Prerequisites

Hands-on experience you must have before doing this course

- Create UFT Scripts using various recording modes.
- Create local and shared object repository.
- Parameterize the script through data table.
- Implement checkpoints and output values according to the business requirements
- Execute scripts and analyze the results in UFT.

Concepts you must know before doing this course

- Automation Testing
  - Automation life cycle
  - Test tools at various levels of testing and challenges in automation testing.

Recommended resources to learn the prerequisite concepts

- Introduction to automation testing
- UFT Fundamentals

Hardware requirement

- **Operating System:** Windows 7 Service Pack 1(32-bit/64-bit).
- **Memory:** Minimum of 2GB when more than three add-ins are loaded simultaneously. Additional memory is required when loading more add-ins.
- **Hard-Disk Space:** 2GB of free disk space for application files and folders you must also have an additional 1GB of free disk space on the system disk(the disk on which operating system is installed)

Software requirement

- HP Unified Functional Testing version 11.5 or above.

Learning outcomes

UFT automation projects do not prefer test scripts created using record-replay method as their execution speed is slow, resource efficiency is low and they are difficult to maintain. You need to code fast, efficient and easily maintainable UFT scripts using programming techniques in VBScript.

By the end of the course you will be able to write advanced UFT scripts using VBScript which will be much more faster, efficient and highly maintainable than recorded ones.

Upon completion of this course, you will be able to

- Create UFT scripts through static and dynamic descriptive programming which results in creation of faster test scripts
- Create dictionary objects and arrays for passing data to the test scripts
- Create efficient data driven tests through database, excel and file automation
- Debug test scripts using inbuilt functionalities of UFT
- Analyze the automation framework that your project uses and classify its type.

Need for advanced UFT scripts – 1

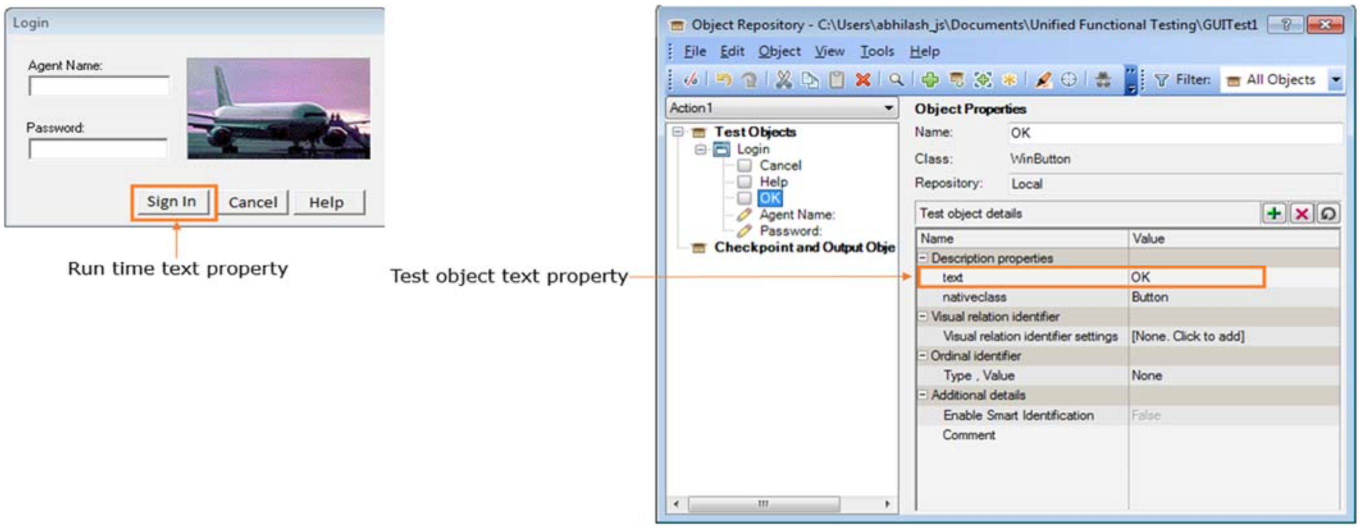
Scenario 1:

Consider that you have created UFT scripts for the following scenarios using local object repository for the flight reservation application.

1. Login
2. Login - >Insert order
3. Login->Open order -> Fax order
4. Login->Open order->Update order

Now, the developer changed the name of 'OK' button in Login dialog box to 'Sign In' due to a new requirement.

This change would create a mismatch between the corresponding run-time object in the application and test object in your object repository as shown in the below images.


Login
Agent Name:
Password:
Sign In
Cancel
Help
Run time text property
Test object text property
Object Repository - C:\Users\abhilash_js\Documents\Unified Functional Testing\GUITest1
File Edit Object View Tools Help
Filter: All Objects
Action1
Test Objects
Login
Cancel
Help
OK
Agent Name:
Password:
Checkpoint and Output Obje

Object Properties

Name: OK

Class: WinButton

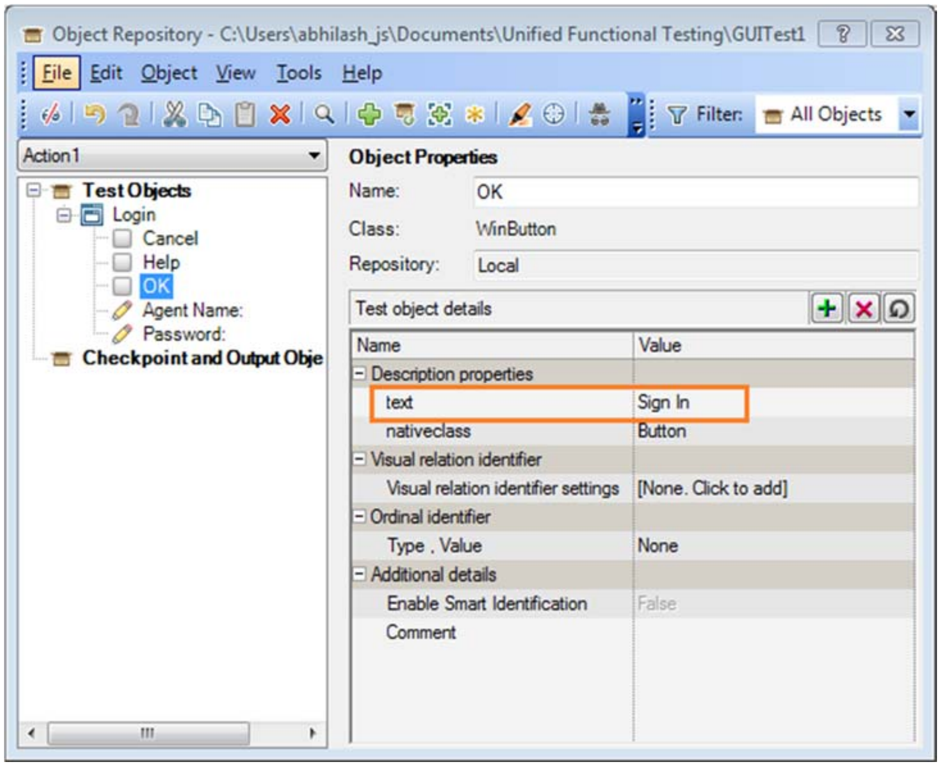
Repository: Local

Test object details

Name	Value
Description properties	
text	OK
nativeclass	Button
Visual relation identifier	
Visual relation identifier settings	[None. Click to add]
Ordinal identifier	
Type , Value	None
Additional details	
Enable Smart Identification	False
Comment	

This mismatch would cause all your 4 scripts to fail as they are all using the changed object.

