UiPath Studio Workflow Automation Manual

Table of Contents

Basic Workflows

1. Iterating Over an Array and Displaying Values

Objective

To create a workflow in UiPath Studio that initializes an array, iterates through its elements, and displays each value using a message box.

Tools Required

- UiPath Studio

Workflow Components

1. **Sequence**: Structures the workflow logically.
2. **Assign Activity**: Initializes the array.
3. **For Each Activity**: Loops through the array elements.
4. **Message Box Activity**: Displays each element.

Steps to Perform the Experiment

Step 1: Open UiPath Studio

1. Launch UiPath Studio on your machine.
2. Create a new project or open an existing one.

Step 2: Create a New Sequence

1. Navigate to **New File > Sequence** and name it `Main`.
2. Drag a **Sequence** container to the Designer Panel.

Step 3: Configure Variables

1. Open the **Variables Panel** at the bottom of UiPath Studio.
2. Add the following variable:
    - **Name**: `arrayVar`
    - **Type**: `Integer[]`
    - **Default Value**: `New Integer(){1,2,3,4,5}`

Step 4: Initialize the Array

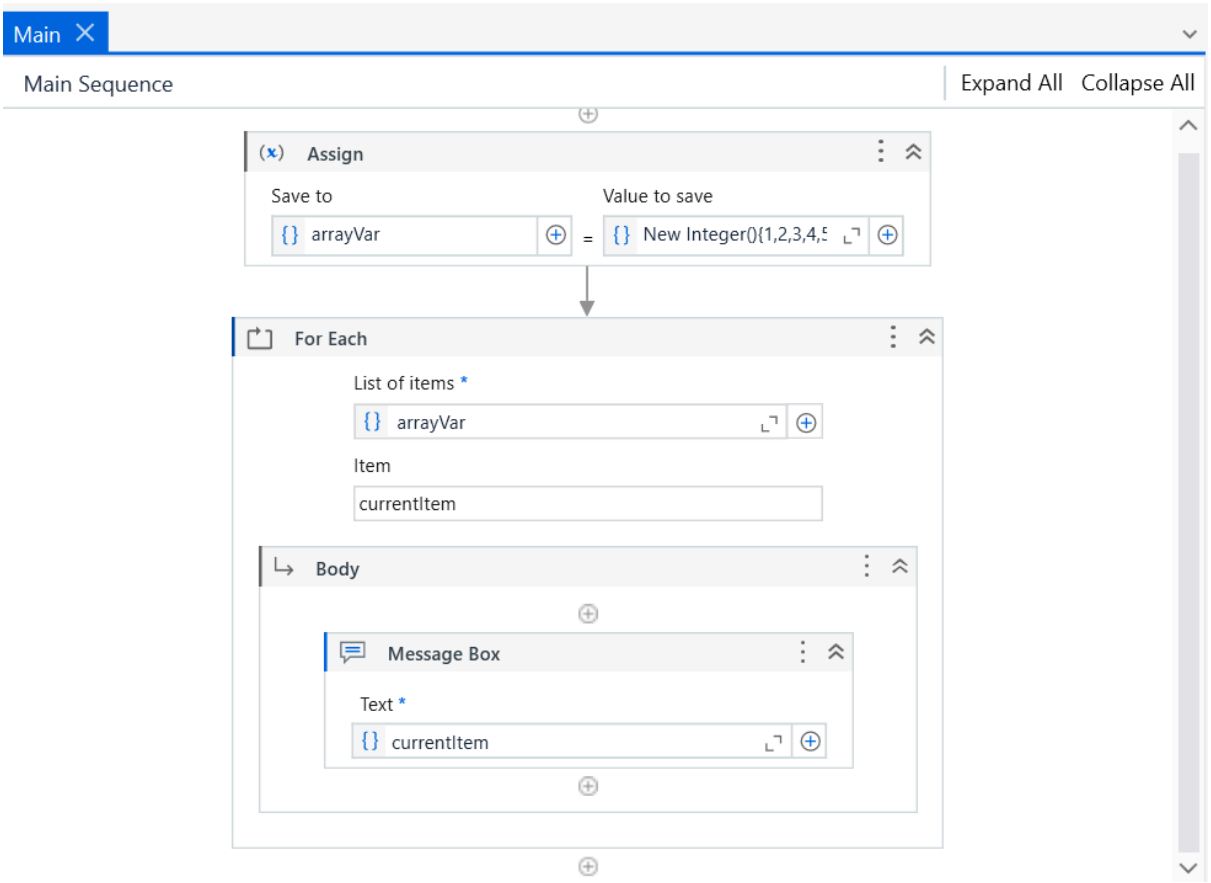
1. Drag an **Assign Activity** into the sequence.
2. Configure it as:
    - **To**: `arrayVar`
    - **Value**: `New Integer(){1,2,3,4,5}`

Step 5: Add a For Each Activity

1. Drag a **For Each** activity below the Assign activity.
2. Configure it as:
    - **Values**: `arrayVar`
    - **TypeArgument**: `Integer` (Set from the Properties Panel)

Step 6: Display Each Array Value

1. Drag a **Message Box** activity into the body of the For Each loop.
2. Configure its **Text** property to: `[currentItem.ToString()]`



Final Workflow Overview

1. **Sequence**: Contains the entire logic.
2. **Assign Activity**: Initializes the array.
3. **For Each Loop**: Iterates over the array elements.
4. **Message Box**: Displays each array value in a message box during iteration.

Expected Output

The workflow iterates through the array and displays:

- 1
- 2
- 3
- 4
- 5

Each value appears in a separate message box.

Error Handling

- **Null Array**: Ensure the array is initialized before the For Each loop.
- **Runtime Errors**: Use **Try-Catch** blocks to handle unexpected exceptions.
- **Debugging**: Run the workflow in **Debug Mode** to inspect the loop execution.

2. Calculating Factorial and Using an Application

Objective

To calculate the factorial of a user-provided number using a While loop in UiPath and interact with the Snipping Tool application.

Tools Required

- UiPath Studio
- Windows Snipping Tool Application

Workflow Components

1. **Input Dialog**: Collects user input.
2. **Assign Activities**: Initializes variables.
3. **While Loop**: Performs the factorial calculation.
4. **Message Box**: Displays the factorial result.
5. **Use Application/Browser**: Launches and interacts with the Snipping Tool.

Steps to Perform the Experiment

Step 1: Open UiPath Studio

1. Launch UiPath Studio.
2. Create a new project or open an existing one.

Step 2: Create a New Sequence

1. Navigate to **New File > Sequence** and name it `Main`.
2. Drag a **Sequence** container into the Designer Panel.

Step 3: Configure Variables

1. Open the **Variables Panel** at the bottom of UiPath Studio.
2. Add the following variables:

Name	Type	Scope	Default Value

`inputVar`	Integer	Sequence	None
`factVar`	Integer	Sequence	None
`copyInput`	Integer	Sequence	None

Step 4: Create Input Dialog for User Input

1. Drag an **Input Dialog** activity into the sequence.
2. Configure the properties:
    - **Title**: `"Enter Number"`
    - **Label**: `"Enter number to calculate factorial:"`
    - **Result**: `inputVar`

Step 5: Initialize Variables

1. Drag an **Assign Activity** below the Input Dialog.
    - **To**: `factVar`
    - **Value**: `1`
2. Add another Assign activity:
    - **To**: `copyInput`
    - **Value**: `inputVar`

Step 6: Calculate Factorial Using a While Loop

1. Drag a **While** activity below the Assign activities.
2. Configure its **Condition** property to: `inputVar > 0`.
3. Inside the While loop, add a **Sequence** and configure the following:
    - **Multiply factVar**:
        - Drag an **Assign Activity** into the Sequence.
        - **To**: `factVar`
        - **Value**: `factVar * inputVar`
    - **Decrement inputVar**:
        - Drag another Assign Activity.
        - **To**: `inputVar`
        - **Value**: `inputVar - 1`

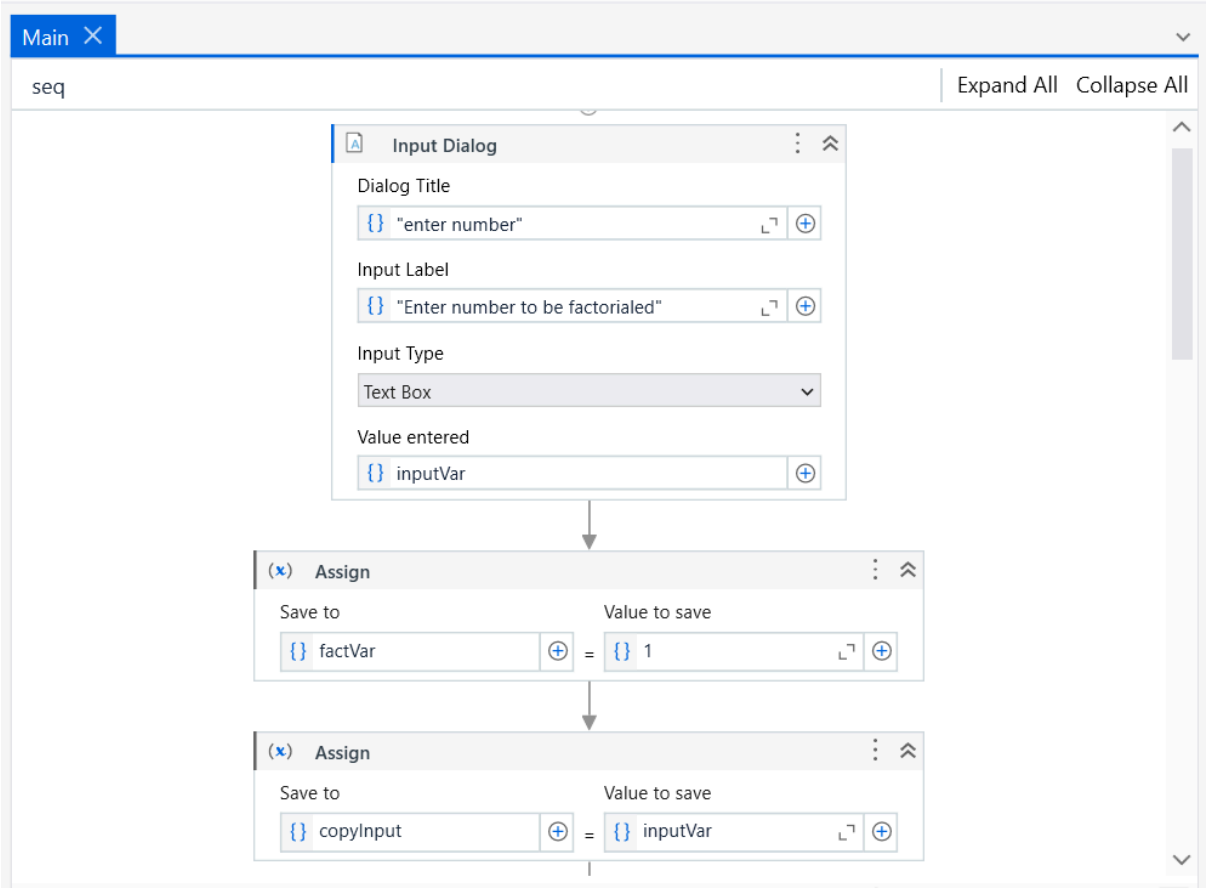
Step 7: Display the Factorial Result

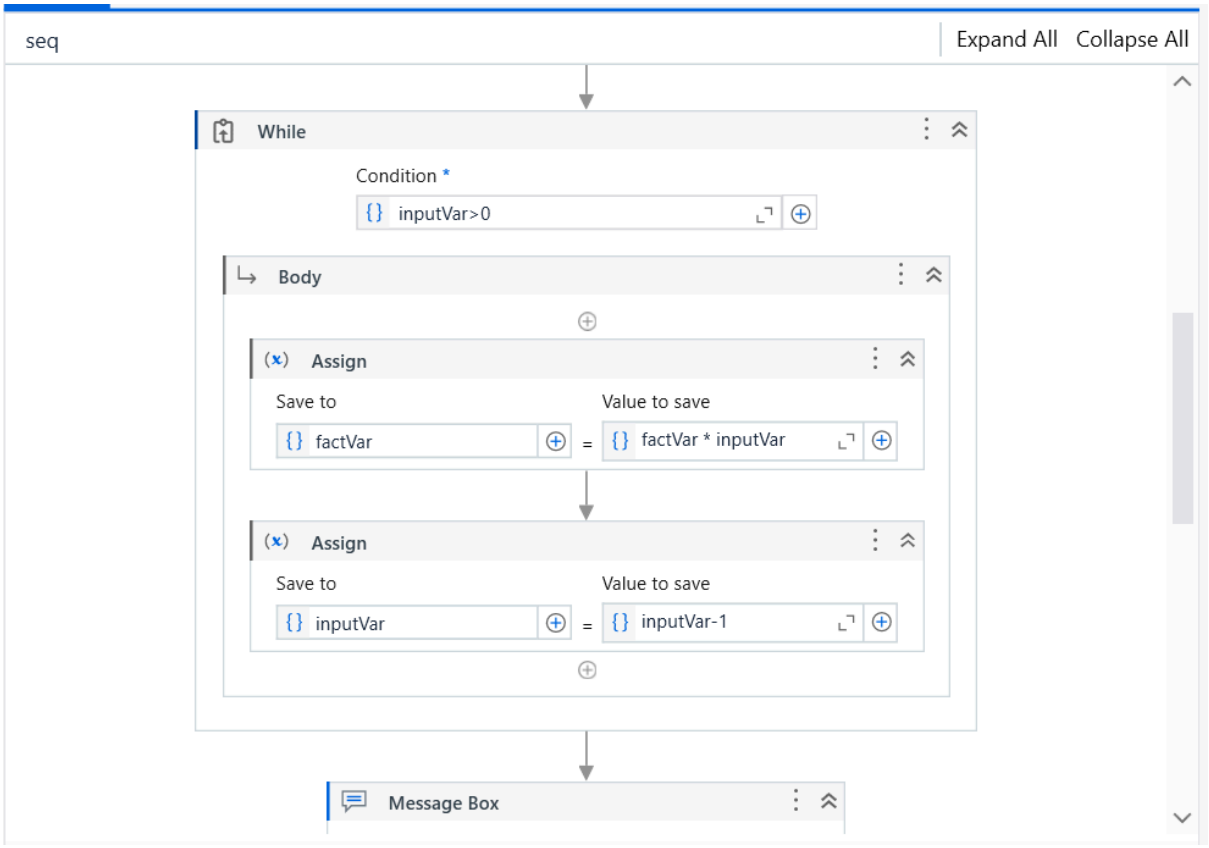
1. Drag a **Message Box** activity below the While loop.

