TYPES OF WORK FLOWS

Sequences

Sequences are the smallest type of workflow. 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 workflow or as part of a state machine or flowchart.

For example, you can create a sequence to take information from a .pdf file and add it to a spreadsheet, and reuse it in a different setting, while changing just a few properties.

**Note:** Sequences do not use connectors.

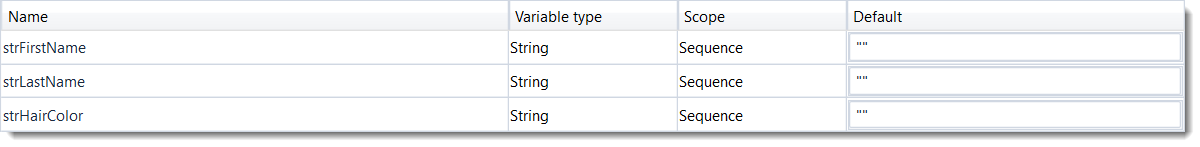
Example of a Sequence

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

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

**Note:** You can also add a **Sequence** activity to the **Main** panel to create a new sequence.

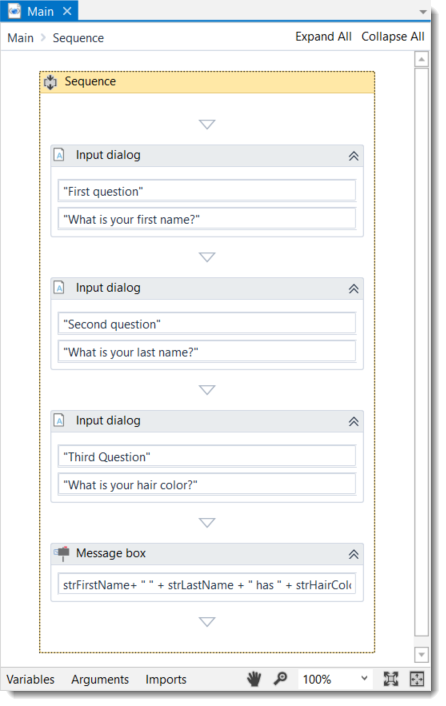
1. In the **Name** field type a name for the workflow, such as “First Sequence,” and click **Create**. The **Main** panel is updated accordingly.
2. Create three **String** variables such as strFirstName, strLastName, and strHairColor, so that you can store data from the user in them. Leave the **Default**field empty, to indicate that there is no default value.



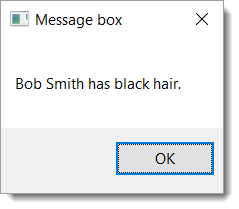
1. Drag three **Input Dialog** activities to the **Main** panel, one under the other.
2. Select the first **Input Dialog** and, in the **Properties** panel, add a **Label** asking for the first name of the user, and a custom **Title**.
3. In the **Result** field, add the strFirstName variable. This indicates that this variable is going to be updated with the value added by the user at this point.
4. Repeat steps 6 - 7 for the second and third **Input Dialog** activities to ask the user for his last name and hair color, and store them in the strLastName and strHairColor variables.
5. Add a **Message Box** activity under the third **Input Dialog** window.
6. Select the **Message Box** and, in the **Properties** panel, in the **Text** field, add the variables and a string to enable you to display all information gathered from the user, such as: strFirstName + " " + strLastName + " has " + strHairColor + " hair.".

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

The final workflow should look as in the following screenshot.



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



[Click here to download this example.](https://www.uipath.com/hubfs/Documentation/WorkflowExamples/FirstSequence.xaml?t=1492088952431)

Flowcharts

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

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

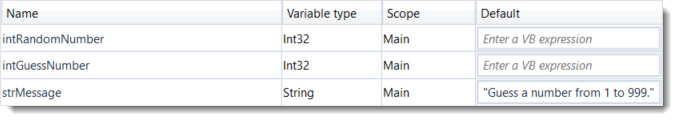
Example of a Flowchart

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

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

**Note:** You can also add a **Flowchart** activity to the **Main** panel to create a new flowchart workflow.

1. In the **Name** field type a name for the workflow, such as “First Flowchart,” and click **Create**. The **Main** panel is updated accordingly.
2. Create two **Int32** variables (intRandomNumber, intGuessNumber) and a **String** one (strMessage).
3. Set the default value of the strMessage variable to "Guess a number from 1 to 999.0". The intRandomNumber stores a random number between 1 and 999, intGuessNumber stores the user's guess, and strMessage stores the message that is going to be displayed to prompt the user.

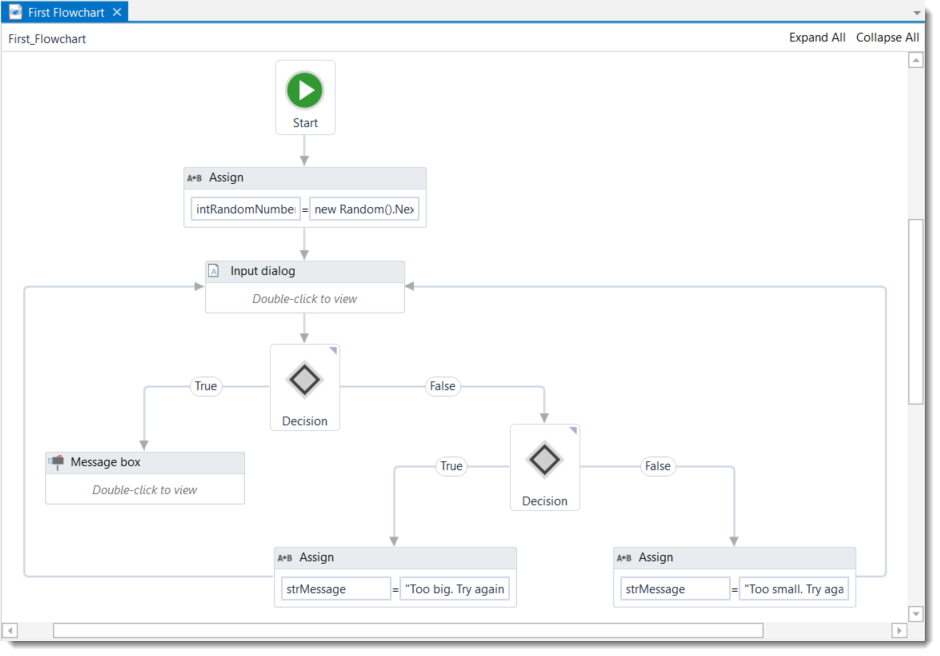


1. Add an **Assign** activity to the **Main** panel, and connect it to the **Start** node.
2. In the **Properties** panel, in the **To** field add the intRandomNumber variable.
3. In the **Value** field, type new Random().Next(1,999).

**Note:** This field uses the Random() function to generate a random number between 1 and 999.

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

The final workflow should look as in the screenshot below.



[Click here to download this example.](https://www.uipath.com/hubfs/Documentation/WorkflowExamples/FirstFlowchart.xaml?t=1492088952431)

State Machines

A state machine is a type of workflow 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.

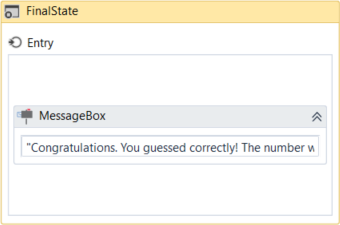
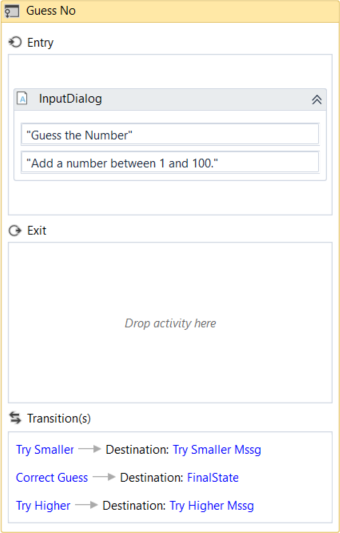
Another important aspect of state machines are transitions, as they also enable you to add conditions based on which to jump from one state to another. These are represented by arrows or branches between states.

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

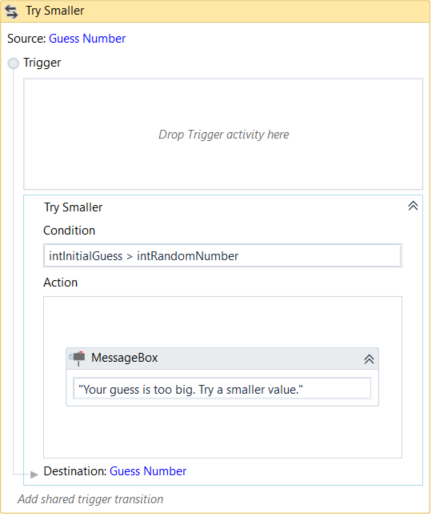
**Note:** You can only create one initial state, yet it is possible to have more than one **Final State**.

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.