You can fix this failure by changing the OK button object's 'text' property, which is highlighted in the below image, from 'OK' to 'Sign In'


Object Repository - C:\Users\abhilash_js\Documents\Unified Functional Testing\GUITest1
File Edit Object View Tools Help
Filter: All Objects
Action1
Test Objects
Login
Cancel
Help
OK
Agent Name:
Password:
Checkpoint and Output Obje

Object Properties

Name: OK

Class: WinButton

Repository: Local

Test object details

Name	Value
Description properties	
text	Sign In
nativeclass	Button
Visual relation identifier	
Visual relation identifier settings	[None. Click to add]
Ordinal identifier	
Type , Value	None
Additional details	
Enable Smart Identification	False
Comment	

However, you have four different tests for you have to change the text property of OK button. In a real-time automation test suite which has around 200 scripts it will take approximately 400 minutes to change the test object properties which is a huge amount.

Advanced UFT scripting will help you in handling such scenarios with ease by enabling you to write efficient scripts which dont need object repository at all. Such scripts are highly maintainable and thus increasing the efficiency of the scripts

Need for advanced UFT scripts - 2

Scenario 2:

Consider that you have a very big test script having 300 lines of code associated with a shared object repository.

The execution time of this script will be very high since each line of code is associated with a corresponding object in the repository.

Therefore, UFT will do the below steps for each line in order to execute your test script

1. Understand the logical name of object mentioned in the line
2. Find out the same object in the object repository
3. Read the mandatory/assistive property for the same object
4. Find the corresponding object in the AUT which is having the same mandatory/assistive property
5. Perform the action

Doing the above five steps for each and every line will increase the overall execution time for such big test scripts.

Using advanced UFT scripting, you can avoid the object repository usage by giving the test object properties directly in the script and thus by increasing the test execution speed by a considerable margin.

We will discuss, in detail, the implementation of these scripting process in the upcoming pages.

Need for advanced UFT scripts - 3

Scenario 3:

Consider that you have to script the following scenario in flight reservation application using UFT.

> *Login to the application using the set of details given in the excel sheet. Once logged in, open the corresponding order number. You have to check whether the customer name displayed in the application is matching with the corresponding customer name in the database. If matching you have to write the Result column as 'Pass' else 'Fail'.*

The above scenario can be done using parameterization with the help of datatable. But importing the excel sheet, writing the data in data table, exporting the datasheet, etc. inefficiently consume a lot of processing time.

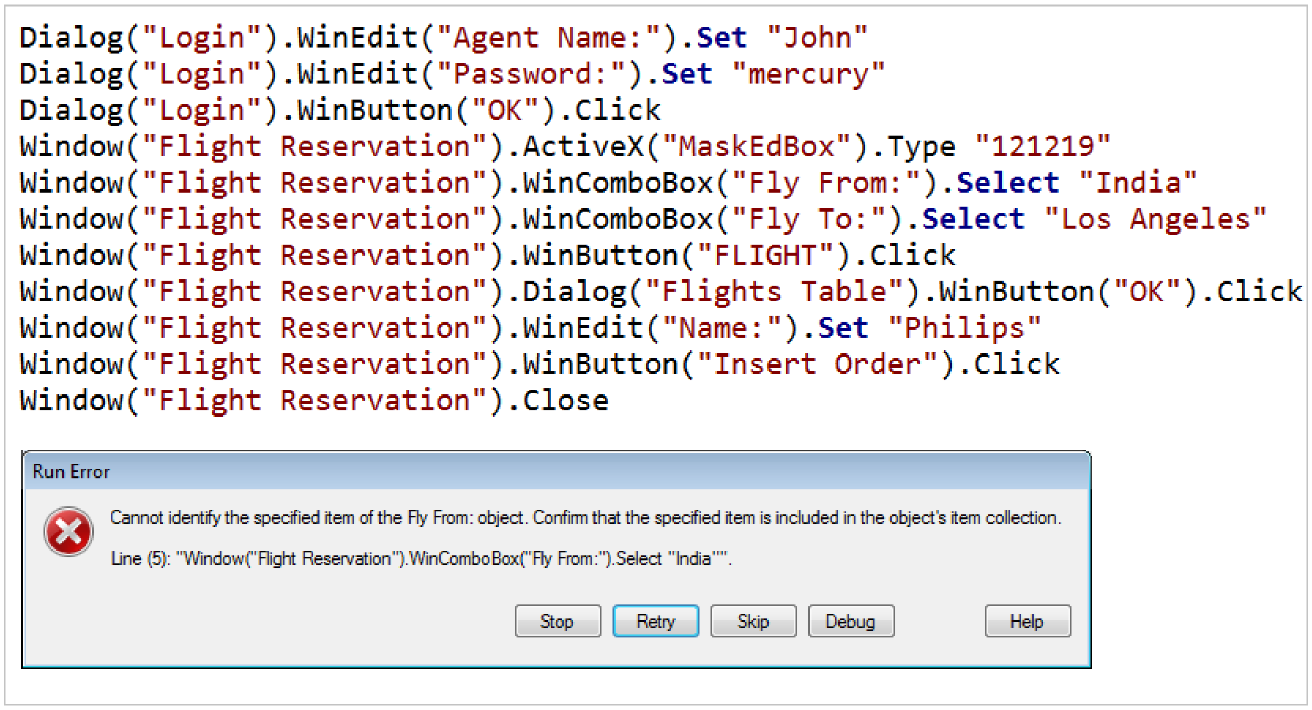
Using advanced VBScript, you can read and write the data directly from the excel sheet without the involving data table and thus increasing the efficiency of your script.

For fetching the data from data table you can use data table output value, but the same process can be done faster using advanced VBScript and by thus we will be able to create a complete package of a well efficient UFT script.

Need for advanced UFT scripts – 4

Scenario 4:

Consider that an 'insert order' script for our flight reservation application is throwing the run time error shown below because the developer mis-spelt the word 'India' in the 'Fly from' drop down list.



```
Dialog("Login").WinEdit("Agent Name:").Set "John"
Dialog("Login").WinEdit("Password:").Set "mercury"
Dialog("Login").WinButton("OK").Click
Window("Flight Reservation").ActiveX("MaskEdBox").Type "121219"
Window("Flight Reservation").WinComboBox("Fly From:").Select "India"
Window("Flight Reservation").WinComboBox("Fly To:").Select "Los Angeles"
Window("Flight Reservation").WinButton("FLIGHT").Click
Window("Flight Reservation").Dialog("Flights Table").WinButton("OK").Click
Window("Flight Reservation").WinEdit("Name:").Set "Philips"
Window("Flight Reservation").WinButton("Insert Order").Click
Window("Flight Reservation").Close
```

```
Run Error

Cannot identify the specified item of the Fly From: object. Confirm that the specified item is included in the object's item collection.

Line (5): "Window("Flight Reservation").WinComboBox("Fly From:").Select "India"".
```

This error will stop the test execution and require a manual intervention to handle the error message shown.

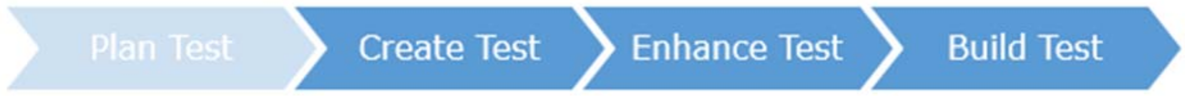
You can handle such scenarios using advanced UFT concepts like recovery scenario. Thus you can make your automated scripts run without any manual intervention even if there are unexpected bugs in the application.