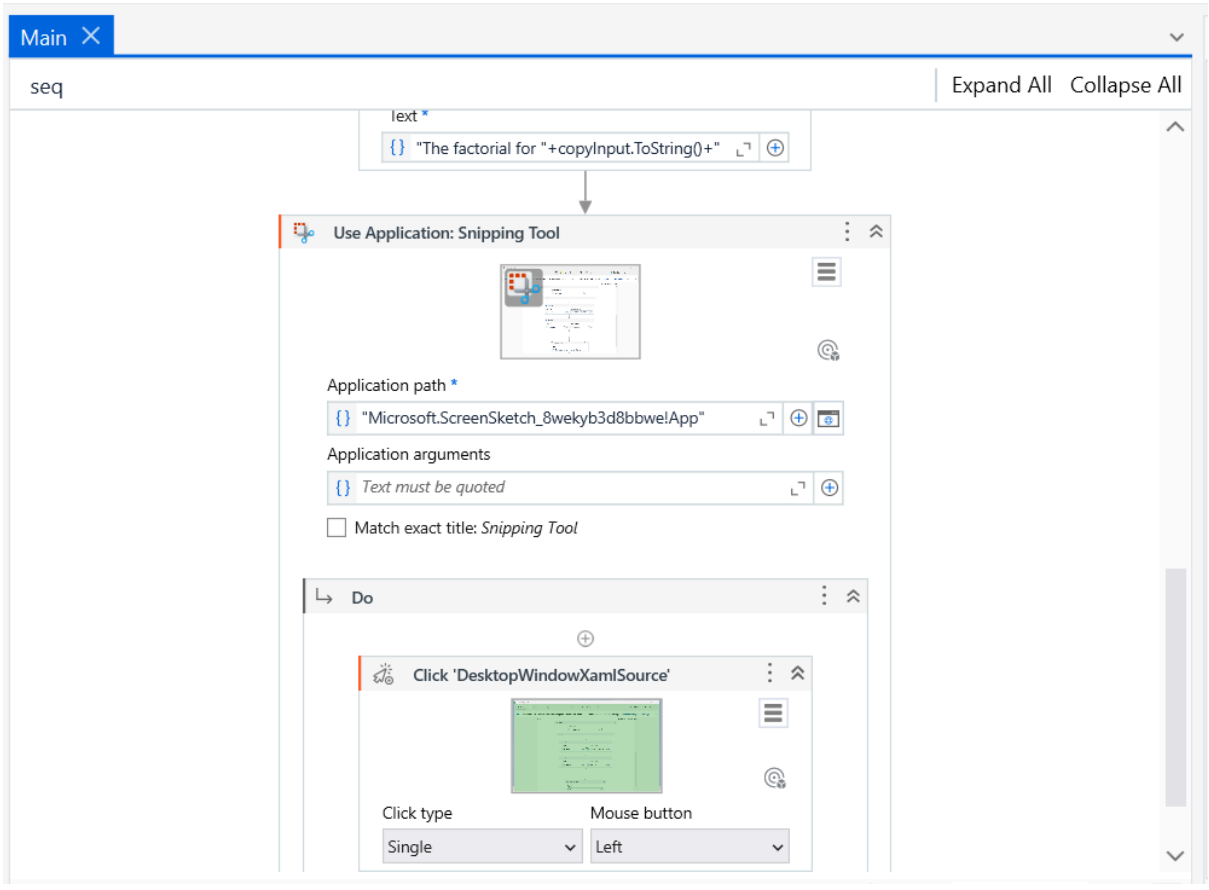
Configure its **Text** property to: `"The factorial of " + copyInput.ToString() + " is " + factVar.ToString()`

Step 8: Interact with the Snipping Tool Application

1. Drag a **Use Application/Browser** activity below the Message Box.
2. Configure its properties:
   - **Application**: Snipping Tool
   - **Attach Mode**: ByInstance
3. Inside the Do container of the Use Application activity:
   - Drag a **Click Activity**.
   - Use UiPath's recorder or manually indicate a UI element on the Snipping Tool (e.g., a button or window).







Final Workflow Overview

1. **Input Dialog**: Collects a number from the user.
2. **Variable Initialization**: Sets initial values for factorial calculation.
3. **While Loop**: Calculates the factorial by decrementing the input and multiplying the result.
4. **Message Box**: Displays the calculated factorial.
5. **Snipping Tool Interaction**: Opens and interacts with the Snipping Tool.

Expected Output

1. The user enters a number in the Input Dialog.
2. The workflow calculates the factorial using the While loop.
3. The factorial result is displayed in a Message Box.
4. The Snipping Tool application opens, and the specified action is performed (e.g., clicking a button).

Error Handling

- **Input Errors**:
  - Use input validation to ensure the user enters a positive integer.
  - Add a **Try-Catch Block** to handle non-integer inputs.
- **Factorial Calculation**:
  - Ensure `inputVar` is properly initialized before entering the While loop.
  - Handle edge cases like 0 factorial (result should be 1).
- **Snipping Tool Issues**:
  - Add error handling for scenarios where the Snipping Tool is not installed or the indicated element is missing.

3. Trigger Notification Every 5 Seconds

Objective

To create a UiPath workflow that continuously displays a notification every 5 seconds using a While loop.

Tools Required

- UiPath Studio

Workflow Components

1. **While Activity**: Executes an infinite loop.
2. **Message Box**: Displays the notification message.
3. **Delay Activity**: Pauses execution for 5 seconds between notifications.

Steps to Perform the Experiment

Step 1: Open UiPath Studio

1. Launch UiPath Studio.
2. Create a new project or open an existing one.

Step 2: Create a New Sequence

1. Navigate to **New File > Sequence** and name it `Main`.
2. Drag a **Sequence** container into the Designer Panel.

Step 3: Add a While Activity

1. Drag a **While** activity from the Activities Panel into the Sequence container.
2. Configure its **Condition** property to: `True`.
    - This creates an infinite loop, continuously executing the body of the While activity.

Step 4: Add Activities Inside the While Loop

1. **Message Box Activity**
    - Drag a **Message Box** activity into the body of the While activity.

Configure its **Text** property to:
arduino
Copy code

```
"Notification: This message is displayed every 5 seconds."
```

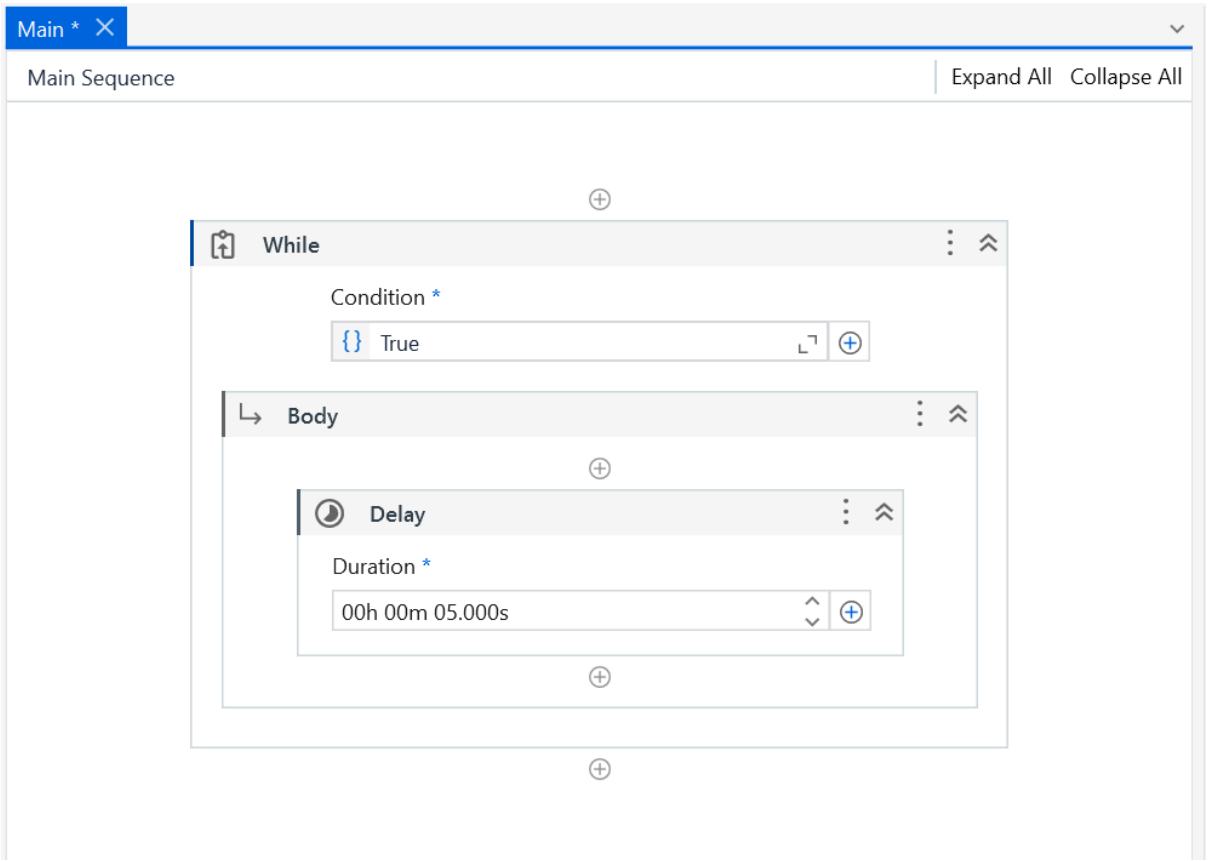
    -
2. **Delay Activity**
    - Drag a **Delay** activity below the Message Box in the While loop.

Configure its **Duration** property to:`00:00:05`

    - This introduces a 5-second pause before the next iteration of the loop.

Step 5: Save and Run the Workflow

1. Save the project by clicking the **Save** icon.
2. Click **Run** to execute the workflow.
    - The process will display a notification message every 5 seconds in a loop.



Final Workflow Overview

1. **While Activity**:
    - Executes its body repeatedly while the condition is `True`.
    - In this case, the condition is always `True`, creating an infinite loop.
2. **Message Box**:
    - Displays a notification with the message: `"Notification: This message is displayed every 5 seconds."`
3. **Delay**:
    - Introduces a 5-second pause between consecutive notifications to avoid continuous execution without breaks.

Expected Output

- The workflow displays a notification message every 5 seconds, indefinitely.

Error Handling

1. **Infinite Loop Termination**:
    - The While loop runs indefinitely because the condition is always `True`.
    - To stop the loop, use the **Stop** button in UiPath Studio or close the application running the workflow.
2. **Message Box Issues**:
    - If the message box does not appear, ensure UiPath Studio has permissions to display notifications.
3. **System Performance**:
    - An infinite loop can consume resources over time. Use this workflow for controlled testing purposes only.

4. Conditional Input Handling with UiPath

Objective

To create a workflow that processes numeric input and displays a message based on conditional logic:

- Checks if the input is greater than 10.
- Checks if the input is equal to 10.
- Handles other cases with appropriate messages.

Tools Required

- UiPath Studio

Workflow Components

1. **Input Dialog**: Prompts the user for a numeric input.
2. **If Activity**: Evaluates if the input is greater than 10.
3. **IfElseIf Block**: Handles the condition where the input equals 10 or other cases.
4. **Message Box**: Displays the result based on the condition evaluated.

Steps to Perform the Experiment

Step 1: Open UiPath Studio

1. Launch UiPath Studio.

2. Create a new project or open an existing one.

Step 2: Create a New Sequence

1. Navigate to **New File > Sequence** and name it `Main`.
2. Drag a **Sequence** container into the Designer Panel.

Step 3: Configure Variables

1. Add a variable `strVar` of type **Int32** to store the numeric input.

Step 4: Add an Input Dialog Activity

1. Drag the **Input Dialog** activity into the sequence.
2. Configure its properties:
    - **Label**: `"Please enter the number"`
    - **Title**: `"Input number"`
    - **Result**: `strVar`

Step 5: Add an If Activity

1. Drag an **If** activity below the Input Dialog.

Set its **Condition** to:`[strVar > 10]`

2. In the **Then** block:

Add a **Message Box** activity with the text:`[strVar.ToString() + " is greater than 10"]`

3. In the **Else** block:
    - Add an **IfElseIf** activity to handle other conditions.

Step 6: Configure the IfElseIf Activity

1. Inside the Else block, add an **IfElseIf** activity.

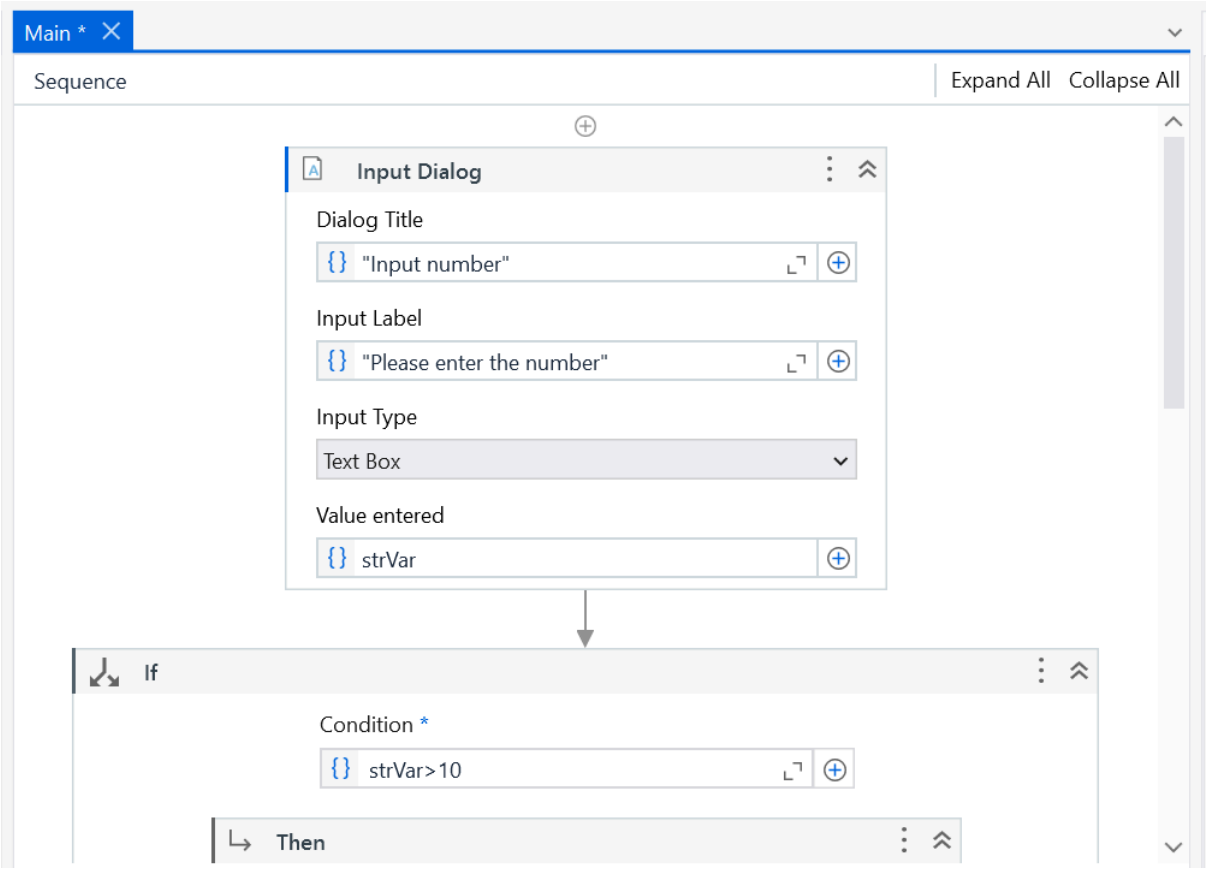
Configure the first condition to:`[strVar = 10]`