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



Example of How to Use a State Machine

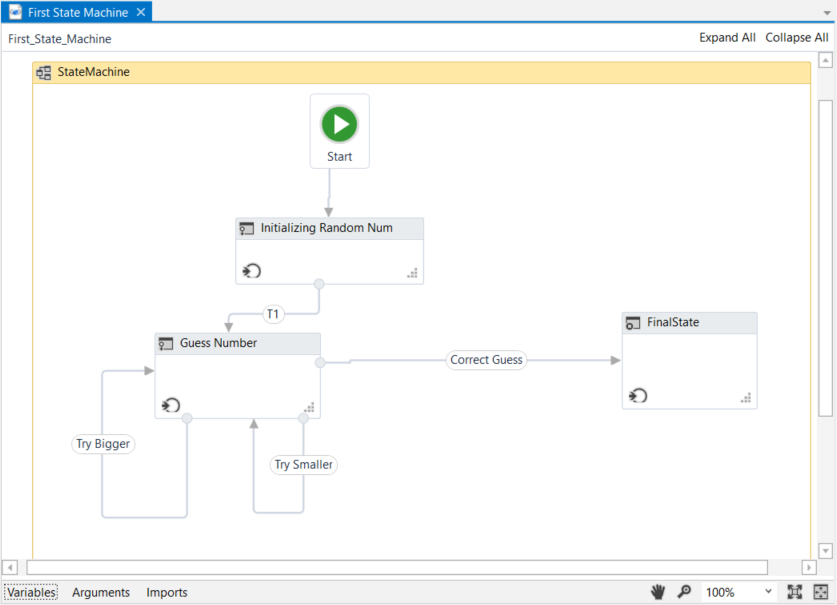
To exemplify how to use a state machine, we are going to build the guessing game we did in the previous chapter, only we will try to guess a number between 1 and 100.

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

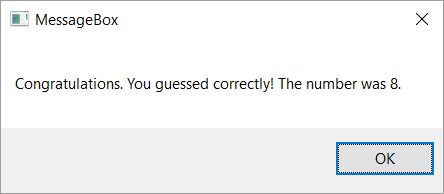
**Note:** You can also add a **State Machine** activity to the **Main** panel to create a new state machine workflow.

1. Create two integer variables, intGuess and intRandomNumber. The first variables stores your guess, while the second stores the random number.
2. Add a **State**activity to the **Main** panel and connect it to the **Start** node. This is the intial state, and it is used to generate the random number.
3. Double-click the activity. This **State** activity is displayed explanded in the **Main** panel.
4. In the **Properties** panel, in the **DisplayName** field, type Initializing Random Number. This enables you to easily tell states apart.
5. In the **Entry** section, add an **Assign** activity.
6. In the **To** field, add the intRandomNumber variable.
7. In the **Value** field, type new Random().Next(1.100). This expression generates a random number.
8. Return to the main workflow view and add a new **State** activity.
9. Connect it to the previously added activity.
10. Double-click the last added **State** activity. This activity is displayed expanded in the **Main** panel.
11. In the **Properties** panel, in the **DisplayName** field, type Guess Number. This state is used to prompt the user to guess a number.
12. In the **Entry** section, add an **Input Dialog** activity.
13. Select the **Input Dialog**, and in the **Properties** panel, add an appropriate **Label** and **Title** to prompt the user to guess a number between 1 and 100.
14. In the **Result** field, add the intGuess variable. This variable stores the user’s guess.
15. Return to the main workflow view and create a transition that points from the **Guess Number** state to itself.
16. Double-click the transition. The transition is displayed expanded in the **Main** panel.
17. In the **Properties** panel, in the **DisplayName** field, type Try Smaller. This message is displayed on the arrow, enabling you to run through your workflow easier.
18. In the **Condition** section, type intGuess > intRandomNumber. This verifies if the user’s guess is bigger than the random number.
19. In the **Action** section, add a **Message Box** activity.
20. In the **Text** field, type something similar to "Your guess is too big. Try a smaller number." This message is displayed when the user’s guess is bigger than the random number.
21. Return to the main workflow view and create a new transition that points from the **Guess Number** state to itself.
22. Double-click the transition. The transition is displayed expanded in the **Main** panel.
23. In the **Properties** panel, in the **DisplayName** field, type "Try Bigger.". This message is displayed on the arrow, enabling you to run through your workflow easier.
24. In the **Condition** section, type intGuess < intRandomNumber. This verifies if the guess is smaller than the random number.
25. In the **Action** section, add a **Message Box** activity.
26. In the **Text** field, type something similar to "Your guess is too small. Try a bigger number.". This message is displayed when the users guess is smaller than the random number.
27. Return to main workflow view and add a **Final State** activity to the **Main** panel.
28. Connect a transition from the **Guess Number** activity to the **Final State**.
29. In the **Properties** panel, in the **DisplayName** field, type Correct Guess.
30. In the **Condition** field, type intGuess = intRandomNumber. This is the condition on which this workflow steps to the final state and end.
31. Double-click the **Final State** activity. It is displayed expanded in the **Main** panel.
32. In the **Entry** section, add a **Message Box** activity.
33. In the **Text** field, type something similar to "Congratulations. You guessed correctly! The number was " + intRandomNumber.ToString + ".". This is the final message that is to be displayed, when the user correctly guesses the number.

The final workflow should look as in the following screenshot.



1. Press F5. The workflow is executed correctly.



[Click here to download this example.](https://www.uipath.com/hubfs/Documentation/WorkflowExamples/FirstStateMachine.xaml?t=1492088952431)

VARIABLES

# Managing Variables

## Managing Variables

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

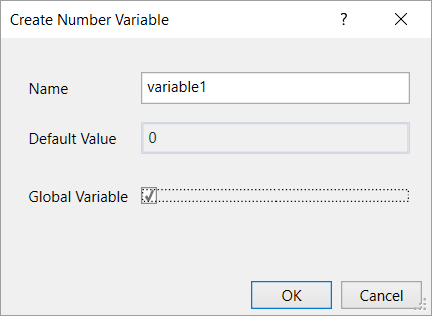
The data stored within a variable is called a value, and it can be of multiple types. In UiPath, we support a large amount of types, ranging from generic value, text, number, data table, time and date, to UiElements.

### Creating Variables

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

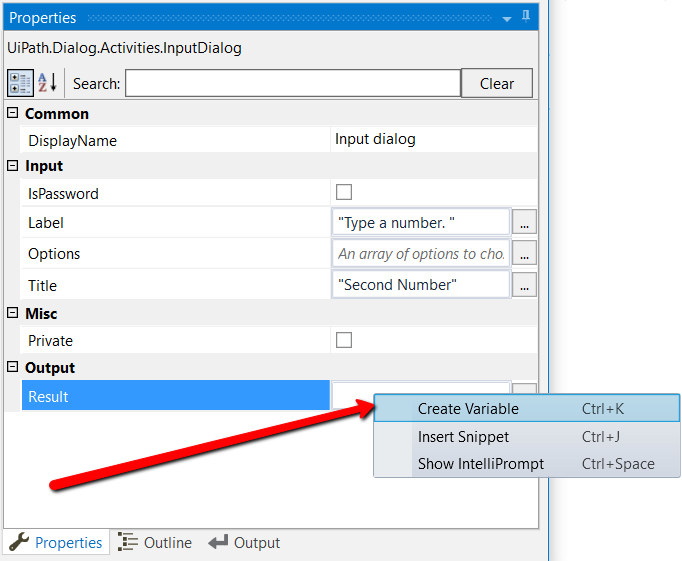
##### **From the Design ribbon:**

1. On the **Design** ribbon tab, in the **Variables** group, select **Create Variable > [Type of variable]**. The **Create Variable** window is displayed.



1. Fill in the required fields and click **OK**. The variable is created and you can view and edit it in the **Variables** panel.

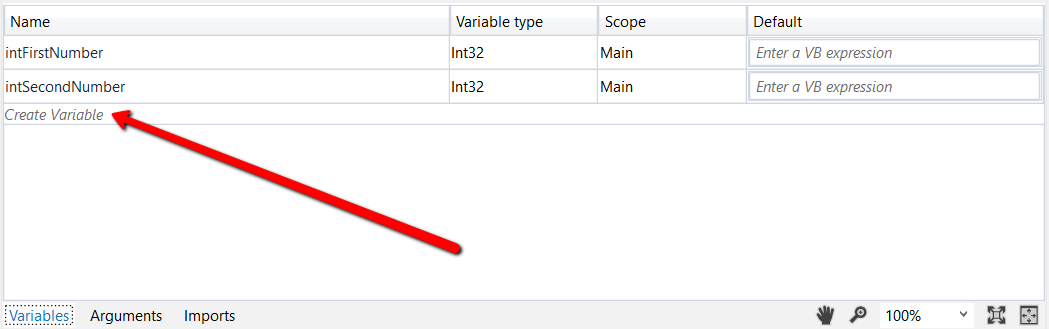
##### **From the context menu or with a keyboard shortcut:**



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

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

##### **From the Variables panel:**



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

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

### Removing Variables

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

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

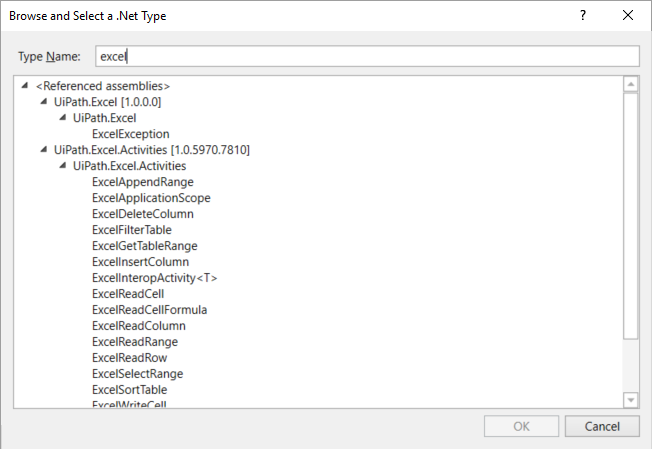
You can also remove all the variables that are not used in your currently opened workflow:

1. On the **Design** ribbon tab, in the **Variables** group, select **Manage Variables > Remove Unreferenced**. Note that the **Variables** panel only contains the variables used in your workflow.