Advanced UFT scripting

From the last four scenarios, you would have understood how advanced UFT scripting will help in creating faster, efficient and highly maintainable scripts.

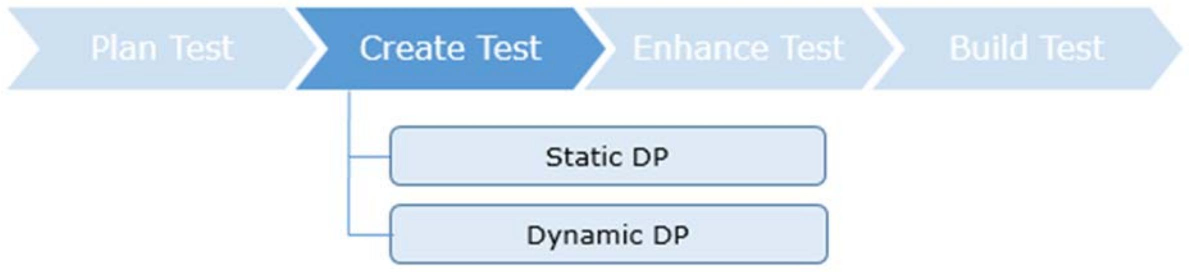
The steps involved in creating such scripts are similar to the ones involved in creating a recorded test script.

Below are the 4 major steps performed while using UFT, which you already learned in UFT foundation course.



In this course we will be focusing on the three highlighted steps to create advanced UFT scripts.

Create test - Descriptive programming

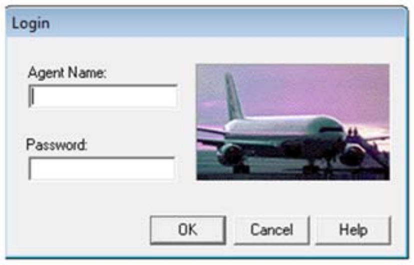


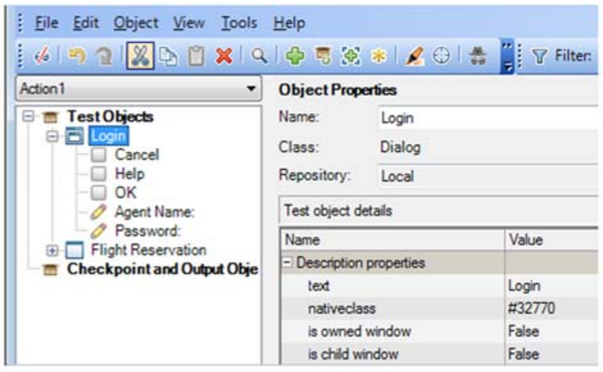
Without recording, you can create UFT test scripts using a method called as Descriptive Programming (DP).

Why descriptive programming

Whenever you record any application flow, UFT adds the test objects in the Object Repository(OR). During playback, UFT identifies the run time object in the AUT using its properties stored in the OR.

If UFT identifies a matching object in the AUT, the script will be executed.





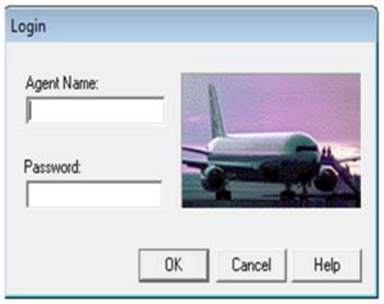
Object Properties

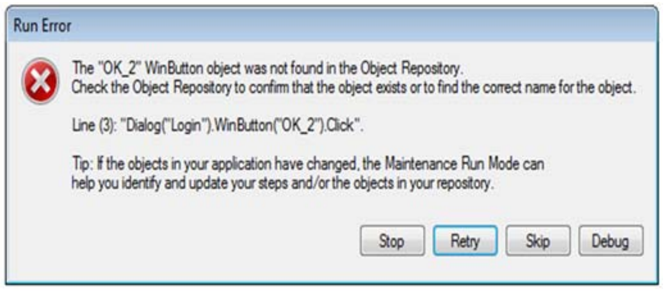
Name: Login
Class: Dialog
Repository: Local

Test object details

Name	Value
Description properties	
text	Login
nativeclass	#32770
is owned window	False
is child window	False

If UFT fails to identify a matching object in the AUT, then it will display an error message like the one shown below.




Run Error

The "OK_2" WinButton object was not found in the Object Repository.
Check the Object Repository to confirm that the object exists or to find the correct name for the object.

Line (3): "Dialog("Login").WinButton("OK_2").Click".

Tip: If the objects in your application have changed, the Maintenance Run Mode can help you identify and update your steps and/or the objects in your repository.

What is descriptive programming

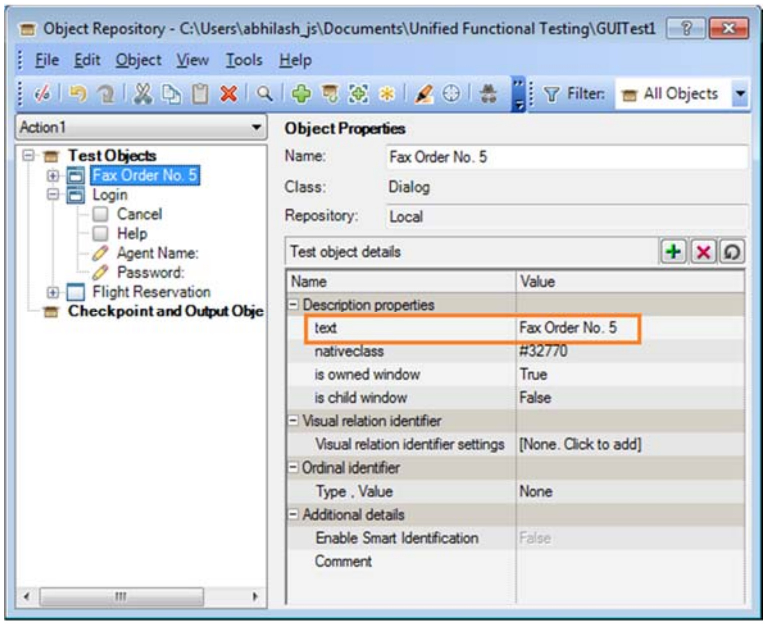
Descriptive programming is a technique to describe the test object programmatically in the test script itself.

- It aids UFT to identify the objects in AUT even if the test objects are not present in the associated object repository by detailing the object's property in the script itself.
- It will help you in even creating UFT scripts without the object repository.

You can use descriptive programming in the following scenarios

1. Handling dynamic object property

You can use descriptive programming when the object properties in the application under test (AUT) are dynamic in nature i.e. it changes with each execution. For example the text property of the Fax window will keep on changing on insertion of new order.If you have a script for inserting an order and then faxing the order ,it will throw you error due to this text property mismatch. You can handle such properties easily with descriptive programming.


