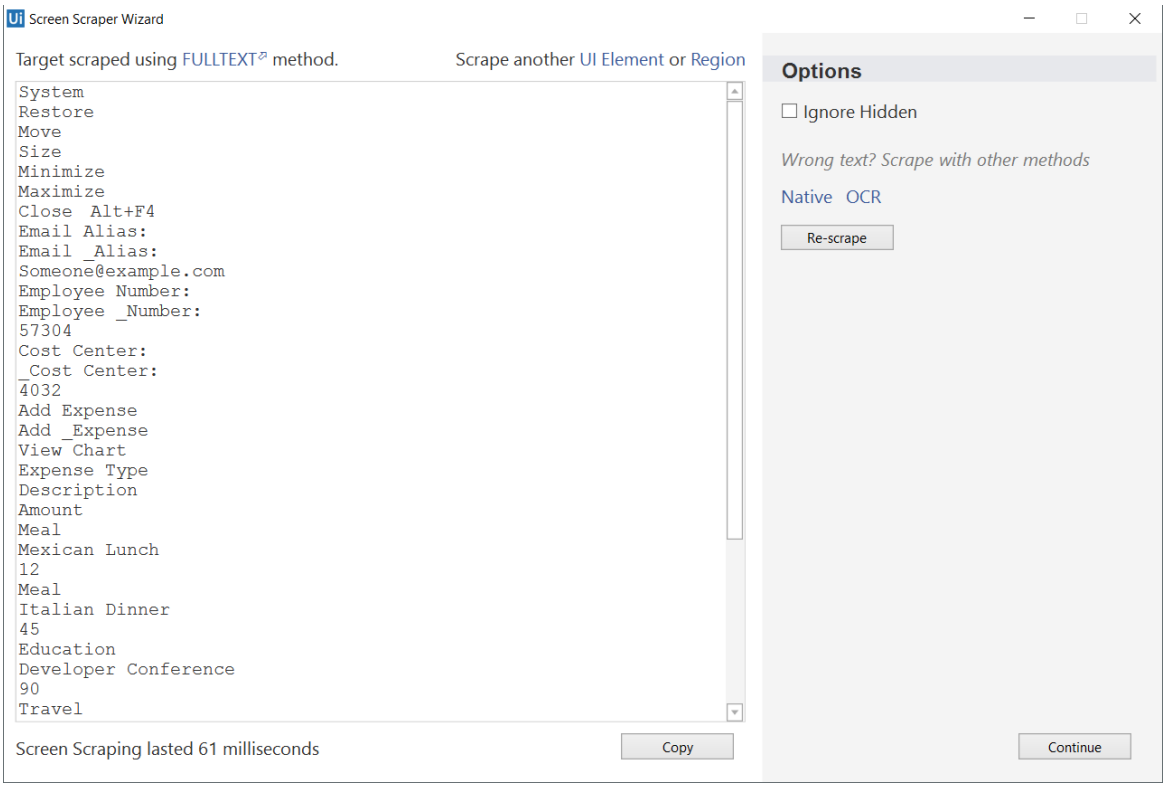


Now let's try scraping an application and see the results. We use a dummy expense app, which you can download here.



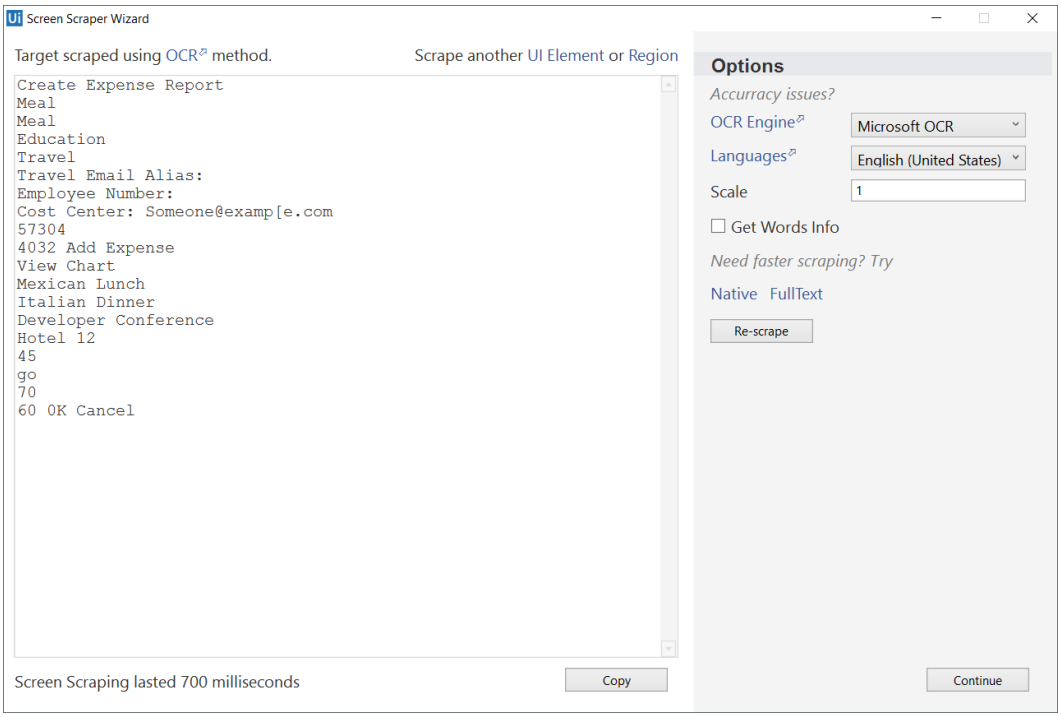
If we scrape this entire window, we receive the following results:

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

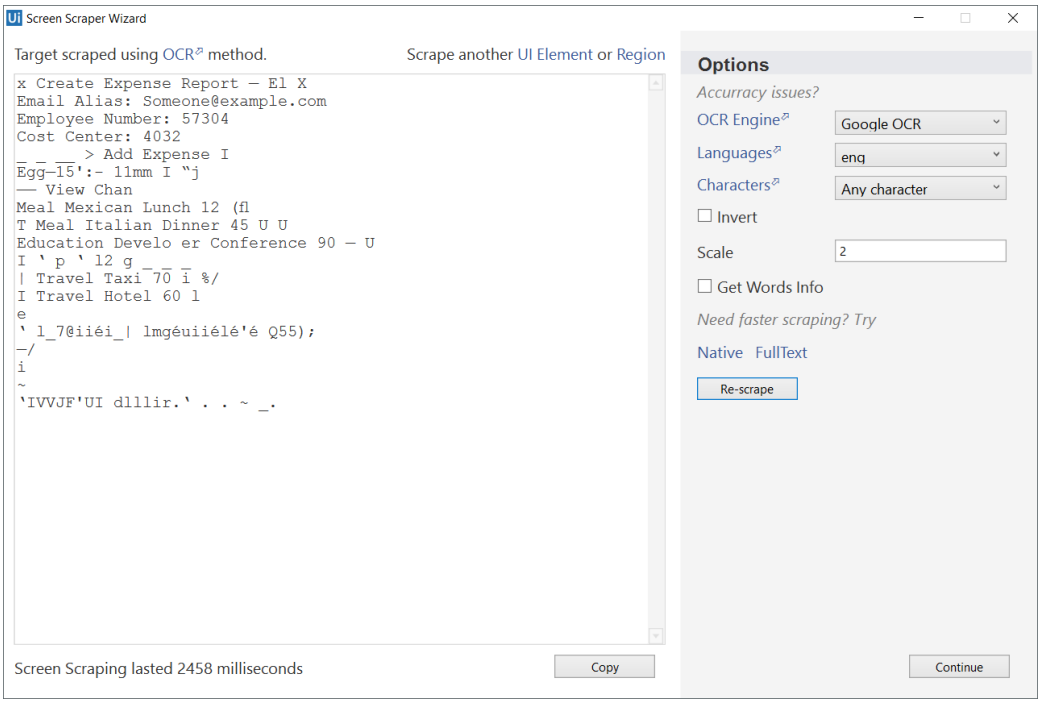


2. **Native** does not work on this UI as it does not make use of GDI to render text.

3. **Microsoft OCR** works pretty well, although accuracy is still not 100%.



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



Relative Scraping

SUGGEST EDITS

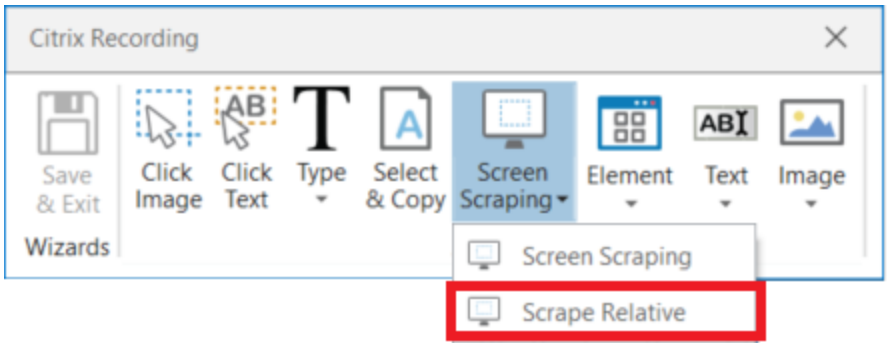
Relative Scraping is a technique that enables you to retrieve text from UI elements by using OCR technology. In situations where selectors cannot be found, the target UI objects are identified by using image recognition activities to look for adjacent labels or other elements.

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.

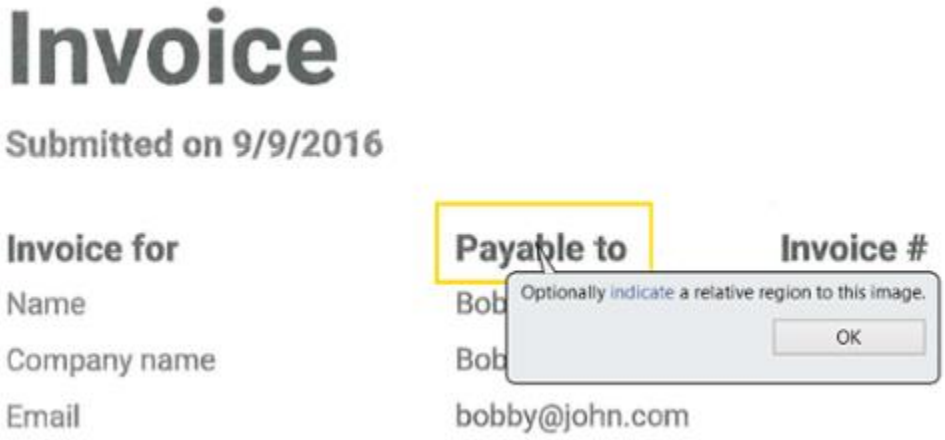
To use the **Scrape Relative** functionality, do the following:

1. Start the **Citrix Recording Wizard**.

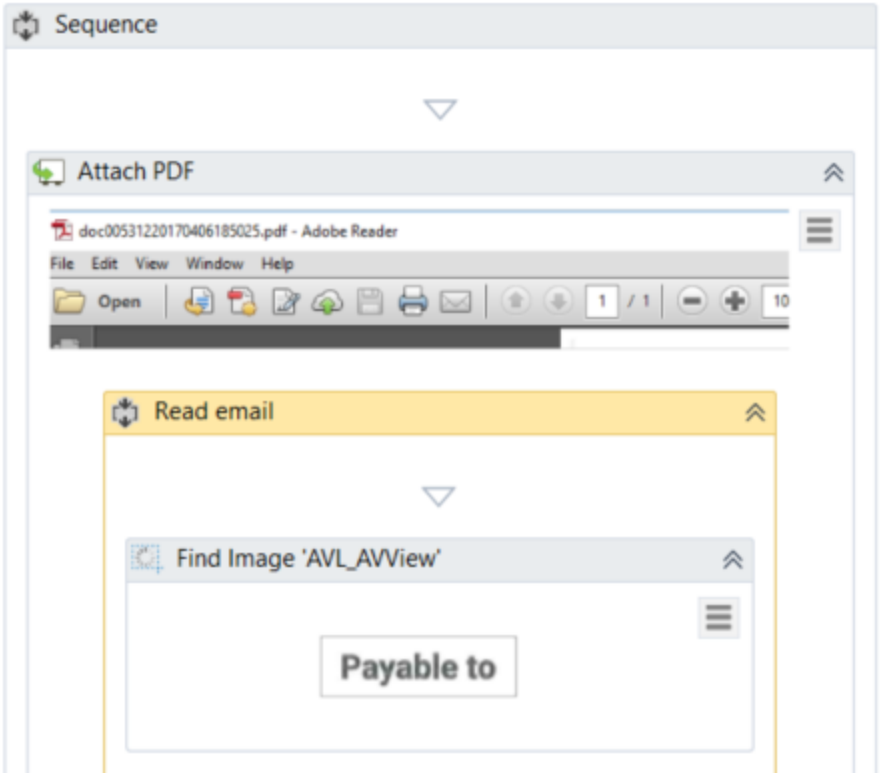
2. Click **Screen Scraping > Scrape Relative**.



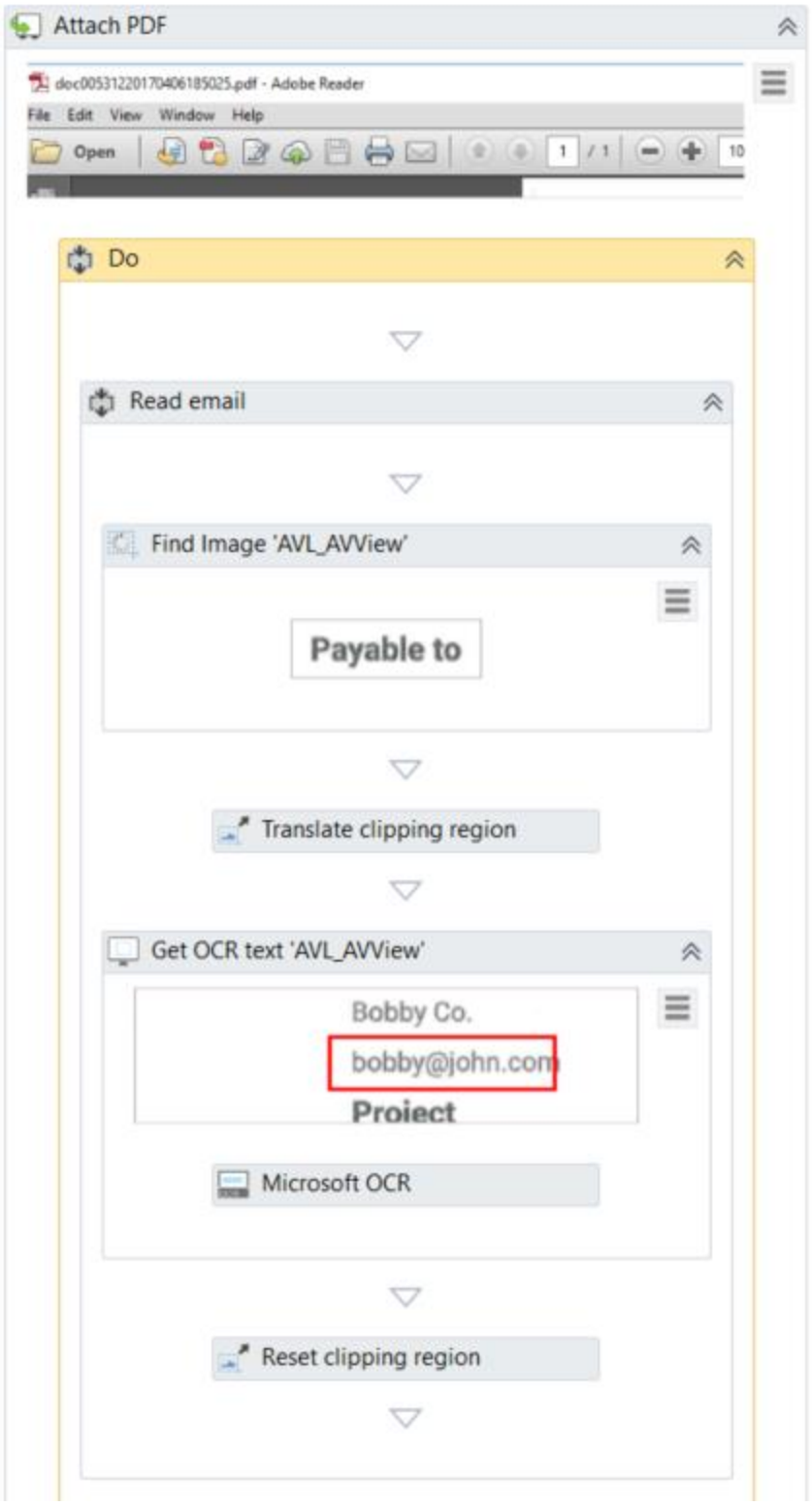
3. Select an **anchor**, which is the relative element used to identify the location of the target, such as the label of a text field.



An **Attach Window** container is generated that sets focus to the app window and contains a **Find Image** activity that locates the position of the anchor on the screen.



4. 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** 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.


Attach PDF
doc005312201704061850 25.pdf - Adobe Reader
File Edit View Window Help
Open
1 / 1
Do
Read email
Find Image 'AVL_AVView'
Payable to
Translate clipping region
Get OCR text 'AVL_AVView'
Bobby Co.
bobby@john.com
Project
Microsoft OCR
Reset clipping region

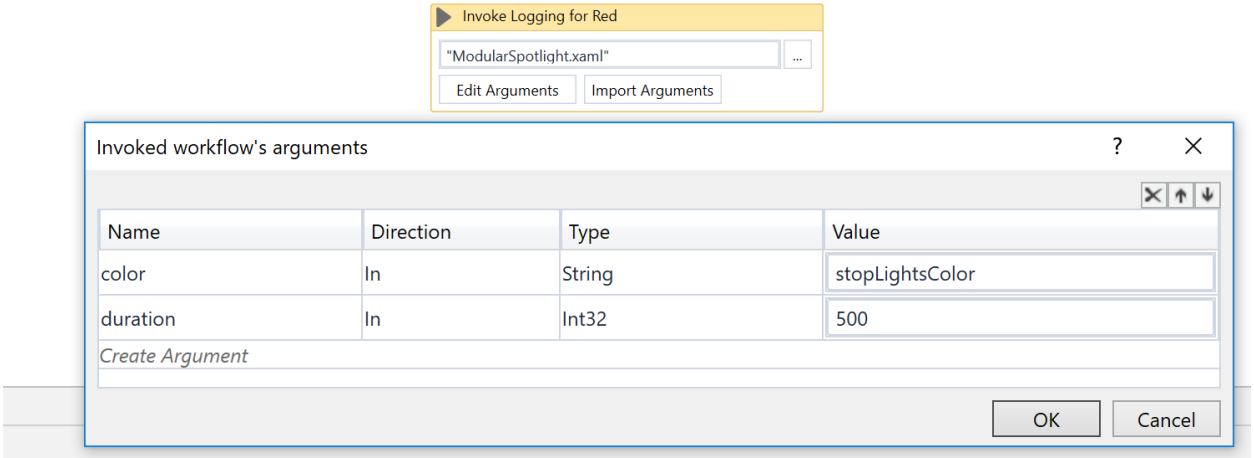
If you used **Write Line** earlier, these are very similar. However, **Log Message** activities have more versatility as they can write information lines as well as troubleshooting lines.

Arguments and Variables:

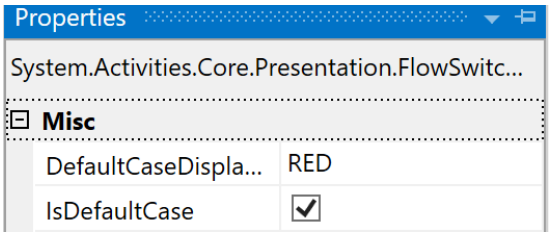
Name	Direction	Argument type	Default value
color	In	String	"red"
duration	In	Int32	50
Create Argument			

Create 2 arguments in a Sequence and the .xaml File.

Now Take invoke WorkFlow activity and Import Arguments as shown below.



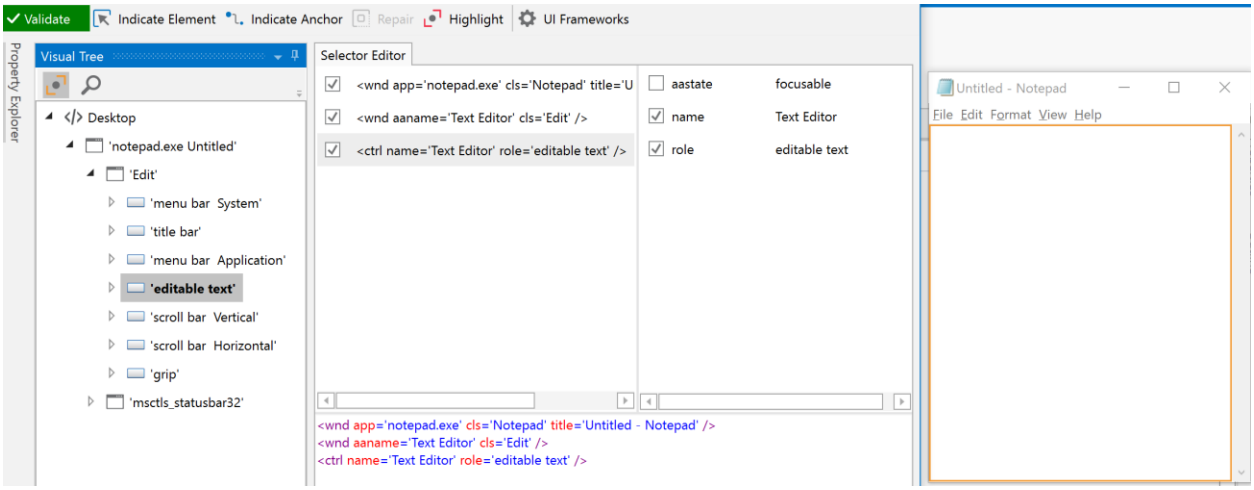
In Switch Case, we have the Default case as shown below.