### Browsing for .Net Variable Types

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

1. In the **Variable** panel, from the **Variable Type** drop-down list, select **Browse for Types**. The **Browse and Select a .Net Type** window is displayed.



1. In the **Type Name** field, type a keyword for the variable you are looking for, such as excel. Note that the result section is updated, displaying all the .Net variable types that contain your keyword.
2. Select one and click **OK**. A new variable is created with the selected type and is displayed in the **Variables** panel.

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

### Promoting Variables to Global Scope

Some variables, when created directly in an activity (from the context menu of an activity), are automatically given the smallest scope they belong to. To make them available in your entire workflow, do the following:

1. Click the smallest container in a workflow.
2. On the **Design** ribbon tab, in the **Variables** group, select **Manage Variables > Promote to Global Scope**. All the variables used in the selected container now have a global scope.

Naming Best Practices

When creating very large workflows, it can be very easy to forget what every variable does. That is why it is important to have a good naming system in place.

We recommend that you always use descriptive names, such as userName for a variable that stores the name of a user.

Additionally, you might want to keep track of the type of variable you create, and that is why adding a short descriptor in the front of each variable name can be useful, such as int for integers.

Finally, we recommend that you write variable names in [camel case](https://en.wikipedia.org/wiki/CamelCase), so that you can read them easier.

Example of how to name your variables:

|  |  |
| --- | --- |
| **Variable Type** | **Variable Name** |
| Generic Value | genVariableName |
| Text | strVariableName |
| Number | intVariableName |
| True or False | boolVariableName |
| Date and Time | timVariableName |
| Data Table | datVariableName |

The Variables Panel

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

|  |  |
| --- | --- |
| **Field** | **Description** |
| **Name** | Mandatory.  The name of your variable. If you do not add a name to a variable, one is automatically generated. For more information on how to name your variables, see [Naming Best Practices](https://www.uipath.com/guides/naming-best-practices). |
| **Variable Type** | Mandatory.  Enables you to choose the type of variable. The following options are available:   * [Boolean](https://www.uipath.com/guides/true-or-false-variables) * [Int32](https://www.uipath.com/guides/number-variables) * [String](https://www.uipath.com/guides/text-variables) * Object * [Generic Value](https://www.uipath.com/guides/generic-value-variables) * [Array of [T]](https://www.uipath.com/guides/array-variables) * [Browse for Types](https://www.uipath.com/guides/managing-variables) |
| **Scope** | Mandatory.  The are in which a variable is available, such as a specific activity. By default, they are available in the entire workflow. |
| **Default** | Optional.  The default value of the variable. If this field is empty, the variable does not have a default value. |

Types of Variables

# Generic Value Variables

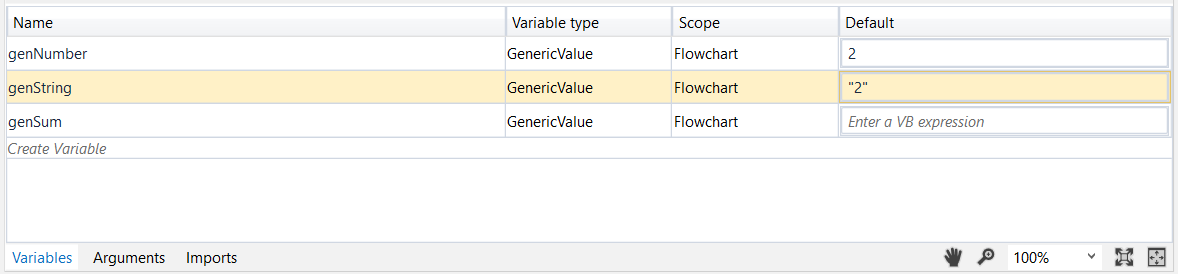
The generic value variable is a type of variable with a wide range that can store any kind of data, including text, numbers, dates and arrays, and is particular to UiPath Studio.

Generic value 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 workflow.

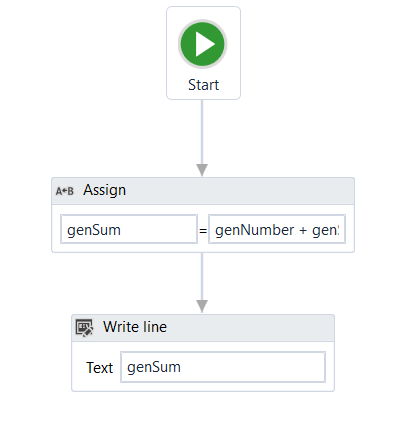
### Example of Using a Generic Value Variable

To demonstrate how a generic value variable can be converted and used, let's create a workflow that displays in the **Output** panel the sum of two numbers, using generic value variables with different types of values.

1. Create a flowchart.
2. Create three generic value variables, genNumber, genString and genSum.
3. In the **Default** column for the genNumber variable, type 2, and for the genStringvariable, type"2." The first value is interpreted as as integer, and the second one as a string.



1. Add an **Assign** activity to the **Main** panel and connect it to the **Start** node.
2. In the **Properties** panel, in the **To** field, enter the genSum variable.
3. In the **Value** field, type genNumber+genString.
4. Add a **Write Line** activity and connect it to the **Assign** one.
5. In the **Properties** panel, in the **Text** field, enter the genSum variable.



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

# Text Variables

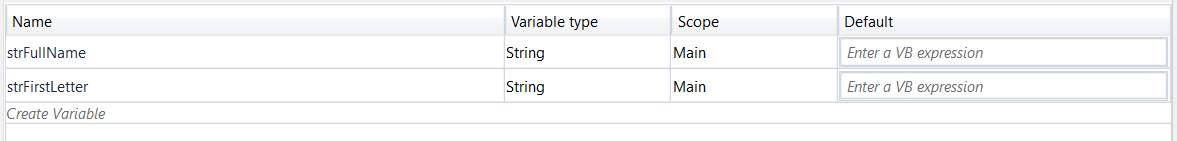
A text or string variable is a type of variable that can store only [strings](https://en.wikipedia.org/wiki/String_%28computer_science%29). These types of variables can be used, for example, to store names, passwords or information extracted from a table.

**Note:** All strings in UiPath Studio have to placed in between quotes.

### Example of Using a Text Variable

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

1. Create a sequence.
2. Create two simple string variables, strFullName and strFirstLetter.

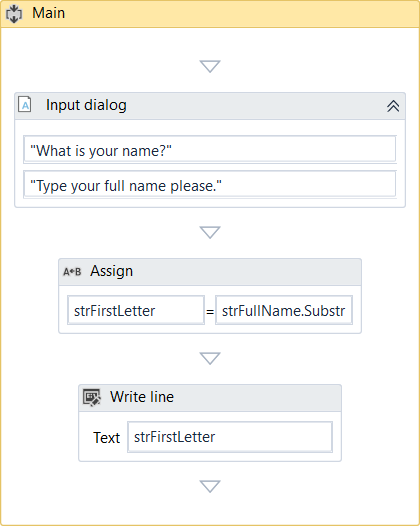


1. Add an **Input Dialog** activity to the **Main** panel.
2. In the **Properties** panel, in the **Label** field, type "Type your full name please.".
3. In the **Title**field, type "What is your name?".
4. In the **Result** field, add the StrFullName variable. This variable stores whatever the user writes when prompted with the **Input Dialog** activity.
5. Add an **Assign** activity under the **Input Dialog** one.
6. In the **Properties** panel, in the **To** field, add the strFirstLetter variable.
7. In the **Value** field, type strFullName.Substring(0,1). The strFirstLettervariable is assigned the new value created by the strFullName.Substring(0,1)expression.

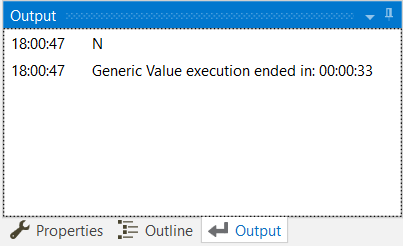
**Note:** This field uses the Substring() function to find the first character from the string added by the user in the **Input Dialog**.

1. Add a **Write Line** activity under the **Assign** one.
2. In the **Properties** panel, in the **Text** field, enter the strFirstLetter variable. The **Output** panel is going to display the first letter of what the user wrote in the **Input Dialog**.

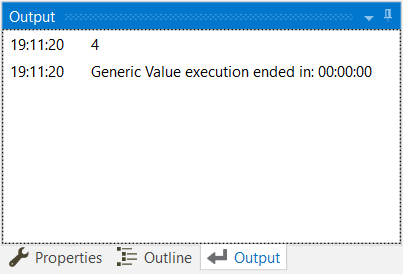
The workflow should look as in the following screenshot.



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



[Click here to download this example.](https://www.uipath.com/hubfs/Documentation/WorkflowExamples/TextVariable.xaml?t=1492088952431)



This means that UiPath Studio knows that the genNumber is an integer and knows how to transform the generic value genString variable to an integer, so that it can add it to the first one.

However, keep in mind that this was our goal from the beginning. If we wanted to display the two variables in the **Output** panel as strings using this exact method, it would not have worked.