Object Repository - C:\Users\abhilash_js\Documents\Unified Functional Testing\GUITest1
File Edit Object View Tools Help
Filter: All Objects
Action1
Test Objects
Fax Order No. 5
Login
Cancel
Help
Agent Name:
Password:
Flight Reservation
Checkpoint and Output Obje

Object Properties

Name: Fax Order No. 5

Class: Dialog

Repository: Local

Test object details

Name	Value
Description properties	
text	Fax Order No. 5
nativeclass	#32770
is owned window	True
is child window	False
Visual relation identifier	
Visual relation identifier settings	[None. Click to add]
Ordinal identifier	
Type , Value	None
Additional details	
Enable Smart Identification	False
Comment	

2. Huge object repository

UFT adds all the properties of the objects in object repository. For a very big application, object repository will be bulky and it may affect the performance of the tool while identifying the object. By using descriptive programming, you can make the execution of the script faster by using only the required properties of the objects.

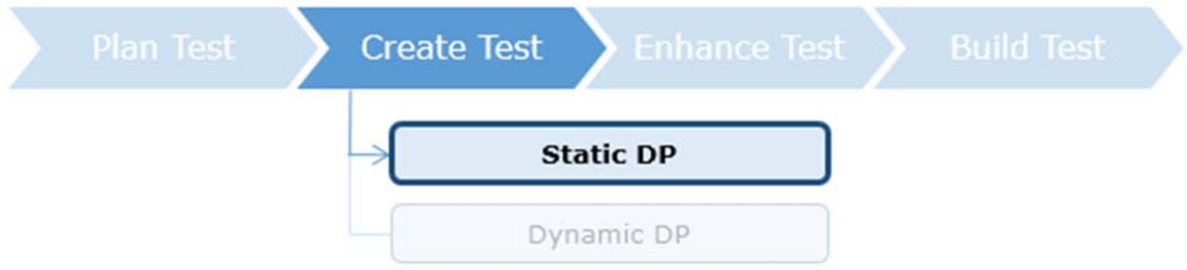
3. Object repository in 'read only' or 'shared' mode

If the object repository is in read only mode, you will not be able to modify the object property when you need to. This will lead to unnecessary delays scripting tasks for the project. If the object repository is in 'shared' mode, any change that you make in the object property will be affecting all other users' test which had this object repository associated. This may cause total failure of a lot of tests.

For handling all these issue you can go for descriptive programming.

In the upcoming pages, let us see what are static and dynamic DP and how you can implement them.

Descriptive programming - Static DP



You can use **static descriptive programming** to access an object in the AUT by specifying the properties and values directly in the script statement. You can give as many number of properties and its corresponding values for an object inside the script.

Syntax

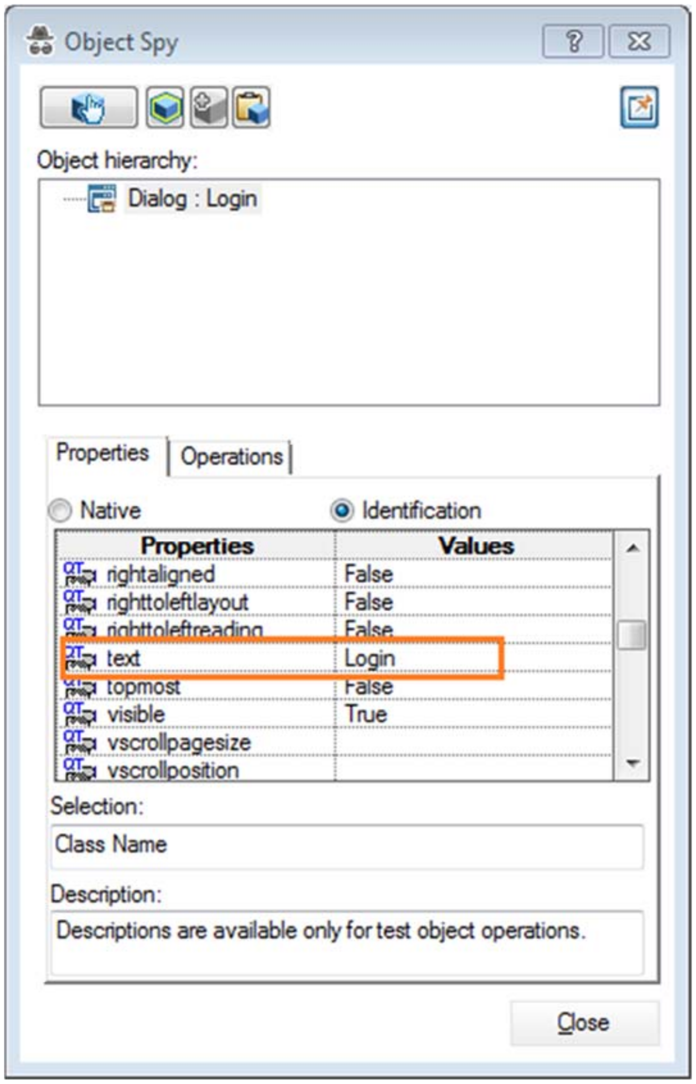
```
TestObject (“PropertyName:=PropertyValue”)
```

Example

To enter the text 'John' as Agent Name in the flight reservation login dialog box using only static DP (without object repository), the steps you need to follow are:

Step 1: Open Flight GUI Login dialog box and open a new test in UFT

Step 2: Using objectspy, spy the 'Login' dialog box. From the available list of properties associated with the object, pick one such that it helps the tool to uniquely identify the object. In this case, the property “text” is used to uniquely identify Login dialog box of Flight GUI application

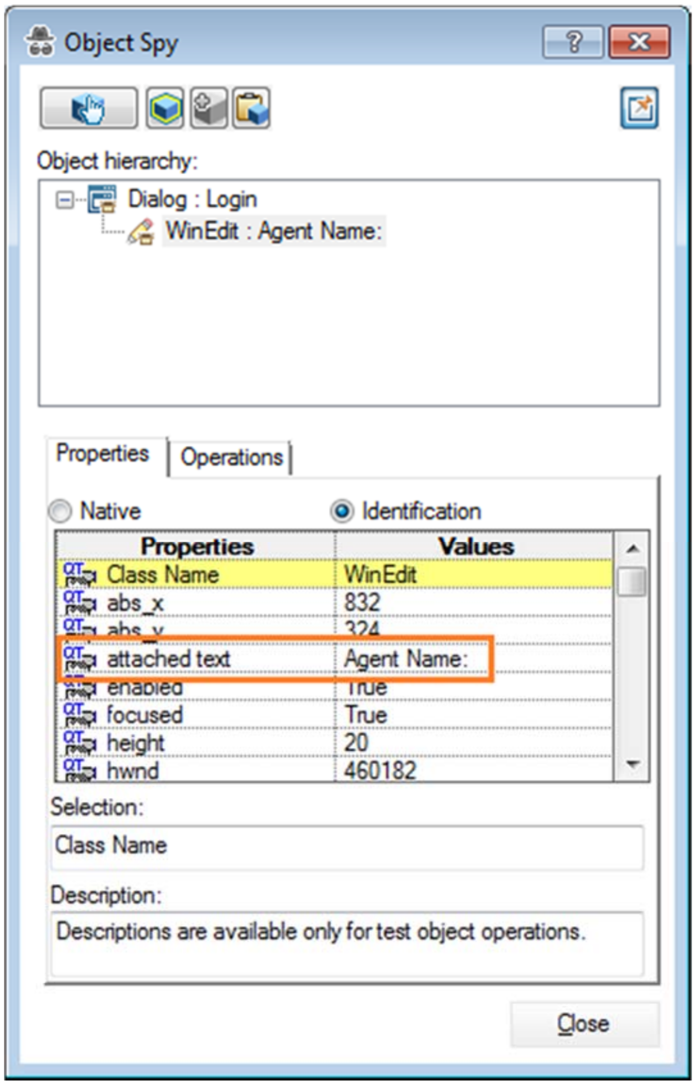


Properties	Values
rightaligned	False
righttoleftlayout	False
righttoleftreading	False
text	Login
topmost	False
visible	True
vscrollpagesize	
vscrollposition	

Class Name

Descriptions are available only for test object operations.