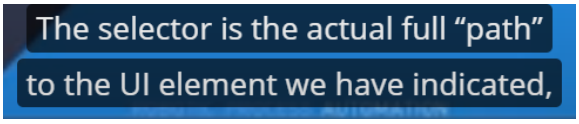


SELECTORS

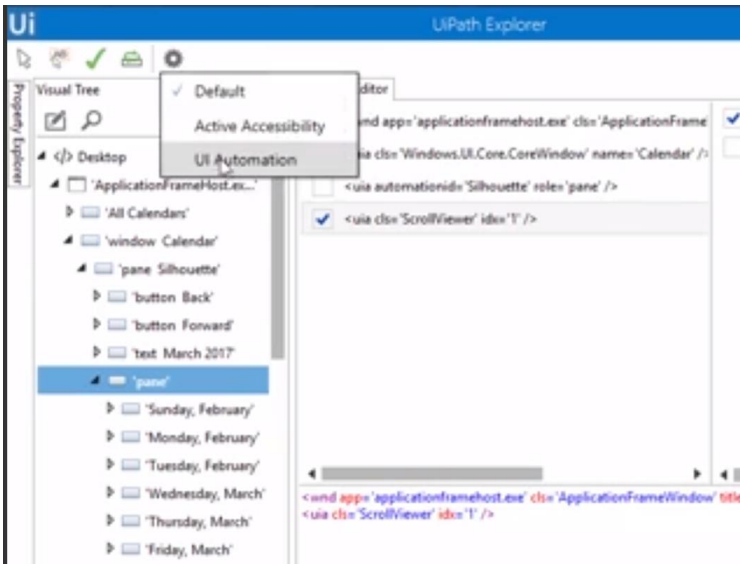
Every application works as a container and has UI elements.Same applies to browser.

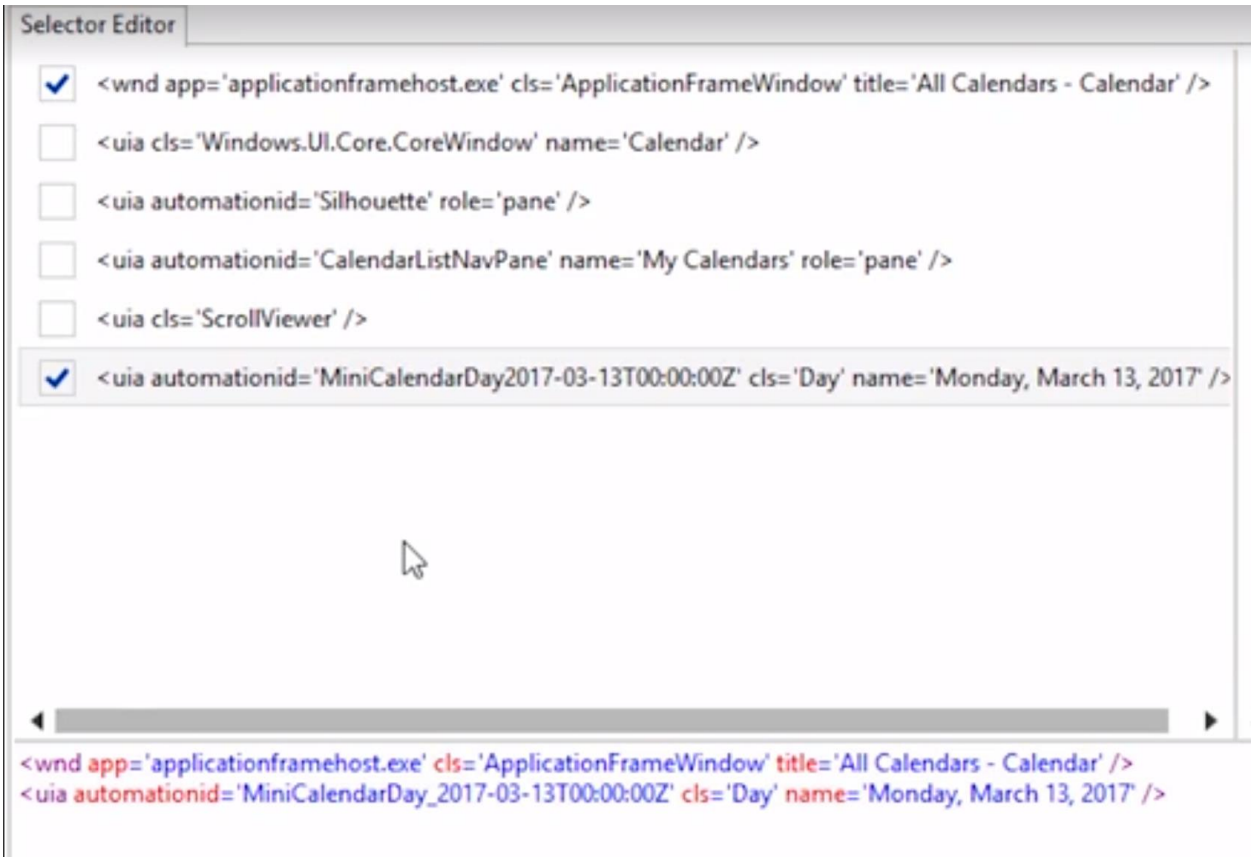
Use UI Explorer to see selectors on NotePad. Also, we can use highlight option to see the selected Area.



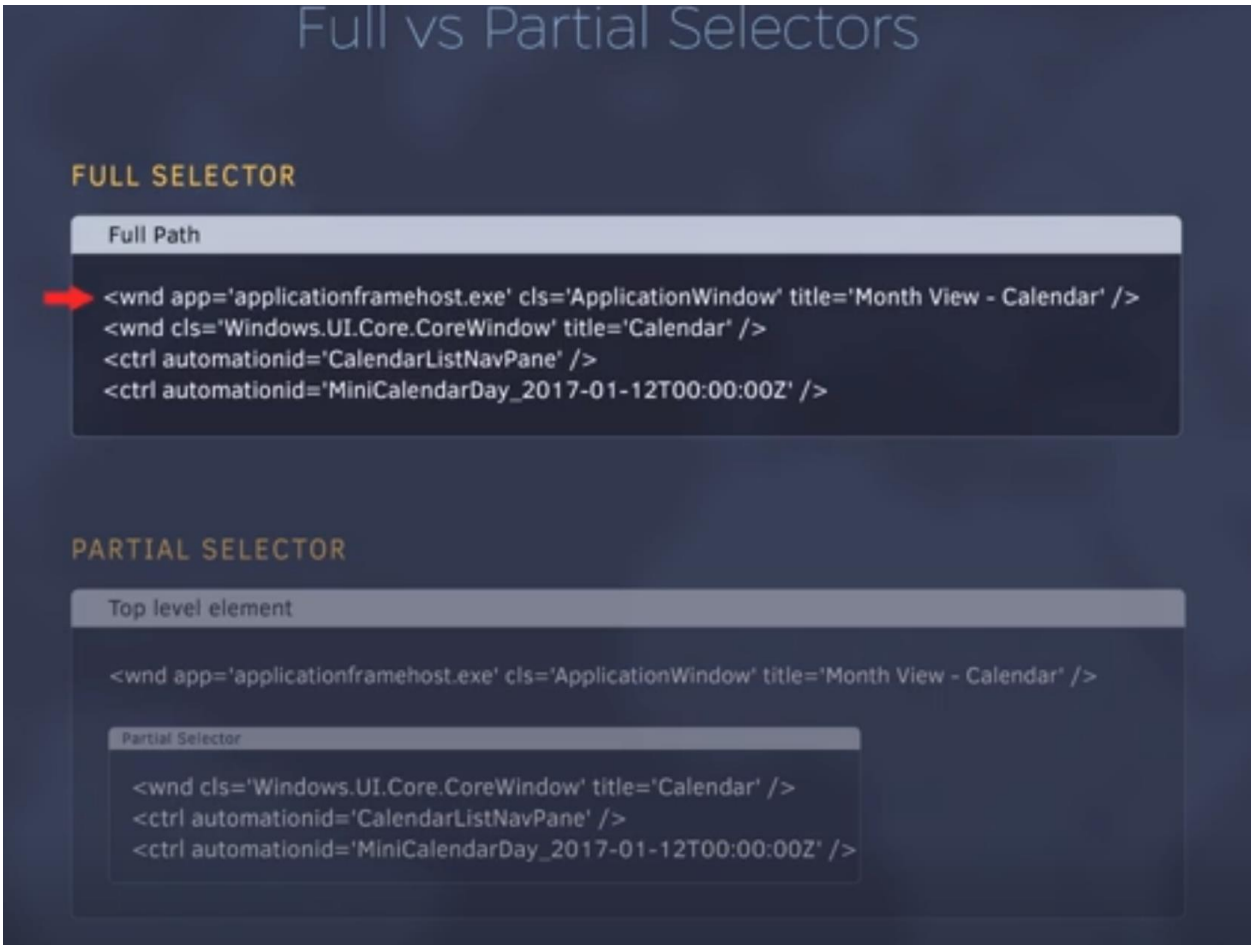


If Individuals Elements are not recognized , try changing the current UI Framework from settings nad indicate again.





Above Screen show : Raw Selector and the Final Selectors. Checking and Unchecking adds or remove from the final selectors.



Partial does not have Top Element.

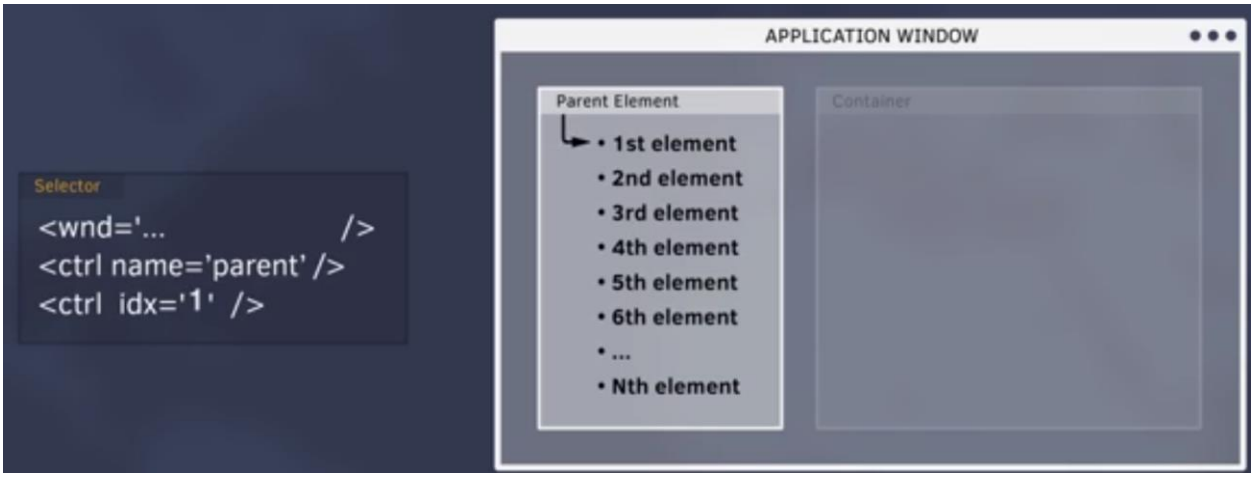
To Build Dynamic selectors in the situations where the The filenames itself keeps on changing. We make use of the wild Cards,

If an selectors is not working as expected ,we have to edit the selectors with * or ? .

We have something called, **Attach to Live element**, that edits the selectors so that it could work in both the cases.It basically puts the * for us.

INDEX

Index main use is to tell UI path to pick 1st second or the nth element.



Let say we want to always click on the forth element in a drop down. Ui path automation will always pick a Name. what we can do is in the ui explorer, delete the name , **Ui path will automatically add a index in its replacement.**

Sometimes the index appears in the selectors when we actually want to locate a specific element. Not necessarily the nth element. Now that's a Sign that the Ui element does not have enough specific attribute to uniquely identify the element.

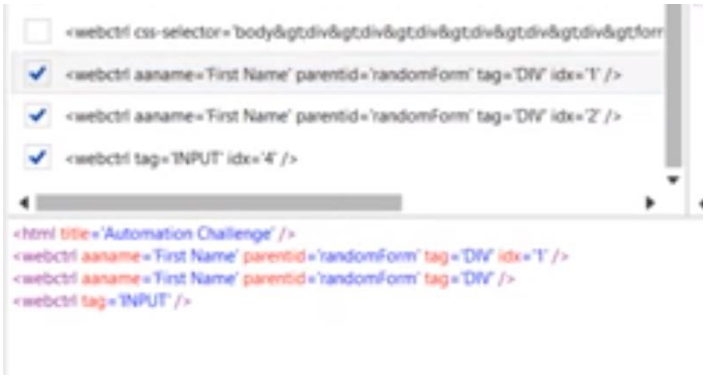
Dynamic Selector

Take Input from user and build a dynamic selector. we have to update the **automation_id** of the element.

Robots Test:

Let say we have pages , we have to fill data in a webpage wrt to excel.But after each submit, the textboxes changes their position.

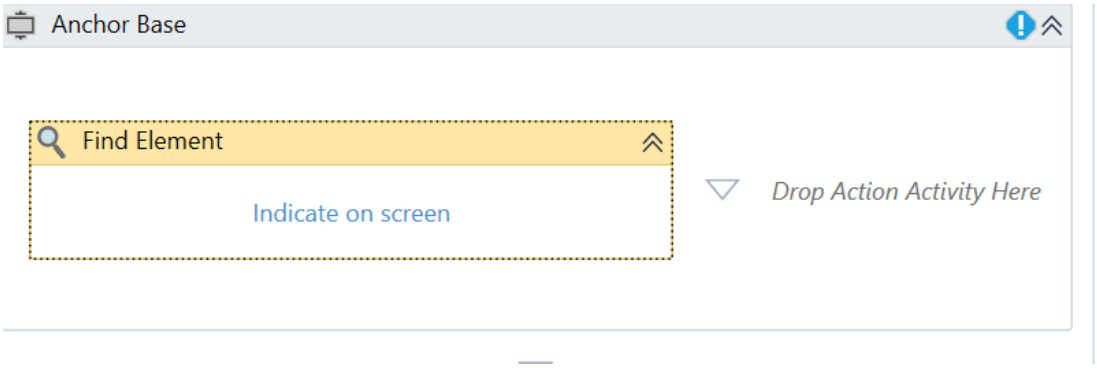
1. Look for the Ui Explorer of the Textbox and modify the selector by adding parents node.



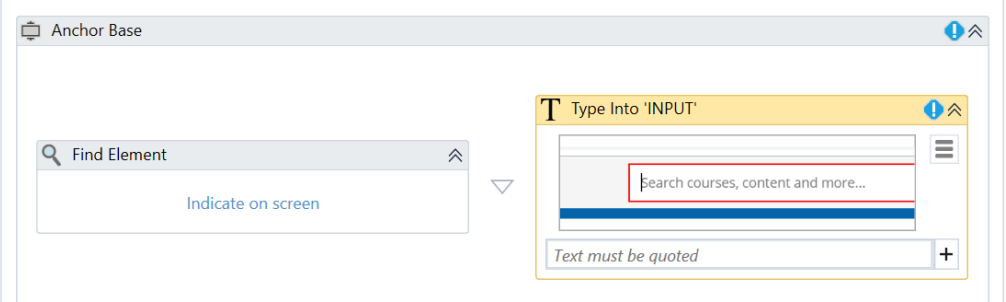
   But moreover adding too many parents to a selector causes errors at the slightest change in the any of the parents.

   <u>Alternative:</u>

2.Anchor Base is an container for an anchor and an action activity. We can use find element of find image as an anchor

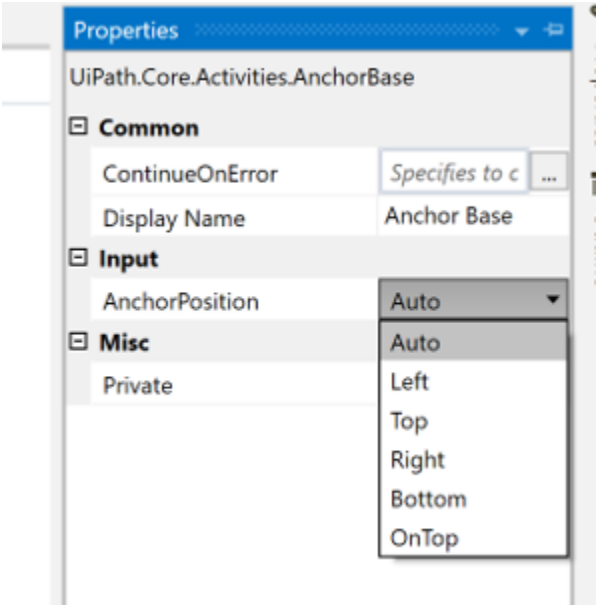


Output of find element: UIElement object – which can be later used as a reference in the element property of the target. The found element will be our anchor which requires a reliable selector. Anchor will be a LABEL .Since Label has a stable selectors.



Drag a TypeInto activity and Indicate the TextBox on screen. Here Ui path is using the anchor as a Selector.

Anchor Base a useful property to specify the position of he anchor.



Anchor Base uses the screen position so it does not work in the background.

Solution :Nav Tag – In UiExplorer , Select the select relative button. Other values Supported by the Nav Tag are the Next and the Prev , which goes to the nest and the previous element in the tree respectively.

We can use Anchor base or the relative selectors whenever we are not having reliable selectors on the screen.

The difference is that Anchor base uses the element position and does not work in the background.

Relative selector does work in the background, but relies on the internal structure of the application.

About Selectors

SUGGEST EDITS

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.

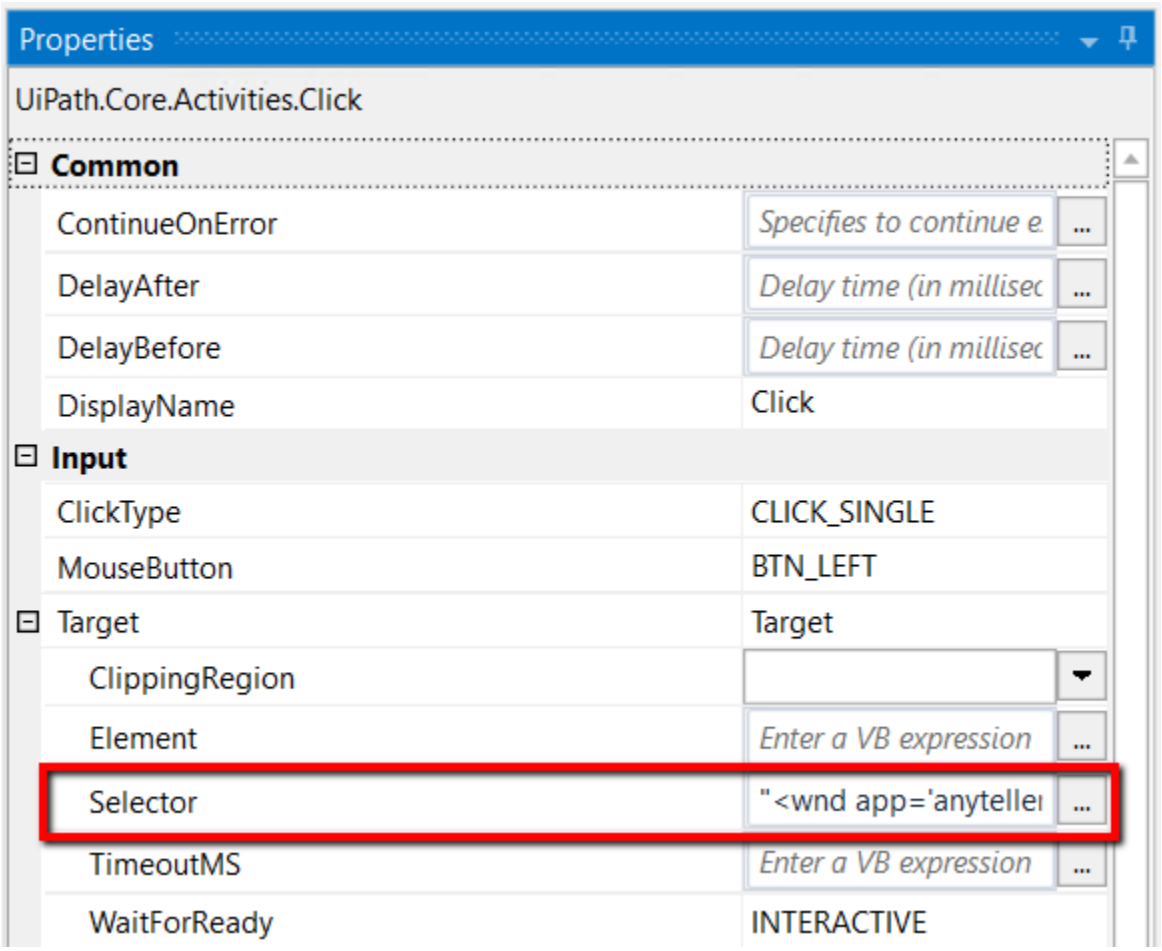
Each node has one or more attributes that help you correctly identify a specific level of the selected application.

Each node has the following format:

```
<ui_system attr_name_1='attr_value_1' ... attr_name_N='attr_value_N'/>
```

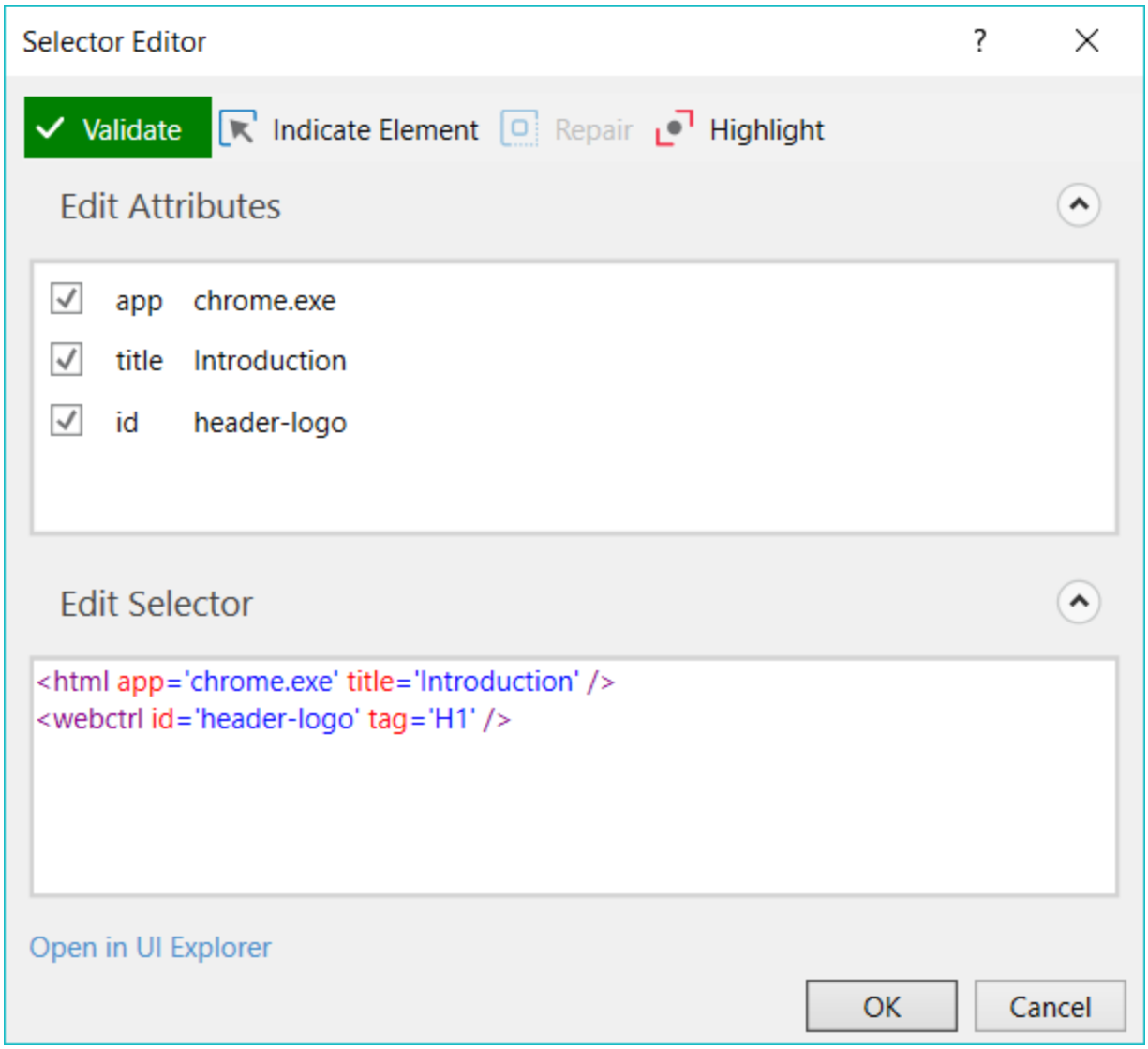
Every attribute has an assigned value. It is important to pick attributes with a constant value. If the value of an attribute changes each time the app is started, then the selector will not be able to correctly identify the element.