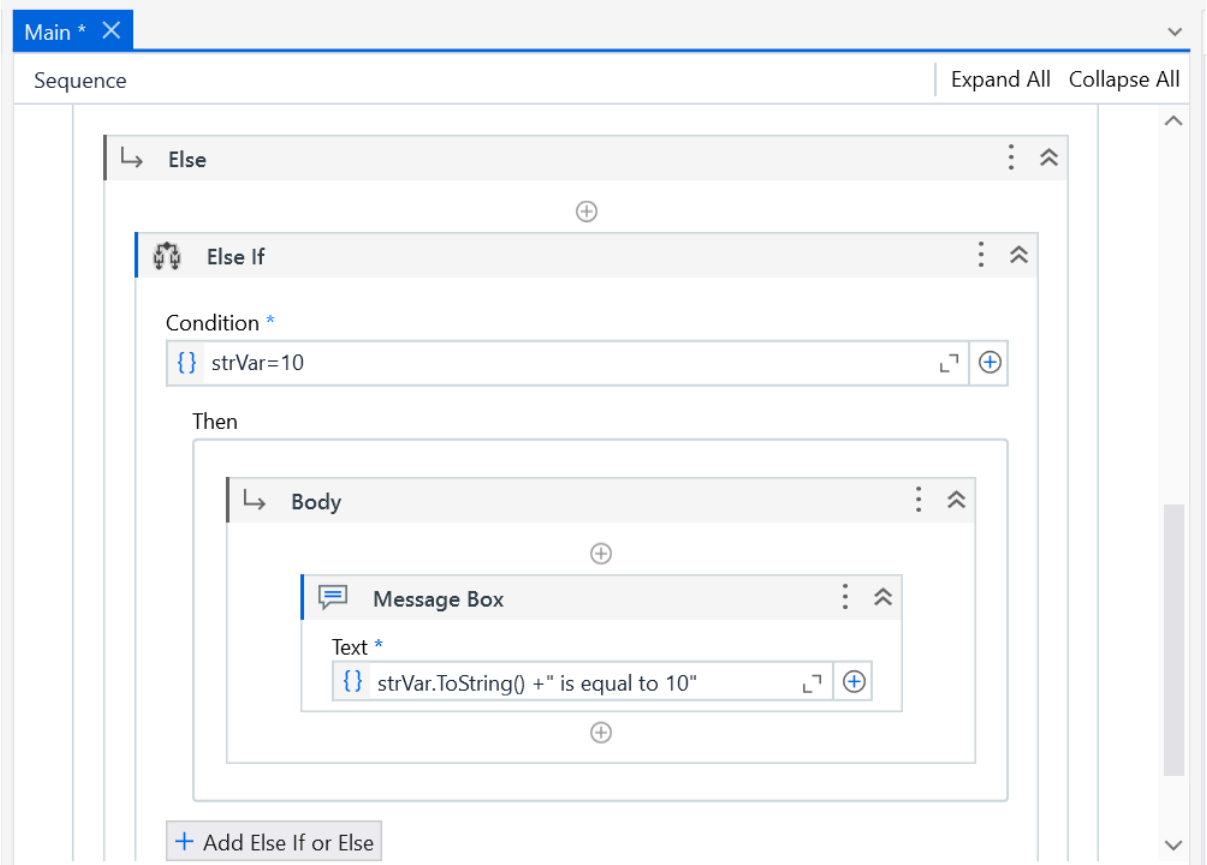
2. In the **Then** block:

Add a **Message Box** activity with the text: `[strVar.ToString() + " is equal to 10"]`

3. In the **Else** block:

Add a **Message Box** activity with the text: `[strVar.ToString() + " is less than 10"]`





Final Workflow Overview

1. **Input Dialog**:
   - Prompts the user to enter a number and stores the value in `strVar`.
2. **If Activity**:
   - Checks if the number is greater than 10.
   - Displays `"Number is greater than 10"` if the condition is true.
3. **IfElseIf Block**:
   - Checks if the number equals 10 and displays `"Number is equal to 10"`.
   - Handles cases where the number is less than 10 and displays `"Number is less than 10"`.

Expected Output

- If the user enters a number greater than 10: Displays `"Number is greater than 10"`.
- If the user enters 10: Displays `"Number is equal to 10"`.
- If the user enters a number less than 10: Displays `"Number is less than 10"`.

Error Handling

1. **Input Errors**:
    - If the user enters non-numeric input, the workflow may throw an error.
    - Use Try-Catch blocks to handle exceptions and prompt the user again.
2. **Validation**:
    - Add validation logic to ensure only numeric input is processed.

5. Dynamic Grade Classification Using UiPath

Objective

To create a workflow that classifies a student's grade based on the selected score range using the **Switch** activity in UiPath. The workflow will prompt the user to select a grade range and display the corresponding grade classification.

Tools Required

- **UiPath Studio**

Workflow Components

1. **Input Dialog**: Prompts the user to select a grade range from predefined options.
2. **Switch Activity**: Evaluates the user's selection and determines the grade classification.
3. **Message Box**: Displays the grade classification based on the user's selection.

Steps to Perform the Experiment

Step 1: Open UiPath Studio

1. Launch **UiPath Studio**.
2. Create a new project or open an existing one.

Step 2: Create a New Sequence

1. Navigate to **New File > Sequence** and name it `Main`.
2. Drag a **Sequence** container into the **Designer Panel**.

Step 3: Configure Variables

1. Open the **Variables Panel** at the bottom of UiPath Studio.
2. Add the following variable:

Name	Type	Scope	Default Value
`userOption`	String	Sequence	None

Step 4: Add an Input Dialog Activity

1. Drag an **Input Dialog** activity into the sequence.
2. Configure its properties:
   - **Label**: `"Please select the range"`
   - **Title**: `"Select range"`
   - **OptionsString**: `"85-100;70-84;60-69;<60;<50;"`
   - **Result**: `userOption`
3. This activity will prompt the user to select one of the provided grade ranges.

Step 5: Add a Switch Activity

1. Drag a **Switch** activity below the Input Dialog.
2. Set the **Expression** property to: `[userOption]`
3. Set the **TypeArgument** to `String`.

Step 6: Configure Cases in the Switch Activity

1. **Case `"85-100"`**:
   - Add a **Message Box** activity.
   - Set the **Text** property to: `"Student is A grade"`
2. **Case `"70-84"`**:
   - Add a **Message Box** activity.
   - Set the **Text** property to: `"Student is B grade"`
3. **Case `"60-69"`**:
   - Add a **Message Box** activity.
   - Set the **Text** property to: `"Student is C grade"`
4. **Case `"<60"`**:
   - Add a **Message Box** activity.
   - Set the **Text** property to: `"Student is D grade"`
5. **Case `"<50"`**:
   - Add a **Message Box** activity.
   - Set the **Text** property to: `"Student has Failed"`
6. **Default**:

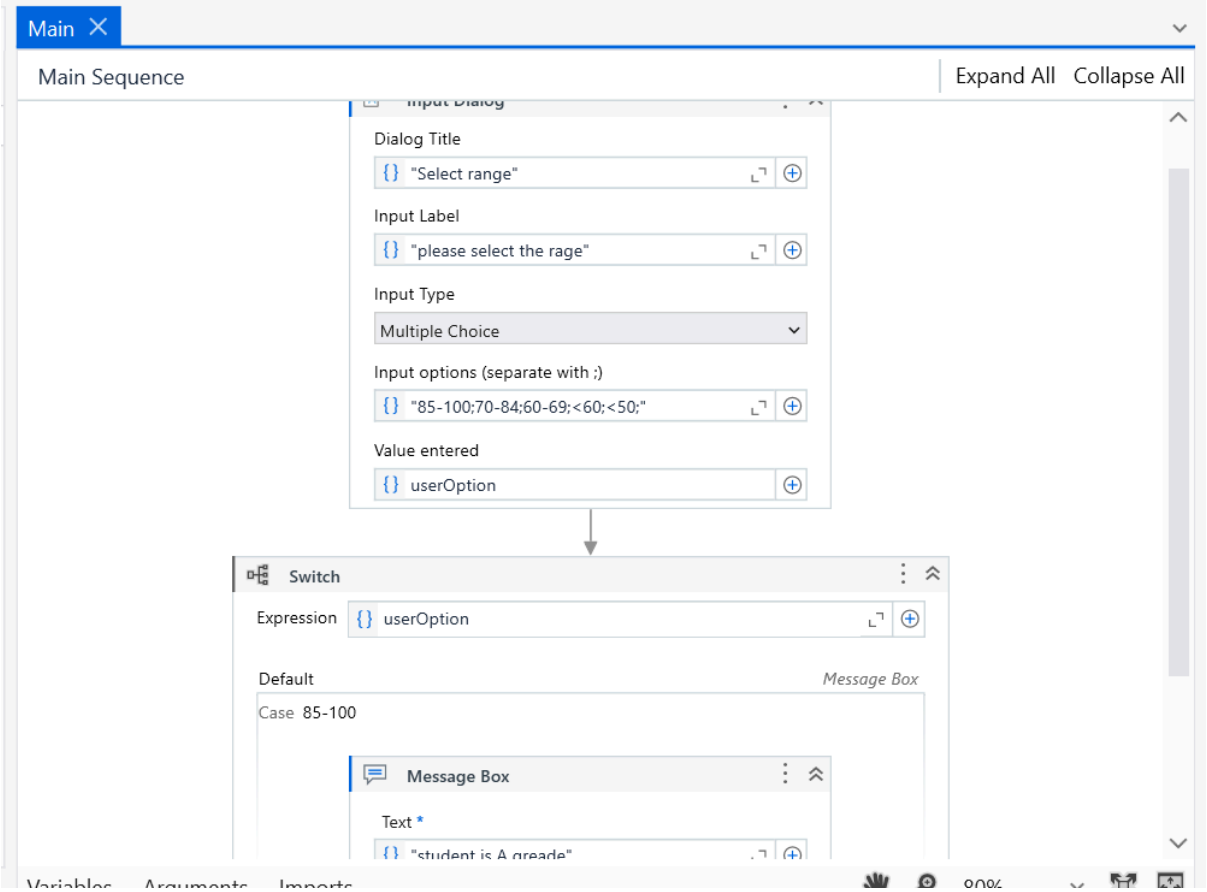
    - Add a **Message Box** activity.
    - Set the **Text** property to: `"Invalid selection"`
7. Ensure that the **Case** keys exactly match the options provided in the **Input Dialog**'s **OptionsString**.

Step 7: Save and Run the Workflow

1. Save the project by clicking the **Save** icon or pressing **Ctrl + S**.
2. Click the **Run** button to execute the workflow.

Final Workflow Overview

1. **Input Dialog**:
    - Prompts the user to select a grade range from the options.
    - Stores the selection in the variable `userOption`.
2. **Switch Activity**:
    - Evaluates the value of `userOption`.
    - Executes the corresponding **Case** based on the user's selection.
3. **Message Box**:
    - Displays the grade classification message associated with the selected range.



Expected Output

- **If the user selects `"85-100"`**:
  - Message Box displays: `"Student is A grade"`
- **If the user selects `"70-84"`**:
  - Message Box displays: `"Student is B grade"`
- **If the user selects `"60-69"`**:
  - Message Box displays: `"Student is C grade"`
- **If the user selects `"<60"`**:
  - Message Box displays: `"Student is D grade"`
- **If the user selects `"<50"`**:
  - Message Box displays: `"Student has Failed"`
- **If the user selects an option not listed**:
  - Message Box displays: `"Invalid selection"`

Error Handling

- **Invalid Input**:
  - Ensure that the options provided in the **OptionsString** are correctly formatted and match the **Case** keys in the Switch activity.

    - The **Default** case handles any invalid selections by displaying `"Invalid selection"`.
- **Case Sensitivity**:
    - The **Switch** activity is case-sensitive. Make sure that the values in the **OptionsString** and the **Case** keys match exactly, including any symbols like `<`.
- **User Selection Issues**:
    - If the user closes the Input Dialog without making a selection, the variable `userOption` may be empty. You can handle this by adding an additional **Case** or modifying the **Default** case to prompt the user again or exit gracefully.

Notes

- **Testing**:
    - Run the workflow multiple times, selecting different options each time to ensure all cases are handled correctly.
- **Extensibility**:
    - You can add more grade ranges and corresponding cases as needed.
    - For a more dynamic approach, consider using numerical comparisons instead of string matches.

Conclusion

This experiment demonstrates how to use the **Switch** activity in UiPath to handle multiple possible input values efficiently. By mapping user selections to specific cases, you can create workflows that respond dynamically to user input without complex nested conditions.

6. Text Automation with Wikipedia and Google Docs

Objective

To create a workflow that automates text retrieval from Wikipedia and pastes it into a Google Docs document. This workflow demonstrates UiPath's ability to integrate web data extraction and text handling.

Tools Required

- **UiPath Studio**
- **Google Chrome or Edge (with extensions enabled)**

Workflow Components

1. **Use Application/Browser**: For interacting with Wikipedia and Google Docs.
2. **Type Into**: To enter search terms on Wikipedia.
3. **Click**: To initiate search and interact with UI elements.
4. **Get Text**: To extract content from the Wikipedia page.
5. **Variables**: To store extracted text and other details.

Steps to Perform the Experiment

Step 1: Open UiPath Studio

1. Launch **UiPath Studio**.
2. Create a new project or open an existing one.

Step 2: Create a New Sequence

1. Navigate to **New File > Sequence** and name it `Main`.
2. Drag a **Sequence** container into the **Designer Panel**.

Step 3: Configure Variables

1. Open the **Variables Panel**.
2. Add the following variables:

Name	Type	Scope	Default Value
`extractedText`	String	Main	None

Step 4: Automate Wikipedia Search

1. **Use Application/Browser Activity**:
   - Drag the **Use Application/Browser** activity into the sequence.
   - Configure the browser type (e.g., Chrome or Edge).
   - Set the **URL** property to: `https://www.wikipedia.org/`.
2. **Type Into Activity**:
   - Drag the **Type Into** activity into the browser container.
   - Configure:

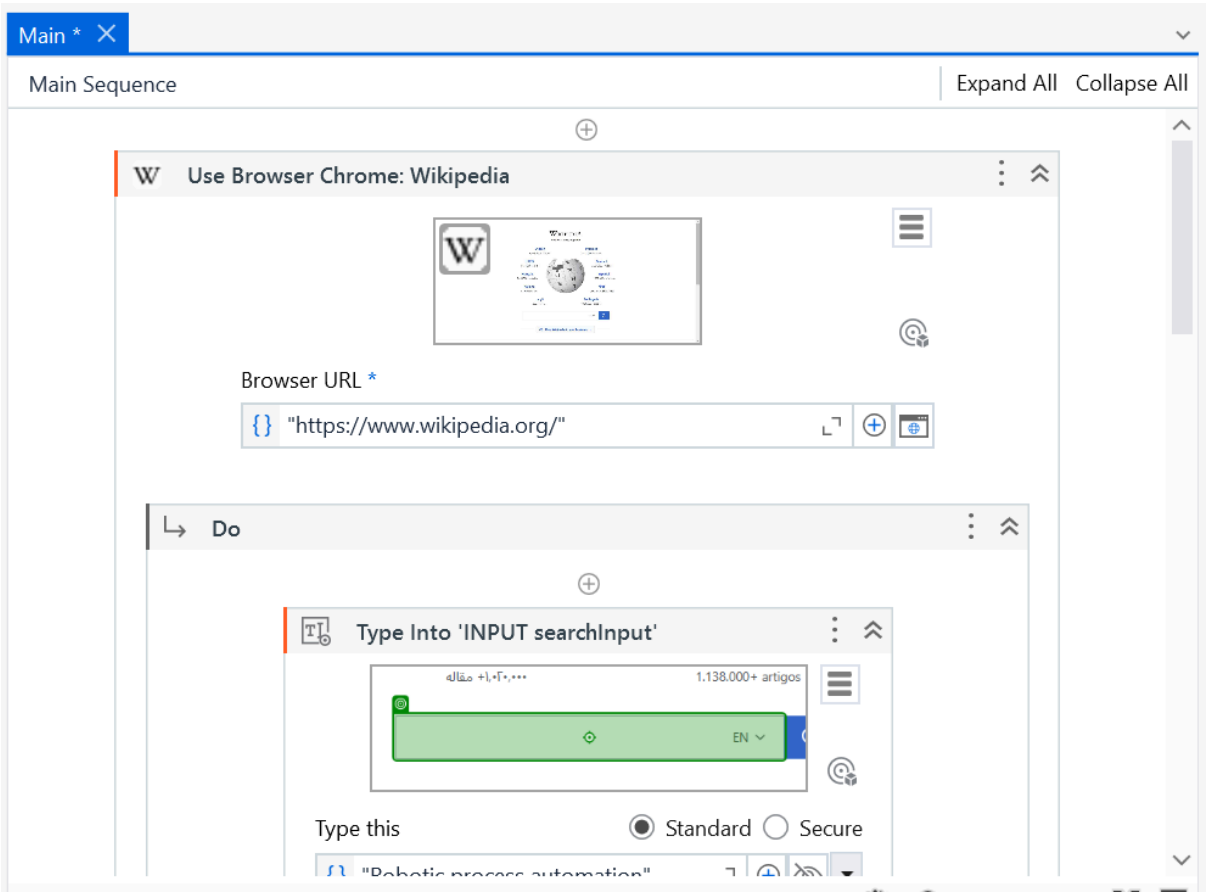
        - **Selector**: Target the Wikipedia search input box (`id='searchInput'`).
        - **Text**: `"Robotic process automation"`.
3. **Click Activity**:
    - Drag the **Click** activity below the **Type Into** activity.
    - Configure:
        - **Selector**: Target the Wikipedia search button (`type='submit'`).
4. **Get Text Activity**:
    - Drag the **Get Text** activity below the **Click** activity.
    - Configure:
        - **Selector**: Target the first paragraph of the Wikipedia result.
        - **Output**: Assign to `extractedText`.