[Click here to download this example.](https://www.uipath.com/hubfs/Documentation/WorkflowExamples/GenericValueVariable.xaml?t=1492088952431)

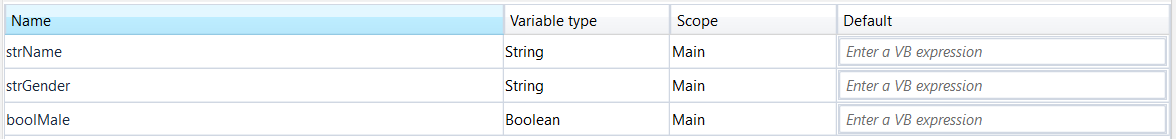
# True or False Variables

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

### Example of Using a True or False Variable

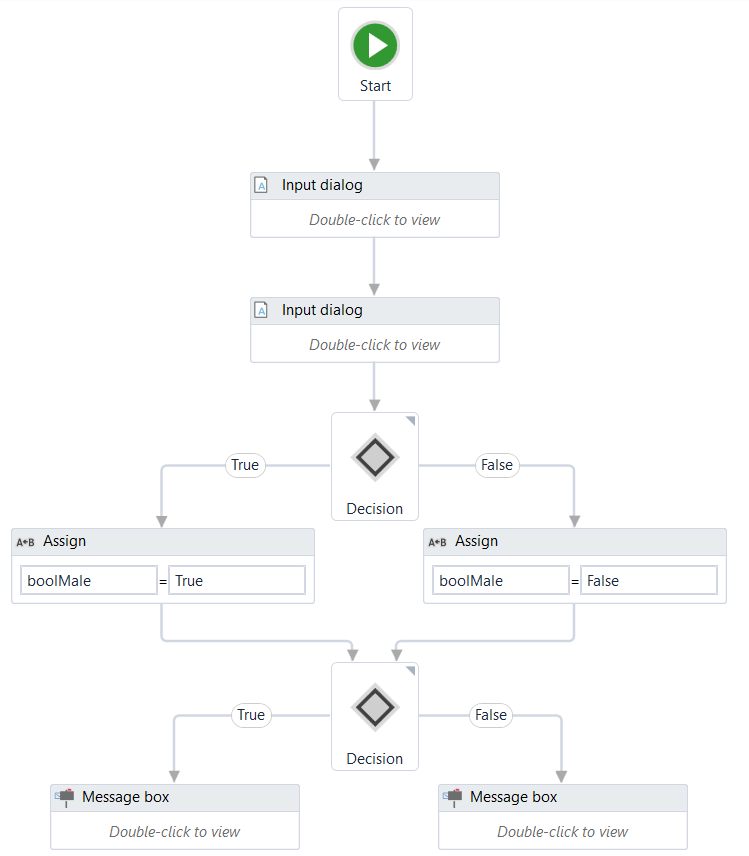
To exemplify how you can work with true or false variables, we are going to create a workflow that asks the user for his name and gender, and displays the results in another window.

1. Create a new workflow.
2. Create two string variables, strName and strGender. The first is going to be used to store the name of the user, and the second to store the user’s gender.
3. Create a boolean variable, boolMale. This variable is used to verify if the user is a male.

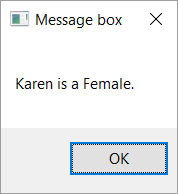


1. Add an **Input Dialog** activity to the **Main** panel and connect it to the **Start** node.
2. In the **Properties** panel, in the **Label** field type "What is your name?".
3. Add a title and, in the **Result** field, add the strName variable.
4. Add another **Input Dialog** activity and connect it to the previous one.
5. In the **Properties** panel, in the **Label** field type "What is your gender?".
6. Add a title and, in the **Result** field, add the strGender variable.
7. Add a **Flow Decision** activity to the **Main** panel, and connect it to the second **Input Dialog**.
8. In the **Properties** panel, in the **Condition** field, type strGender = "Male" or strGender = "male". This activity checks if the user is a male or female.
9. Add two **Assign**activities.
10. Connect one to the **True** branch of the **Flow Decision** activity.
11. In the **Properties** panel, in the **To** field enter the boolMale variable.
12. In the **Value** field, type **True**. This assigns the **True** value to the boolMale variable when the strGender = "Male" or strGender = "male" condition is met.
13. Connect the second **Assign** activity to the **False** branch of the **Flow Decision**.
14. In the **Properties** panel, in the **To** field, enter the boolMale variable.
15. In the **Value** field, type **False**. This assigns the **False** value to the boolMale variable when the strGender = "Male" or strGender = "male" condition is not met.
16. Add a new **Flow Decision** and connect the previously added **Assign** activities to it.
17. In the **Properties** panel, in the **Condition** field, type boolMale = True.
18. Add a **Message Box** activity and connect it to the **True** branch of the **Flow Decision**.
19. In the **Properties** panel, in the **Text** field, type strName + " is a " + strGender + ".". This message displays the name of the user and its gender, if boolMale is true.
20. Add another **Message Box** activity and connect it to the **False** branch of the **Flow Decision**.
21. In the **Properties** panel, in the **Text** field, type strName + " is a " + strGender + ".". This message displays the name of the user and its gender, if boolMale is false.

The final workflow should look like in the following screenshot.



1. Press F5. The workflow is executed. Note that the final **Message Box** displays the message as expected.



[Click here to download this example.](https://www.uipath.com/hubfs/Documentation/WorkflowExamples/TrueOrFalseVariable.xaml?t=1492088952431)

# Number Variables

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

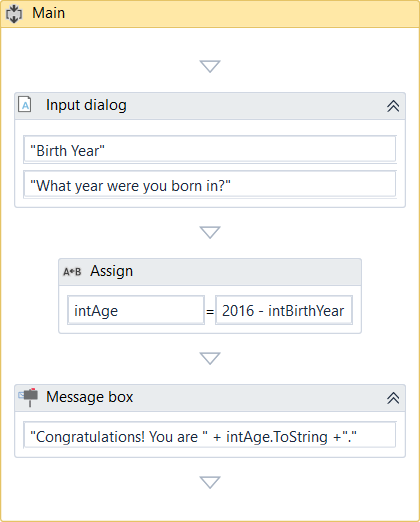
### Example of Using a Number Variable

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

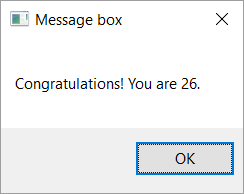
1. Create a new sequence.
2. Create two Int32 variables, intBirthYear and intAge. The first stores the user's birth year and the second, the user's age.
3. Add an **Input Dialog** activity to the sequence.
4. In the **Properties** window, type an appropriate title and label.
5. In the **Result** field,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

**Note:** The .ToString method converts the integer stored in the intAge variable to a string and display it as such.

The final workflow should look as in the following screenshot.



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



[Click here to download this example.](https://www.uipath.com/hubfs/Documentation/WorkflowExamples/NumberVariable.xaml?t=1492088952431)

# Array Variables

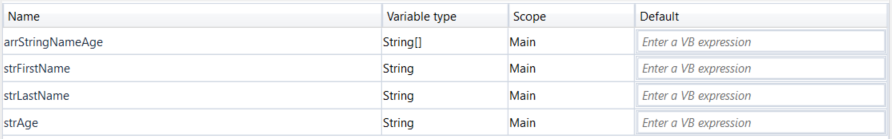
The array variable is a type of variable which enables you to store multiple values of the same type.

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

### Example of Using an Array Variable

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

1. Create a new sequence.
2. Create three string variables, strFirstName, strLastName and strAge, in which to store the information gathered from the user.
3. Create an array of strings variable called arrStringNameAge.



1. Add an **Input Dialog** activity to the **Main** panel.
2. In the **Properties** panel, fill in the **Label** and **Title** fields to ask for the user’s first name.
3. In the **Result** field, add the strFirstName variable. This variable stores the first name of the user.
4. Add another **Input Dialog** activity under the previous one.
5. In the **Properties** panel, fill in the **Label** and **Title** fields to ask for the user’s last name.
6. In the **Result** field, type the strLastName variable. This variable is going to store the last name of the user.
7. Add another **Input Dialog** activity under the previous one.
8. In the **Properties** panel, fill in the **Label** and **Title** fields to ask for the user’s age.
9. In the **Result** field, type the strAge variable. This variable is going to store the age of the user.

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

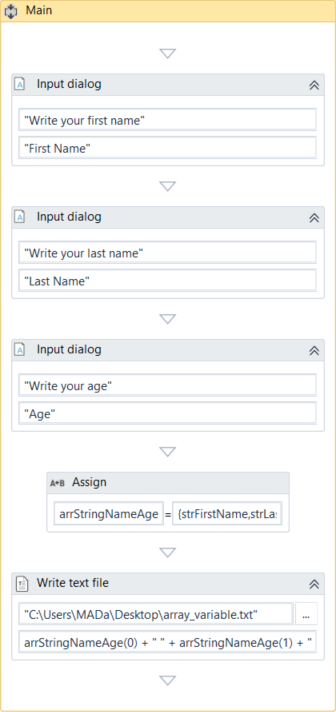
1. Add an **Assign** activity under the last **Input Dialog**.
2. In the **Properties** panel, in the **To** field, type the arrStringNameAge variable.
3. In the **Value** field, type {strFirstName,strLastName,strAge}. This **Assign**activity enables you to store all the values from the initial string variables in the arrStringNameAge one.
4. Add a **Write Text File** activity under the **Assign** one.
5. In the **Properties** panel, in the **FileName** field, type the path of the file you want to write to between quote marks, such as "C:\Users\MADa\Desktop\array\_variable.txt".

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

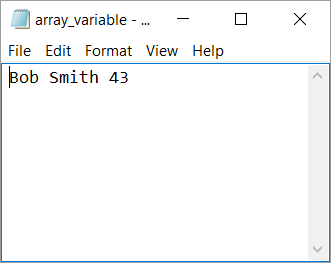
1. In the **Text** field, type arrStringNameAge(0) + " " + arrStringNameAge(1) + " " +arrStringNameAge(2) + " ".