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.



Option	Description
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: • Validate Selector is being validated • Validate Valid selector

- 

Invalid selector
- 

Modified selector, revalidate

The button is correlated with UI Explorer validation states.

Indicate Element

Indicate a new UI element to replace the previous one.

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 the option is disabled with a click. The button is enabled only if the selector is valid.

Edit Attributes

Contains all the application components needed to identify the target application (a window, a button etc.). This section is editable.

Edit Selector

Holds the actual selector. This section is editable.

Open in UI Explorer

Launches the UI Explorer. The option is enabled only for valid selectors.

Note:

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

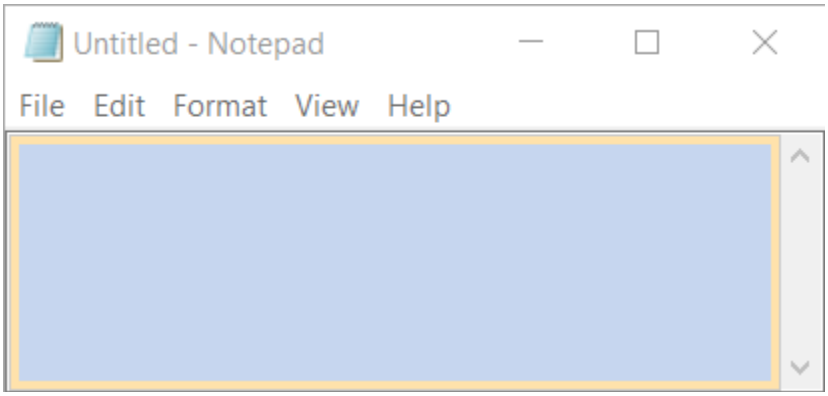
Wildcards are symbols that enable you to replace zero or multiple characters in a string. These can be quite useful when dealing with dynamically-changing attributes in a selector.

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

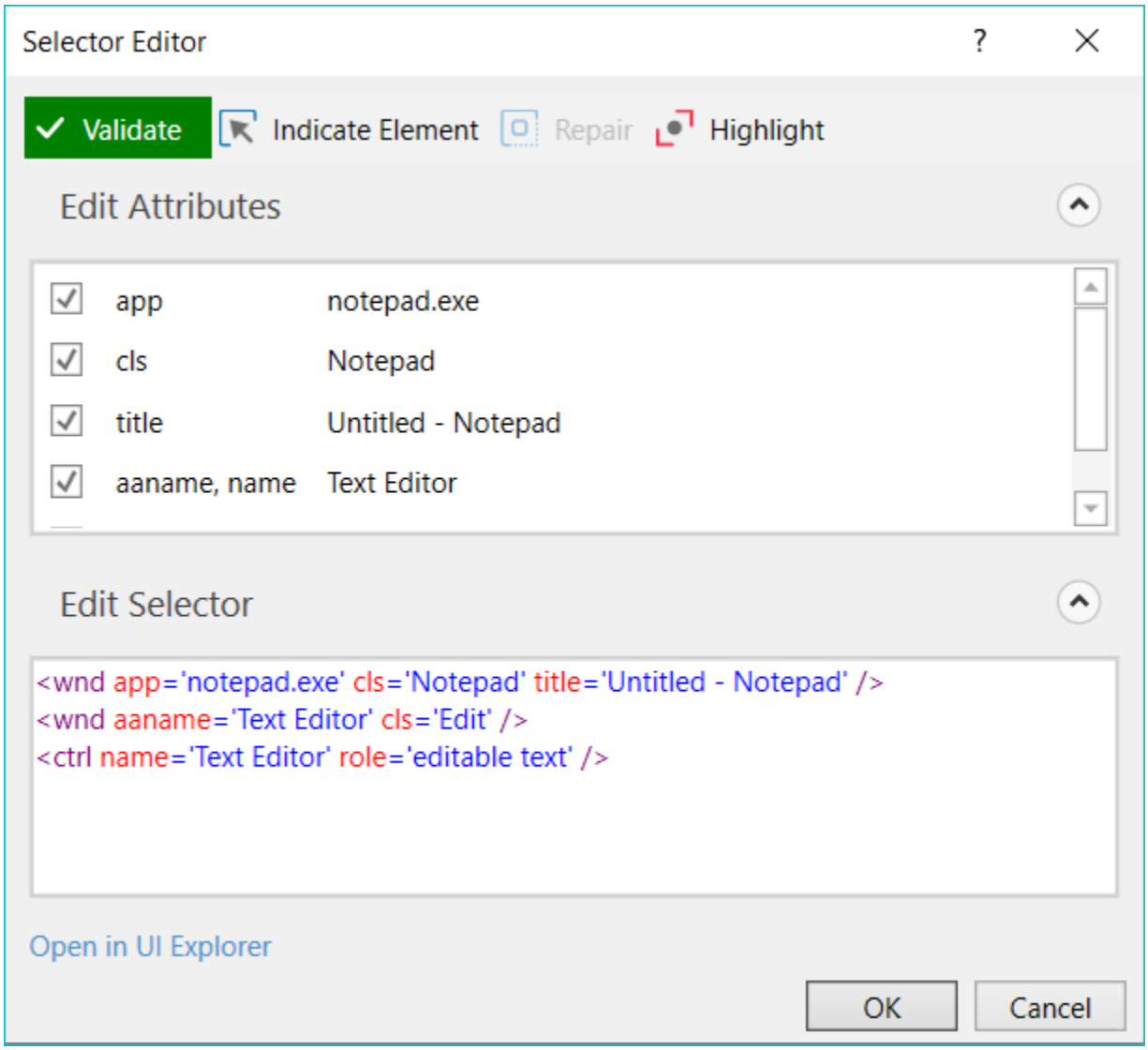
Example of Generating a Selector with Wildcards in the Selector Editor Window

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** 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.

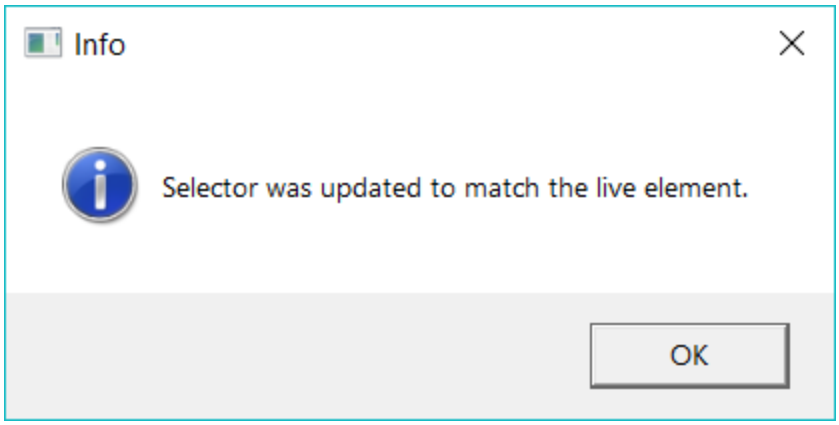


5. In the **Properties** panel, click the Ellipsis [...] button next to the **Selector** field. The **Selector Editor** window is displayed.

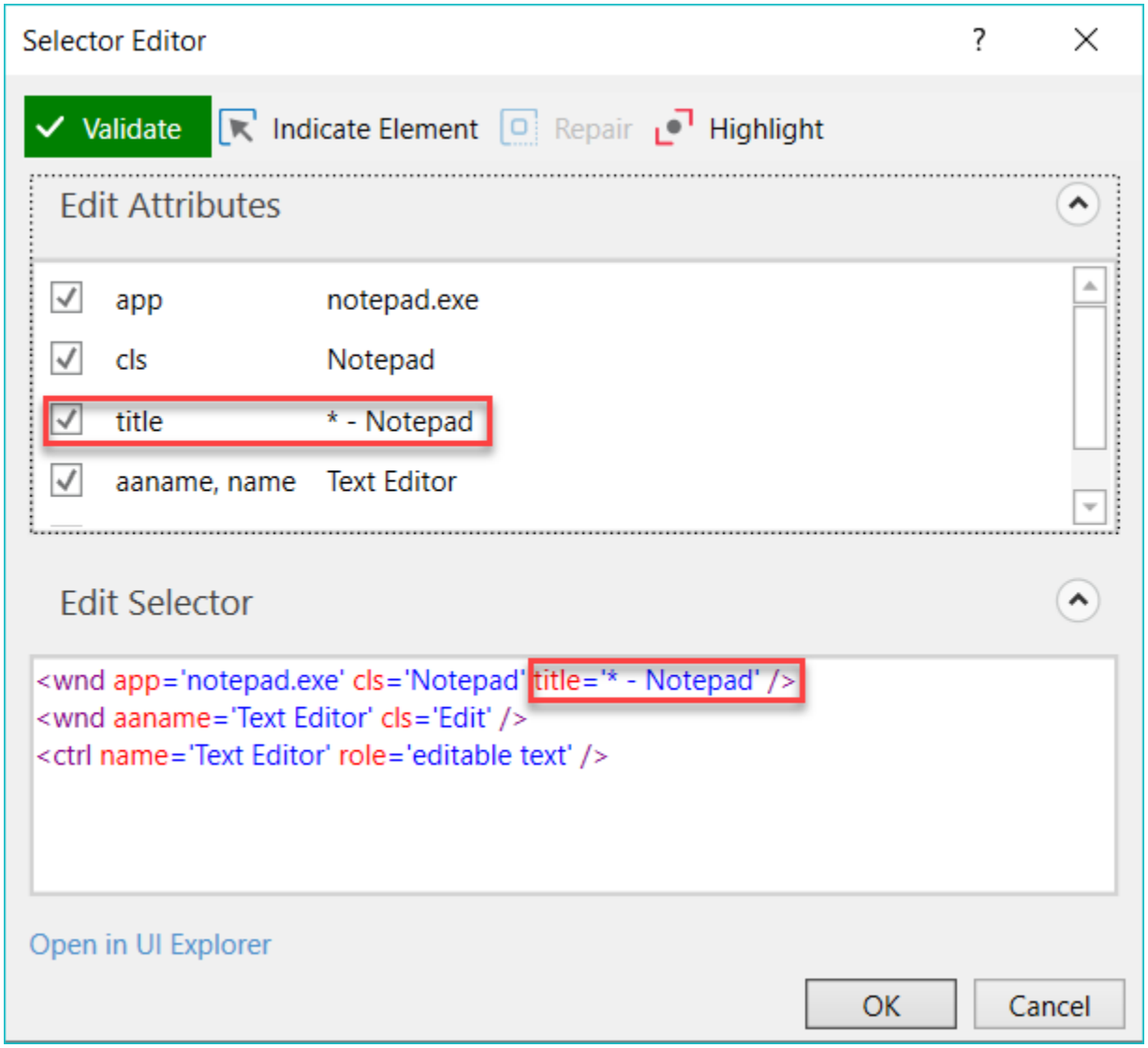


6. Open any .txt file with Notepad. Note that the window title is partially different than the one at step 1.

7. 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.



8. Click **OK**. The **Selector Editor** window and the selector are updated with a wildcard.



Full versus Partial Selectors

SUGGEST EDITS

Full selectors:

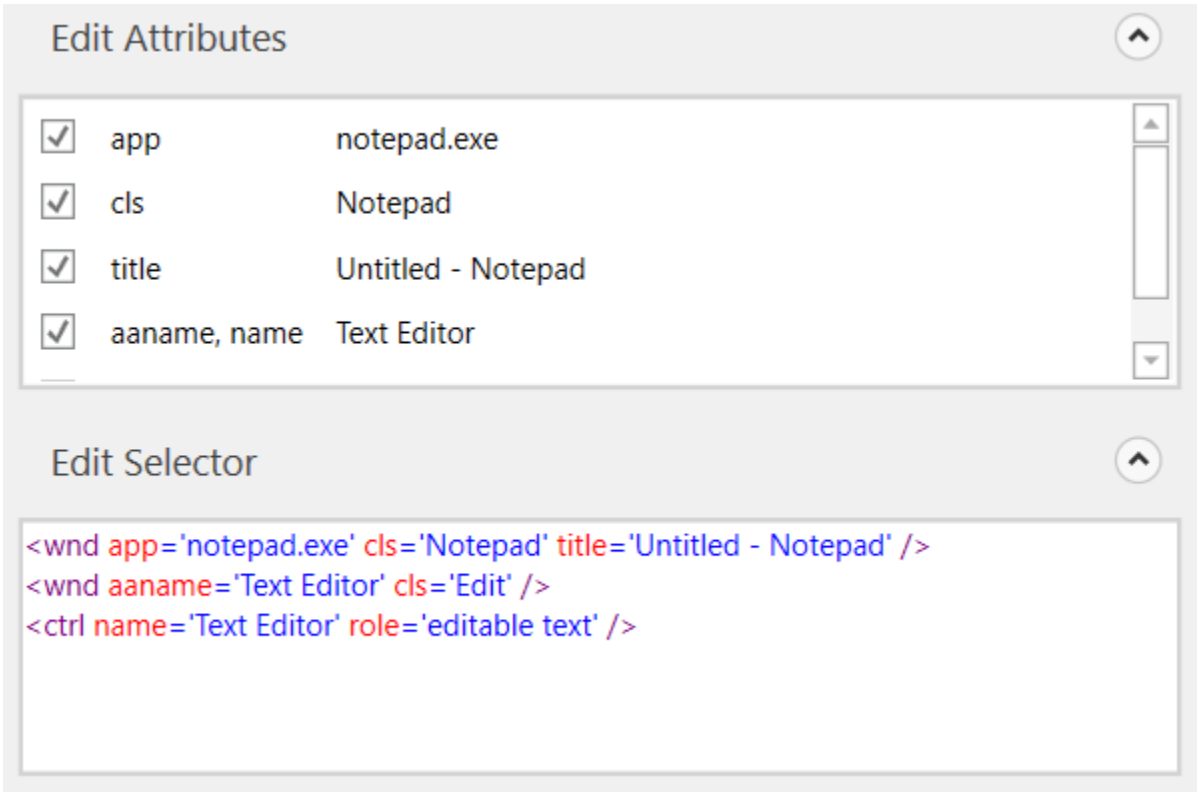
- Contain all the elements needed to identify a UI element, including the top-level window
- Generated by the Basic recorder
- Recommended when switching between multiple windows

Partial selectors:

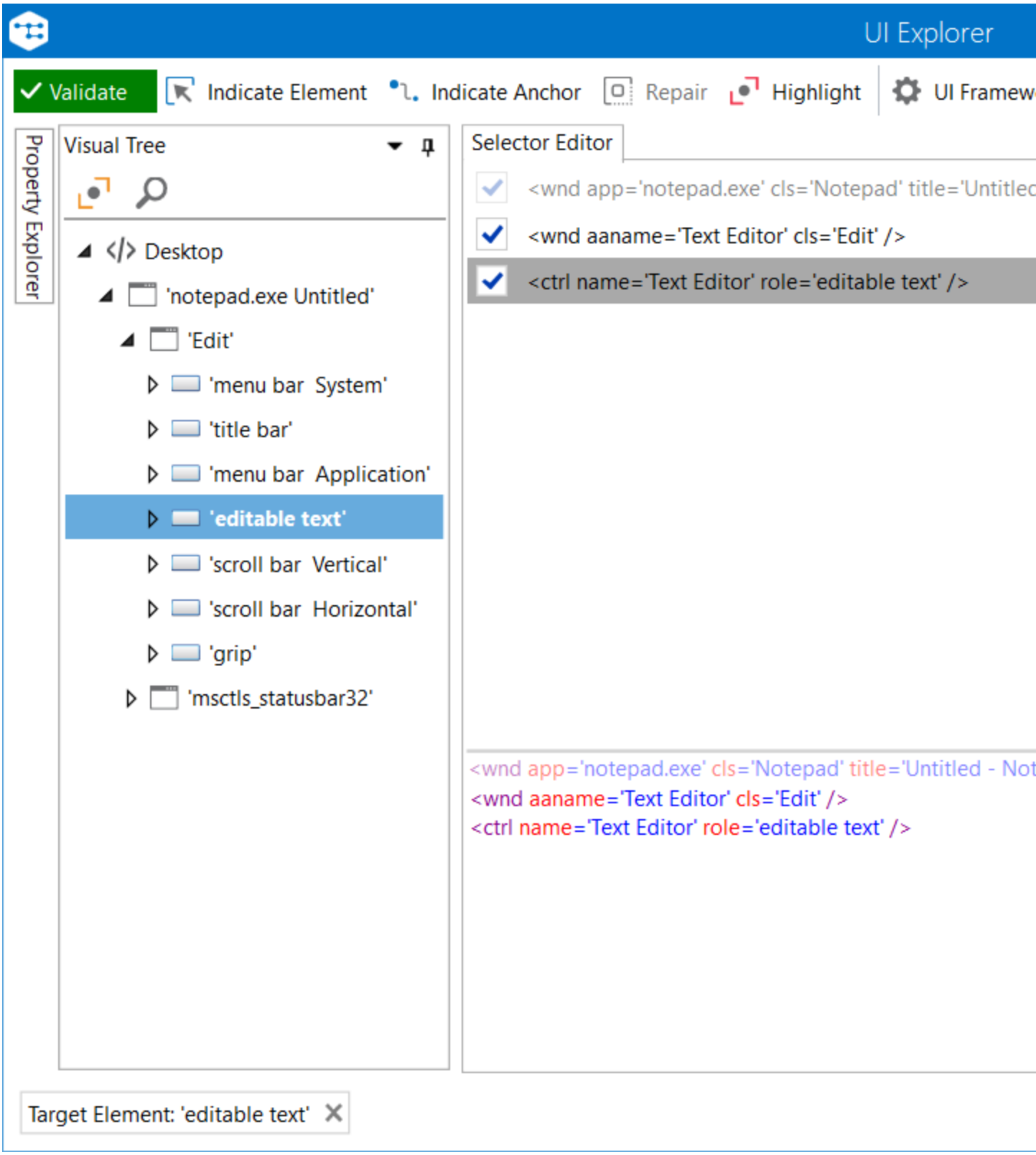
- Generated by the Desktop recorder
- Do not contain information about the top-level window
- Activities containing partial selectors are enclosed in a container (**Attach Browser** or **Attach Window**) that contains a full selector of the top-level window
- Recommended when performing multiple actions in the same window

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

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



Selector Editor and UI Explorer display the full selector, not just the partial one. However, only elements belonging to the partial selector can be edited, the prepended ones are grayed out and read-only.


UI Explorer
Validate
Indicate Element
Indicate Anchor
Repair
Highlight
UI Framew
Property Explorer
Visual Tree
Desktop
'notepad.exe Untitled'
'Edit'
'menu bar System'
'title bar'
'menu bar Application'
'editable text'
'scroll bar Vertical'
'scroll bar Horizontal'
'grip'
'msctls_statusbar32'
Selector Editor
<wnd app='notepad.exe' cls='Notepad' title='Untitled
<wnd aaname='Text Editor' cls='Edit' />
<ctrl name='Text Editor' role='editable text' />
<wnd app='notepad.exe' cls='Notepad' title='Untitled - Not
<wnd aaname='Text Editor' cls='Edit' />
<ctrl name='Text Editor' role='editable text' />
Target Element: 'editable text'

UI Explorer

SUGGEST EDITS

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.