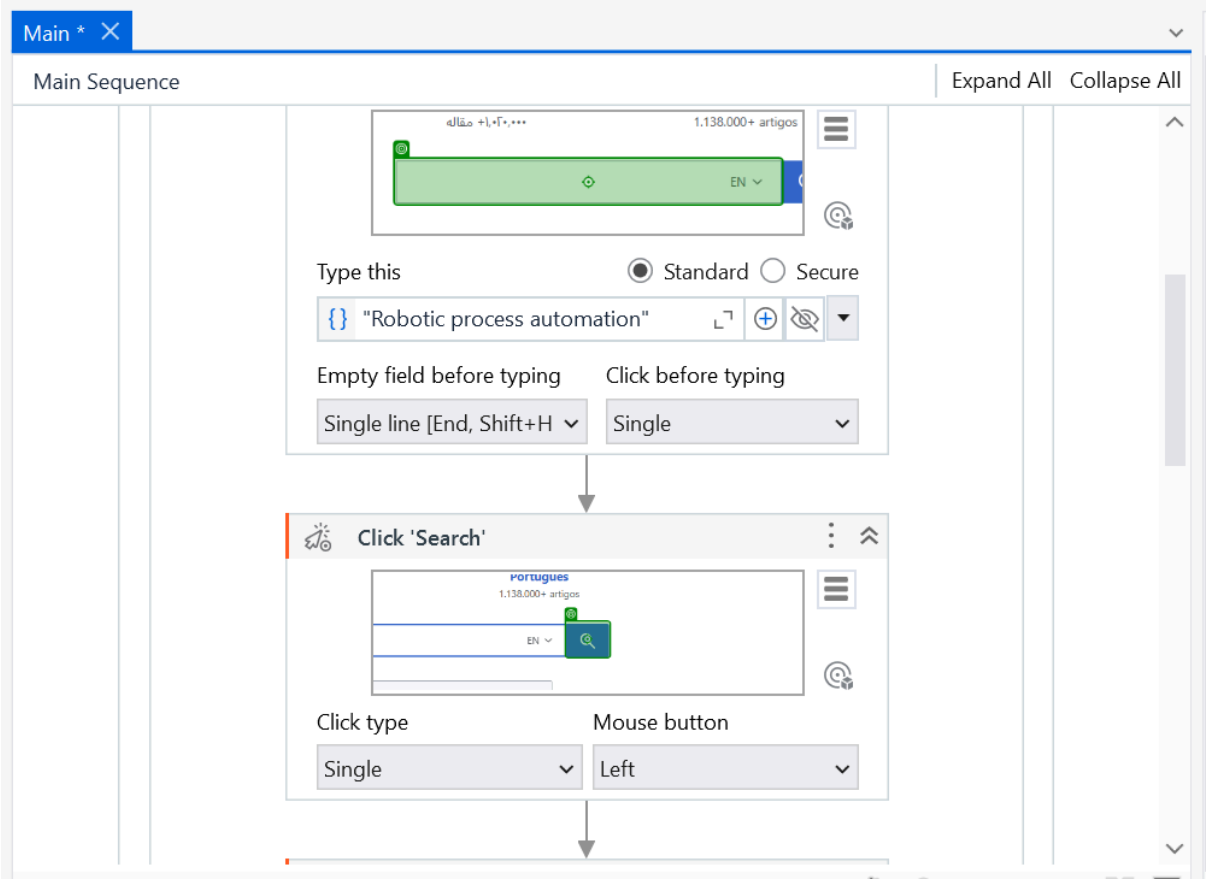
Step 5: Automate Google Docs Interaction

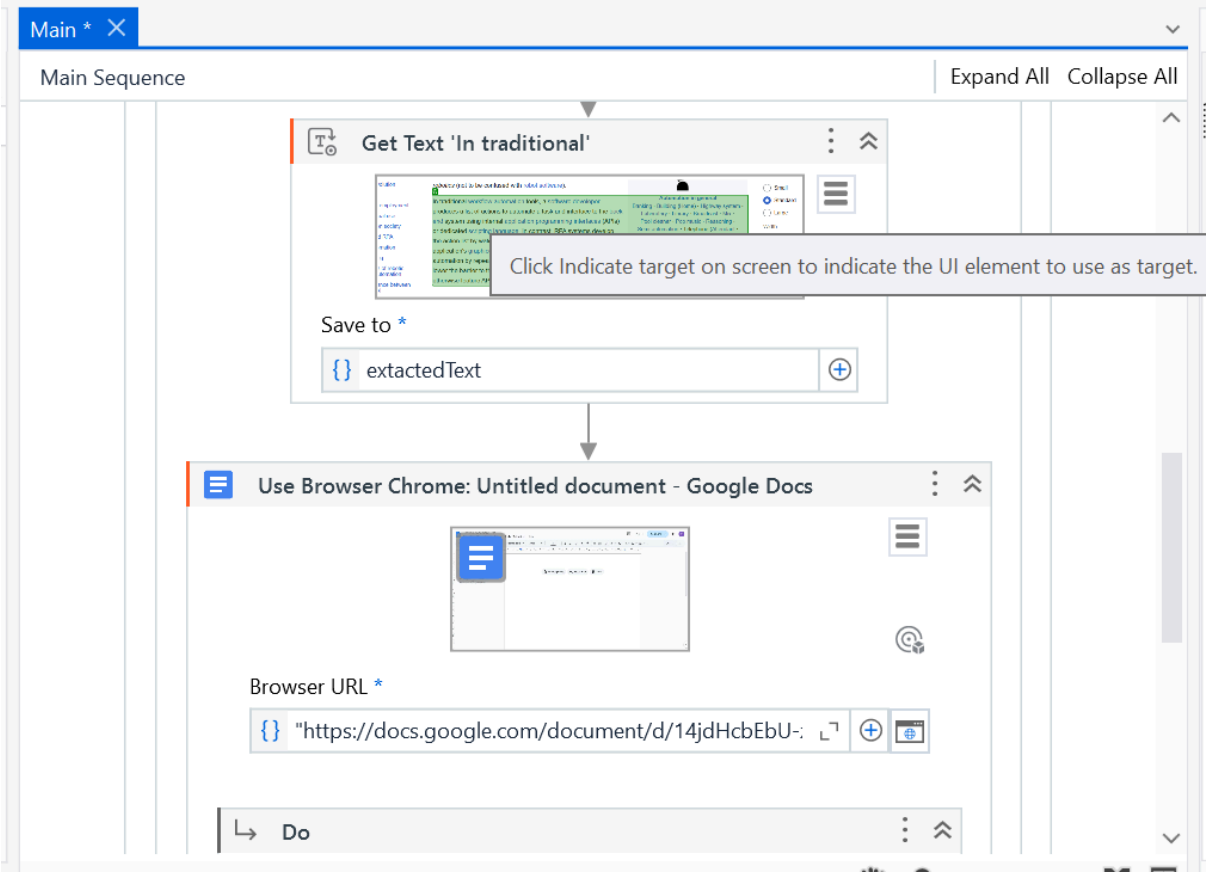
1. **Use Application/Browser Activity**:
    - Drag another **Use Application/Browser** activity into the sequence.
    - Configure the browser type and set the **URL** property to: `https://docs.google.com/document`.
2. **Type Into Activity**:
    - Drag the **Type Into** activity into the Google Docs browser container.
    - Configure:
        - **Selector**: Target the editable area in Google Docs.
        - **Text**: `[extractedText]`.

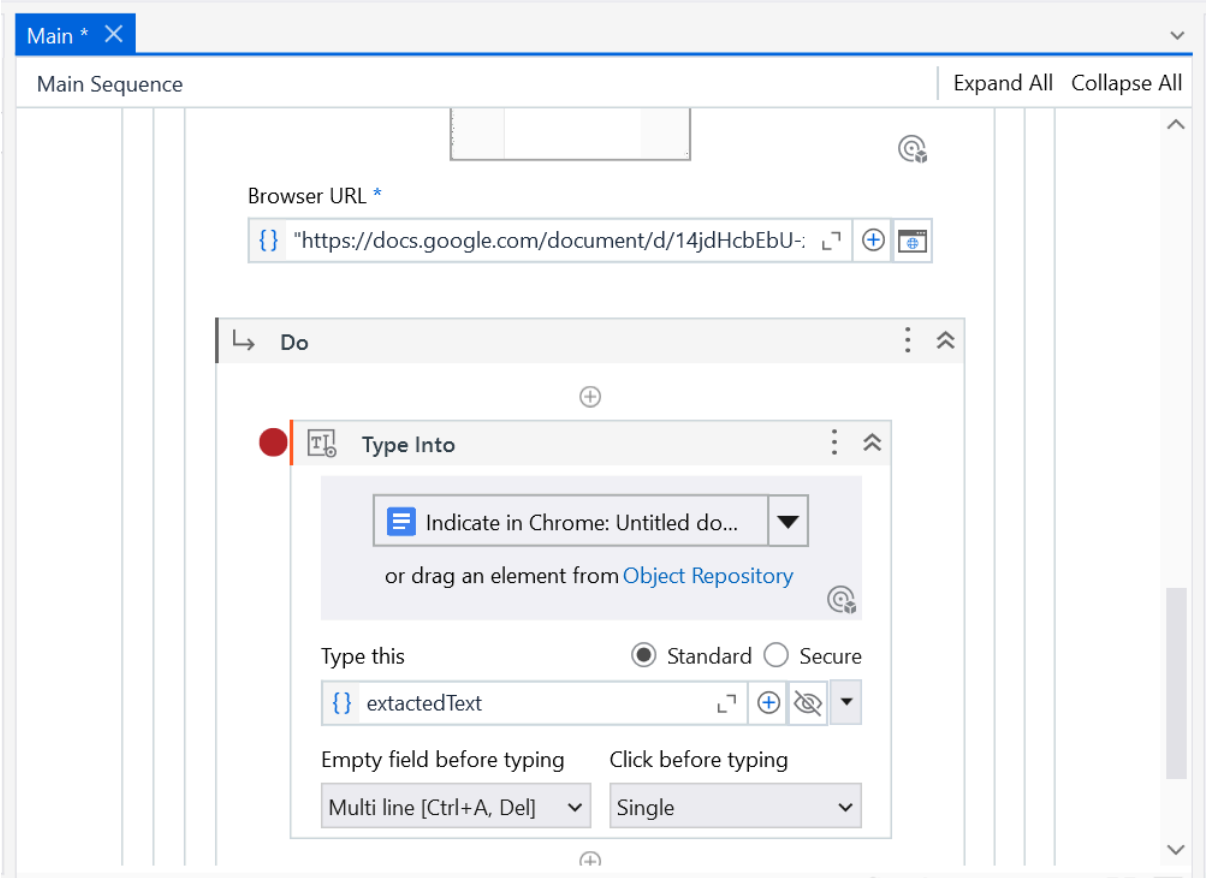
Step 6: Save and Run the Workflow

1. Save the project by clicking the **Save** icon or pressing **Ctrl + S**.
2. Click the **Run** button to execute the workflow.


Main *
Main Sequence
Expand All
Collapse All
Use Browser Chrome: Wikipedia
Browser URL *
"https://www.wikipedia.org/"
Do
Type Into 'INPUT searchInput'
Type this
Standard
Secure


Main *
Main Sequence
Expand All
Collapse All
Type this
Standard
Secure
"Robotic process automation"
Empty field before typing
Click before typing
Single line [End, Shift+H
Single
Click 'Search'
Click type
Mouse button
Single
Left


Main *
Main Sequence
Expand All
Collapse All
Get Text 'In traditional'
Click Indicate target on screen to indicate the UI element to use as target.
Save to *
{} extactedText
Use Browser Chrome: Untitled document - Google Docs
Browser URL *
{} "https://docs.google.com/document/d/14jdHcbEbU-:
Do


Main *
Main Sequence
Expand All
Collapse All
Browser URL *
{} "https://docs.google.com/document/d/14jdHcbEbU-:
Do
Type Into
Indicate in Chrome: Untitled do...
or drag an element from Object Repository
Type this
Standard
Secure
{} extactedText
Empty field before typing
Click before typing
Multi line [Ctrl+A, Del]
Single

Final Workflow Overview

1. **Wikipedia Automation**:
    - Opens Wikipedia in a browser.
    - Searches for "Robotic process automation."
    - Extracts text from the first paragraph.
2. **Google Docs Automation**:
    - Opens Google Docs in a browser.
    - Pastes the extracted text into a new document.

Expected Output

- The workflow opens Wikipedia, searches for "Robotic process automation," and extracts text from the first paragraph.
- The extracted text is pasted into a new Google Docs document.

Error Handling

- **Selector Issues**:
    - Ensure all selectors are accurate and validated during development.
    - Use the **UI Explorer** tool to refine selectors.
- **Network Issues**:
    - Add a **Retry Scope** or use **Try-Catch** blocks to handle connectivity issues.
- **Empty Extraction**:
    - Validate that `extractedText` is not empty before proceeding to Google Docs.
- **Browser Compatibility**:
    - Ensure browser extensions are installed and enabled for UiPath automation.

Notes

- **Testing**:
    - Test the workflow for various search terms to ensure accurate extraction.
    - Verify that the text is pasted correctly in Google Docs.
- **Extensibility**:
    - Add functionality to format the text in Google Docs.
    - Include additional data extraction steps for Wikipedia.

Conclusion

This workflow demonstrates the integration of web data extraction and text automation in UiPath. It highlights the capability to automate repetitive tasks across multiple web applications seamlessly.

7. Extracting Data and Writing to Excel

Objective

To automate the extraction of structured data from a website and save it to an Excel file. This experiment highlights UiPath's capabilities in web data extraction and Excel file handling.

Tools Required

- **UiPath Studio**
- **Google Chrome or Edge (with extensions enabled)**
- **Excel Application** or a workbook-compatible editor.

Workflow Components

1. **Use Application/Browser**: For interacting with the website.
2. **Extract Table Data**: For extracting structured data from a webpage.
3. **Write Range Workbook**: For saving data to an Excel file.
4. **Variables**: To store extracted data and other details.

Steps to Perform the Experiment

Step 1: Open UiPath Studio

1. Launch **UiPath Studio**.
2. Create a new project or open an existing one.

Step 2: Create a New Sequence

1. Navigate to **New File > Sequence** and name it `Main`.
2. Drag a **Sequence** container into the **Designer Panel**.

Step 3: Configure Variables

1. Open the **Variables Panel**.
2. Add the following variables:

Name	Type	Scope	Default Value
`ExtractDataTable`	DataTable	Main Sequence	New System.Data.DataTable()
`extractedText`	String	Inside Do	None

Step 4: Set Up Browser Automation

1. **Use Application/Browser Activity**:
   - Drag the **Use Application/Browser** activity into the sequence.
   - Configure the browser type (e.g., Chrome or Edge).
   - Set the **URL** property to: `https://books.toscrape.com/`.