Step 3: Using objectspy, spy the 'Agent Name' edit box and pick a property from the available list of properties associated with the object such that it helps the tool to uniquely identify the object. In this case, the property “attached text” is used to uniquely identify the Agent Name edit box



Object hierarchy:

- Dialog : Login
  - WinEdit : Agent Name:

Native [✓ Identification]

Properties	Values
Class Name	WinEdit
abs_x	832
abs_y	324
attached text	Agent Name:
enabled	True
focused	True
height	20
hwnd	460182

Selection:

Class Name

Description:

Descriptions are available only for test object operations.

Step 4: With reference to the static DP syntax and the identified properties, below given is the static DP syntax which is used to enter 'John' as 'Agent Name' in the Login dialog box

```
Dialog("text:=Login").WinEdit("attached text:=Agent Name:").Set "John"
```

If you want to specify multiple descriptions for identifying the object, you can do it using the following statement

```
Dialog("text:=Login").WinEdit("window id:=3001","attached text:=Agent Name:").Set "John"
```

Note: While writing descriptive programming, you need to maintain the hierarchy of the objects as designed in the application.

Demo 1 : Descriptive programming - Static DP

Highlights:

- Implementation of static descriptive programming using web application as AUT

Demo Steps:

Step 01: Open Flight GUI application using Systemutil.run method

```
SystemUtil.Run  "C:\Program Files\HP\Unified Functional Testing\samples\Flights Application\FlightsGUI.exe"
```

Step 02: Using object spy, identify the unique property for window, the edit boxes and the button. Write the static DP code for entering details in edit boxes and click button

```
WpfWindow("devname:=HP MyFlight Sample Application").WpfEdit("devname:=agentName").Set "john"
WpfWindow("devname:=HP MyFlight Sample Application").WpfEdit("devnamepath:=password;;").set "hp"
WpfWindow("devname:=HP MyFlight Sample Application").WpfButton("devname:=okButton").Click
```

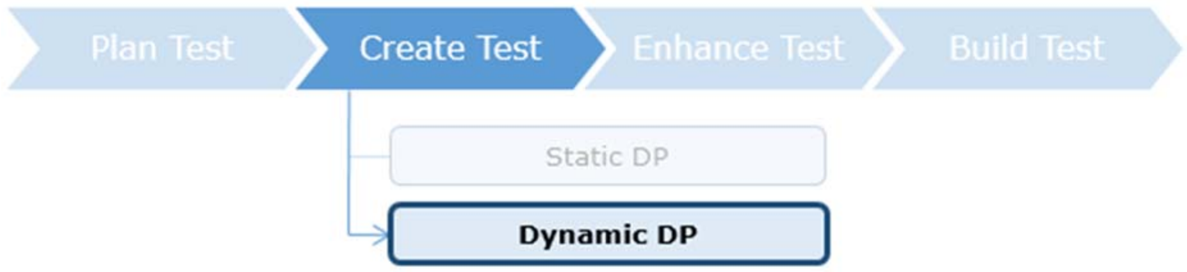
Step 03: Wait for five seconds for the application and then close the application.

```
Wait (5)

WpfWindow("devname:=HP MyFlight Sample Application").Close
```

Step 04 The final script would look like the one given below. Execute the whole script and observe the output.

```
SystemUtil.Run  "C:\Program Files\HP\Unified Functional Testing\samples\Flights Application\FlightsGUI.exe"

WpfWindow("devname:=HP MyFlight Sample Application").WpfEdit("devname:=agentName").Set "john"

WpfWindow("devname:=HP MyFlight Sample Application").WpfEdit("devnamepath:=password;;").set "hp"

WpfWindow("devname:=HP MyFlight Sample Application").WpfButton("devname:=okButton").Click

Wait (5)

WpfWindow("devname:=HP MyFlight Sample Application").Close
```

Descriptive programming - Dynamic descriptive programming



Description objects

If you refer to the same object multiple times using static DP, specifying the properties and values in each statement will get tiresome. To avoid repetition, you can create **description objects**, one of the UFT utility objects.

Dynamic descriptive programming

The method used to access an object in the AUT by creating a description object for describing the properties and values is known as **dynamic descriptive programming**.

Using Dynamic DP we can handle **child objects and web tables** effectively which we will learn in the following pages

Syntax

```
Set ObjectRef = Description.Create
ObjectRef (“PropertyName”).Value = PropertyValue
```

Where

> `Description.Create` is used to create properties collection object.
>
> `Set ObjectRef` will create a reference to the object created using `Description.Create`
>
> You can set the values and properties for the particular object using the second statement specified in the syntax.

Example

To enter the text 'John' as Agent Name in the flight reservation login dialog box using only dynamic DP, the steps you need to follow are:

Step 1: Open Flight GUI application and open a new test in UFT

Step 2: Using objectspy, identify the most suitable property and its value for the window and for the 'Agent Name' edit box.

Step 3: With reference to the dynamic DP syntax and the identified properties, below given is the dynamic DP syntax which you can use to create application window, Agent Name edit box, password edit box and “ok” button description objects.

```
Set myWindow = Description.Create
myWindow("devname").Value = "HP MyFlight Sample Application"
myWindow("regexpwndtitle").Value = "HP MyFlight Sample Application"

Set uname = Description.Create
uname("devname").Value = "agentName"

Set pwd = Description.Create
pwd("devname").Value = "password"

Set btnok = Description.Create
btnok("devname").Value = "okButton"
```

Step 4: After creating the objects, you have to write the below script for using those objects and to write the data in the edit box.

```
WpfWindow(myWindow).WpfEdit(uname).Set username
WpfWindow(myWindow).WpfEdit(pwd).Set password
WpfWindow(myWindow).WpfButton(btnok).Click
```

Descriptive programming - Dynamic DP - Child objects

Child objects are the objects like edit box, link, button, etc. that are contained within the parent object like a page.

As a tester, you might have to script scenarios to validate child objects like, for example, checking if the number of links in Sparsh home page is 120. You can do this using **ChildObjects** method.

ChildObjects method

- You can access all the property and values of all the child objects in a webpage/window using ChildObjects method.
- It returns the collection of child objects contained within the given parent object.
- You can implement this using dynamic DP programming.
- To access the objects within an application, the hierarchy of the objects used during designing of application should be followed.

Syntax

```
TestObject. ChildObjects(Description)
```

Example

Let us print the count of all the buttons and their names from the Login dialog box using child objects method.

Step 1: Open Flight GUI Login window and open a new test in UFT

Step 2: Using objectspy, identify the most suitable property and its value for the application window and create the description objects for window, edit boxes and ok button.

Step 3: On **"Search Flight"** window, identify the class name of the child object for which you have to get the details. It can be identified by spying on any one of the child object. In this case the class name is "WpfComboBox". Write the following statements to create a description object named **"cmbBox_child"** for drop down combo boxes.

```
WpfWindow(myWindow).WpfEdit(pwd).Set password
Set cmbBox_child = Description.Create
cmbBox_child("micclass").value = "WpfComboBox"
```

In descriptive programming, **micclass** keyword is used to set the value for **ClassName** property.