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

The final workflow should look as in the following screenshot.



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



[Click here to download this example.](https://www.uipath.com/hubfs/Documentation/WorkflowExamples/ArrayVariables.xaml?t=1492088952431)

# Date and Time Variables

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

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

### Example of Using a Date and Time Variable

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

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

**Note:** The default value attributed to the timSpan variable uses the day.hh:mm:ssformat.

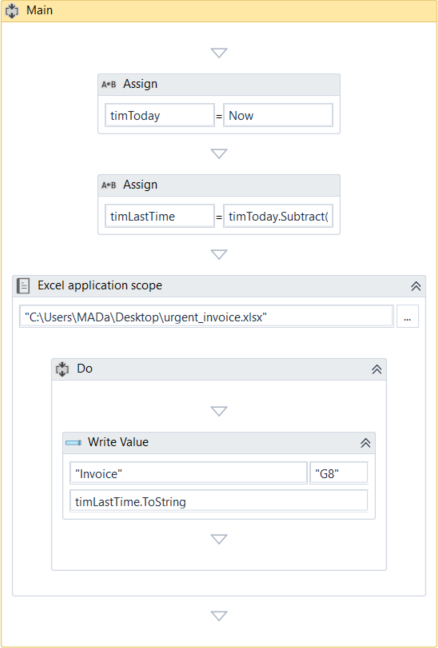
1. Add an **Assign** activity to the **Main** panel.
2. In the **Properties** panel, in the **To** field, add the timToday variable.
3. In the **Value** field, type **Now**. This gives you the date and time when the workflow is executed, in the dd/MM/yyyy and hh:mm:ss formats.
4. Add another **Assign** activity under the previous one.
5. In the **Properties** panel, in the **To** field, add the timLastTime variable.
6. In the **Value** field, type timToday.Subtract(timSpan). This is going to subtract the default value of the timSpan variable from the current date and time, stored in the timToday variable.
7. Add an **Excel Application Scope** activity under the last **Assign** one.

**Note:** If you do not have Excel activities installed on your version of UiPath Studio, use the Manage Packages functionality to get them.

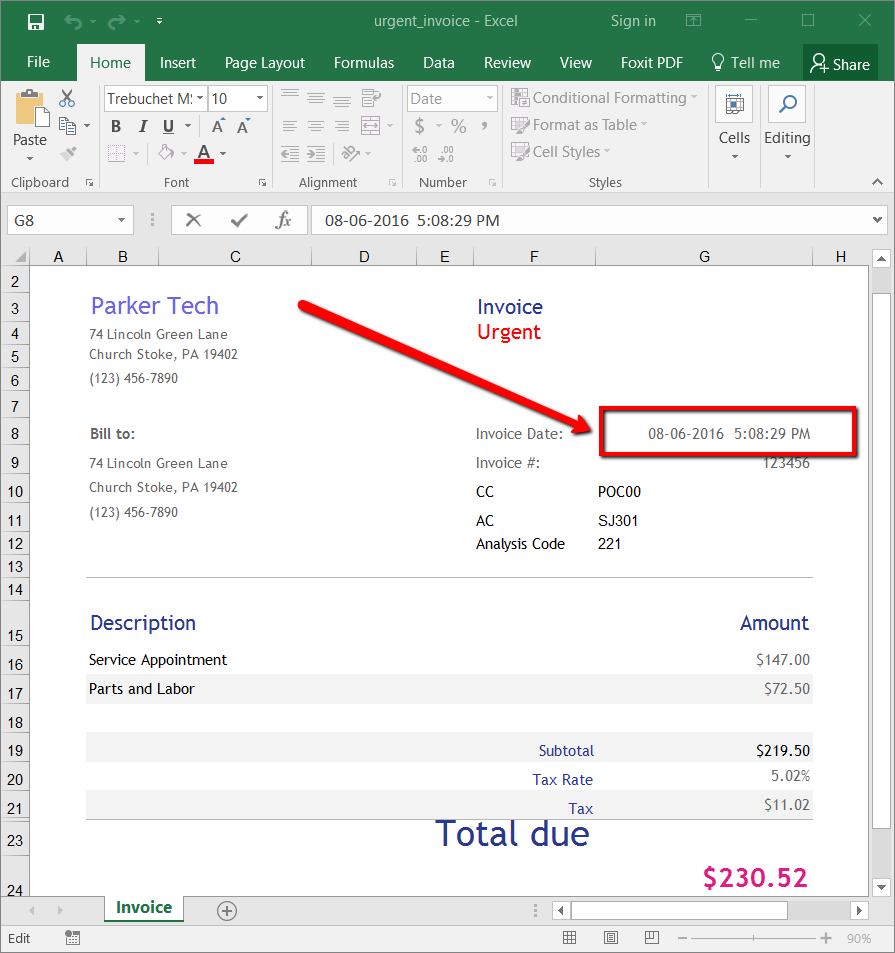
1. In the **Properties** panel, in the **WorkbookPath** field, type the path of the Excel file you want to write to, between quotation marks. In our case, "C:\Users\Username\Desktop\urgent\_invoice.xlsx".

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

1. Add a **Write Value** activity in the **Excel Application Scope** activity.
2. In the **Properties** panel, in the **Range** field, type the coordinates of an Excel cell between quotation marks. In our case, “G8.”
3. In the **Sheet Name** field, type the name of the sheet in which you want to write. In our case, “Invoice”. Note that if the sheet does not exist, it is going to be created.
4. In the **Value** field, type timLastTime.ToString. This transforms the value of the timLastTime variable to a string and writes it to the coordinates previously given.

The final workflow should look as in the following screenshot.

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



[Click here to download this example.](https://www.uipath.com/hubfs/Documentation/WorkflowExamples/DateAndTimeVariable.xaml?t=1492088952431)

# Data Table Variables

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

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

### Example of Using Data Table Variables

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

The initial file is a database of people, their age, location and e-mail address. In this example, we are going to extract their names and e-mail addresses and append them to an Excel spreadsheet that already contains similar information.

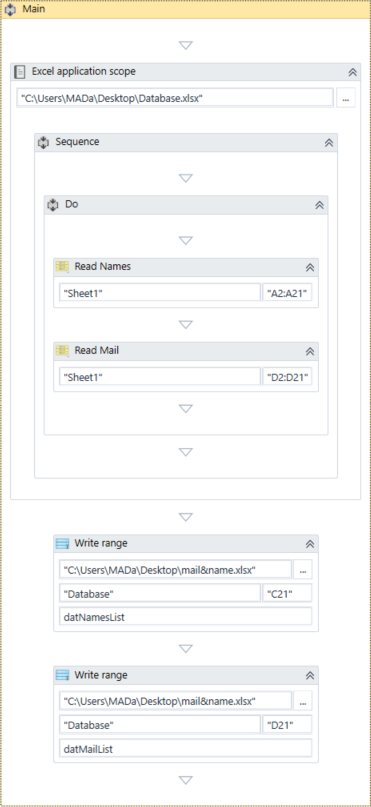
1. Create a new sequence.
2. Add an **Excel Application Scope** activity to the sequence. This activity is required for most of the Excel-related activities.

**Note:** If you do not have Excel activities installed on your version of UiPath Studio, use the [Manage Packages](https://www.uipath.com/guides/managing-packages) functionality to get them.