Step 5: Extract Data from Webpage

1. **Extract Table Data Activity**:
   - Drag the **Extract Table Data** activity into the **Do** container of the browser scope.
   - Use the Table Extraction wizard to select data:
     - Identify table elements (e.g., book titles, prices).
     - Include additional attributes if necessary (e.g., book URLs).
   - Configure:
     - **Output DataTable**: Assign to `ExtractDataTable`.
     - Enable pagination by selecting the "Next" button in the wizard.

Step 6: Output Data as Text

1. **Output Data Table Activity**:
   - Drag the **Output Data Table** activity below the **Extract Table Data** activity.
   - Set **Input DataTable** to `ExtractDataTable` and **Output Text** to `extractedText`.
2. **Write Line Activity**:
   - Drag the **Write Line** activity below the **Output Data Table** activity.
   - Set **Text** to `[extractedText]`.

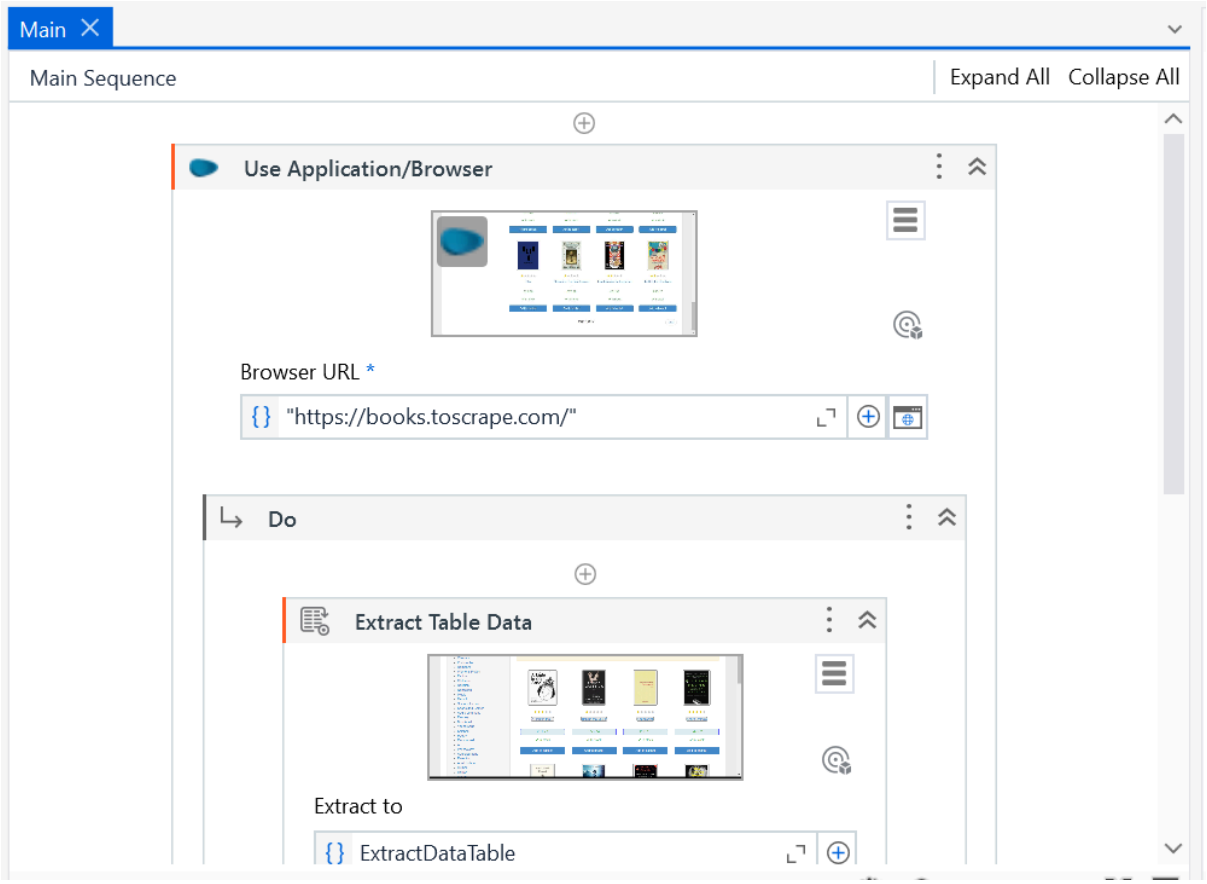
Step 7: Write Data to Excel

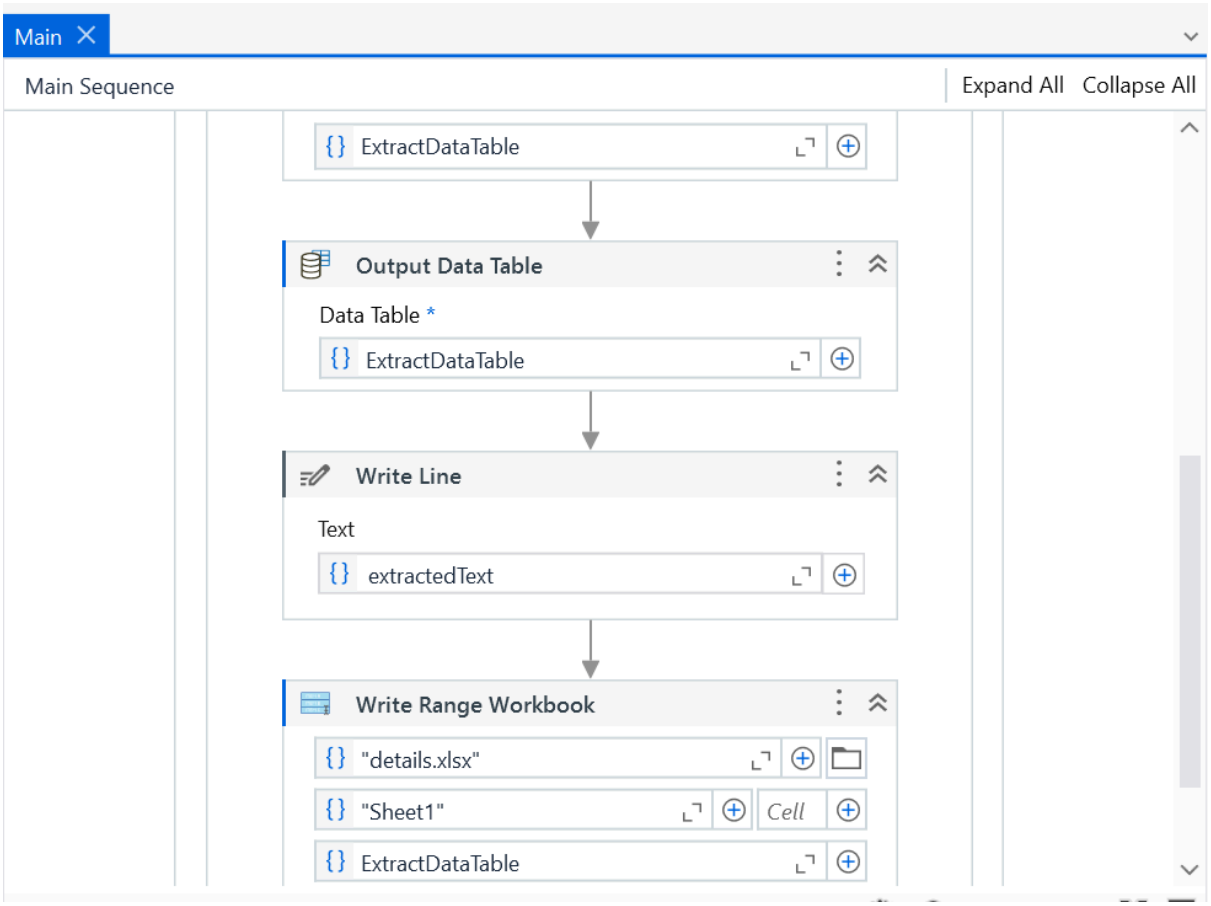
1. **Write Range Workbook Activity**:

- Drag the **Write Range Workbook** activity below the **Write Line** activity.
- Configure:
  - **Workbook Path**: `details.xlsx`.
  - **Sheet Name**: `Sheet1`.
  - **DataTable**: `ExtractDataTable`.
  - Enable **Add Headers** to include column names in the output.

Step 8: Save and Run the Workflow

1. Save the project by clicking the **Save** icon or pressing **Ctrl + S**.
2. Click the **Run** button to execute the workflow.





Final Workflow Overview

1. **Use Application/Browser**:
   - Opens the specified webpage (`https://books.toscrape.com/`).
   - Initiates table data extraction.
2. **Extract Table Data**:
   - Extracts structured data such as book titles, prices, and URLs.
   - Handles pagination using the "Next" button.
3. **Output Data Table**:
   - Converts the extracted data table into a string format for debugging.
4. **Write to Excel**:
   - Saves the extracted data into an Excel file named `details.xlsx`.

Expected Output

- The workflow extracts book information (e.g., titles, prices, URLs) from the website and saves it into an Excel file with the following columns:
  - Book Title
  - Price
  - URL

Error Handling

- **Selector Issues**:
  - Ensure all selectors are accurate and validated during development.
  - Use the **UI Explorer** tool to refine selectors.
- **Pagination Issues**:
  - Ensure the "Next" button is correctly identified.
  - Validate the pagination logic to avoid infinite loops.
- **Excel Writing Errors**:
  - Verify file permissions and paths before writing data.
  - Use Try-Catch blocks to handle file-related exceptions.
- **Browser Compatibility**:
  - Ensure browser extensions are installed and enabled for UiPath automation.

Notes

- **Testing**:
  - Test the workflow for multiple pages to ensure the pagination logic works.
  - Verify the extracted data for accuracy and completeness.
- **Extensibility**:
  - Add more attributes for extraction (e.g., book ratings, stock status).
  - Include additional data processing steps if required.

Conclusion

This workflow demonstrates UiPath's ability to automate web data extraction and integrate with Excel for data processing. It highlights the potential for automating data collection from websites for analytics or reporting.

8. Extracting Text from PDFs

Objective

To automate the process of extracting text from multiple PDF files in a folder and display the extracted text using UiPath Studio. This experiment demonstrates file handling, text extraction, and looping mechanisms.

Tools Required

- **UiPath Studio**
- A folder containing PDF files for testing (e.g., `C:\Users\student\Documents\UiPath\pdfAutomation\Pdf files`).

Workflow Components

1. **For Each File in Folder**: Iterates through all PDF files in a specified directory.
2. **Extract PDF Text**: Extracts text content from a PDF file.
3. **Message Box**: Displays the extracted text for verification.
4. **Variables**: Used to store file paths and extracted text.

Steps to Perform the Experiment

Step 1: Open UiPath Studio

1. Launch **UiPath Studio**.
2. Create a new project or open an existing one.

Step 2: Create a New Sequence

1. Navigate to **New File > Sequence** and name it `Main`.
2. Drag a **Sequence** container into the **Designer Panel**.

Step 3: Configure Variables

1. Open the **Variables Panel**.
2. Add the following variables:

Name	Type	Scope	Default Value
`extractedText`	String	Inside For Each	None

Step 4: Process Files in a Folder

1. **For Each File in Folder Activity**:
    - Drag the **For Each File in Folder** activity into the sequence.
    - Configure:
        - **Folder**: Path to your PDF folder (e.g., `C:\Users\student\Documents\UiPath\pdfAutomation\Pdf files`).
        - **Filter**: `*.pdf` to process only PDF files.

- **Include Subdirectories**: Enable this to process files in subfolders.
- **Order By**: `NameAscFirst` to process files alphabetically.

Step 5: Display File Path

1. **Message Box Activity**:
   - Drag a **Message Box** activity into the **Do** container of the **For Each File** activity.
   - Set the **Text** property to: `[CurrentFile.FullName]`.
   - This displays the full path of each PDF being processed.

Step 6: Extract PDF Text

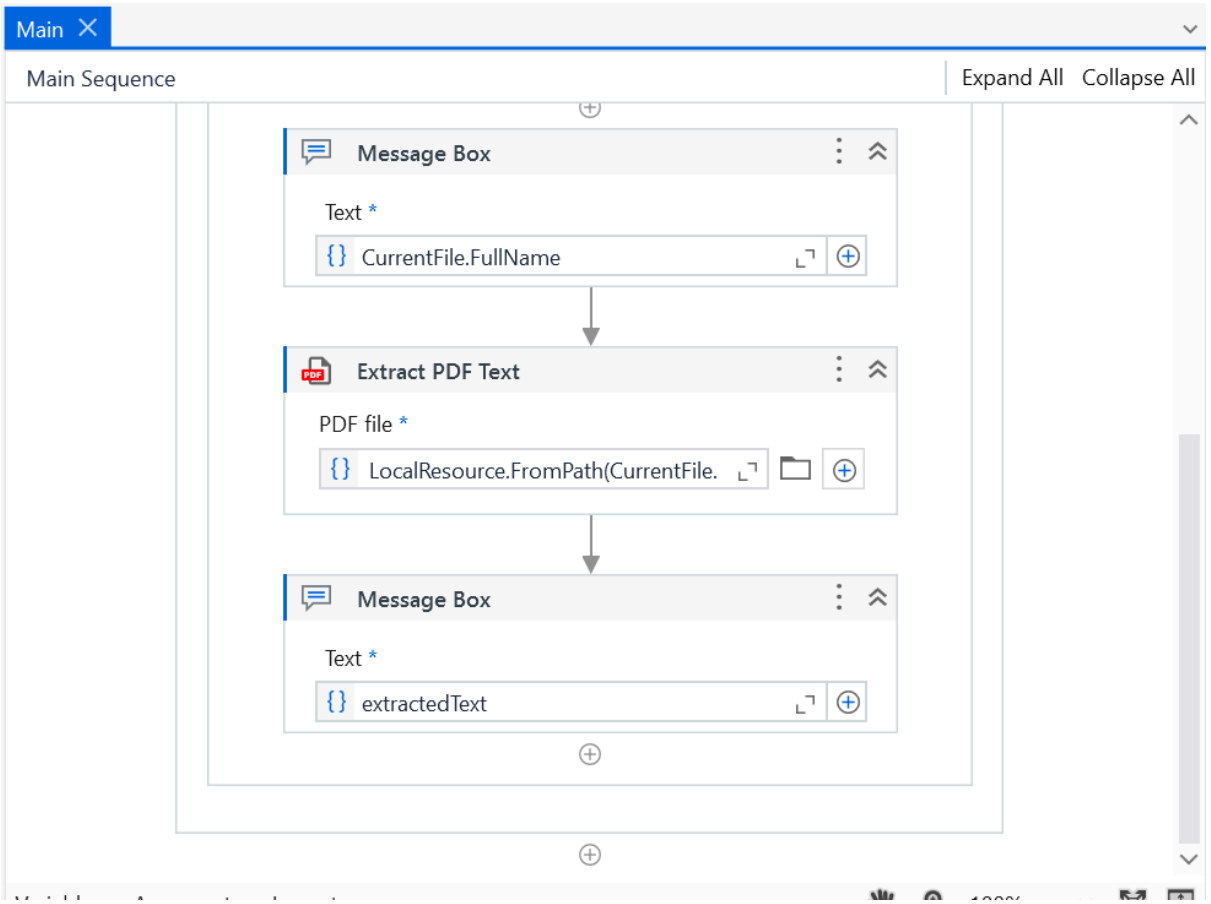
1. **Extract PDF Text Activity**:
   - Drag the **Extract PDF Text** activity below the **Message Box**.
   - Configure:
     - **PDF File**: `[LocalResource.FromPath(CurrentFile.FullName)]`.
     - **Output Text**: Assign to the `extractedText` variable.