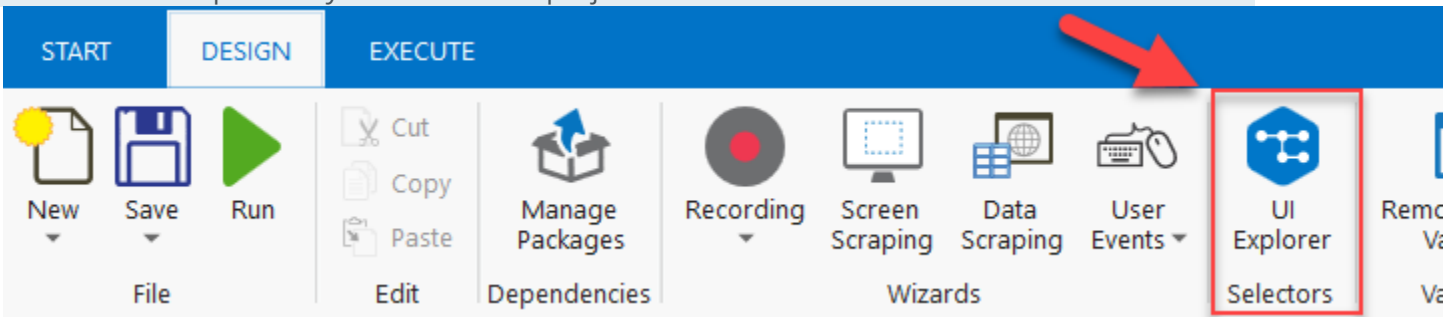
Note:

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:

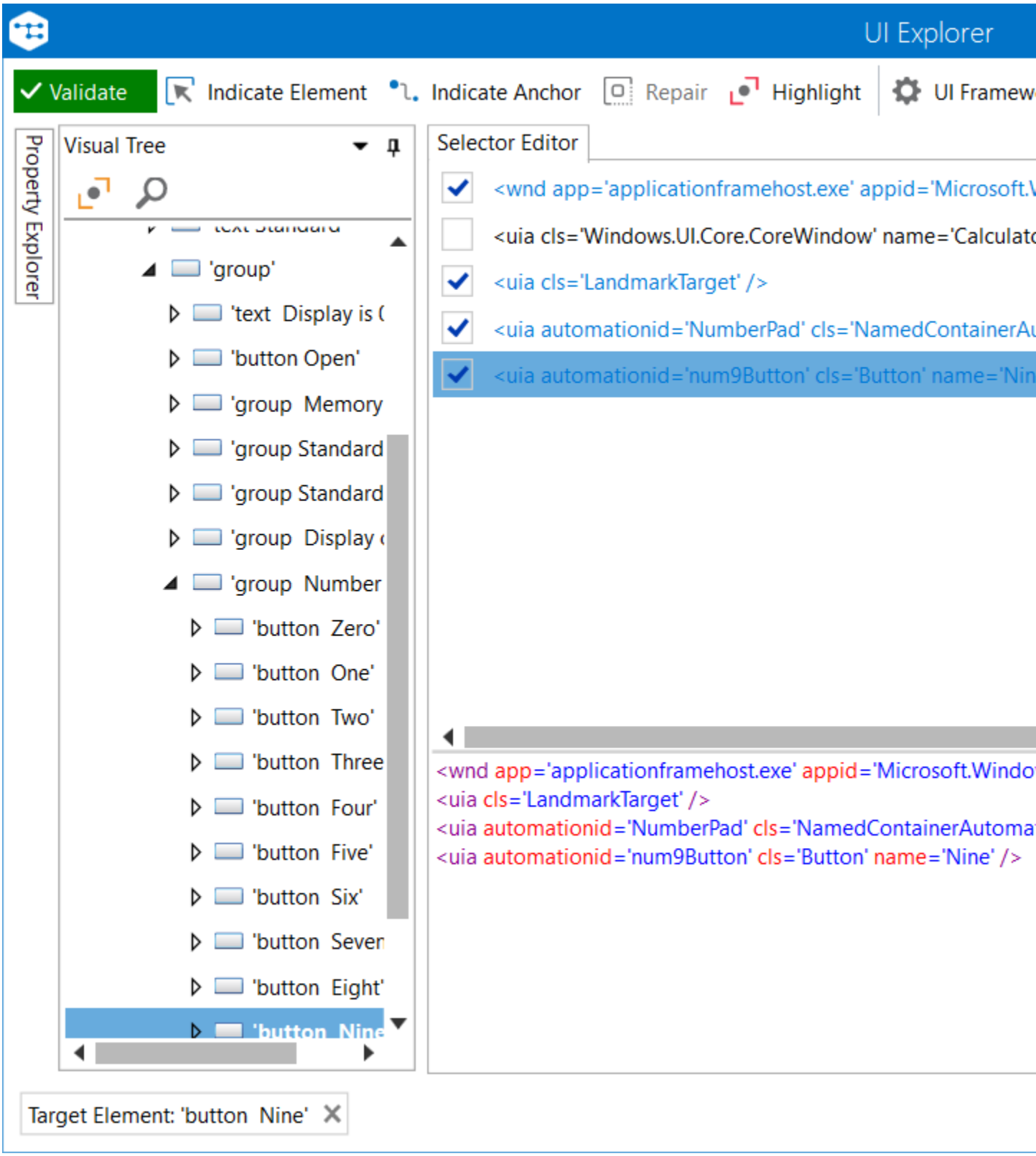
The version of the UIAutomation package that is currently used is displayed in the lower right corner in the **UI Explorer** you have opened. This version varies, as launching the **UI Explorer** from the **Tools** page uses the default **UI Automation**version shipped with the Studio version you are 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 UI 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.

rameworks

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 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 – 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**, 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

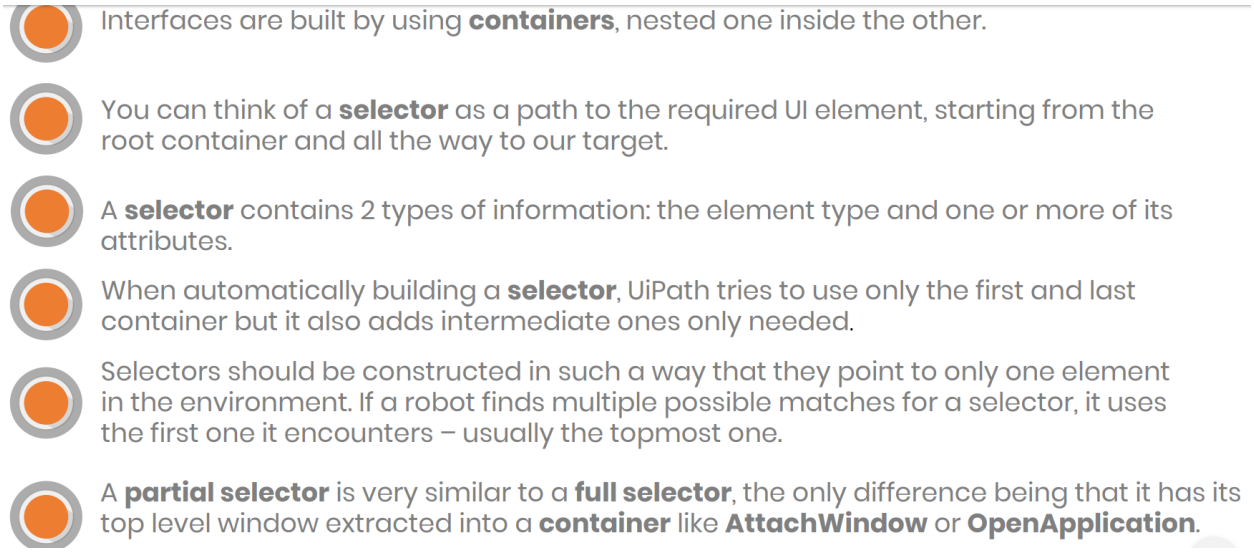
Displays all the available attributes of a selected node (from the **Selector Editor** panel).

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

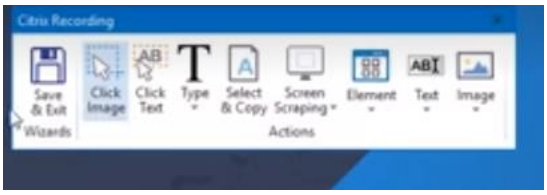
Displays all the attributes that a specified UI object can have, including the ones that do not appear in the selector. They cannot be changed.

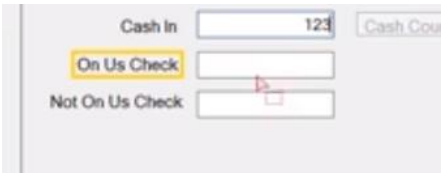


- **Basic recording** generates **full selectors** while **Desktop recording** generates only **partial** ones.
- You can build **dynamic selectors** using the 2 available **wildcards**: **question mark** (**?**) takes the place of a single character and **asterisk** (*) that replaces any number of characters or by using variables.
- Use **Attach to live element** to update an existing selector to also match a 2nd element.
- Use the selector's **idx** property to get a certain occurrence of an element that is found multiple times.
- Anchor Base activity and Select Relative Element in UiExplorer can be very useful to build reliable automations when the selectors might not be very stable.

- Avoid using the **idx** attribute if its value is larger than 2, unless you don't have other options. Always try to add other attributes to make sure the selector is stable.
- When it's likely that you will have interference from other applications and windows it's best to use partial selectors.

CITRIX AUTOMATION





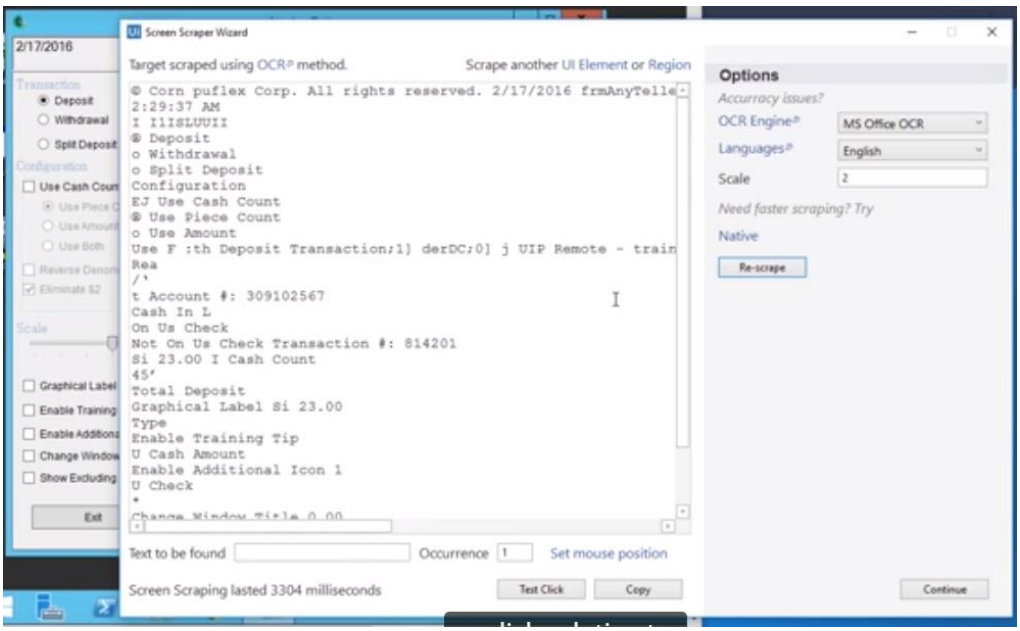
Click Image and Indicate Position and Type.

We have something called Click Text as shown below.

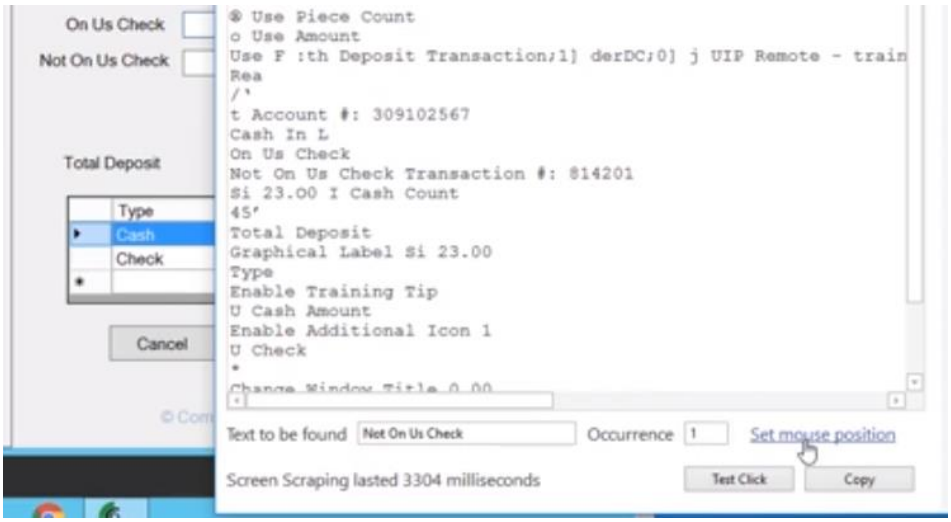
Click text uses OCR to scan the screen of the virtual machine for text. Hence it works even if things like windows-theme or text-size are different.

Click-Image is faster and reliable but more sensitive to such graphical variations and can fail if colors or background details change. So, Its More prone to error in fluctuating environments.

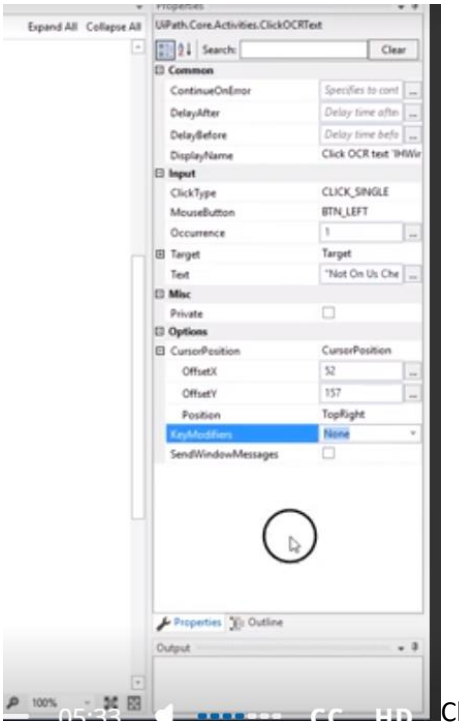
Below is what Click Text output Looks Like



Specify the text to be found and set Mouse position.



For the OCR Click in the generated workflow we have different click options available.



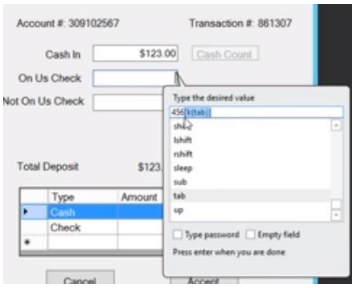
ClickType ,Mouse Button and KeyModifiers

On virtual machines, ClickTxt relies on OCR to find the desired results.so even one misidentified texts can throw it off track.

ClickImage is very reliable unless the theme or the graphical representation changes.

Alternative: Avoid Mouse interactions and use Keyboard buttons.

TypeInto then add an Tab from the scroll below.



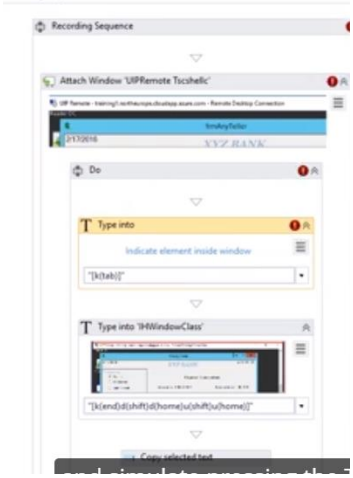
We can combine input in a single TypeInto.

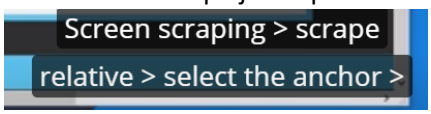
Textk[Tab]Textk[Tab]Textk[Tab] and when the focus in on Accept (Press Space)

Get text out of Citrix:

To get text out of citrix automation.

1. Select and Copy

- Works only for selectable texts.
- Copy selected texts works on the entire Virtual machine area . So all Selectable texts are scrapped.
- Manually add a TypeInto and Press Tab to get to the desired location.
- 

2. Scrape Relative:
    - Allows us to scrape just a portion of an image relative to an anchor.
    - 

    - And Indicate the area to scrape.
    - 

    - Then Play with OCR and different OCR Techniques to get desired results.
    - This is a combination of 4 actions:
      1. Find anchor Image.
      2.Get OCR Text
      3.Action that resets the clipping region because it is a shared resource and this avoids interference with other operations.

About Image and Text Automation

SUGGEST EDITS

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.

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 behavior 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

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, there are several technologies that can be used for these activities, each with their own advantages in certain situations.

Double Click, **Click**, **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 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 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** activity.

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 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.

Text Activities

SUGGEST EDITS

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, **Click Text** and **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 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 extracts a string and its information from an indicated UI element using the FullText screen scraping method. 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 extracts a string and its information from an indicated UI element using the Native screen scraping method. 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 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 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** 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.

OCR Activities

SUGGEST EDITS

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, **Click OCR Text** and **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 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, 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 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 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

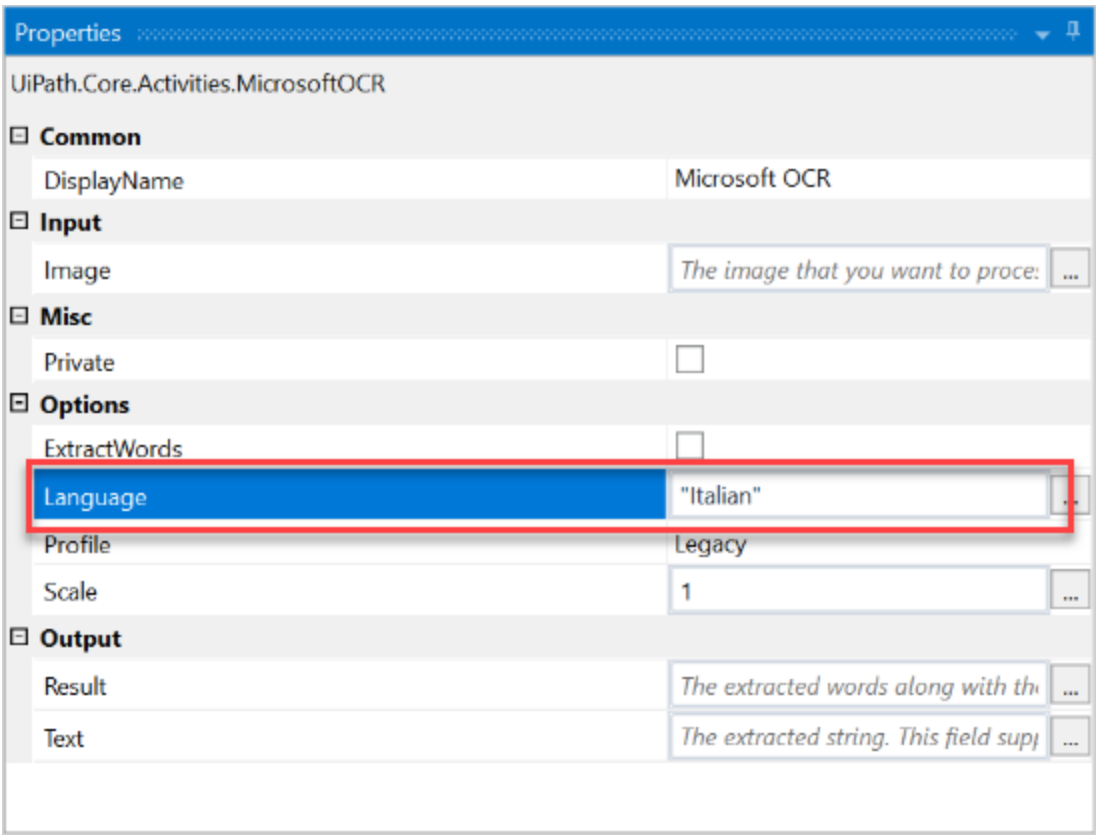
region, and generates selectors for them. If this does not work for you, then manual intervention might be required. This activity returns a boolean variable that states whether the text was found or not.

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.

Installing OCR Languages

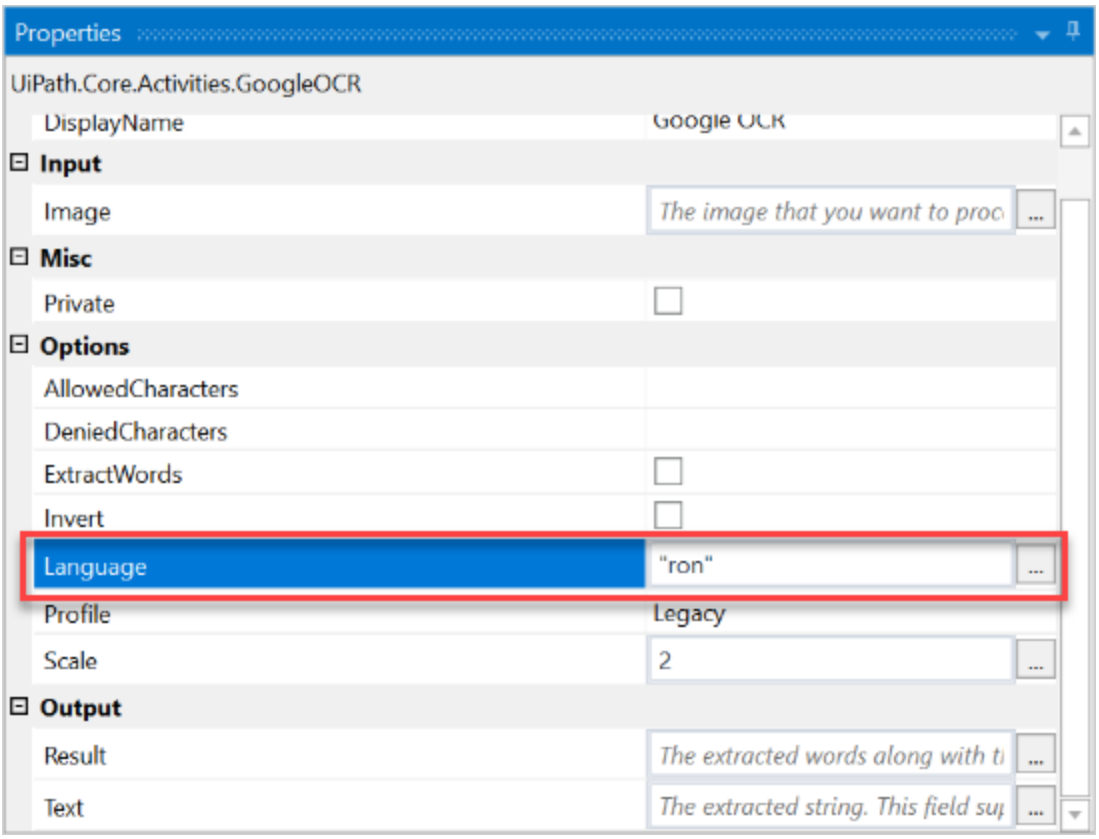
SUGGEST EDITS

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:



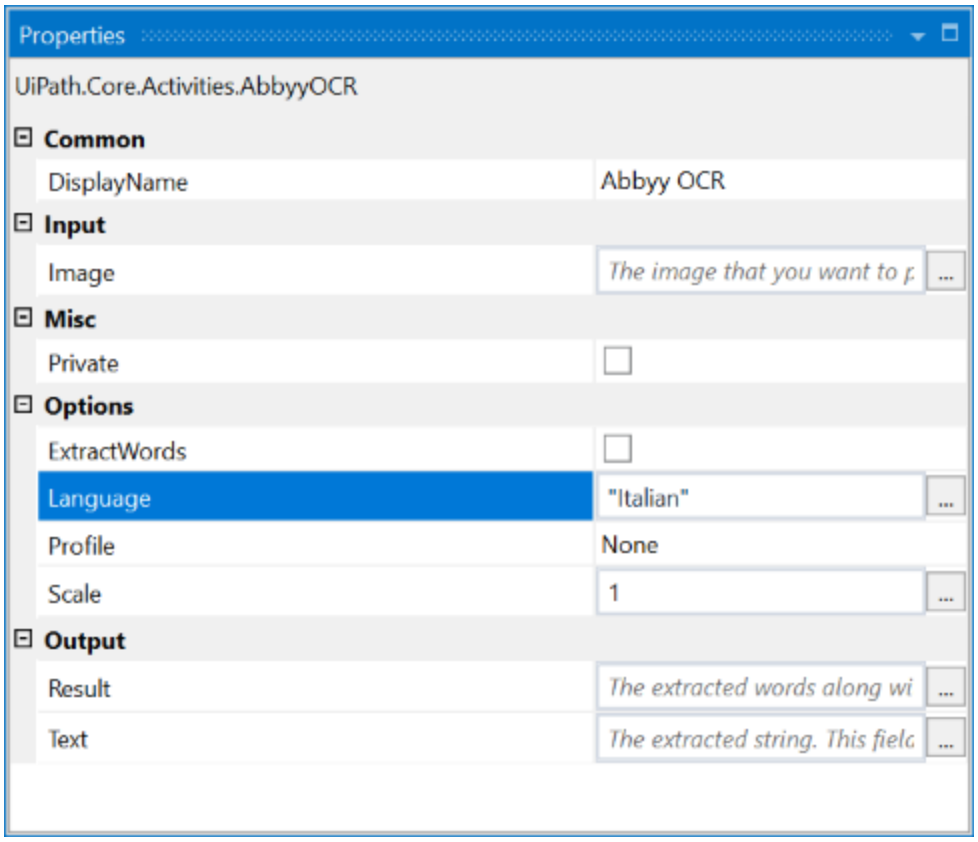
Note:

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

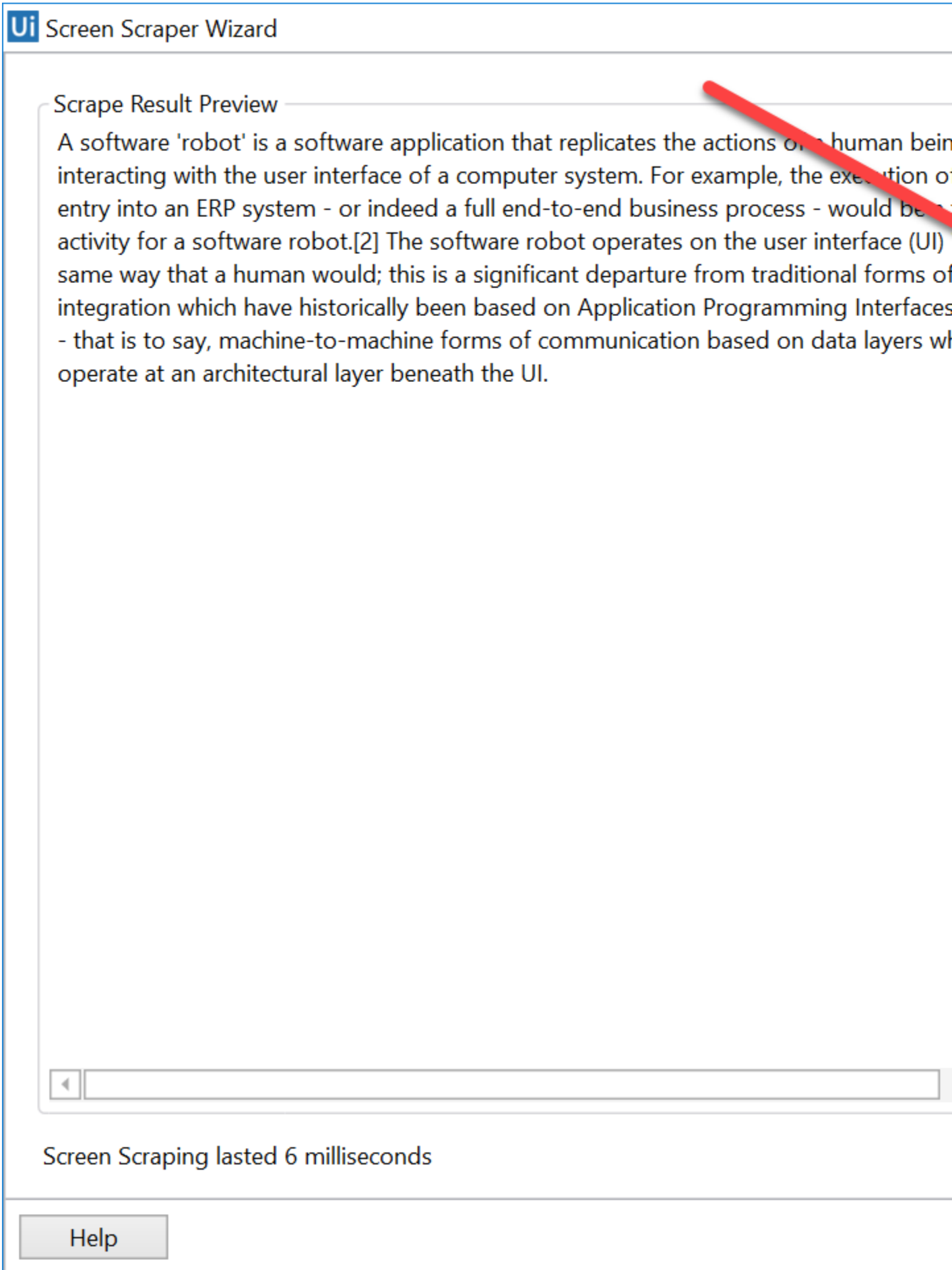


Note:

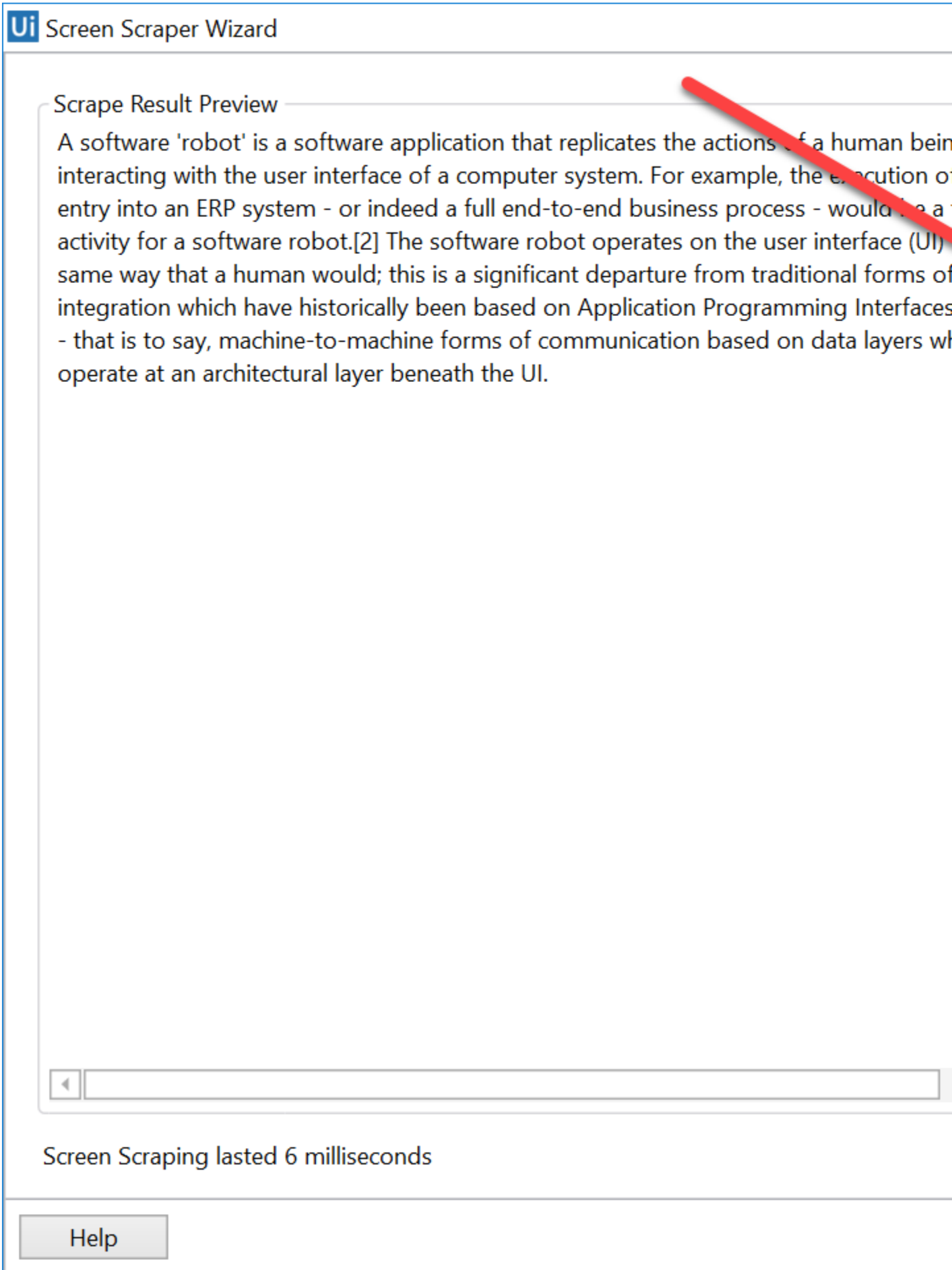
ABBYY FineReader Engine includes the majority of supported OCR languages by default. They can be used right after a successful installation of the engine.



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


Screen Scraper Wizard
Scrape Result Preview
A software 'robot' is a software application that replicates the actions o human bein
interacting with the user interface of a computer system. For example, the exe tion o
entry into an ERP system - or indeed a full end-to-end business process - would be
activity for a software robot.[2] The software robot operates on the user interface (UI)
same way that a human would; this is a significant departure from traditional forms of
integration which have historically been based on Application Programming Interfaces
- that is to say, machine-to-machine forms of communication based on data layers wh
operate at an architectural layer beneath the UI.
Screen Scraping lasted 6 milliseconds
Help

This can also be done for the Google OCR engine.


Screen Scraper Wizard
Scrape Result Preview
A software 'robot' is a software application that replicates the actions of a human bein
interacting with the user interface of a computer system. For example, the execution o
entry into an ERP system - or indeed a full end-to-end business process - would be a
activity for a software robot.[2] The software robot operates on the user interface (UI)
same way that a human would; this is a significant departure from traditional forms of
integration which have historically been based on Application Programming Interfaces
- that is to say, machine-to-machine forms of communication based on data layers wh
operate at an architectural layer beneath the UI.
Screen Scraping lasted 6 milliseconds
Help

Note:

ABBYY OCR is not available in the default **Screen Scraping** window. You can change the language of the OCR engine by modifying the **Language** property.

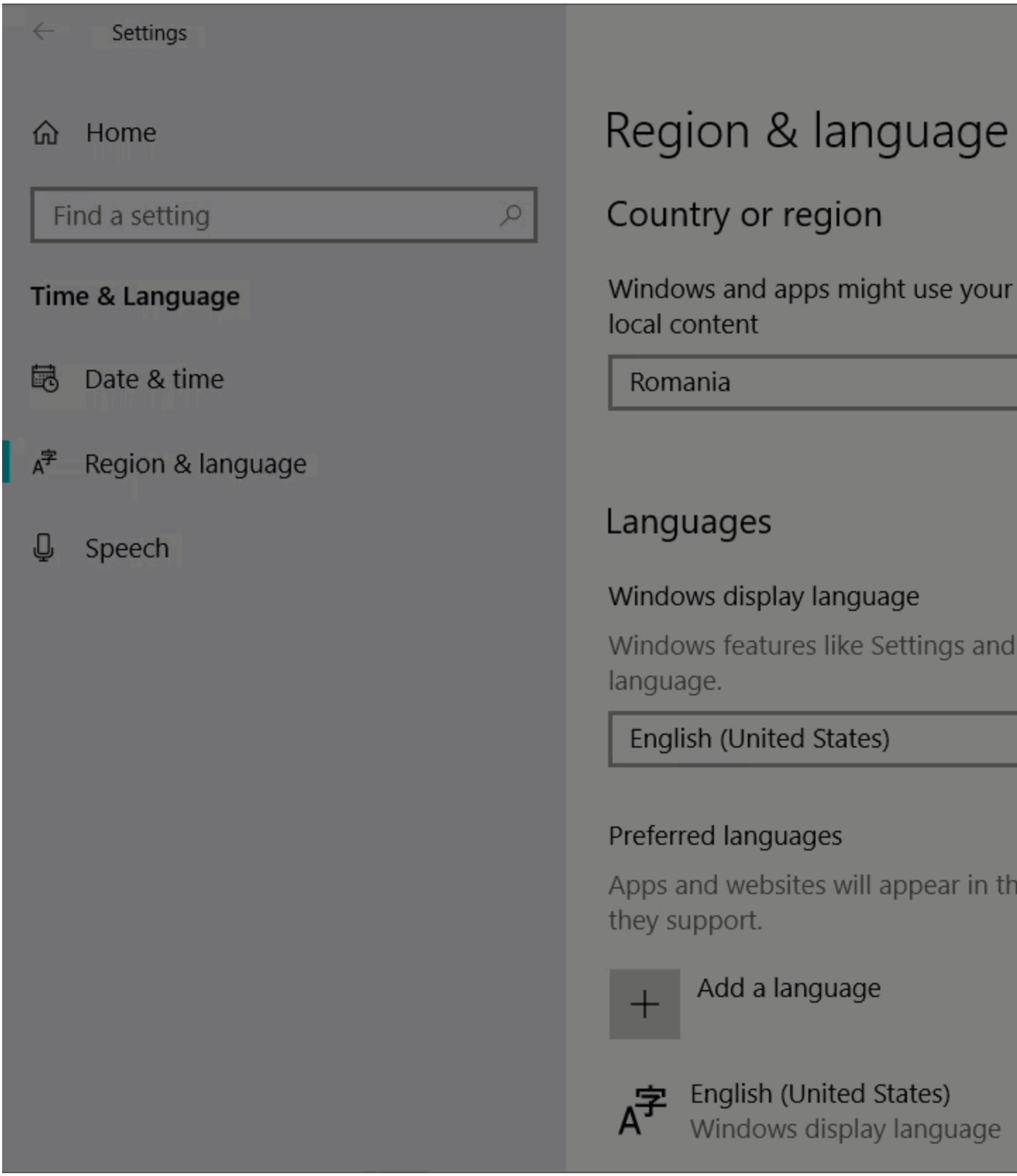
Installing an OCR Engine and Changing the Language

Microsoft OCR

Windows 10

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

1. Go to Start Menu > Settings, the **Windows Settings** window opens. Make sure to maximize the window.
2. Access **Time & Language**, the **Date & time** window opens.
3. On the left side menu, select **Region & language**. Under **Languages**, click **Add a language**.
4. Choose your preferred language and click **Next**. The **Install language features**window opens.
5. Uncheck the **Set as my Windows display language** check box. Click **Install** and wait for the installation to finish.
6. Restart UiPath Studio for new languages to become available. The language can now be used in Studio by adding its name between quotation marks (“Japanese”).


Settings
Home
Find a setting
Time & Language
Date & time
Region & language
Speech
Region & language
Country or region
Windows and apps might use your
local content
Romania
Languages
Windows display language
Windows features like Settings and
language.
English (United States)
Preferred languages
Apps and websites will appear in th
they support.
Add a language
English (United States)
Windows display language

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.

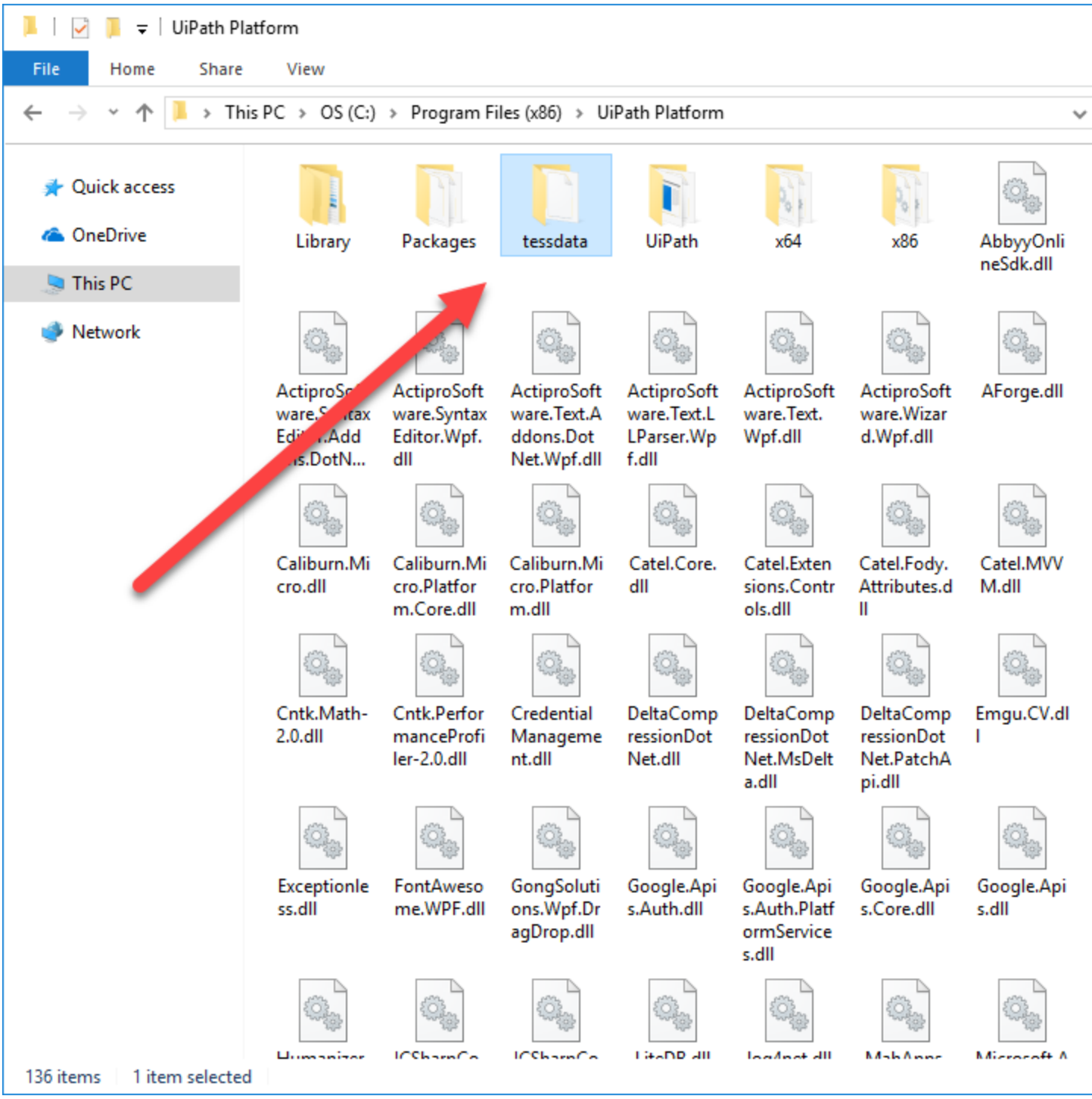
Windows 7 and Windows 8.1

The Microsoft OCR engine needs to be manually installed.

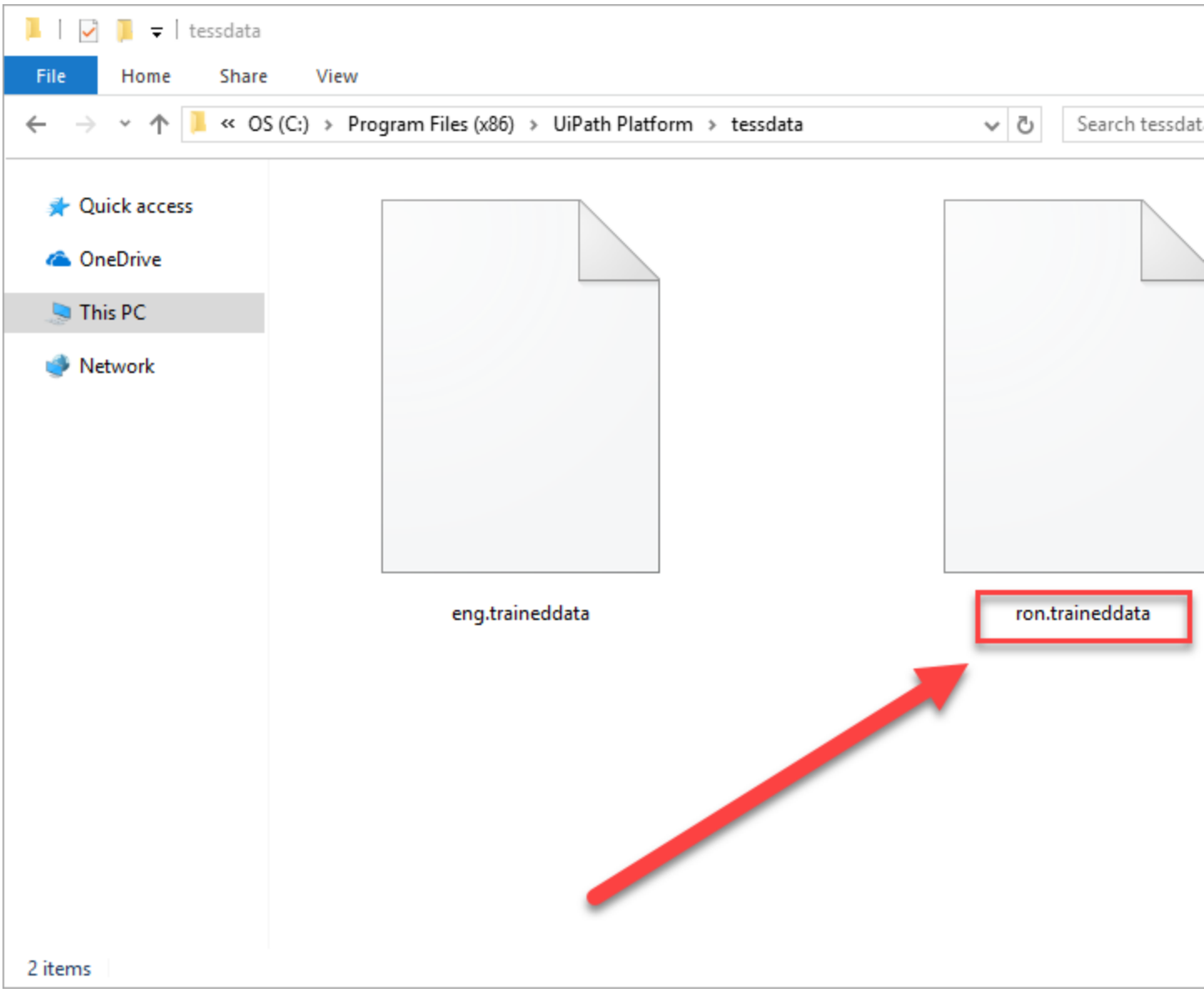
1. Download and install Microsoft SharePoint Designer 2007 .
2. Choose your Office version and language here, and follow the instructions to set up the desired language.

Google OCR

1. Search for the desired language file on this page.
2. Save the file in the **tessdata** folder of the UiPath installation directory (`C:\Program Files (x86)\UiPath\Studio\tessdata`).



3. 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 page.
   1.2. Go to Technical Support & 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.

5. 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.

6. Navigate to the Bin folder in the installation directory. It should look something like `C:\abbyy\fr11\Bin`.

7. 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.

8. Restart UiPath Studio to use ABBYY OCR.

Image Activities

SUGGEST EDITS

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, **Double Click Image** and **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 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 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 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 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.