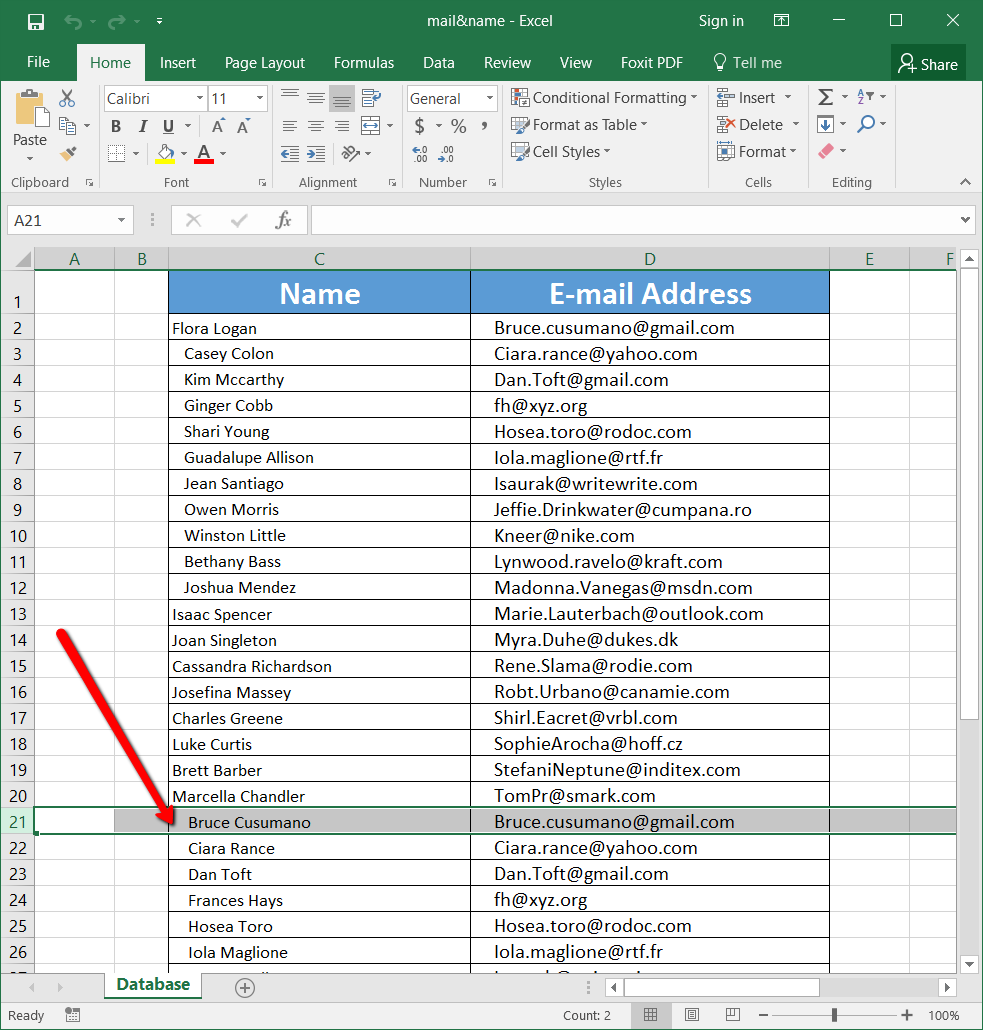
1. Create two data table variables, datNamesList and DatMailList. These are going to be used to store information from the initial Excel spreadsheet.
2. In the **Properties** panel, in the **WorkbookPath** field, type the path of the initial Excel file to be used, between quotation marks.
3. Add two **Read Range** activities and place them one under the other, in the **Excel Application Scope** activity. These are used to get information from the initial spreadsheet.
4. Select the first **Read Range** activity and, in the **Properties** panel, in the **Range** field, type “A2:A21.”. These are the Excel table coordinates that tell UiPath Studio from where to extract information.
5. In the **SheetName** field, do not make any changes as the name of our sheet is the default one, Sheet1.
6. In the **DataTable** field, type the name of the first data table variable, datNamesList. This variable stores all the information available between the A2 and A21 rows.
7. (**Optional**) Change the value in **DisplayName** field to Read Names, so you can easily tell apart this activity from the second one.
8. Select the second **Read Range** activity, and in the **Properties** panel, in the **Range** field, type “D2:D21.” These are the Excel table coordinates that contain the e-mail information we want to extract.
9. In the **DataTable** field, specify the datMailList variable. This variable retains all the mail information we require.
10. Add a **Write Range** activity to the **Main** panel, under the **Excel Application Scope**. This activity is used to write the stored information to another Excel file.

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

1. In the **Properties** panel, in the **WorkbookPath** field, type the path of the Excel file to be used to store all the information gathered at the previous steps.
2. In the **DataTable** field, type the datNamesList variable.
3. In the **SheetName** field type Database, and in the **StartingCell**, type “C21.“ This is the starting cell in which information from the initial file is to be added.
4. Add another **Write Range** activity and place it under the first one.
5. In the **Properties** panel, fill in the **WorkbookPath** and **SheetName** fields as for the previous **Write Range** activity.
6. In the **Starting Cell** field, type "D21."
7. In the **DataTable** field, type the datMailList variable.



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



[Click here to download this example.](https://www.uipath.com/hubfs/Documentation/WorkflowExamples/DataTableVariables.xaml?t=1492088952431)

IMPORTED NAMESPACES

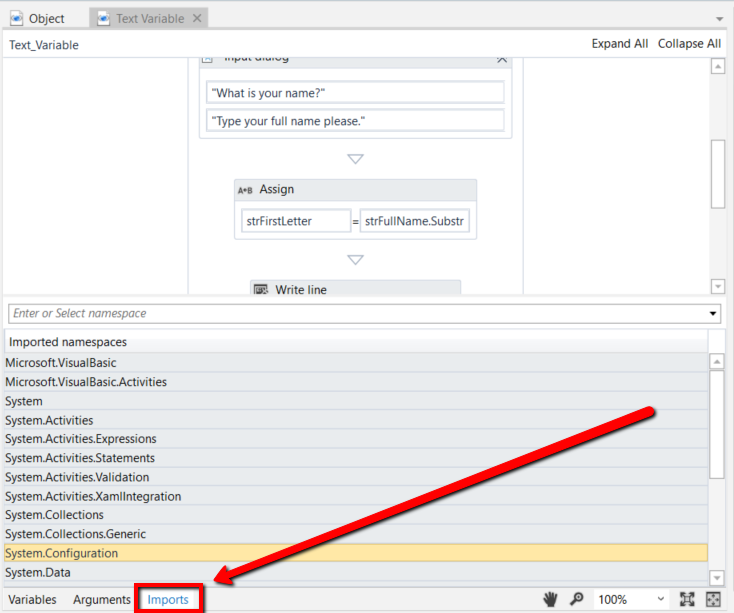
About Imported Namespaces

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

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

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

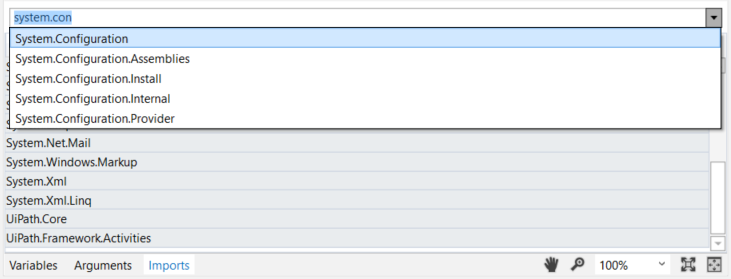
To open this panel, click **Imports** in the **Main** panel.



Importing New Namespaces

To add new namespaces to your library:

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



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

To remove a namespace, select it and press Delete. Note that namespaces can only be deleted if they are invalid. For example, you can delete a namespace if the assembly that contains it is no longer referenced by the project.

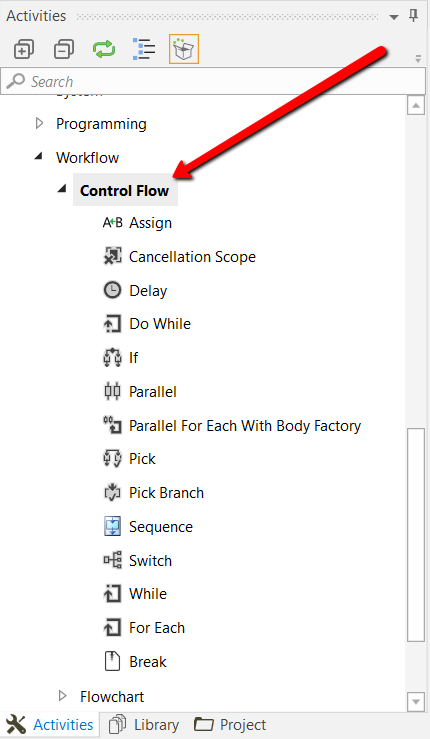
CONTROL FLOW

About Control Flow

An important aspect of successfully working with UiPath Studio is understanding and knowing how to control your workflow. As in computer science, in UiPath this concept is referred to as control flow.

A proper control flow can be achieved through the intelligent use of variables, and of certain activities.

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



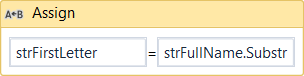
They enable you to define rules and automate decisions for a given workflow, through if…else or for each statements or loops, as well as add delays so that you can perfectly time two activities.

Loops represent an important part of workflows as they enable you to easily check dependencies between variables, activities and conditions. They are created once and enable you to iterate data a specified number of times, until a condition is met, once for each item in a collection or indefinitely.

Control flow Activities

# The Assign Activity

The **Assign** activity is a pretty 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 [Do While Activity](https://www.uipath.com/guides/the-do-while-activity) chapter), sum up the value of two or more variables and assign the result to another variable (see the example in the [Generic Value Variables](https://www.uipath.com/guides/generic-value-variables) chapter), assign values to an array (see the [Array Variables](https://www.uipath.com/guides/array-variables) chapter) 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

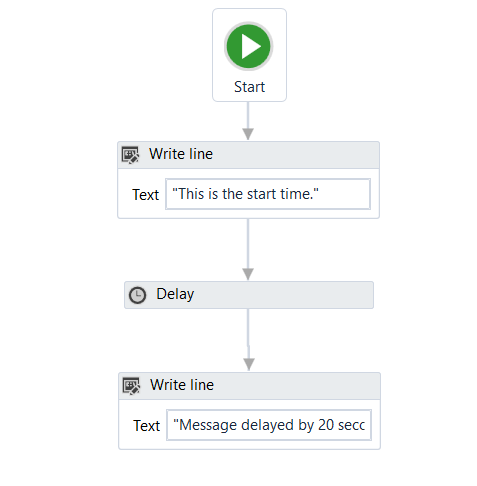
The **Delay** activity enables you to pause the workflow for a custom period of time (in the hh:mm:ss format). This activity proves itself quite useful in workflows that require good timing, such as waiting for a specific application to start or waiting for some information to be processed so that you can use it in another activity.

## Example of Using the Delay Activity

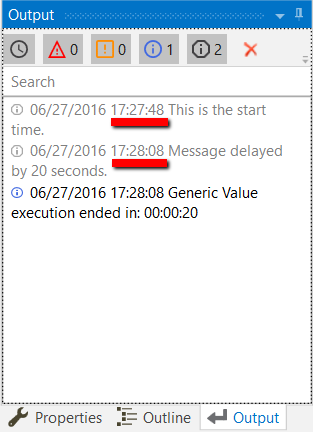
To exemplify how you can use the **Delay** activity, lets create a workflow that writes two messages to the **Output** panel, with a delay of 20 seconds between them.