Step 4: The below statement will fetch all the properties and values of child objects of type "WpfComboBox" into an object variable called **"cmbobj"**

```
Set cmbobj = WpfWindow(myWindow).ChildObjects(cmbBox_child)
```

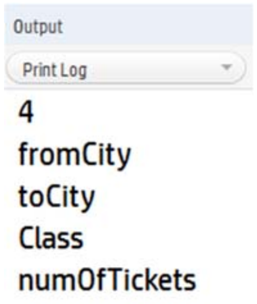
Step 5: For printing the total number of child objects the below statement can be used

```
print cmbobj.count
```

Step 6: For printing the names of all the buttons 'text' property of the buttons can be fetched inside a for loop. The below statements can be used for the same.

```
For index = 1 To cmbobj.count-1 Step 1
       print cmbobj(index).GetROProperty("Devname")
Next
```

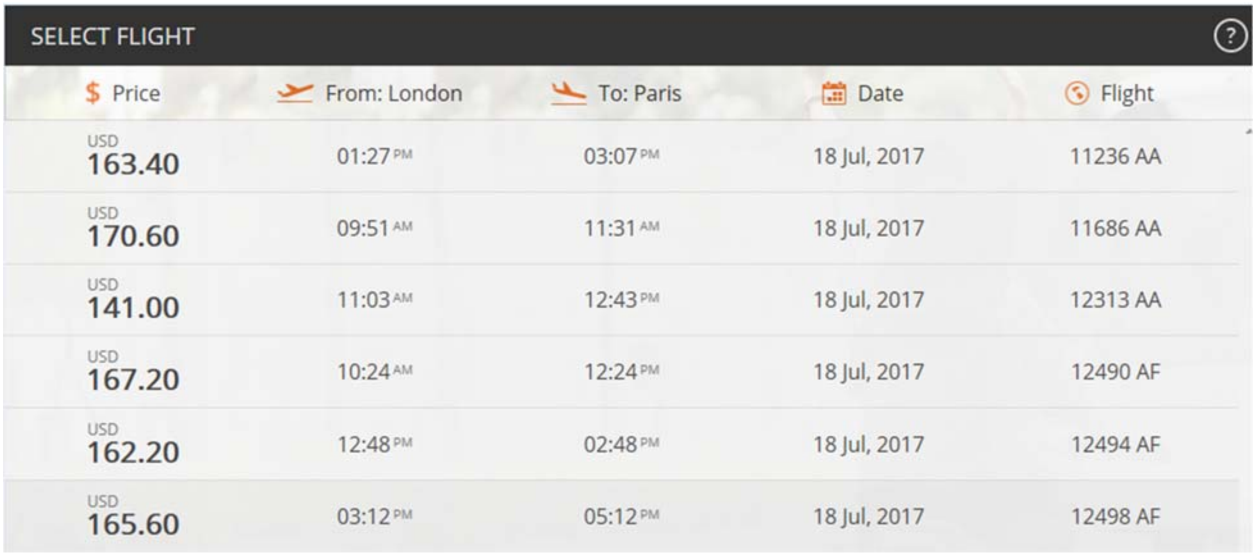
Once all the above six steps are executed you will be getting the below output which shows the count and the names of all the buttons in the Login dialog box



```
4
fromCity
toCity
Class
numOfTickets
```

Descriptive programming - Dynamic DP - table

Tables (web table/ grid table) are objects on a page which are used to present data in a tabular format as shown below



SELECT FLIGHT

$ Price	From: London	To: Paris	Date	Flight
USD 163.40	01:27 PM	03:07 PM	18 Jul, 2017	11236 AA
USD 170.60	09:51 AM	11:31 AM	18 Jul, 2017	11686 AA
USD 141.00	11:03 AM	12:43 PM	18 Jul, 2017	12313 AA
USD 167.20	10:24 AM	12:24 PM	18 Jul, 2017	12490 AF
USD 162.20	12:48 PM	02:48 PM	18 Jul, 2017	12494 AF
USD 165.60	03:12 PM	05:12 PM	18 Jul, 2017	12498 AF

Fetching data from them is not as straight forward as other web objects because for that we have to iterate through the table and to extract the data using various table methods.

- Each cell in a table can contain data and child objects.
- The number of rows and columns can be variable during different instances of the page.

Using a table object and its methods will simplify your scripting tasks related to retrieving data and child objects from them.

Web table methods

Method	Syntax	Description
RowCount	webTableObject.RowCount	Returns the number of rows in the table
ColumnCount	webTableObject.ColumnCount(RowNumber)	Returns the number of columns in a particular row in the table.
GetCellData	webTableObject.GetCellData(RowNumber, ColumnNumber)	Returns the data contained in the specified cell.
GetROProperty	webTableObject.GetROProperty(Property)	Returns the current value of the specified identification property from the object in the application.

Demo 4 : Descriptive programming - Dynamic DP - Web table

Highlights:

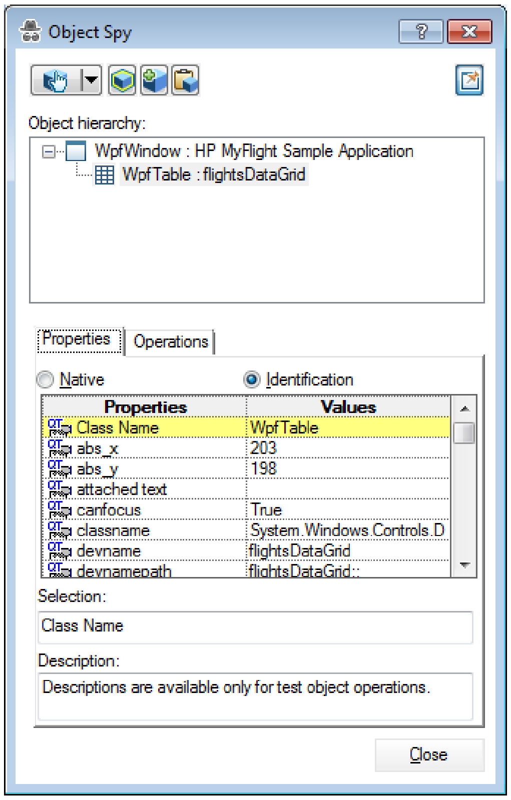
- Implementation of grid table methods

Demo Steps:

Step 1: Open FlightGui application and navigate till “Select flight” window. Open a new UFT test and add following script to it.

```
Set myWindow = Description.Create
myWindow("devname").Value = "HP MyFlight Sample Application"
myWindow("regexpwndtitle").Value = "HP MyFlight Sample Application"

Set uname = Description.Create
uname("devname").Value = "agentName"

Set pwd = Description.Create
pwd("devname").Value = "password"

Set btnok = Description.Create
btnok("devname").Value = "okButton"

SystemUtil.Run  "C:\Program Files\HP\Unified Functional Testing\samples\Fligh
ts Application\FlightsGUI.exe"
WpfWindow(myWindow).WpfEdit(uname).Set "john"
WpfWindow(mywindow).WpfEdit(pwd).set "hp"
WpfWindow(myWindow).WpfButton(btnok).Click

'get child objects
Set cmbBox_child = Description.Create
cmbBox_child("micclass").value = "WpfComboBox"


Set cmbobj = WpfWindow(myWindow).ChildObjects(cmbBox_child)
print cmbobj.count
For index = 0 To cmbobj.count-1 Step 1
    print cmbobj(index).GetROProperty("Devname")
Next

cmbobj(0).Select(2)
cmbobj(1).Select(4)

WpfWindow(myWindow).WpfCalendar("devname:=datePicker").SetDate "28-Jul-2017"
WpfWindow(myWindow).WpfButton("name:=FIND FLIGHTS").Click
```