- Since you cannot use selectors on the image that is sent in a Citrix environment, the approach is a little different than in other scenarios: you perform actions in relation to some elements.
- In a Citrix environment, **Click Image** and **Click OCR Text** are your best friends for inputting.
- **Click OCR Text** uses **OCR** to find the target text.
- Both **Click OCR Text** and **Click Image** offer options for modifier keys, single or double click and left or right mouse buttons.
- Remember you can tweak the **Click Image**'s **Accuracy** property to get the desired results. This value goes from 0 to 1 and higher values mean higher similarity between the images.
- Getting information out of such environments is done through the use of 2 methods: Copy Selected Text (**Select & Copy** from the recorder) and **Scrape Relative**. Just keep in mind that for **Copy Selected Text** to work you need Clipboard sharing enabled betw local and Citrix environment.
- **Scrape Relative** scrapes only a portion of an image, relative to an anchor.

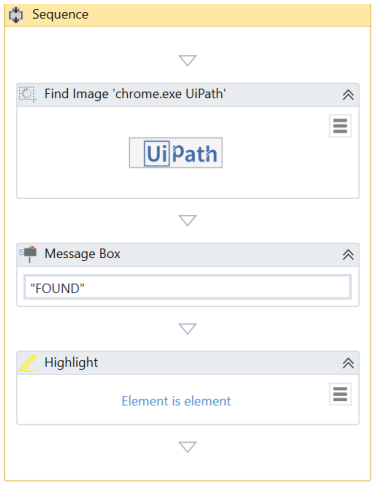
- Click Image is fast and reliable, but sensitive to graphical variations, while **Click OCR Text** is immune to such changes but susceptible to OCR failures.
- To avoid the problems that come with **Click OCR Text** and **Click Image** you should use keyboard shortcuts as much as possible.
- Selecting too much or too little of an image might lead to it not being found or a false positive.
- To get good results when matching images, the resolution should be at least equal to the one used when recording the workflows. You can also compensate such losses by lowering the **Accuracy** factor.
- Pay attention to how the application layout changes on different resolutions when using coordinate based techniques.

Advanced Citrix Automation

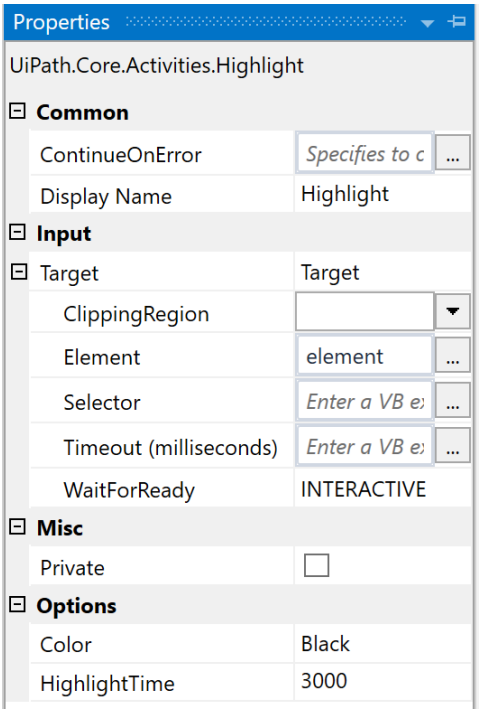
To automate on citrix , we have to first evaluate the state of an application on order to verify when the application is ready so that we can verify when we can type the username and password.

Incase of desktop automation , the uipath is smart enough to guess the state and know when to perform the automation.

In case of citrix, Wait for the Load icon to disappear and wait for the Site Favicon.can be achieved by Find Image and highlight the element found.



The highlight has the below properties to identify element better.



For Login we can use ClickRelative and click Image ,But a better way is to Use Keyboard Shortcuts.

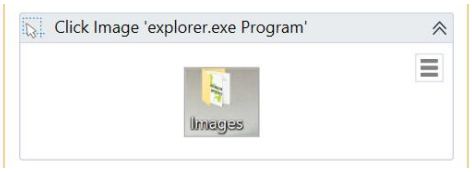
EmailAddress > Tab > password > Tab > Tab > Tab > Enter.

So Let's say , The Automation gets the image favicon but the input controls are not loaded . so the automation will fail, Bad solution will be to add an delay Activity but we might end up adding an extra long delay activity causin performance impact.

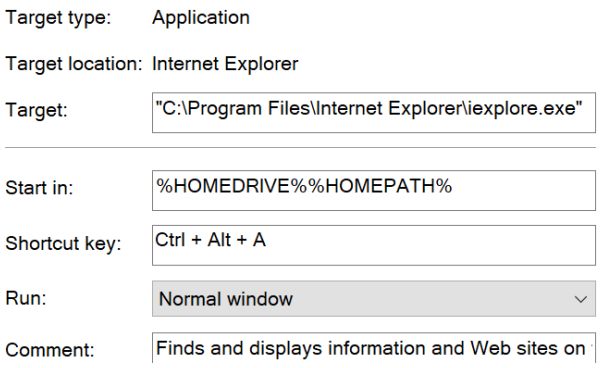
Optimal solution will be to add a new Find activity activity. Eg: Button Load.

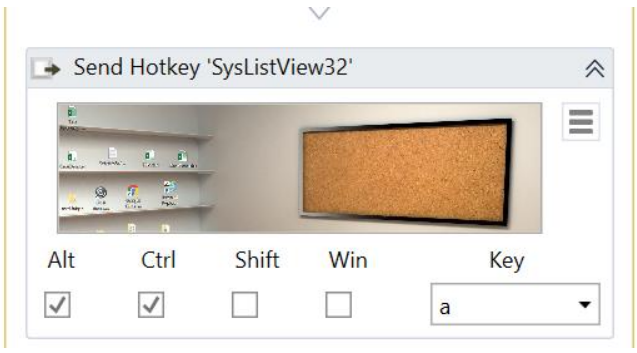
Citrix Automation: Starting Apps

1. Click Image activity and in button option: double click. Select an image with No portion of the background image selected.

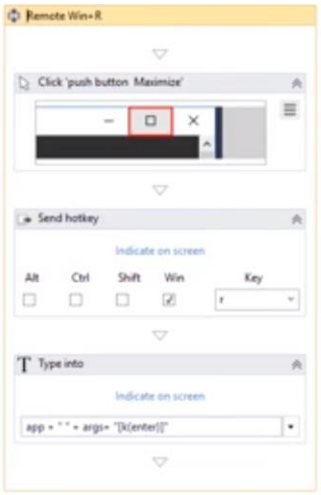


2. Assign an shortcut Key to the app and add an sendhotkey activity.





3. Open Run Command in Citrix and specify the target path and passing arguments.



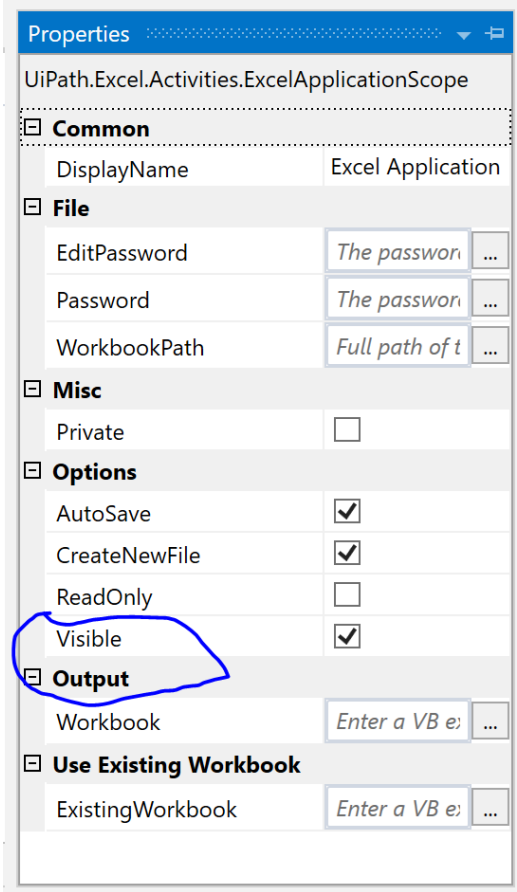
Go through lesson 7 and 8 practice exercises.

- Use **Find Image** to wait for a page or an application to finish loading.
- It's easier to automate such environments by using **keyboard shortcuts**.
- If you want to start an application by double clicking its desktop icon pay attention to the image selection, since it will be vulnerable to background color changes or even state changes such as when the icon is selected.
- A better way of starting an application is by assigning it a **keyboard shortcut**.
- If you want to run an application with parameters you can use **Win + R** and put in the full path of the application and add the parameters after it.

- Always use **keyboard shortcuts** when possible since it's faster and more reliable.
- When waiting for an application to load, don't add a delay to your actions - you should instead use **Find Image** or other similar techniques.
- When using **command prompt** to run an application, you can either try to use the **Win** + **R** combination or, if this doesn't work even in fullscreen mode, try opening Start menu and typing "cmd".

Excel and Data Tables: Basic Interactions

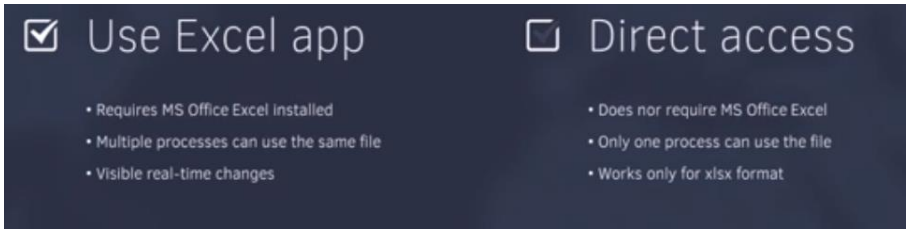
1. Activities are available under App Integration >> Excel.
2. First Activity is always the Excel Application Scope.



Excel application scope has a attribute called Visible -it tells uipath to either read the file using MS Excel or if its unchecked , the read operation will be performed internally directly on the file.

The Default visible option requires excel to be installed on the machine and all actions will be performed through it so we will be able to see in the real time what's happening. Helpful in debugging.

If visible option is unchecked, we do not need MS office to be installed on machine and whole process happens in background.



3. Read range : Reads the data from the range specified and outputs a Datatable. Range Parameter is empty by default , which means the whole sheet will be read.However Range can be specified as two colon separated cells (“A9:A60”).

4. Output DataTable : Does not prints the contents on the output Pane.It just comverts it to a string so that we can print it properly. Use the output of the output datatable as an input to Message box.

5. Since we need to write the data to a new excel file, we would need a new excel application scope. If the File does not exists the file will be created.

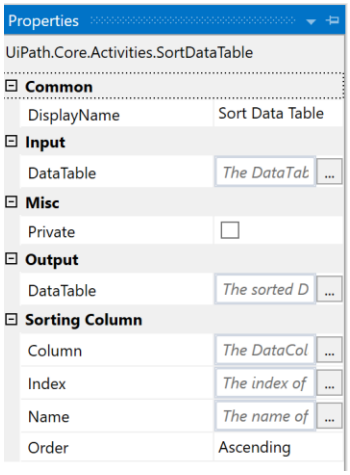
6. WriteRange: this is used to write the content to the new excel. If there is already data in the excel, the data will be replaced by ye new one, same as the paste command.

7. Add Headers is the option to select or De-select the Headers.

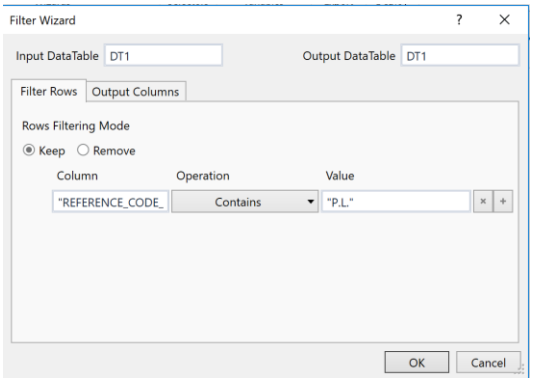
8. Append Range : same as write range, but it does not overwrite the existing data.but appends to the existing one.

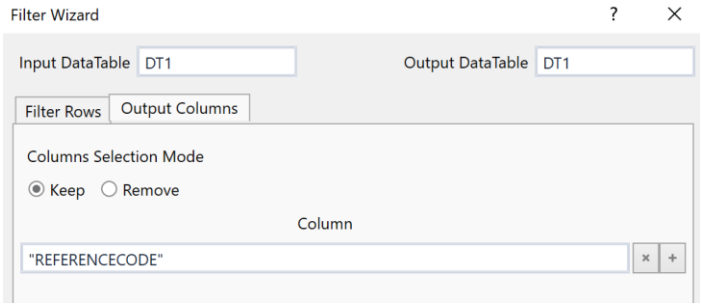
9. Build Data Table : Available at : Programming >> DataTable >> Build Data table : Used to create a new DataTable with specified number of rows and columns.

10. Sort Data Table: To Sort the DataTable.

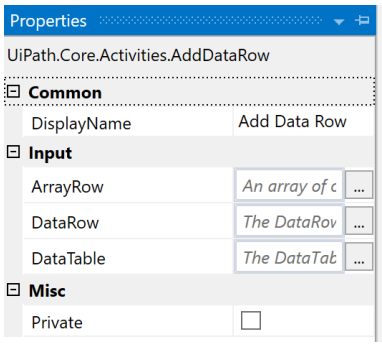


11. Read Cell and Write Cell : To Read and write values in a particular cell.
12. Select Range : Select the range specified that later can be used to copy or delete.
13. Filter Data Table :





14. Build Data Table : Create a new Data table with name ,age and income.
    a. Now For each row in the DataTable generated from excel read range. Iterate and add it to the new datatable created above.
    b. Add 3 GetRow Item activity ,which outputs the value of a specific cell to the current row.
    c. To indicate which cell we want,we can use an index or name of the column, its advisable to use the name , because if the data changes , the index will vary.
    d. Inside the for each , Add an AddDataRow Activity , Parameters .



Datatable : the new datable created by builddatatable.
ArrayRow :Array of the 3 Getrow item.

Practical :

Add Data in A1 and B1 to C1

A	B	C	D
159	36	195	
3359	35	3394	
9072	8996	18068	
623	56	679	
11	324	335	
6643	88942	95585	
1	6	7	
3213	4324	7537	
7732	43	7775	
952	13	965	
3577	91	3668	

1. Excel Application scope: Read the Excel in a Datatable.
2. For Each Row in the Datatable.
    - Get the rowindex for writing data in C1 ,C2 ,C3 .

Next, use a **For Each Row** activity and sum the first two columns.

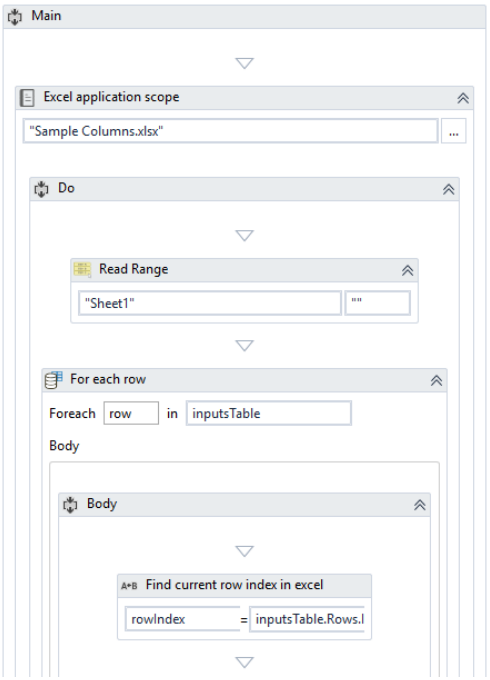
- Find an add a **For Each Row** activity and add it below the **Read Range** activity
    - Set it to loop through the DataTable created earlier, **inputsTable**
- Create an Int32 variable called **rowIndex** - this will keep track of what row to write on later
- Find and add an **Assign** activity inside the body of the **For Each Row** activity
    - Assign **inputsTable.Rows.IndexOf(row) +1** to **rowIndex**
        - This sets the value of rowIndex to the match the current row in the loop
        - The + 1 is because Excel Rows start counting at 1, whereas DataTables start at an index of 0 - this difference needs to be compensated for
- Below that activity, find and add two **Get Row Item** activities
    - For the first one, set the column index to 0 and the row to **row** (the temporary loop variable)
    - In the output parameter, use the Ctrl+K shortcut to create a variable called **valueA**
    - For the second one, set the column index to 1 and the row to **row**

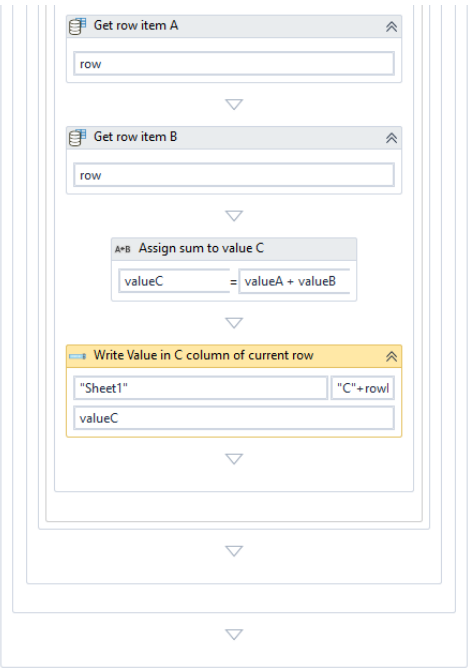
- In the output parameter, use the Ctrl+K shortcut to create a variable called **valueB**

- Find and add an **Assign** activity below
  - Assign **valueA + valueB** to **valueC** (Use the variable creation shortcut here as well)

- Find and add a **Write Cell** activity next
  - Keep the sheet as Sheet1
  - Set the range (the location in the sheet to write to) to **"C" + rowIndex**.**ToString**
    - Throughout the loop, this will evaluate to "C1" then "C2," and so on down the third column
  - Set the value to **valueC**

- This is what the final workflow should look like:



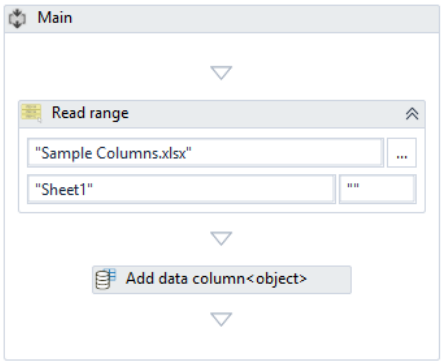


Part B:

In this part, the file will be read without an Excel Application Scope because the automation will be done internally.

- Find and add a **Read Range** activity into the main sequence.
    - Set the WorkBook path to the full path of the **Sample Columns.xlsx** workbook
    - Set the Range to “” so the entire sheet is read
    - In the output parameter, use the shortcut Ctrl+K to create a DataTable called **inputsTable**
- Find and add an **Add Data Column** activity below
    - Set the ColumnName to “C”
    - Set the DataTable parameter to **inputsTable**

- Set the argument type to object

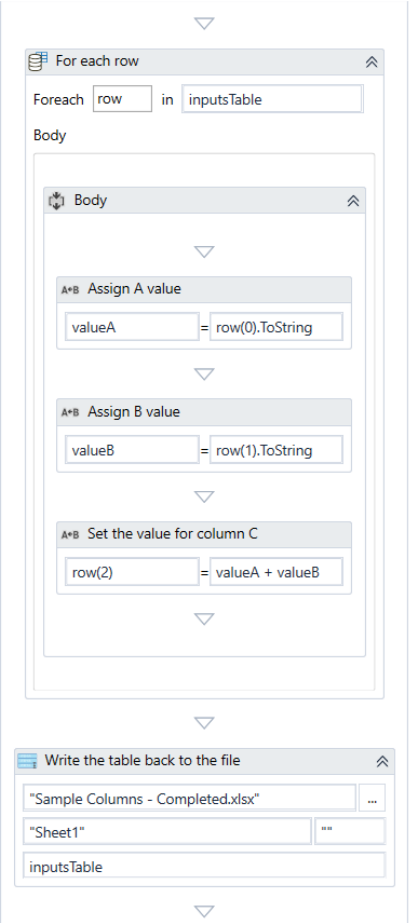


- Find and add a **For Each Row** activity below that
    - Set the activity to loop through **inputsTable**
- Find and add two **Assign** activities (necessary variables should be created with the shortcut):
    - The first one assigns **row(0).ToString** to **valueA**
    - The second one assigns **row(1).ToString** to **valueB**
        - These convert the row object values to more usable string values
- Find and add another **Assign** activity that assigns to **row(2)** this value:
    - **Integer.Parse(valueA) + Integer.Parse(valueB)**
    - This statement converts the string values to integer values using a Visual Basic method and then adds them together
- Lastly, find and add a **Write Range** activity below and outside the **For Each** activity - this will be writing the manipulated DataTable to a new sheet.
    - Set the DataTable to **inputsTable**
    - The sheet name should remain as Sheet1
    - The starting cell should be left blank, as “”

o Set the workbook path to a desired path that ends with the file name **Sample Columns - Completed.xlsx**

- UiPath will create a new file if this one doesn't already exist

- This is what the rest of the completed workflow should look like:



Part C:

This part is mostly a matter of using an Excel command for the rows that need adding. It should be completely contained in an Excel Application Scope.

- Find and add an **Excel Application Scope** activity and add it to the main sequence

o As usual, set the path of **Sample Columns.xlsx**

- Set the visibility option on by checking the box

- Find and add a **Read Range** activity

- The sheet should remain as Sheet1

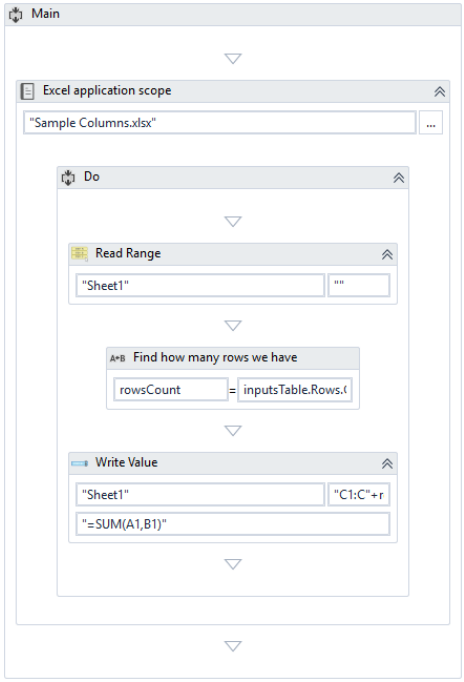
- Set the output to a newly created DataTable called **inputsTable**

Count how many rows there are so the formulas can be applied to the proper section of the sheet.