1. Create a new flowchart.
2. Add a **Write Line** activity and connect it to the **Start** node.
3. Select the activity, and in the **Text** field, type "This is the start time.".
4. Add a **Delay** activity and connect it to the previously added activity.
5. Select the activity, and in the **Properties** panel, in the **Duration** field, type 00:00:20. This is the 20 seconds delay that is going to be between the two logged messages.
6. Add another **Write Line** activity and connect it to the workflow.
7. In the **Text** field, type "Message delayed by 20 seconds.".

The final workflow should look as in the following screenshot.



1. Press F5. The workflow is executed. Note that, in the **Output** panel, the two messages added in the Write Line activities are written twenty seconds apart.



[Click here to download this example.](https://www.uipath.com/hubfs/Documentation/WorkflowExamples/Delay.xaml?t=1492088952431)

# The Do While Activity

The **Do While** activity enables you to execute a specified part of your workflow while a condition is met. When the specified condition is no longer met, the workflow exists the loop.

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

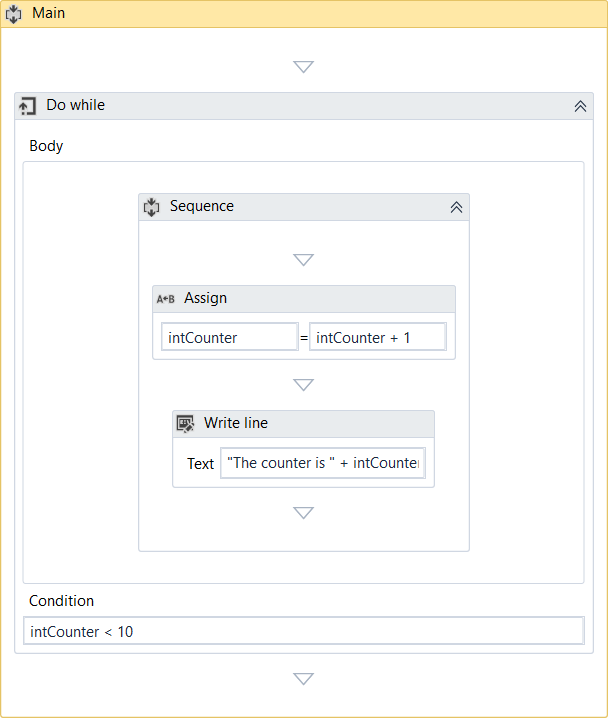
**Note: Do While** activities are evaluated only after the body has been executed once.

## Example of Using a Do While Activity

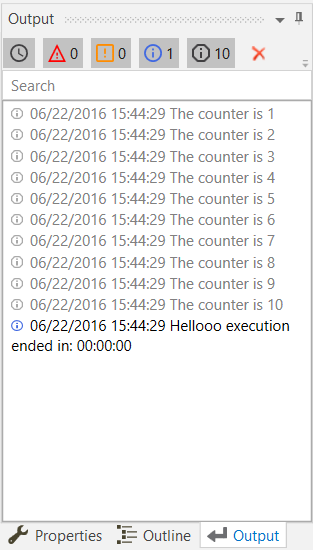
To exemplify how to use a **Do While** activity, let's create a workflow that increments an integer variable from 0 to 10, and displays a message every time it is incremented.

1. Create a new sequence.
2. Create an integer variable, intCounter, with a default value of 0.
3. Add a **Do While** activity to the **Main** panel.
4. Select the **Assign** activity, and in the **Properties** panel, in the **To** field, add the intCounter variable.
5. In the **Value** field, type intCounter + 1. This helps you increment the value of the variable with one.
6. Add a **Write Line** activity, under the **Assign** one.
7. In the **Text** field, type “The counter is ” + intCounter.ToString. This writes the value of the counter in the **Output** panel each time it is incremented.
8. In the **Condition** section of the **Do While** activity, type intCounter < 10. The body of the **Do While** activity is repeated until the value of the intCounter variable is bigger than 10.

The final workflow should look as in the following screenshot.



1. Press F5. The workflow is executed. Note that the **Output** panel displays the message indicated in the **Write Line** activity.



[Click here to download this example.](https://www.uipath.com/hubfs/Documentation/WorkflowExamples/DoWhile.xaml?t=1492088952431)

Top of Form

# 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.

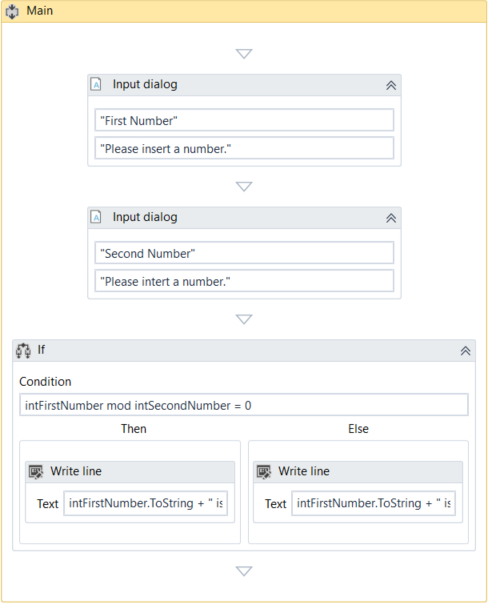
**Note:** The **If** activity is almost identical to the **Flow Decision** one. However, the latter can only be used in flowcharts.

## Example of Using an If Activity

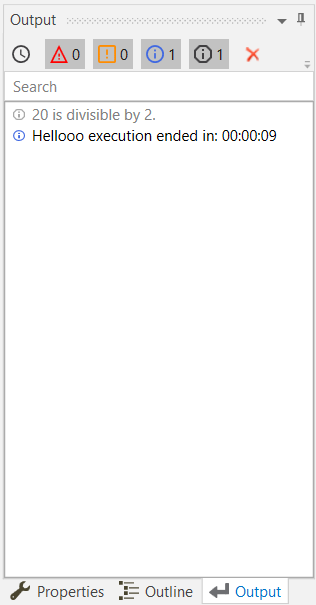
To exemplify how you can use the **If** activity, let's create a workflow that asks the user for two numbers, checks to see if one is divisible by the other, and depending on the result, displays a different message in the **Output** panel.

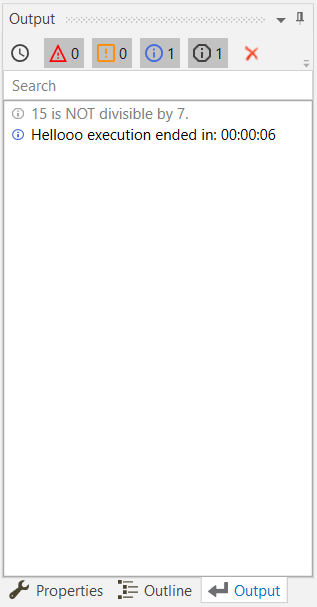
1. Create a new sequence.
2. Create two integer variables, intFirstNumber and intSecondNumber for example.
3. Add two **Input Dialog** activities to the **Main** panel.
4. In the **Properties** panel, type labels and titles for both activities and, in the **Result**fields, add the intFirstNumber and intSecondNumber variables.
5. Add an **If** activity to the **Main** panel, under the previously added **Input Dialog** ones.
6. In the **Condition** section, type intFirstNumber mod intSecondNumber = 0. This expression checks if the first number is divisible to the second one, using the mod operator.
7. In the **Then** section, add a **Write Line** activity.
8. In the **Text** field, type intFirstNumber.ToString + " is divisible by "+ intSecondNumber.ToString + ".". This is the message that is displayed if the first number is divisible by the second one.
9. In the **Else** section, add another **Write Line** activity.
10. In the **Text** field, type intFirstNumber.ToString + " is NOT divisible by "+ intSecondNumber.ToString + ".". This is the message that is displayed if the first number is not divisible by the second one.

The final workflow should look as in the following screenshot.



1. Press F5. The workflow is executed.
2. Add numbers when prompted. Note that the **Output** panel displays the result, depending on the values added in the **Input Dialog** windows.





[Click here to download this example.](https://www.uipath.com/hubfs/Documentation/WorkflowExamples/DoWhile.xaml?t=1492088952431)

# The Switch Activity

The **Switch** activity enables you to select one choice out of multiple, based on the value of a specified expression. This activity supports multiple types of arguments, such as integer, string or array, and more complex ones such as data tables.

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

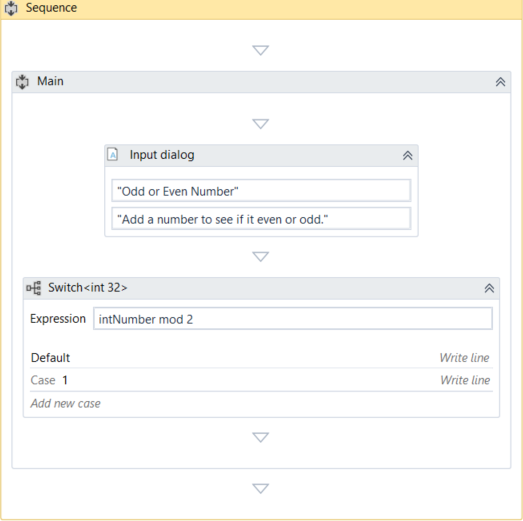
The **Switch** activity can be useful to categorize data according to a custom number of cases. For example, you can use it to store data into multiple spreadsheets or sort through names of employees.