Step 2: Create a description object for the grid table displaying flight details. For getting the table property value you can spy on any element inside the table and in the hierarchy select as “WpfTable”


Object Spy
Object hierarchy:
WpfWindow : HP MyFlight Sample Application
WpfTable : flightsDataGrid
Properties
Operations
Native
Identification
Selection:
Class Name
Description:
Descriptions are available only for test object operations.
Close

Properties	Values
Class Name	WpfTable
abs_x	203
abs_y	198
attached text	
canfocus	True
classname	System.Windows.Controls.D
devname	flightsDataGrid
devnamepath	flightsDataGrid;;

Code for creating the object is given below

```
'create object for WpfTable
Set flightTable = Description.Create
flightTable("devname").Value = "flightsDataGrid"
flightTable("wpftypename").Value = "datagrid"
```

Step 5: Create a "Flighs" column in the datatable to save the flight numbers

```
datatable.GetSheet("Global").AddParameter "Flights",1
```

Step 6: Flight numbers are available in 5th column. Using getcelldata method you can fetch the flight numbers and write inside the datatable

```
For row = 0 To rowcount-1 Step 1
    DataTable.SetCurrentRow(row+1)
    flightNum = WpfWindow(myWindow).WpfTable(flightTable).GetCellData(row,4)
    DataTable("flight",1) = flightNum
Next
```

Step 7: Export the datatable and close the window

```
Datatable.export "C:\flights.xls"

WpfWindow(myWindow).Close
```

Enhance test

So far you had seen how to create a test using various descriptive programming methods.

Once the test is created there are lot of methods to enhance your test to tune it according to the requirement.We have the above listed advaned enhancement concepts available in UFT and lets discuss one by one in detail.

Arrays

Why to use arrays

Applications can have various types of data in various formats. During testing you are supposed to pass these values to the application or have to extract the data from the application. If we have to deal with data of same type and of almost same size,then the best way is to use arrays.

Arrays are commonly used to organize data so that a related set of values can be easily sorted or searched. Using arrays you can hold more than one value in a single variable at a time. When a series of values are stored in a single variable, then it is known as array variable.

For example, a search engine may use an array to store Web pages found in a search performed by the user. When displaying the results, the program will output one element of the array at a time. This may be done for a specified number of values or until all the values stored in the array have been output. While the program could create a new variable for each result found, storing the results in an array is much more efficient way to manage memory.

Arrays

What is an array

A variable is a container to store a value. Sometimes we need to hold more than one value in a single variable at a time. When a series of values are stored in a single variable, then it is known as an array variable.

An array variable is used to store multiple values in a single variable

Array can contain a series of values of same data type

An array is a memory space referred by a common name

Array Declaration

Arrays are declared in the same way as variables. The only difference is that an array variable declaration uses paranthesis while a normal variable declaration does not.

There are three different methods of declaring an array.

Method 1: Using Dim without size

Dim arr ()

Method 2: Using Dim and Size

Dim arr (array size)

Dim arr (3)

Method 3: Using 'Array' parameter

Dim arr

arr= Array("jojo","jhon","james")

Note: VBScript Arrays can store any type of variable in an array. Hence, an array can store an integer, string or characters in a single array variable

Determining the size and number of elements in an Array

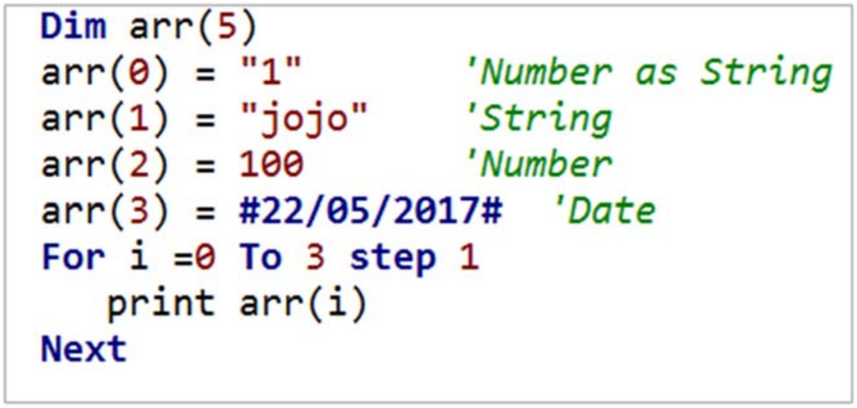
Consider the array arr(3)

Although, the array size is indicated as 3, it can hold 4 values as array index starts from ZERO.

Array index cannot be negative.

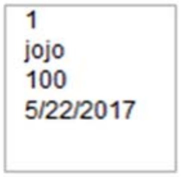
Example

Let's see how to assign the values to an array:



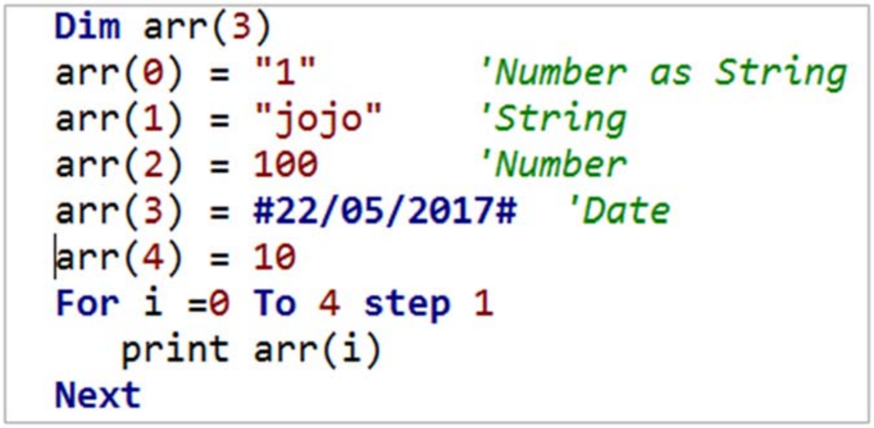
```
Dim arr(5)
arr(0) = "1"          'Number as String
arr(1) = "jojo"       'String
arr(2) = 100          'Number
arr(3) = #22/05/2017#   'Date
For i =0 To 3 step 1
   print arr(i)
Next
```

Output



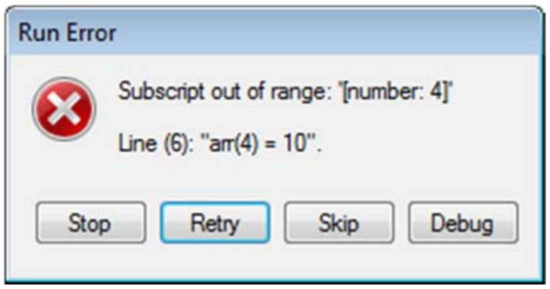
```
1
jojo
100
5/22/2017
```

Consider the second case



```
Dim arr(3)
arr(0) = "1"          'Number as String
arr(1) = "jojo"       'String
arr(2) = 100          'Number
arr(3) = #22/05/2017#   'Date
arr(4) = 10
For i =0 To 4 step 1
   print arr(i)
Next
```

The above script will throw the below error as we are trying to save the data in arr(4) which is not there.