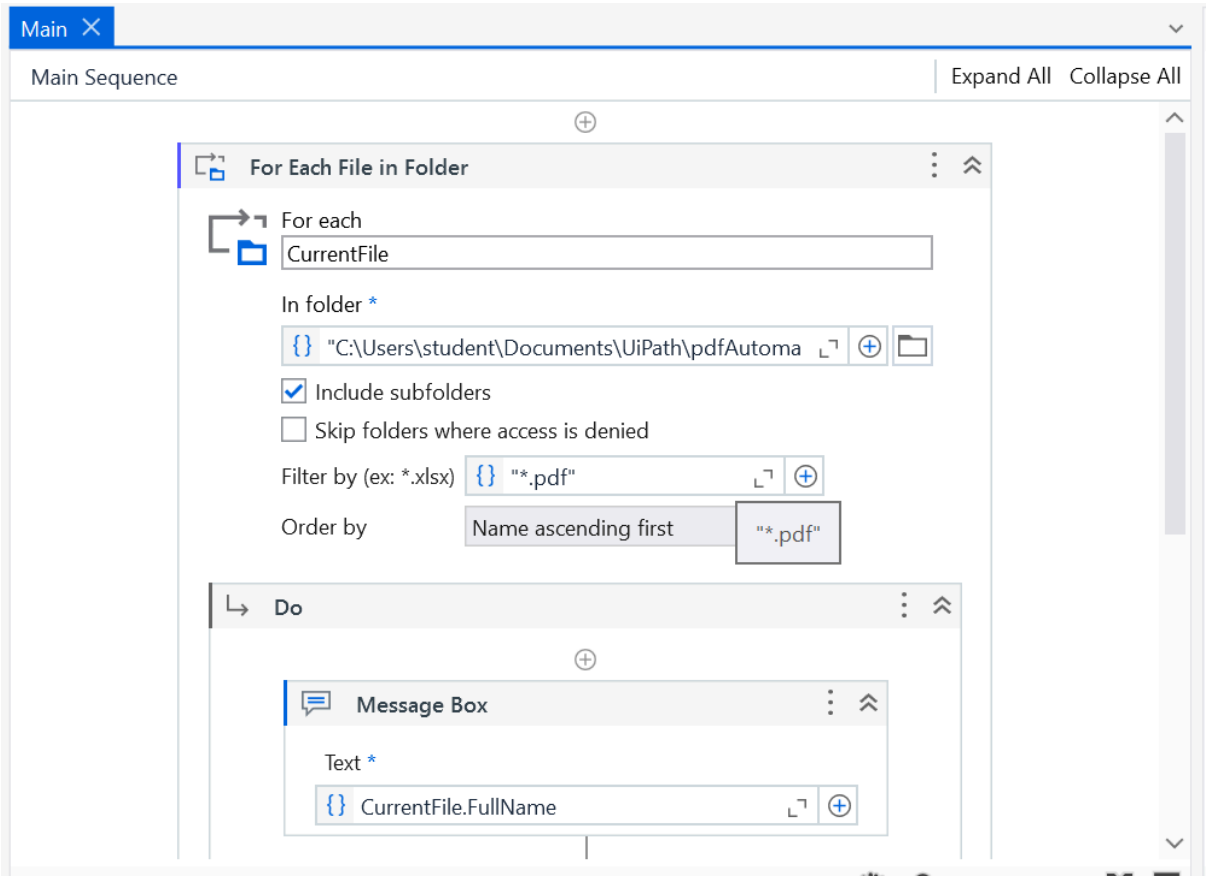
Step 7: Display Extracted Text

1. **Message Box Activity**:
   - Drag another **Message Box** activity below the **Extract PDF Text** activity.
   - Set the **Text** property to: `[extractedText]`.
   - This displays the extracted text for verification.

Step 8: Save and Run the Workflow

1. Save the project by clicking the **Save** icon or pressing **Ctrl + S**.
2. Click the **Run** button to execute the workflow.


Main
Main Sequence
Expand All
Collapse All
Message Box
Text *
CurrentFile.FullName
Extract PDF Text
PDF file *
LocalResource.FromPath(CurrentFile.
Message Box
Text *
extractedText


Main
Main Sequence
Expand All
Collapse All
For Each File in Folder
For each
CurrentFile
In folder *
"C:\Users\student\Documents\UiPath\pdfAutoma
Include subfolders
Skip folders where access is denied
Filter by (ex: *.xlsx)
"*.pdf"
Order by
Name ascending first
"*.pdf"
Do
Message Box
Text *
CurrentFile.FullName

Final Workflow Overview

1. **For Each File in Folder**:
    - Iterates through all PDF files in the specified directory.
    - Displays the file path of each PDF.
2. **Extract PDF Text**:
    - Extracts text from the current PDF file.
3. **Message Box**:
    - Displays the extracted text for verification.

Expected Output

- **File Path Display**:
    - The full path of each PDF file is displayed in a message box.
- **Extracted Text Display**:
    - The text content of each PDF is displayed in a message box.

Error Handling

- **File Access Errors**:
    - Use Try-Catch blocks to handle issues such as file not found or access denied.
    - Display appropriate error messages for user clarity.
- **PDF Extraction Errors**:
    - Handle unsupported formats or password-protected files gracefully.
    - Log extraction issues for debugging purposes.

Notes

- **Testing**:
    - Test the workflow with a variety of PDF files (e.g., text-based, scanned, or complex layouts).
    - Verify that the extracted text matches the content of the PDFs.
- **Extensibility**:
    - Add functionality to save the extracted text into a text file or database.
    - Incorporate OCR (Optical Character Recognition) activities for scanned PDFs.

Conclusion

This workflow highlights UiPath's capabilities in automating PDF text extraction and file handling. It serves as a foundational component for building more complex document processing solutions.

9. Generating COVID-19 Reports and Sending Emails

Objective

To automate the creation of COVID-19 test reports from structured data, export the reports as PDF documents, and send them as email attachments to the respective recipients using UiPath Studio.

Tools Required

- **UiPath Studio**
- An Excel file with COVID-19 test data (e.g., `details.xlsx`).
- Microsoft Word or a compatible text editor for report formatting.
- Gmail or Outlook for sending emails.

Workflow Components

1. Generate DataTable

- **Purpose**: Convert the input text into a structured DataTable for processing.

2. Process Each Row

- Extract patient details like test date, region, result, test type, comments, name, and email.

3. Create and Export Report

- Generate a Word document containing patient-specific information and export it as a PDF.

4. Send Email

- Attach the generated report and send it to the patient's email.

5. Update Excel File

- Change the background color of cells in column C to green for processed rows.

Steps to Perform the Experiment

Step 1: Open UiPath Studio

1. Launch **UiPath Studio**.
2. Create a new project or open an existing one.

Step 2: Create a New Sequence

1. Navigate to **New File > Sequence** and name it `COVID-19 Report`.
2. Drag a **Sequence** container into the **Designer Panel**.

Step 3: Configure Variables

1. Open the **Variables Panel**.
2. Add the following variables:

Name	Type	Scope	Default Value
`resultDataTable`	DataTable	Main Sequence	None
`strTable`	String	Main Sequence	None
`patientName`	String	Inside For Each	None
`emailIdStr`	String	Inside For Each	None
`testDate`	DateTime	Inside For Each	None
`testRegion`	String	Inside For Each	None
`testResult`	String	Inside For Each	None
`testType`	String	Inside For Each	None
`comments`	String	Inside For Each	None
`textHasToBeTyped`	String	Inside Word Scope	None
`mailIdArray`	String[]	Inside Word Scope	None

Step 4: Generate DataTable from Text

1. **Generate Data Table Activity**:
    - Drag the **Generate Data Table** activity into the sequence.
    - Configure:

Input: `Test Date,Region,Test Result,Test Type,Comments,Name,Email ID`
`2024-12-05,NYC,Positive,PCR,Isolating,Yuvan,yuvanyuvaa763@gmail.com`
`2024-12-04,LA,Negative,Antigen,Follow-up test,Rakesh,rakesh.s@cmritonline.ac.in`
`2024-12-05,Chicago,Pending,PCR,-,Madhavi,alukamadhavi@cmritonline.ac.in`
`2024-12-04,Boston,Negative,PCR,Negative result,AMK,alagumuthukrishnan@cmritonline.ac.in`
`2024-12-05,NYC,Positive,Antigen,-,Jayashree,jaishriwankhede58@gmail.com`

        - **Output**: Assign to `resultDataTable`.
2. **Output Data Table Activity**:
    - Drag the **Output Data Table** activity below the **Generate Data Table** activity.
    - Assign the output to `strTable`.
3. **Message Box Activity**:
    - Drag a **Message Box** activity below the **Output Data Table** activity.
    - Set the **Text** property to: `[strTable]`.

Step 5: Process Each Row

1. **For Each Row Activity**:
    - Drag the **For Each Row** activity into the sequence.
    - Configure:
        - **DataTable**: `[resultDataTable]`.

2. Inside the **Body** of the For Each Row, add **Get Row Item Activities** for the following columns:

Column Index	Variable
0	`testDate`
1	`testRegion`
2	`testResult`
3	`testType`

4	`comments`
5	`patientName`
6	`emailIdStr`

Step 6: Create and Export Word Document

1. **Word Application Scope**:
   - Drag the **Word Application Scope** activity into the sequence.
   - Set the file path to `Doc1.docx`.
2. Inside the **Word Scope**:
   - **Assign Activity**:
     - **To**: `textHasToBeTyped`

Value:
```
"----------------------------------------------------COVID-19
Test Report----------------------------------------------------"
+ Environment.NewLine + "Patient Name: " + patientName
+ Environment.NewLine + "Test Date: " + testDate.ToString()
+ Environment.NewLine + "Test Result: " + testResult
+ Environment.NewLine + "Test Type: " + testType
+ Environment.NewLine + "Comments: " + comments
+ Environment.NewLine +
"------------------------------------------------------"
+ Environment.NewLine + "Name: " + patientName + Environment.NewLine
+ "Email ID: " + emailIdStr
```

   - **Word Append Text**:
     - Set **Text** to `textHasToBeTyped`.
   - **Word Export to PDF**:
     - Set the file path to `generatedReport.pdf`.

Step 7: Send Email with PDF Attachment

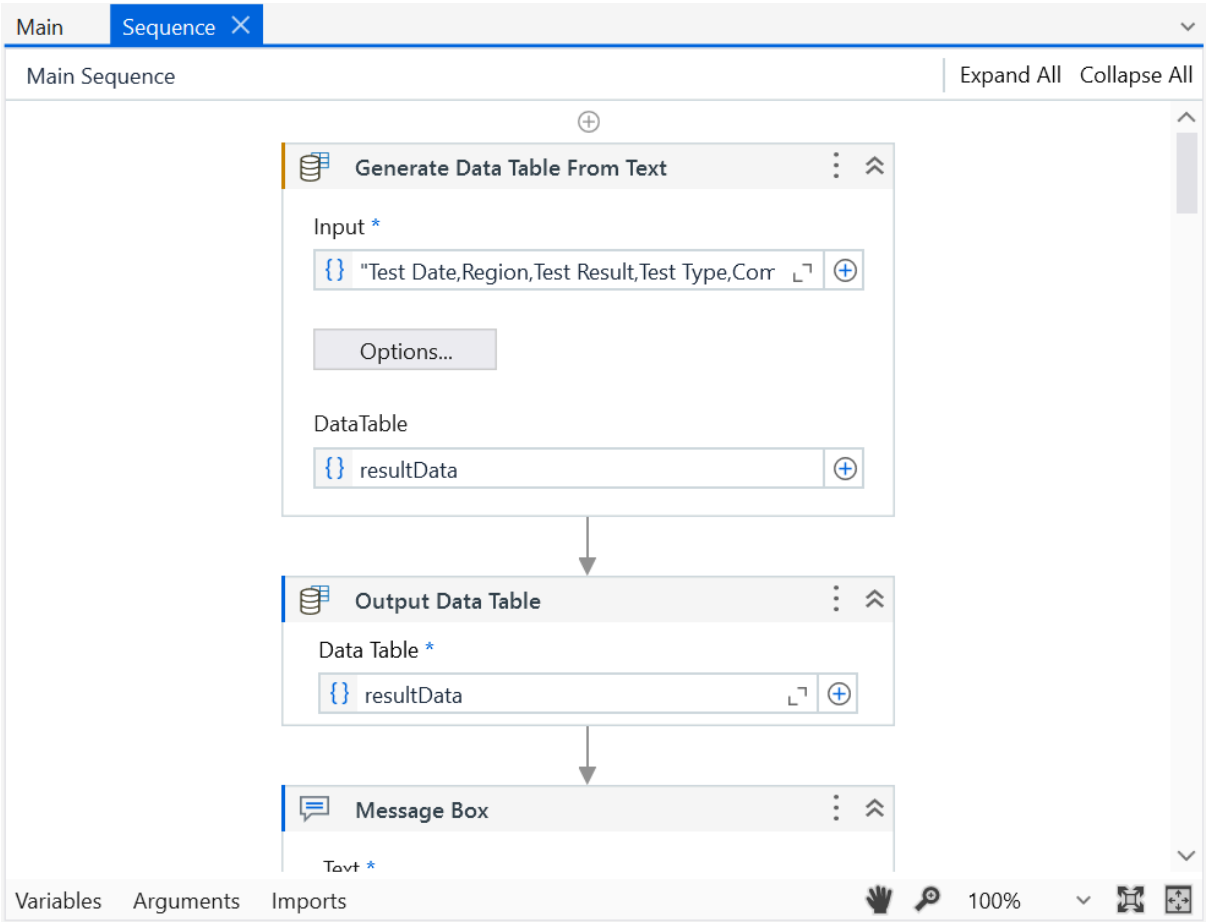
1. **Assign Email Array**:
   - Use an **Assign Activity** to assign:
     - **To**: `mailIdArray`
     - **Value**: `new String[] { emailIdStr }`.
2. **Send Email Activity**:
   - Drag the **Send Email** activity into the sequence.
   - Configure:

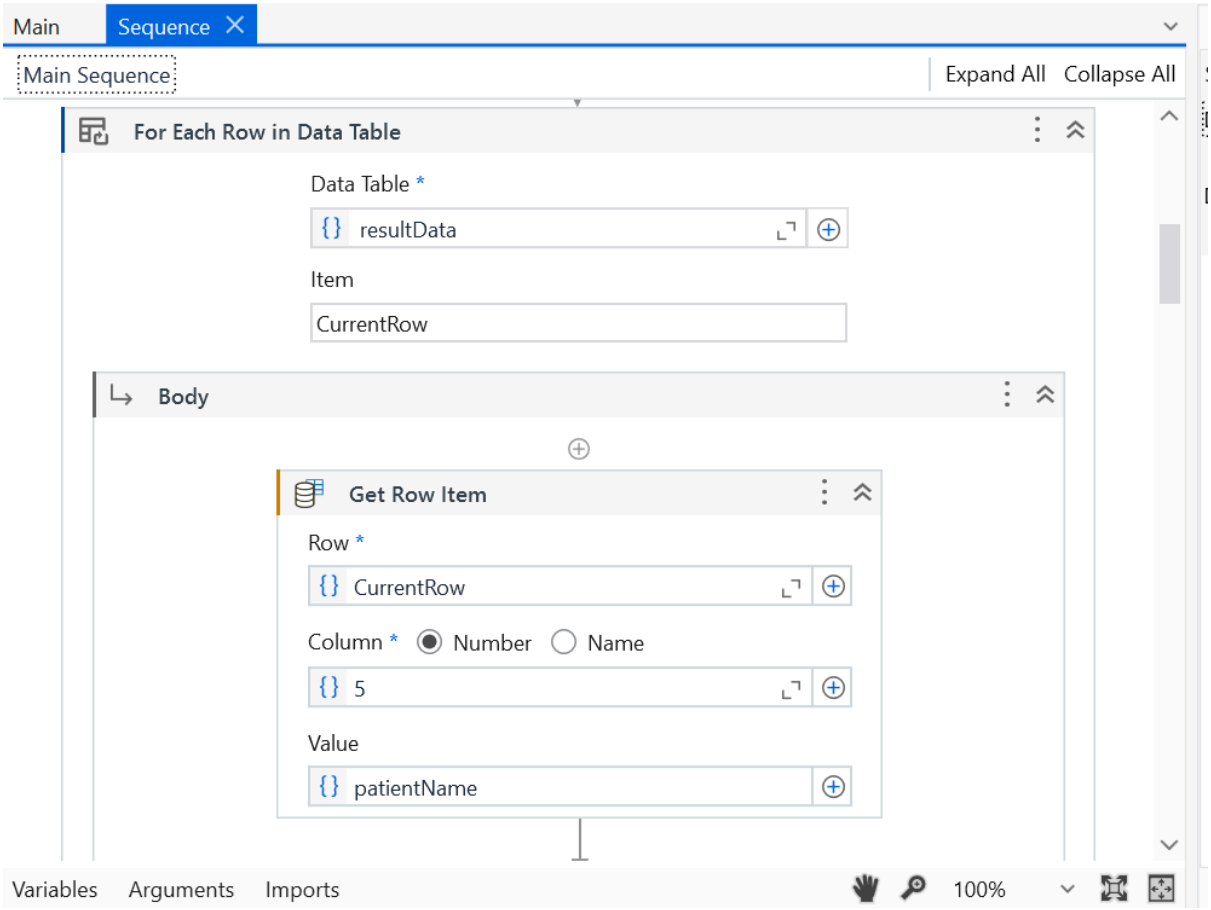
        - **To**: `[mailIdArray]`
        - **Subject**: `"COVID-19 Report"`

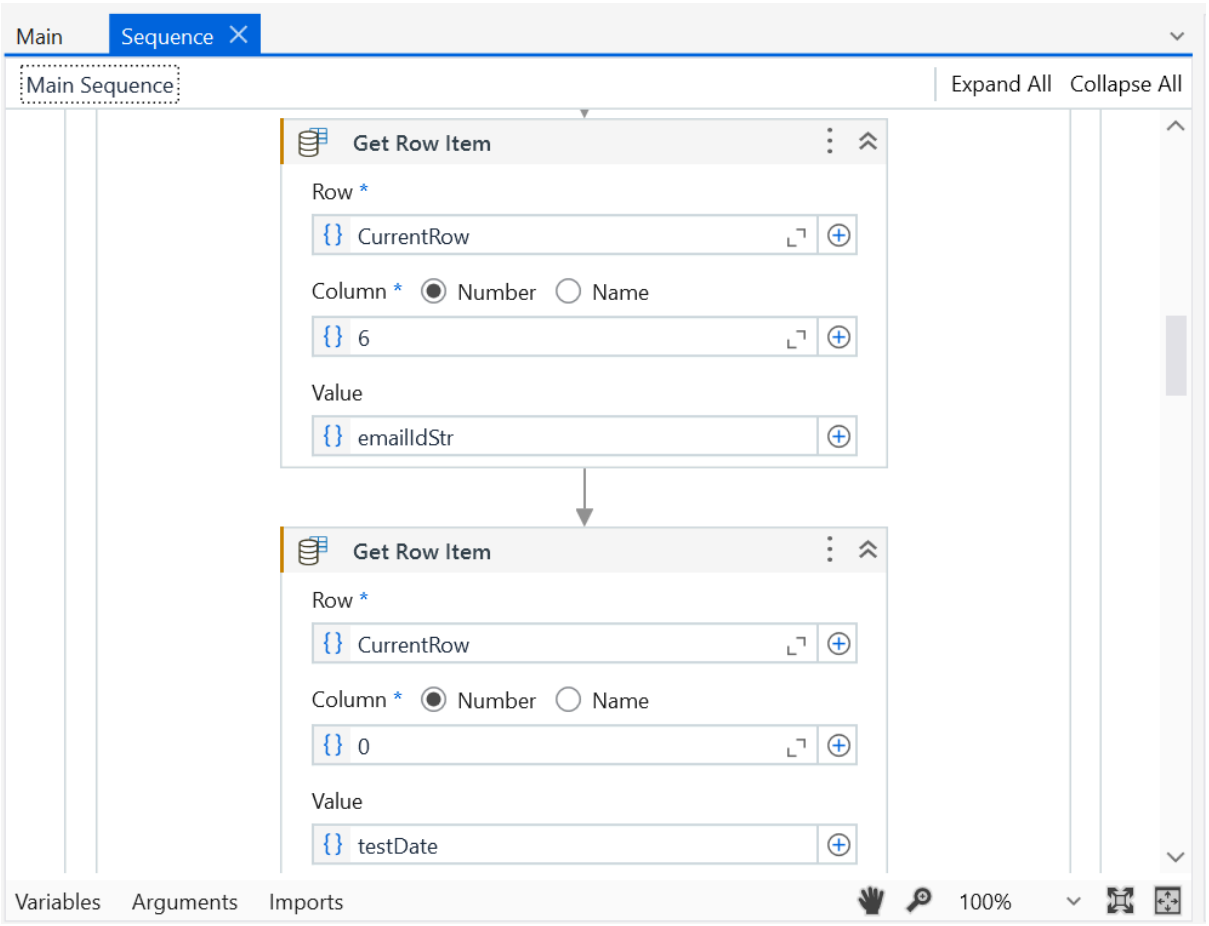
Body:`"Hi " + patientName + "," + Environment.NewLine + "Please find the lab report attached." + Environment.NewLine + "Thanks"`

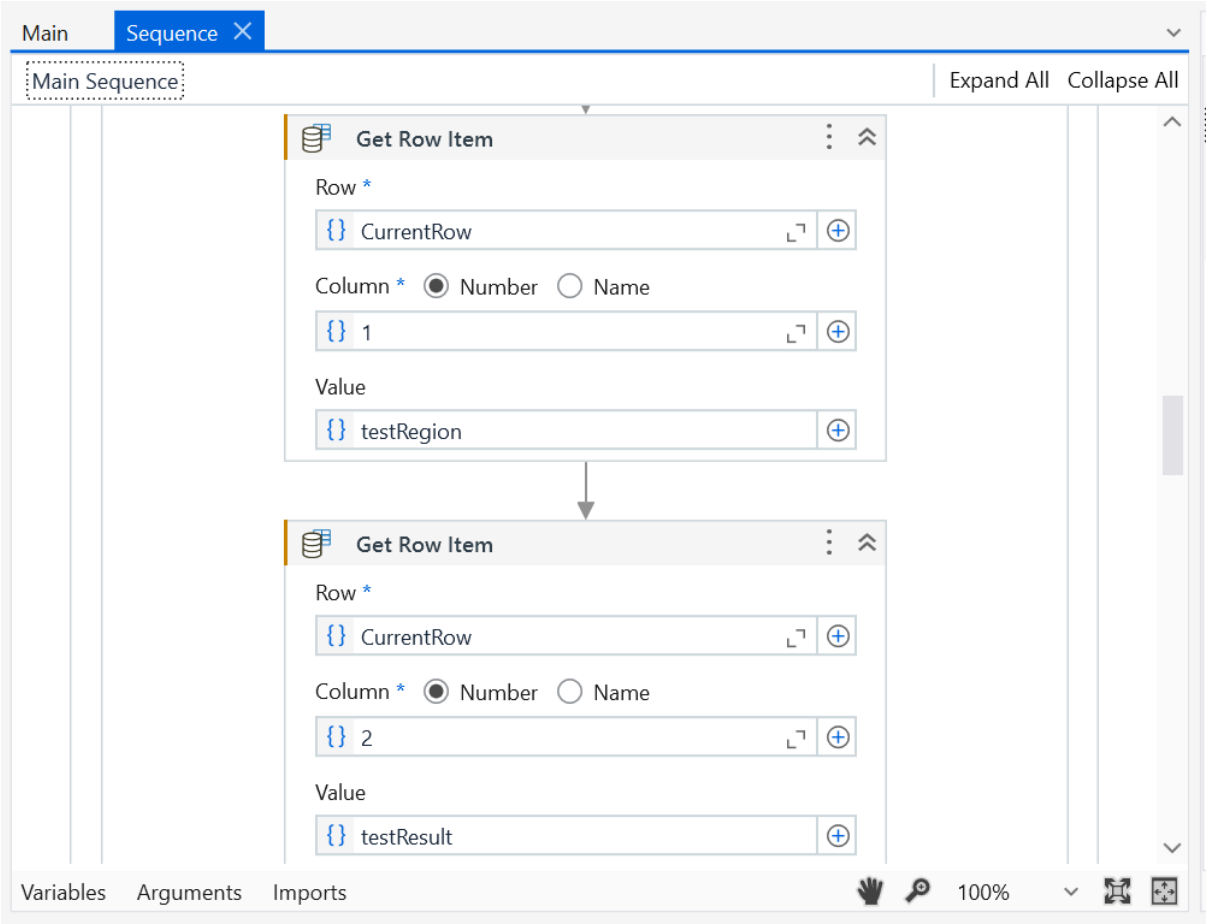
Step 8: Update Excel File

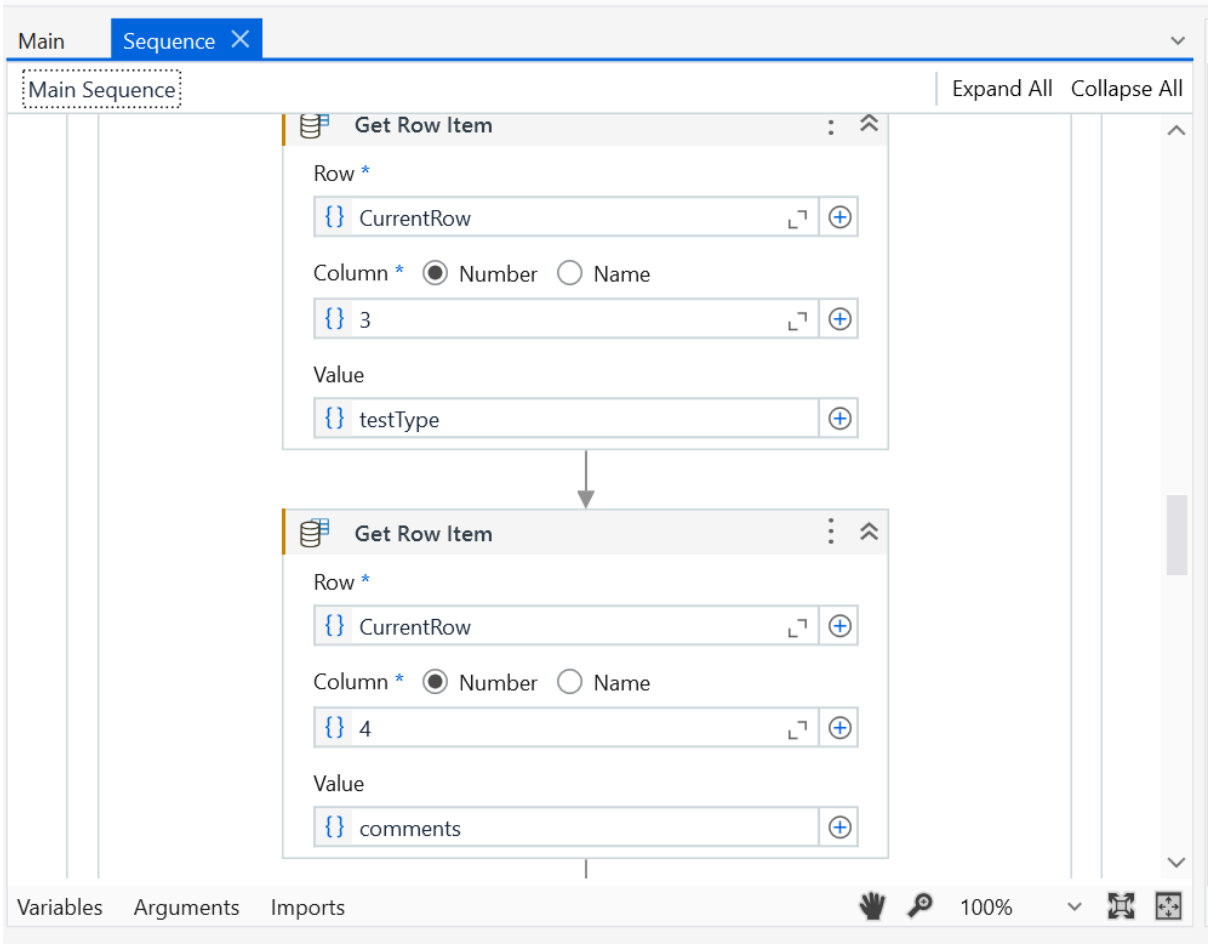
1. **Add Excel Application Scope**:
    - File Path: `details.xlsx`.
2. Inside the scope:
    - **For Each Row Activity**:
        - DataTable: `resultDataTable`.
    - **Set Range Color Activity**:
        - Range: `"C" + (CurrentRow.Table.Rows.IndexOf(CurrentRow) + 1).ToString()`.
        - Sheet Name: `"Sheet1"`.
        - Color: `System.Drawing.Color.Green`.