## Example of Using a Switch Activity

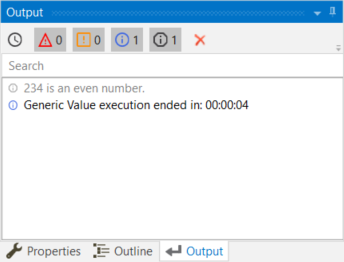
To exemplify how to use the Switch activity, we are going to create a workflow that asks the user for a number, checks if is odd or even, and depending on that, a different message is written to the **Output** panel. Since all odd numbers divided by two have a remainder equal to 1, this workflow needs only two cases (0 and 1), yet keep in mind that this activity supports multiple cases.

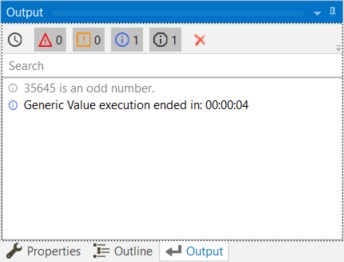
1. Create a new sequence.
2. Create an integer variable, intNumber.
3. Add an **Input Dialog** activity to the **Main** panel.
4. Add a **Title** and **Label** to prompt the user for a number.
5. In the **Result** field, add the intNumber variable.
6. Add a **Switch** activity, under the **Input Dialog**.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

The final workflow should look as in the following screenshot.



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





[Click here to download this example.](https://www.uipath.com/hubfs/Documentation/WorkflowExamples/SwitchActivity.xaml?t=1492088952431)

# 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.

## Example of Using a While Activity

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

1. Create a new sequence.
2. Create a new integer variable, intCounter, with the default value of 10.
3. Add a **While** activity to the **Main** panel.
4. In the **Condition** field, type intCounter < 100. This means that the body of the loop is going to be repeated until the value of the intCounter variable is going to be bigger than 100.
5. In the **Body** section of the **While** activity, add an **Assign** activity.
6. In the **Properties** panel, in the **To** field add the intCounter.
7. In the **Value** field, type intCounter + 1. This increments the value of the intCounter with one.
8. Add an **Append Text** activity under the **Assign** one.

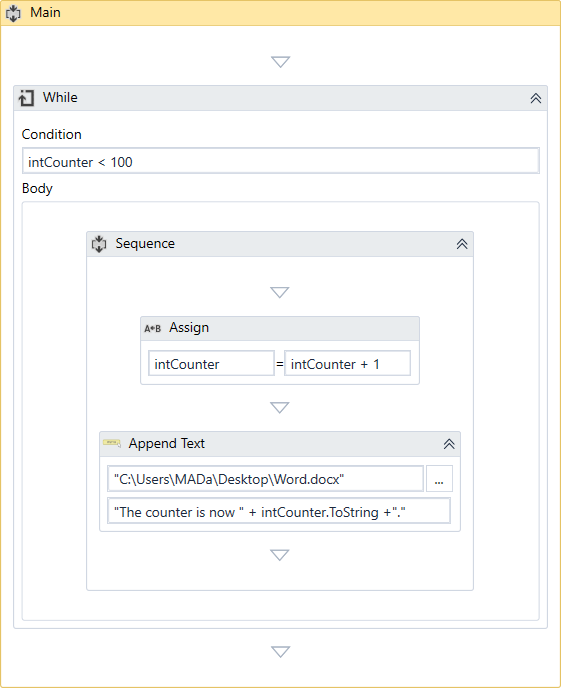
**Note:** This activity is part of the Word activities package. If you do not have it, use the [package manager functionality](https://www.uipath.com/guides/managing-packages) to install it.

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

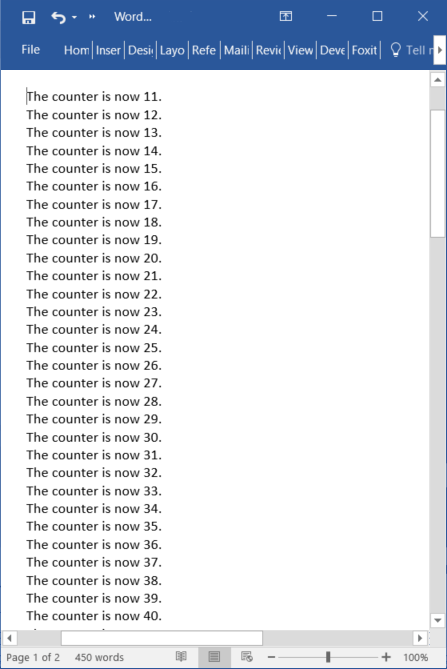
**Note:** Make sure that the Word document is not used when running the workflow, otherwise a message error is displayed and the execution is stopped.

1. In the **Text** field, type "The counter is now " + intCounter.ToString +".".

The final workflow should look as in the following screenshot.



1. Press F5. The workflow is executed.
2. Double-click the Word document specified at step 9. Note that all the numbers between 10 and 100 are written, as expected.



[Click here to download this example.](https://www.uipath.com/hubfs/Documentation/WorkflowExamples/WhileActivity.xaml?t=1492088952431)

# The For Each Activity

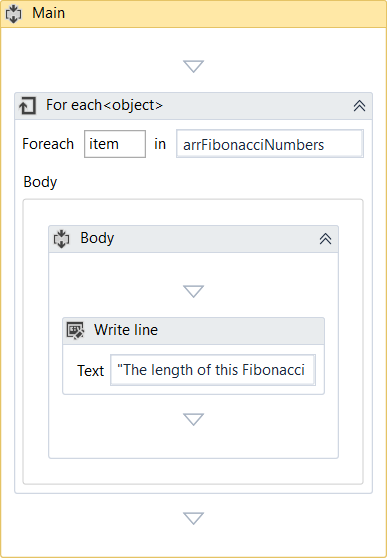
The **For Each** activity enables you to step through arrays, lists, data tables or other types of collections, so that you can iterate through the data and process each piece of information individually.

## Example of Using a For Each Activity

To exemplify how to use a **For Each** activity, we are going to create a workflow that goes through each element of an array of integers and writes the length of the array and each element to the **Output** panel.

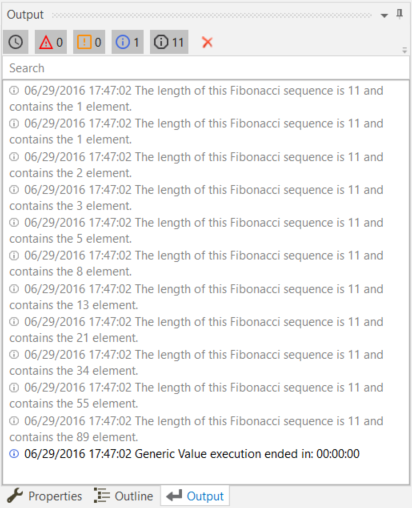
1. Create a new sequence.
2. Create an array of integers variable, arrFibonacciNumbers.
3. In the **Default** field, type the Fibonacci sequence until a desired value, such as {1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89}.
4. Add a **For Each** activity in the **Main** panel.
5. Do not make any changes to the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

The final workflow should look as in the following screenshot.



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

**Note:** The Length property enables you to find out the total number of array elements.



[Click here to download this example.](https://www.uipath.com/hubfs/Documentation/WorkflowExamples/ForEachActivity.xaml?t=1492088952431)

# The Break Activity

The **Break** activity enables you to stop the loop at a chosen point, and then continues with the next activity.

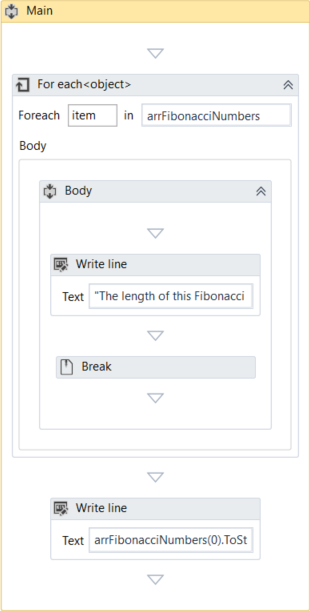
**Note:** The **Break** activity can only be used within the **For Each** activity.

## Example of Using a Break Activity

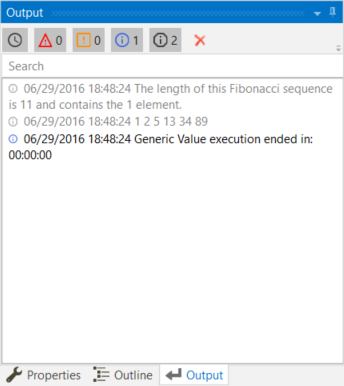
To exemplify how to use the **Break** activity we are going to build upon the workflow created for the [For Each activity](https://www.uipath.com/guides/the-for-each-activity). This new workflow writes only the first iteration of the loop and a few elements of the array to the **Output** panel.

1. In the **Body** of the **For Each** activity, under the **Write Line**, add a **Break** activity.
2. Under the **For Each**, add a new **Write Line** activity.
3. In the **Text** field, type arrFibonacciNumbers(0).ToString + " " + arrFibonacciNumbers(2).ToString + " " + arrFibonacciNumbers(4).ToString + " " + arrFibonacciNumbers(6).ToString +" " + arrFibonacciNumbers(8).ToString +" " + arrFibonacciNumbers(10).ToString + " ". This means that only the indicated elements of the array are going to be written to the **Output** panel.

The final workflow should look as in the following screenshot.



1. Press F5. The workflow is executed. Note that the **Output** panel only displays the first iteration of the loop and the specified array elements from the **Write Line** activity.



[Click here to download this example.](https://www.uipath.com/hubfs/Documentation/WorkflowExamples/ForEachActivityWithBreak.xaml?t=1492088952431)

Bottom of Form