- Find and add an **Assign** activity below the **Read Range** activity
  - Assign **inputsTable.Rows.Count** to a newly created generic variable called **rowsCount**

- Find and add a **Write Cell** activity, it should be set to:
  - Write on Sheet1
  - Write in the range from “C1:C” + rowsCount
    - This sets the range of rows in Column C to write the formula in
  - Write the value “=SUM(A1,B1)”
    - In Excel, this value will automatically iterate through the descending rows

- This is what the final workflow should look like:



Lesson 9 - Practice 2 Outline

Consolidate Multiple Sheets Into One

A. Create a workflow that adds Sheet1, Sheet2, Sheet3.. SheetN data from an Excel file to a new file called "Consolidated.xlsx"

- All the sheets have the same column headers: Name and Id
- The final sheet should preserve the column headers. Assume all Sheets have the same columns
- N is a variable that can be set to the number of sheets

B. From "Consolidated.xlsx", remove all the rows with the "ID" value of less than 10 and display that sorted data in a message box.

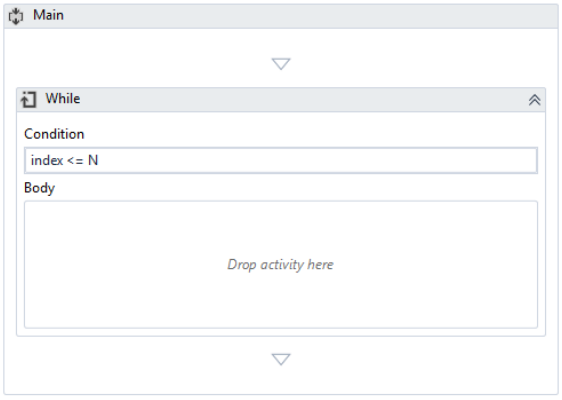
Download **.zip resource** and **extract Excel file**.

Practical Exercise - Walkthrough

Since there are two workflow parts, organize the project as a Flowchart.

A. First create a While loop that the entire automation will be contained in. This will make sure all sheets are indexed through until the end.

- Find and add a **While** activity to the main container
- Create an int32 variable called **Index**
    - Set its default value to 1
- Create an int32 variable called **N**
    - Set its default value to 3 (it can be changed depending on the number of sheets)
- In the properties of the **While** activity:
    - In the condition parameter, write: index <= N
        - This makes sure that as long as the index is less than what **N** is set to, the automation will keep aggregating sheets
- This is what it should look like:



Next, activities need to be added that will read the sheet in each iteration and add them to the main sheet.

- Find and add a **Read Range** activity into the body of the **While** activity

o Set the workbook path to the full path of the sample Excel workbook (downloaded in the outline - previous material)

o Set the sheet name to: “Sheet” + index.ToString

▪ This assumes the workbook sheets will be named Sheet1, Sheet2, …, etc.

o In the output parameter, use the Ctrl-K shortcut to create a DataTable called **partialTable**

If we want the consolidated sheet to have the data from all the sheets, means we have to treat the addition of the first sheet data separately, to add the whole tale, together with the headers. The other sheets data only has to be appended at the end, without adding column headers

- Find and add an **If** activity below the **Read Range** activity

    o Set the condition parameter to: Index = 1

- Find and add a **Write Range** activity to the ‘Then’ branch of the **If** activity

    o Set the workbook path to “**Consolidated**.**xlsx**” - this will create a new workbook with the respective name in the project folder, if it doesn’t already exist.

    o Set the DataTable input to **partialTable**

    o This branch of the **If** will create a workbook with the first sheet as a base

- Find and add an **Append Range** activity to the ‘Else’ branch of the **If** activity

    o Set the workbook path and DataTable input to the same values as the **Write Range** activity

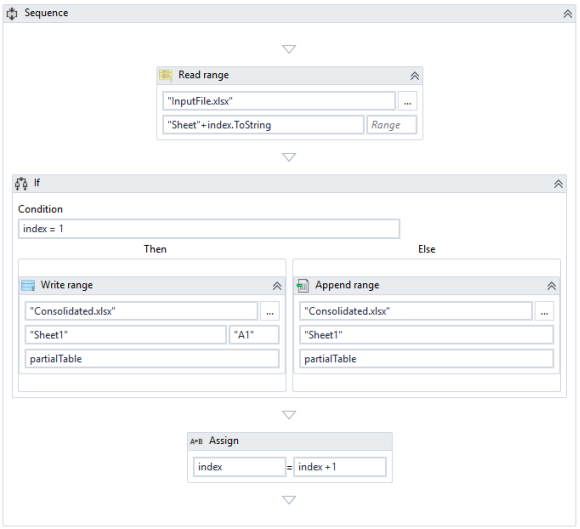
    o This will append the ranged that is read from each of the next sheets to **Consolidated**.**xlsx**

- Finally, add an **Assign** activity below the **If** activity

    o Assign **index + 1** to **index**

- This will increment the loop counter so the sheets will progress from 1 to 2 to 3 and so on…

- This is what the workflow should look like inside the body of the **While** activity:



B.Create a new sequence within the flowchart to filter the data.

- Find and add an **Excel Application Scope** activity to the new container and select the file you created in the previous step, Consolidated.xlsx

- Use a **Read Range** activity to extract the workbook data into a datatable, consolidatedDT.

- Use a **Filter Data Table** activity. Click on the **Filter Wizard**, add **consolidatedDT** as input and output, and select **remove rows**, because we want to get rid of the rows where the ID value is of less than 10.

    - Add a condition for “ID” < 10

- Add an **Output Data Table** activity to the new container, and then add **consolidatedDT** as the input data table. Create a new variable, **stringDT**, for the output text.

- Add a **Message Box** and use **stringDT** as the content.

- When working with Excel files you should work inside the container generated by **Excel Application Scope**, or, better yet, when possible you should use **Workbook** activities because they don't require Excel to be installed and they can work completely in the background.
- The **Visible** property determines if the operations will be performed **internally** or by using Microsoft Excel.
- **Read Range** activity reads a part of an Excel file and stores it in a **DataTable**.
- A **Workbook** is just a reference to an Excel file that can hold many types of data, while a **DataTable** is just a table with rows and columns.
- **Write Range** will overwrite previous entries, while **Append Range** will not, adding the data after the current content instead.
- Both **Read Range** and **Write Range** have the **Add Headers** property that indicates the presence of a first row that contains the column names.

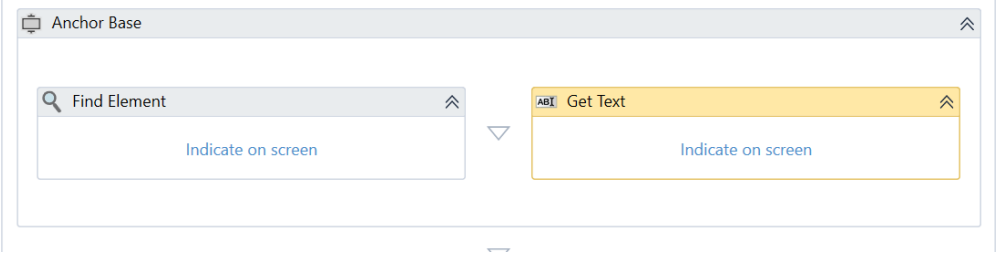
- There are **CSV** activities that are very similar to the Excel ones.
- When creating a **DataTable** with the **Build Data Table** activity, the available data types for a column are spanning all the range of Visual Basic .NET.
- The **Sort Table** and **Filter Table** activities require a table to be defined in Excel.
- An useful tool when working with Excel files is the **Select Range** activity.
- There is a **For Each** correspondent when working with DataTables – **For Each Row**.
- Instead of using an index, if you have headers you can also use column names with the **Get Row Item** activity.

PDF Part 1 - Extracting data from PDF

1. Read pdf Text – Property Range: All or Page Number("3-7").Ignore image part
2. Read Pdf with OCR – It requires a OCR engine.
   - Studio comes with three OCR engines.
   - Google, Microsoft and Abby Fine Reader
   - OCR's are not smart enough to understand the 2-column layout. Abby is an Exception since it can preserve the document structure.
3. Both the above activities can run in background and do not required the pdf to be open.
4. Another option available is Screen Srapping.

PDF Part 1 - Extracting particular data from PDF

1. Get Text.
2. Update selectors in case of multiple file.
3. Use an Anchor Base and Update selector.
   - Find element can be replaced with replaced find Image.
   - Set the zoom of the document to actual size.

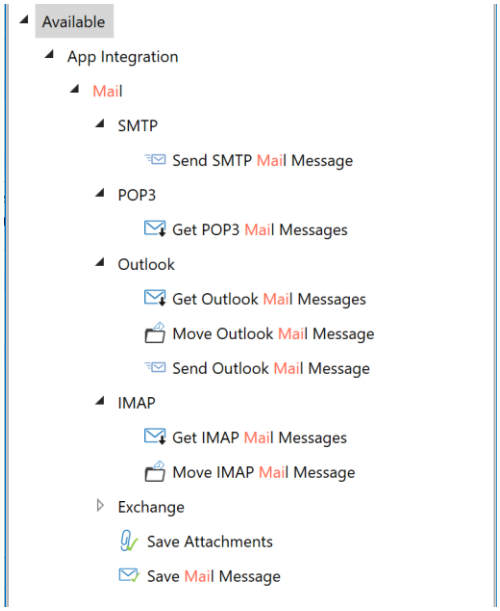


4.Anchor base should have document open and data to be visible.

- You can place PDF activities into 2 categories: one for when processing large chunks of text or whole documents and one for when focusing on specific text elements.
- When looking to extract data from PDF, depending on your file you should choose one of these 2 activities: **Read PDF Text** and **Read PDF With OCR**.
- Both activities can run in the background.
- Another method of grabbing blocks of text is the **Screen Scraping** tool.
- When looking to extract a certain value from PDF files, you can also use **Anchor Base**.

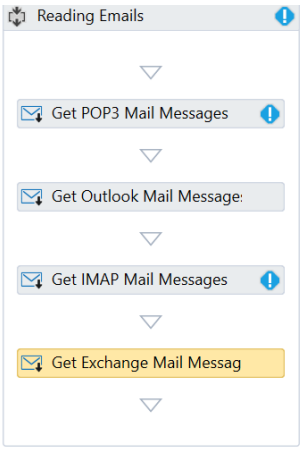
- Use **Read PDF Text** instead of **Read PDF With OCR** when possible since OCR is error prone.
- The **Anchor Base** method can be more reliable than the others since it can handle major structural changes in the file.

Email Automation:



1. SMTP: Simple Mail Transfer Protocol: Used only for sending emails.
2. POP: Post Office Protocol: Old and Obsolete protocol for reading emails but most emails servers support it.
3. IMAP: Internet Message Access Protocol: Used for only reading Messages.
    - Marking Mails as Read
    - Moving Mails to different folders.
4. Exchange: Microsoft Enterprise email solution: Both for sending and receiving emails.
    - Moving Emails between Folders
    - Deleting them.
5. Outlook: Work with the API of the Desktop Application. So, actions already have context.
    - No Need to set up Servers, Users and other technical details.
    - Designed to use the already existing outlook accounts.
6. Finally, 2 generic email activities are available:
    - Save attachment
    - Save Mail Message: To Save to local drives.

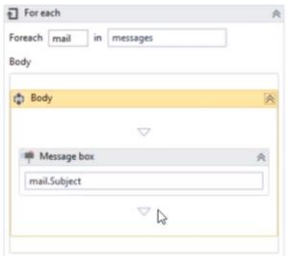
Options to Get emails:



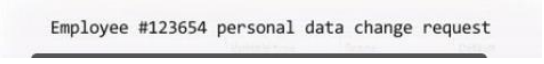
As seen above, POP 3 and IMAP shows validation messages, whereas Outlook and Exchange Don't.

Reason: IMAP and POP3 requires connection parameters whereas, outlook and exchange works with defaults and Autodiscover.

- For the variable created to store email, the type is automatically assigned to List <MailMessages>.
- To Iterate through the list of Mail Messages:
  - Type of each item in the list in foreach should be System.net.mail.
  - Use a message box to display the message subject.(mail.subject).
  -

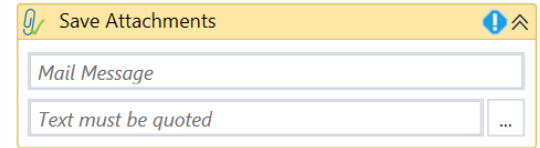


- Filtering an Email based on some known subject and extract the attachment.
  -



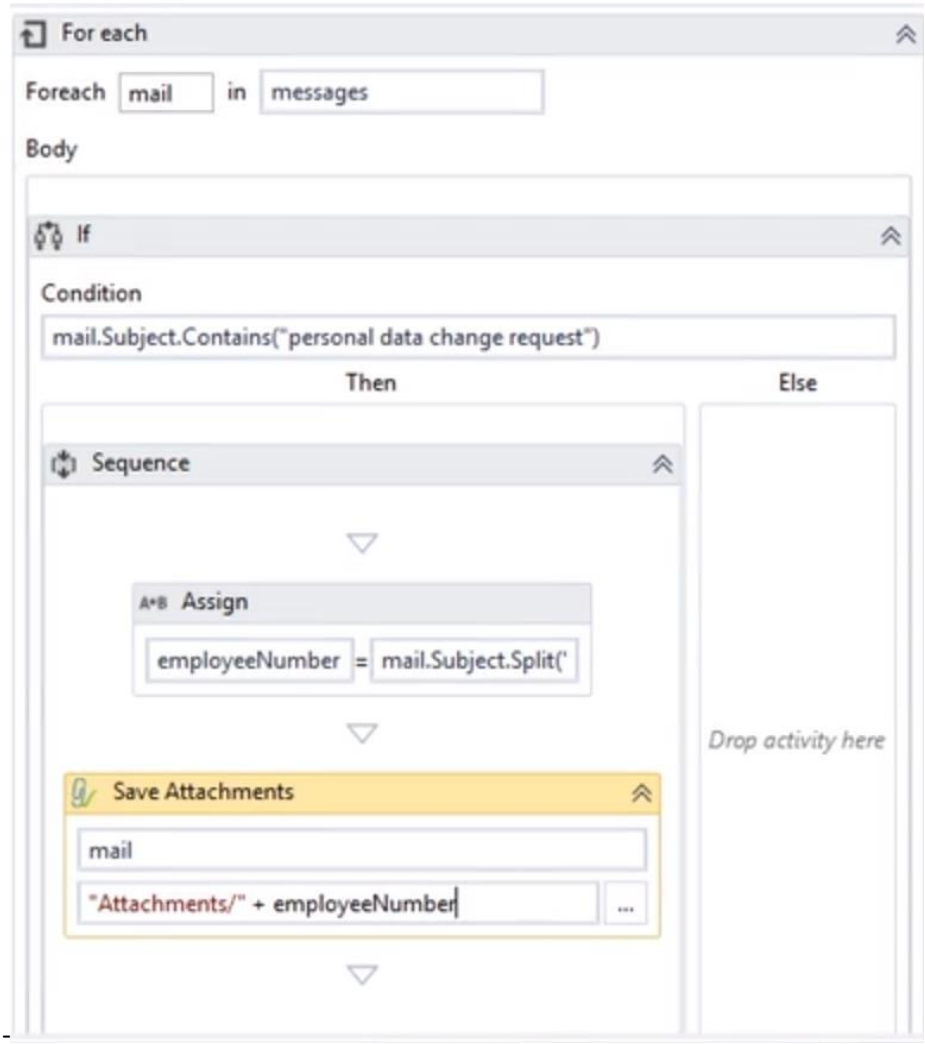
  - Retrieve personal data change request and save attachment folder wise with folder name Employee Id.
  -

- Save attachment activity: To save the attached files.(mail and the folderpath)



- To store the attachments, get the folder names using assign activity.

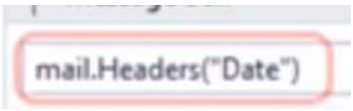
```
mail.Subject.Split("#".ToCharArray)(1).Split(" ".ToCharArray)(0)
```



-

Use getOutlookMailMessages with a custom filter for getting emails received in last 24 hours.

To get TimeStamp of each email.



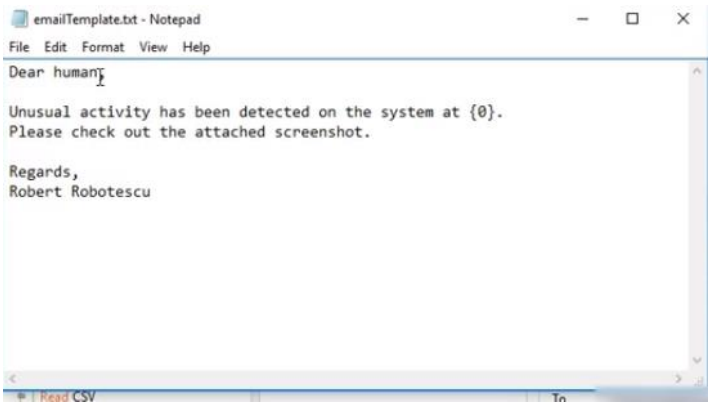
- Drawbacks of Headers: emails has to be retrieved first including attachments making the automation slow.
- Outlook comes with distinct feature that filters the income correspondence based on criteria's like "[subject]" / "[senderEmailAddress]" / ["ReceivedTime"]
- If we only want to process emails received within last 24 hours.

```
"[ReceivedTime] >= "" + Now.AddDays(-1).ToString("MM/dd/yyyy hh:mm tt") +""
```

-

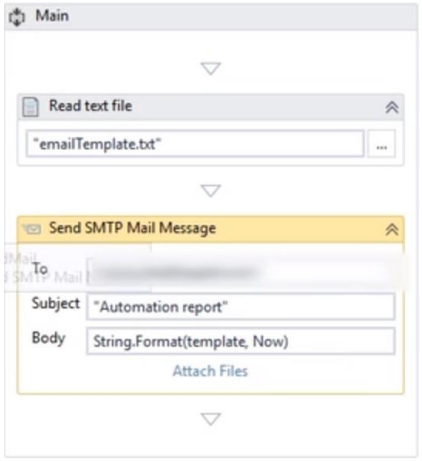
To send error mail with screenshots attached.

1. Read text file : Activity to write mail in template.



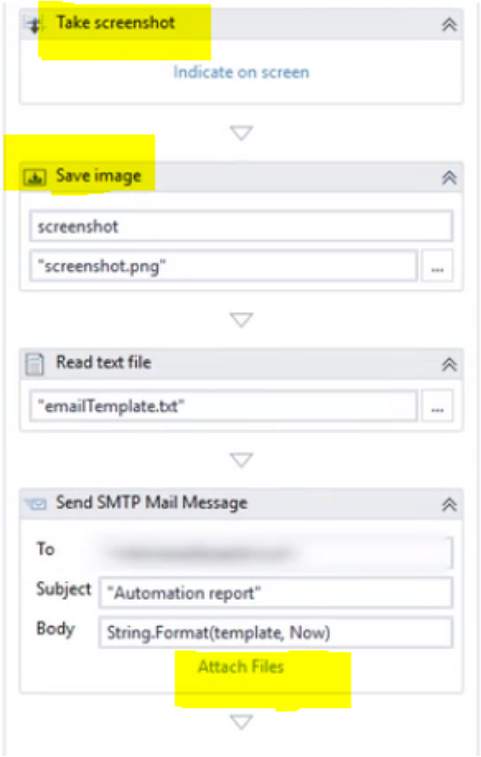
- Load the text in a variable, and use the variable inside the send mail activity.

- Use the string format function to change the placeholders with dynamic content.



-

2. Adding Screenshot to the Mail:



Practical Exercise

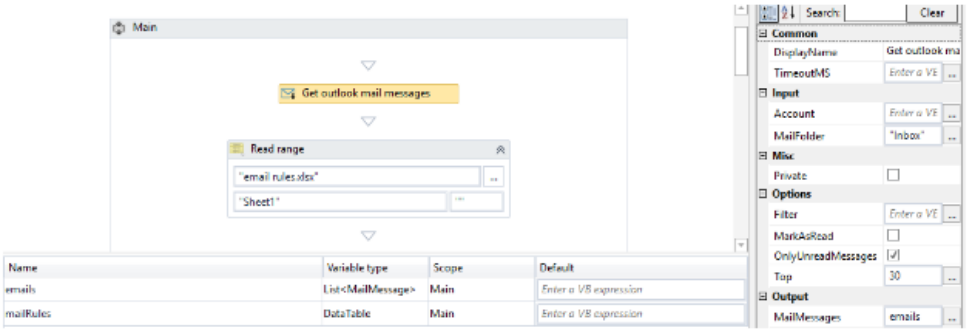
Create a workflow that reads the Inbox and sorts the emails by moving them to various folders, based on "rules" defined in an Excel similar to the one in the attachment.

Practical Exercise - Walkthrough

Depending on what mail server / client you are using, the properties due to be set might be different, however the logic and activities would be similar. When using in this walkthrough Get Outlook Mail Mail Messages, it refers mostly to Get XXX Mail Messages, which XXX could be POP3, IMAP, Exchange, Outlook.

We will start by dragging an activity **Get Outlook Mail Messages**, we configure which account to read from (if we need one which is not the default account in Outlook), the folder to read from and we create a variable in which the emails will be output. Let's call this variable, **emails**

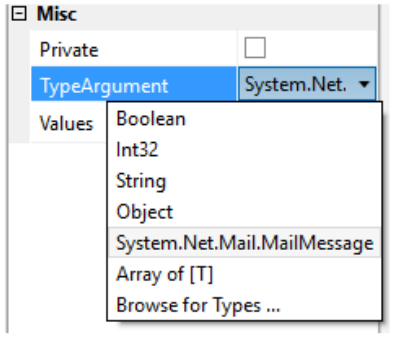
We will also use a **Read Range** activity to read the "rules" from the email rules.xlsx file, provided as input. We will Output the result into a DataTable which we will call **mailRules**



Notice that the **emails** variable, created by us in the output property of Get Outlook Mail Message, is of type *List<MailMessage>*

As we are trying to apply the rules to each email, we will create an iteration on this **emails** list by using a **For each**activity. Considering the list we are iterating consists of *MailMessage* objects, each item from the iteration will be of type *MailMessage* and we need to specify this explicitly.

We click on the **For each** activity and in the property called TypeArgument, we choose from the options in the list, the value System.Net.MailMessage.

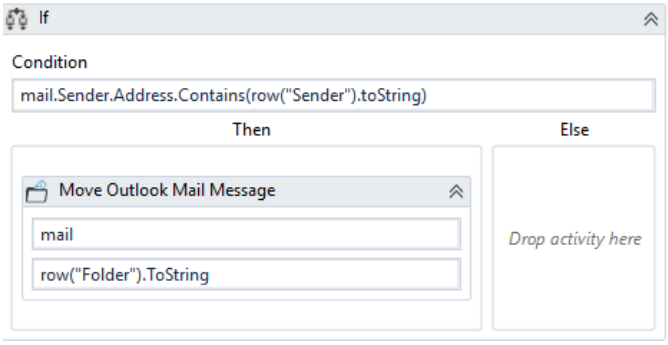


If this option is not on the list, please click on **Browse for Types** and search for the System.Net.Mail.MailMessage in the appearing dialog window.