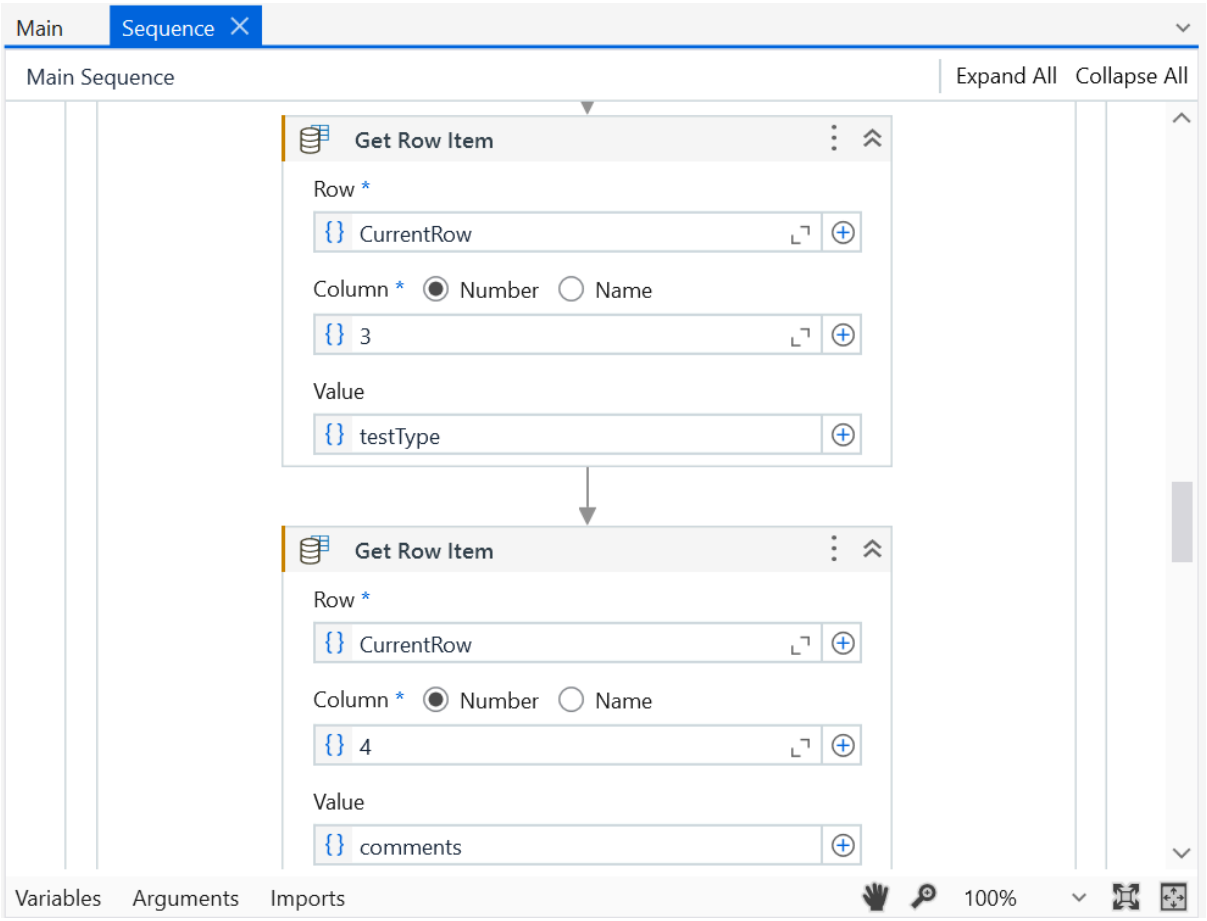

Main
Sequence
Main Sequence
Expand All
Collapse All
Generate Data Table From Text
Input *
"Test Date,Region,Test Result,Test Type,Com
Options...
DataTable
resultData
Output Data Table
Data Table *
resultData
Message Box
Text *
Variables
Arguments
Imports
100%

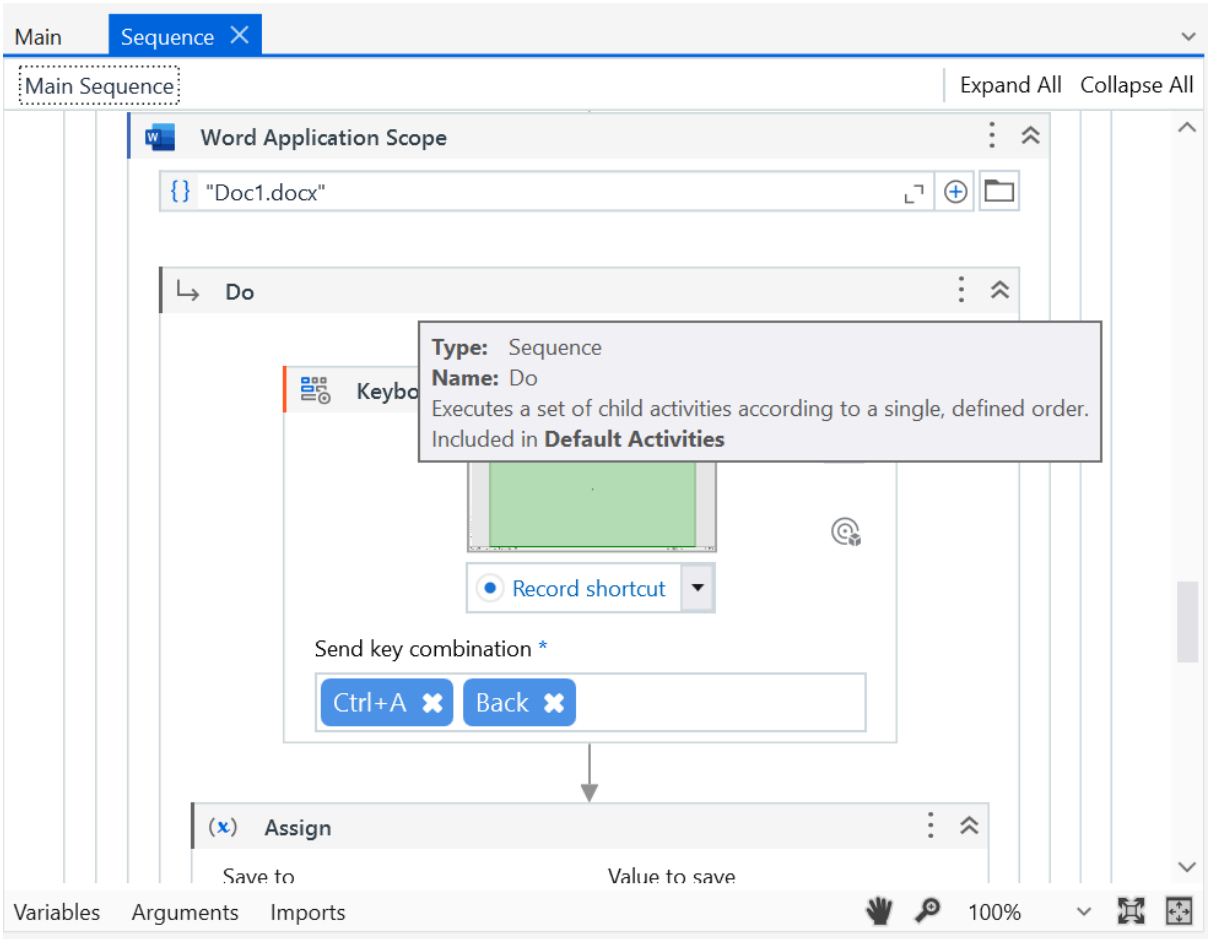

Main
Sequence
Main Sequence
Expand All
Collapse All
For Each Row in Data Table
Data Table *
resultData
Item
CurrentRow
Body
Get Row Item
Row *
CurrentRow
Column *
Number
Name
5
Value
patientName
Variables
Arguments
Imports
100%

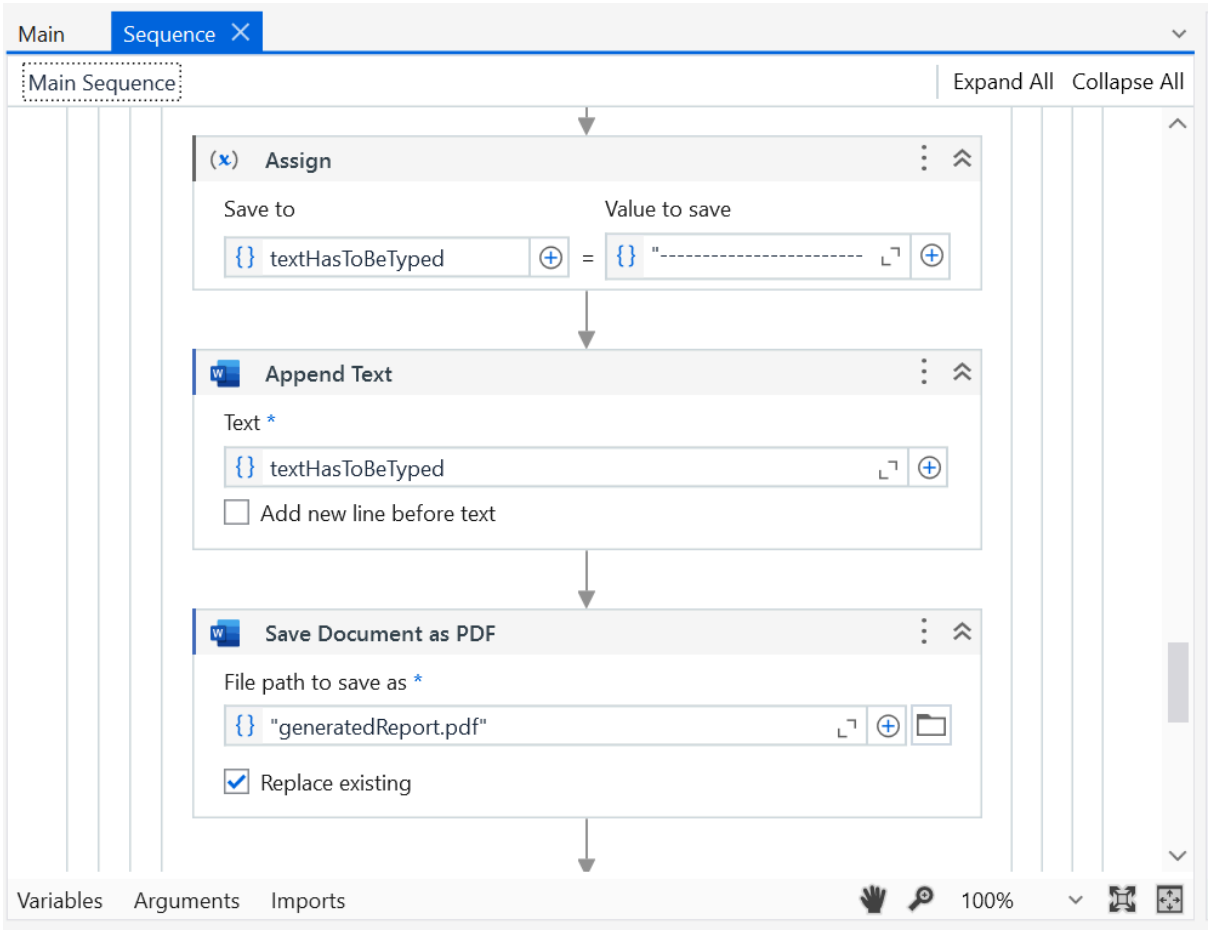

Main
Sequence
Main Sequence
Expand All
Collapse All
Get Row Item
Row *
CurrentRow
Column * Number Name
6
Value
emailIdStr
Get Row Item
Row *
CurrentRow
Column * Number Name
0
Value
testDate
Variables
Arguments
Imports
100%

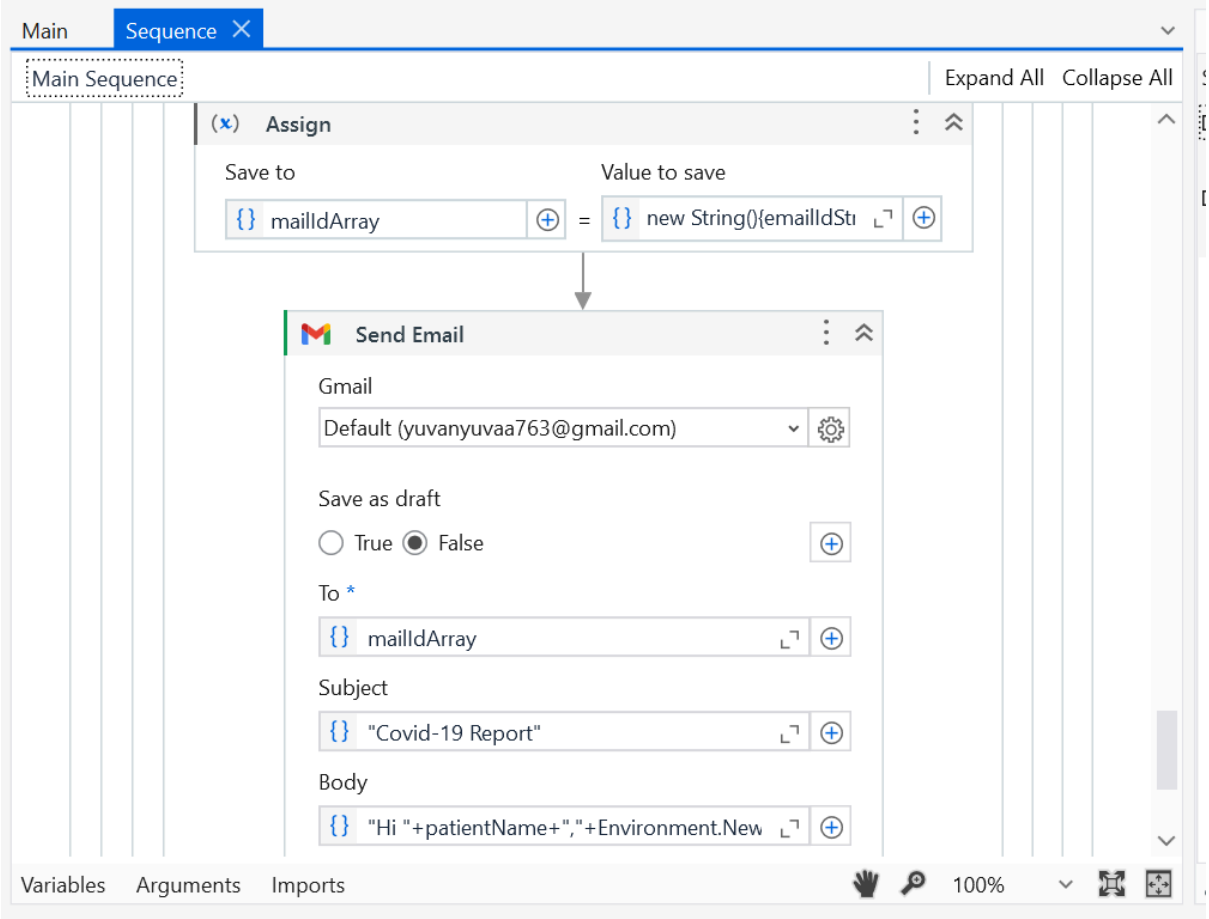

Main
Sequence
Main Sequence
Expand All
Collapse All
Get Row Item
Row *
CurrentRow
Column * Number Name
1
Value
testRegion
Get Row Item
Row *
CurrentRow
Column * Number Name
2
Value
testResult
Variables
Arguments
Imports
100%


Main
Sequence
Main Sequence
Expand All
Collapse All
Get Row Item
Row *
CurrentRow
Column * Number Name
3
Value
testType
Get Row Item
Row *
CurrentRow
Column * Number Name
4
Value
comments
Variables
Arguments
Imports
100%


Main
Sequence
Main Sequence
Expand All
Collapse All
Get Row Item
Row *
CurrentRow
Column *
Number
Name
3
Value
testType
Get Row Item
Row *
CurrentRow
Column *
Number
Name
4
Value
comments
Variables
Arguments
Imports
100%


Main
Sequence
Main Sequence
Expand All
Collapse All
Word Application Scope
"Doc1.docx"
Do
Type: Sequence
Name: Do
Executes a set of child activities according to a single, defined order.
Included in Default Activities
Keybo
Record shortcut
Send key combination *
Ctrl+A
Back
Assign
Save to
Value to save
Variables
Arguments
Imports
100%


Main
Sequence
Main Sequence
Expand All
Collapse All
Assign
Save to
Value to save
textHasToBeTyped
=
"----------
Append Text
Text *
textHasToBeTyped
Add new line before text
Save Document as PDF
File path to save as *
"generatedReport.pdf"
Replace existing
Variables
Arguments
Imports
100%



Final Workflow Overview

1. **Generate Data Table**:
   - Converts input text into a structured DataTable.
2. **For Each Row**:
   - Processes each row to extract patient details and generate individual reports.
3. **Word Application Scope**:
   - Creates a formatted Word document and exports it to PDF.
4. **Send Email**:
   - Sends the generated PDF as an email attachment.
5. **Excel Application Scope**:
   - Opens the Excel file.
   - Iterates through rows to update the background color of column C cells.

Error Handling

1. **DataTable Generation Errors**:
   - Ensure the input text is correctly formatted.
   - Add a Try-Catch block to handle parsing errors.
2. **Email Sending Issues**:
   - Validate email addresses before sending.

    - Add retries in case of network issues.
3. **Excel File Access**:
    - Ensure the Excel file exists and is not open by another application.
    - Add a Try-Catch block around the Excel Application Scope.

Expected Output

1. **Generated Reports**:
    - Individual Word and PDF reports are created for each patient.
2. **Emails Sent**:
    - Each patient receives their COVID-19 report as an email attachment.
3. **Excel Updated**:
    - Column C cells in the Excel file are updated to green for processed rows.

Conclusion

This workflow efficiently automates generating and sending COVID-19 test reports while maintaining a robust and error-resilient system. It demonstrates the use of UiPath's advanced activities, such as Word automation, Excel styling, and email integration.