Now we need to iterate on each of the rules and see if the current mail we are analysing meets any of the criteria. This time we have a DataTable object so we will iterate through it much more comfortably with a **For each row** activity. The only thing you need to specify is the object to iterate on, which in our case is **mailRules**

We now need to check if the rules on the subject apply. As we have access to the current email in the item (we renamed it to mail) variable created automatically and this variable is of type MailMessage, we can access all specific mail properties. We have to see if the Sender.Address property contains the column “From” from the current rule being checked

If that is true, we use a Move Outlook Mail Message and we move the current mail to the folder that is specified in the current row on column “Folder”



We also need to add a break to this sequence, in order not to verify the other rules anymore, if one was already applied and the mail was moved to another folder.

Practical Exercise

Create a workflow that:

- Reads the 6th page of "Session 11 - exercise 2 - UiPathOrchestratorAzureInstallationGuide2016.1" (in the attachment)
- Reads the 2nd page of "Session 11 - exercise 2 - ScannedDoc" (in the attachment)
- Sends an email with both documents attached and with the text from point 1 and text from point 2 as Body

Practical Exercise - Walkthrough

The first pdf document we are trying to extract text from is a native pdf, (the text is selectable, the doc is created from a Word document probably, etc) so we will use the dedicated activity for this case, Read PDF Text

- we click on the dots next to File Name property and browse to the location we have the Orchestrator Installation Guide document saved at
- we replace the **Range** property value from “All” to 6, in order to read just the page 6 from the document
- we create a variable in the Output, to save the retrieved text for later

The second pdf document we a trying to extract from is a scanned pdf (everything is a picture, you cannot select any element, etc) so we will use the dedicated activity for this case, Read PDF With OCR

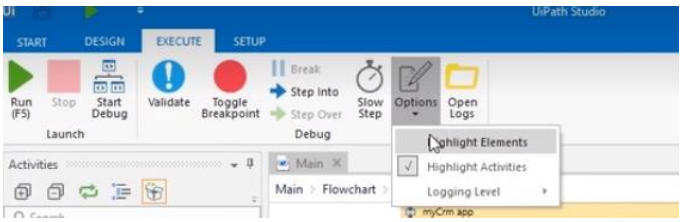
- we click on the dots next to File Name property and browse to the location we have the “Scanned.pdf” document saved at
- we replace the **Range** property value from “All” to 6, in order to read just the page 6 from the document
- we create a variable in the Output, to save the retrieved text for later

- we add a **Send Outlook Mail Message** (can be any Send XXX Mail Message activity)
- set the To property
- set the Subject property
- set the Body property to the concatenated variables that keep the text pieces read from the two documents **installationPDFText + invoicePDFText**

- There are many email activities depending on the protocol you want to use: **SMTP**, **POP3**, **IMAP**, **Exchange** and **Outlook**.
- The get email activities offer mainly the same functionalities:
  - Fetching email from a certain mailbox folder (Outlook, IMAP and Exchange)
  - Fetching only unread messages and marking them as read (Outlook, IMAP and Exchange)
  - Setting a limit for the number of incoming emails
- The **MailMessage** object doesn't directly provide timestamp information, so you will nee to access the "Date" value from the **Headers dictionary**.
- The Outlook activity provides a feature for filtering the incoming messages based on criteria like **Subject** or **ReceivedTime**.
- When loading a template for an email from a file you can use string formatting to populate placeholders with dynamic data.

- **Outlook** and **Exchange** activities are easier to use since they don't require connection parameters.
- If **Outlook** is configured, it's usually your best choice since it requires the least amount of setting up while offering extra features and being compatible with any email protocol.

Debugging and Exceptions



Start Debug: Step by step process execution of workflow.

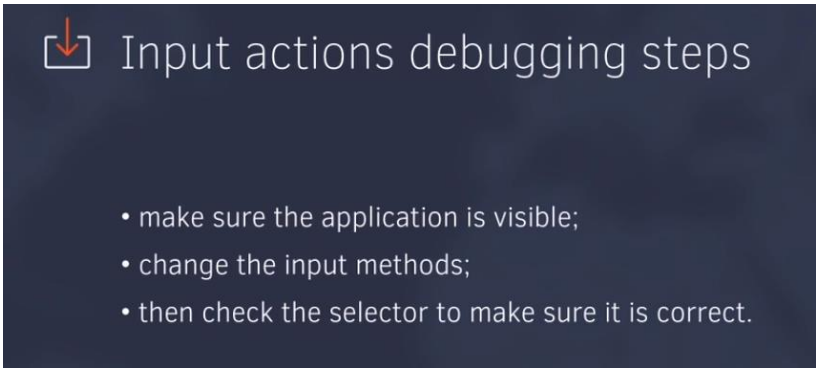
Highlight Element: Works only for input and output.

Properties inspector: Shows current values of all variables.

SlowStep : To slow down the debugging process.

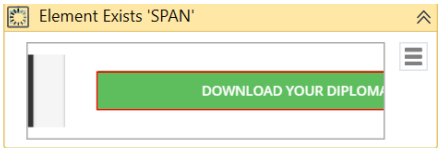
Toggle Breakpoint: (Step Over F10).

Simulate Click : Select multiple activities and update the input methods to simulate type/click to make the workflow work in backround.

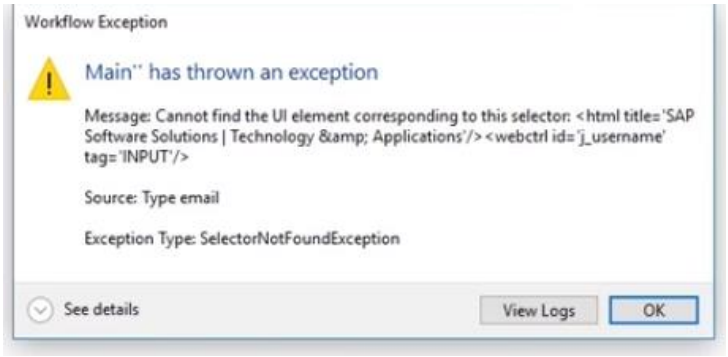


Same we have Image Exists , Find Image , Wait Image vanish.

Element Exists returns – True or False. 3 sec

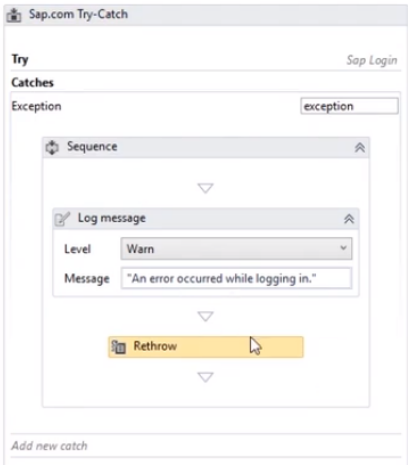


- Find Element stops the automation until an element is found. Default timeout 30 sec.
- Wait Element Vanish : wait for an element to disappear.
- While using try catch: To get the type of the Exception type, look in exception window.



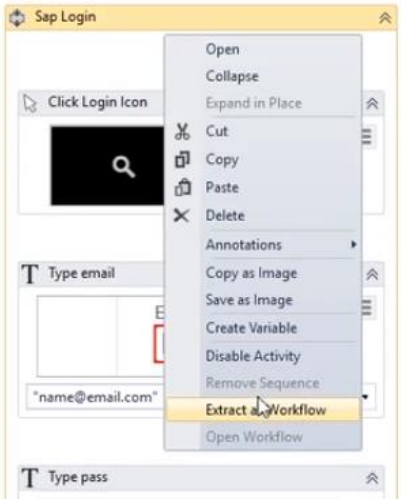
Rethrow Action:

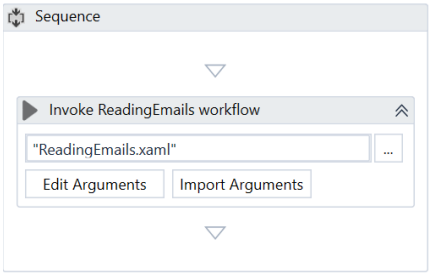
Pop out the occurred Error.



Let's say we are working with a browser automation and the browser itself crashes time to time. In the catch add a new open browser and Log message activities.

Invoke workflow: creates block of reusable code.





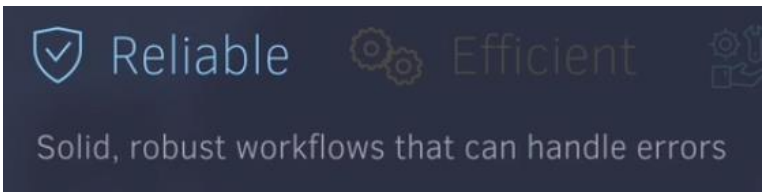
- You can start debugging by pressing the **Start Debug button** from the **Execute tab**.
- When debugging, you can notice 3 things that happen:
  - There is a yellow highlight for the action that is currently executed, and, if suitable, an optional red highlight for the element that is affected by this action
  - The **Locals panel**, where you can check the current value of all the variables
  - A detailed log of all the actions that took place in the workflow
- If you need to slow things down, use the **Slow Step button**, or you can completely pause the execution by using the **Toggle Breakpoint button** and continue the execution step by step by using the **Step Over button**.
- When waiting for an application to load, an activity's default timeout value is 30 seconds, but you can also use activities like **Element Exists**, **Find Element** or **Wait Element Vanish** and their image counterparts.

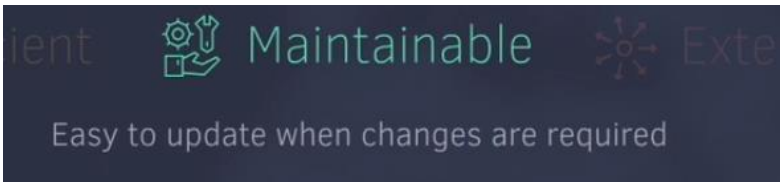
- It's very important to use relevant names for actions and flowcharts, and it pays off on the long term.
- A good strategy if you want to avoid the problems generated by windows that might be on top of the one you want to use is to keep away from the default input method.
- When you are having trouble with the selectors use the **Indicate On Screen** and **Attach to live element** options to "refresh" it.

.

- **Element Exists** doesn't affect your workflow, it just returns a boolean value, whereas the other 2 activities will stop execution until an element is found or it disappeared.
- The **Try Catch** activity should contain the actions that might throw an error inside the **Try** block, the actions to take when an error occurs inside the **Catch** block and, optionally, the actions to always be performed after the other 2 blocks inside the **Finally** block.
- You can have multiple **Catches** for different types of exceptions.
- Even though you caught an exception, you sometimes might want to make sure the workflow actually stops, in that case you can use the **Rethrow** activity.
- You can separate individual components of your automation into different workflow files and then call them using **Invoke Workflow**.

Project Organization Part 1 - Fundamental best practices





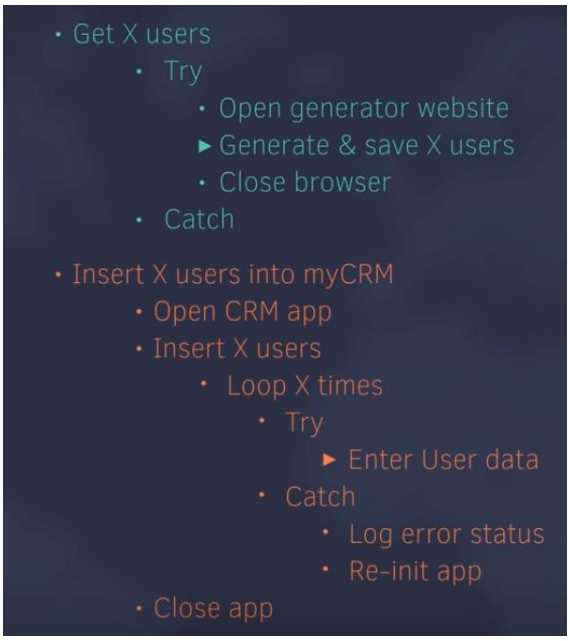
- Pick an appropriate layout for each workflow
  - main WF → flowchart / state machine
  - business logic → flowchart
  - ui interactions → sequence
  - avoid nested IF statements, use flowcharts instead

- Break the whole process in smaller workflows
  - develop and test pieces independently
  - reuse workflows across projects
  - collaborate working on separate files

- Use exception handling
  - put problematic workflows in TryCatch blocks
  - put externally invoked workflows in a T/C
  - implement Recover sequences

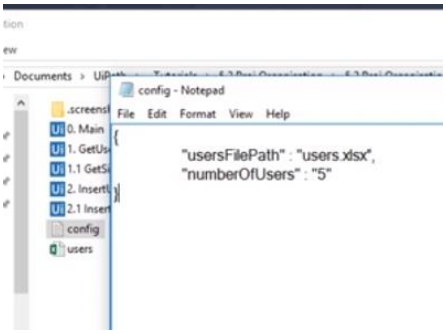
- Make your workflows readable
  - give descriptive names to all components (workflows, activities, variables)
  - leave explanatory notes & comments
  - real time execution progress
  - keep environment settings in a config file

- Keep it clean
  - close applications, windows, web pages

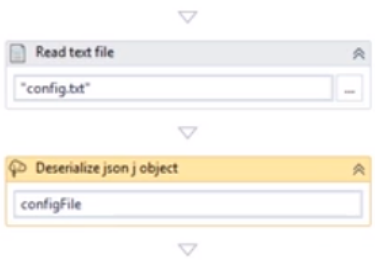


Creating Configurations

1.Inside the project folder make a new text file named as config and use the basic json syntax for configs.

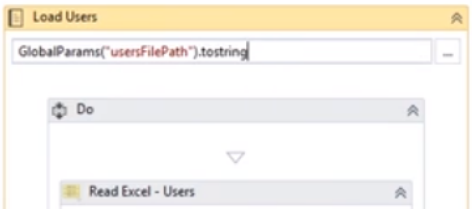


2.

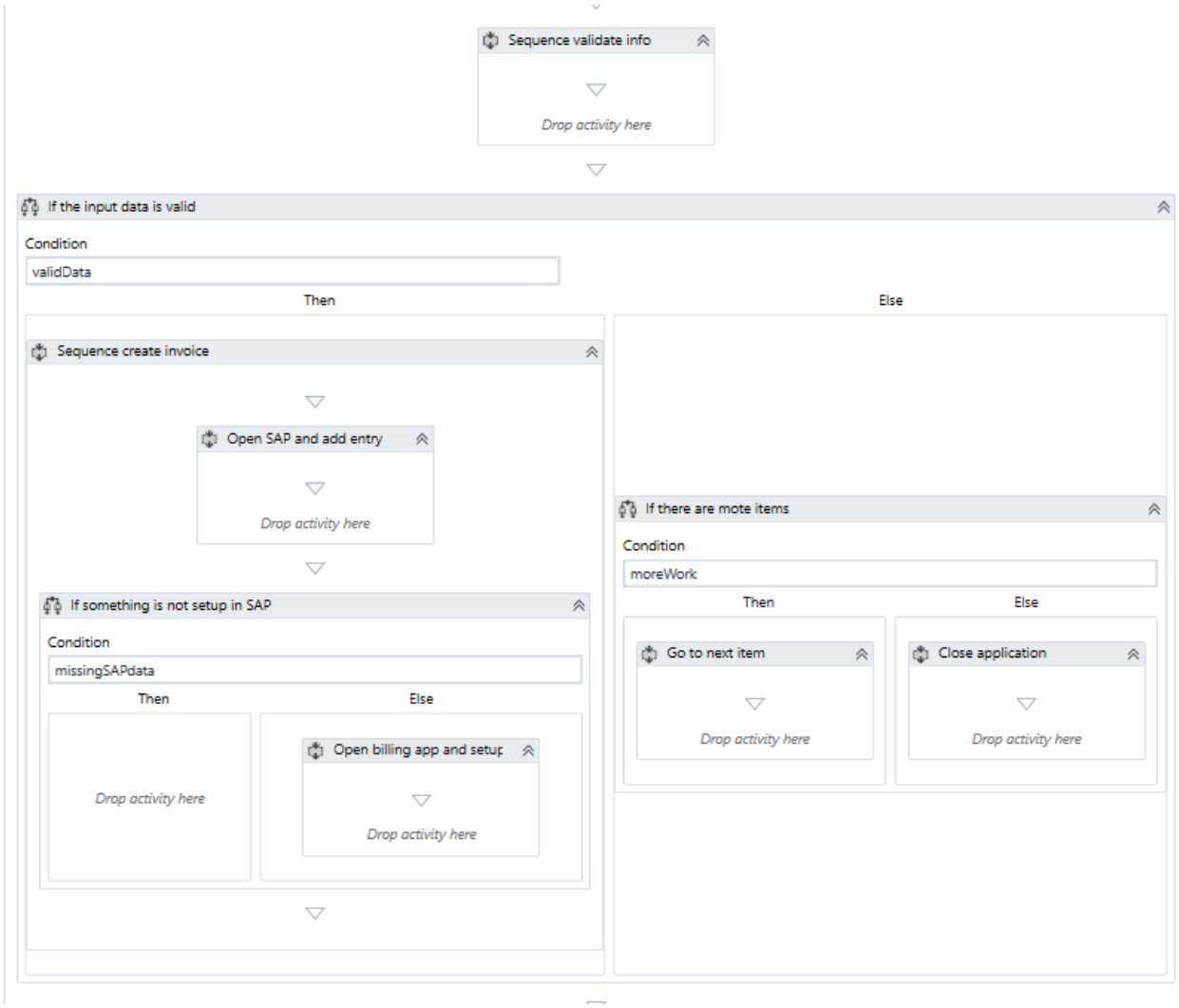


Next use a ReadText and Deserialize json: It takes the information the JsonFile and outputs the object containing all of the variables.

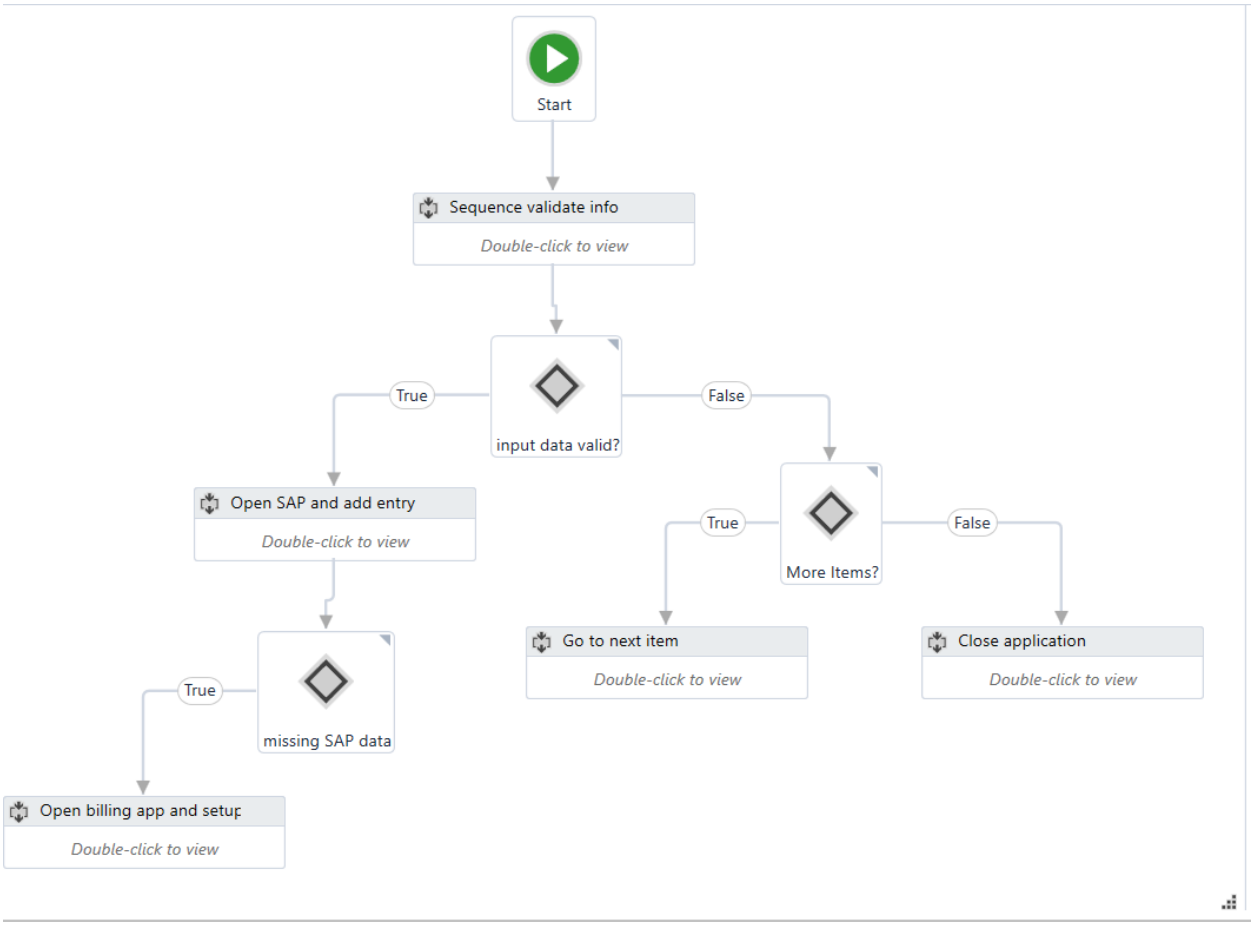
To Use:



BEFORE PROJECT ORGANISATION



AFTER PROJECT ORGANISATION



When developing automation projects, it's best to follow these best practices:

-**Reliability**: Solid and robust workflows that can handle errors and recover gracefully

-**Efficiency**: Maintaining smooth execution while cutting down development time through a variety of methods

-**Maintainability**: In an environment where collaboration and handovers are the way things work, it's important that your project is easy to update

-**Extensibility**: The project needs to be as prepared as possible for the addition of new components

You can select any sequence or flowchart, right click and choose **Extract as Workflow**, thus replacing the selection with an **Invoke Workflow** activity, essentially turning it into programming function, with the parameters being arguments for the workflow.

Make sure you take some time to pick the appropriate layout for each workflow:

-Main: flowchart or state machine

-Business logic: flowchart

-UI interactions: sequence

-Avoid nested IFs by using flowcharts

It's good to break your process into smaller workflows:

-Develop and test pieces independently

-Reuse workflows

-Collaborate more efficiently by working on separate files

Always handle exceptions:

-Place exception prone workflows into Try Catch blocks

-Same goes for externally invoked workflows

-Setup recover sequences

Make sure your workflows are readable:

-Choose descriptive names for all components

-Use explanatory notes and comments

-Log real time execution progress

-Place environment settings in a config file

Always keep things clean by closing the applications when they are no longer needed.