Run Error

Subscript out of range: '[number: 4]'

Line (6): "arr(4) = 10".

Arrays - Keywords associated with arrays

Arrays have two keywords associated with them.

1. Redim statement

2. Preserve

Let us see the syntax and examples of each keyword

1. Redim Statement

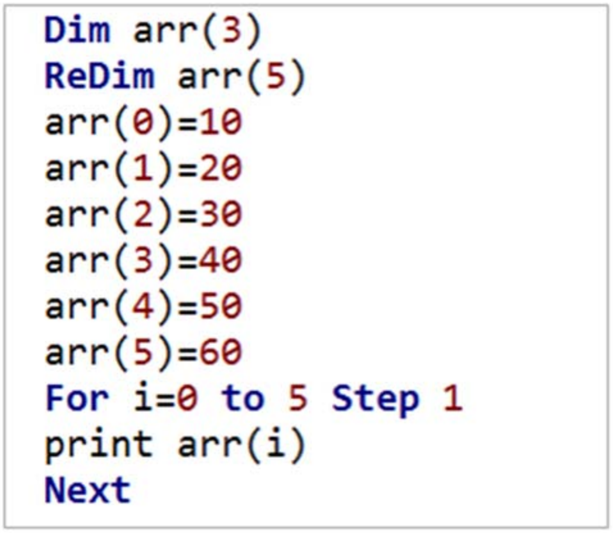
In dynamic array, you must use the keyword Redim for redefining the size of an array. While declaring, dynamic array does not have any size.

Syntax:

ReDim varname(Size)

Example:

In the below example, an array size is redefined



```
Dim arr(3)
ReDim arr(5)
arr(0)=10
arr(1)=20
arr(2)=30
arr(3)=40
arr(4)=50
arr(5)=60
For i=0 to 5 Step 1
print arr(i)
Next
```

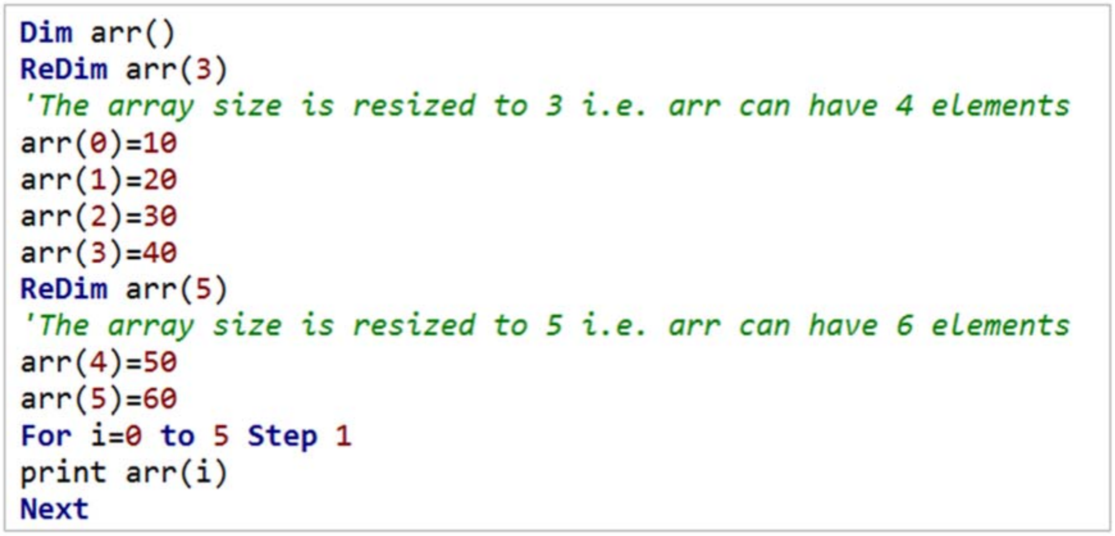
Output:

10 20 30 40 50 60

Note: Upon resizing an array smaller than it was originally, the data in the eliminated elements will be lost.

2. Preserve

This method is used to preserve the data in an existing array when you change the size of the last dimension



```
Dim arr()
ReDim arr(3)
'The array size is resized to 3 i.e. arr can have 4 elements
arr(0)=10
arr(1)=20
arr(2)=30
arr(3)=40
ReDim arr(5)
'The array size is resized to 5 i.e. arr can have 6 elements
arr(4)=50
arr(5)=60
For i=0 to 5 Step 1
print arr(i)
Next
```

Output:

50

60

The re-dimensioning of the array the second time will erase the existing elements in the array i.e. the value of those elements would be blank

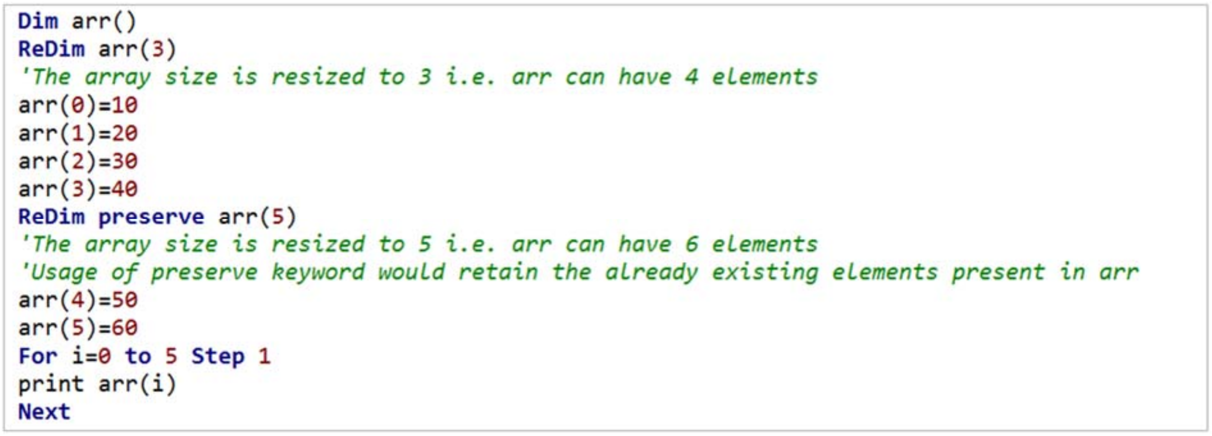
In order to prevent the elements from getting erased, you must add the ‘Preserve’ keyword

Syntax:

ReDim preserve varname(Size)

Example:

In the below example, an array has been redefined and then preserved the values when the existing size of the array is changed



```
Dim arr()
ReDim arr(3)
'The array size is resized to 3 i.e. arr can have 4 elements
arr(0)=10
arr(1)=20
arr(2)=30
arr(3)=40
ReDim preserve arr(5)
'The array size is resized to 5 i.e. arr can have 6 elements
'Usage of preserve keyword would retain the already existing elements present in arr
arr(4)=50
arr(5)=60
For i=0 to 5 Step 1
print arr(i)
Next
```

Output:

```
10
20
30
40
50
60
```

Arrays - Array functions

There are four inbuilt functions in VBScript which will help you to handle arrays effectively.

1. Lower bound
2. Upper bound
3. Split
4. Join

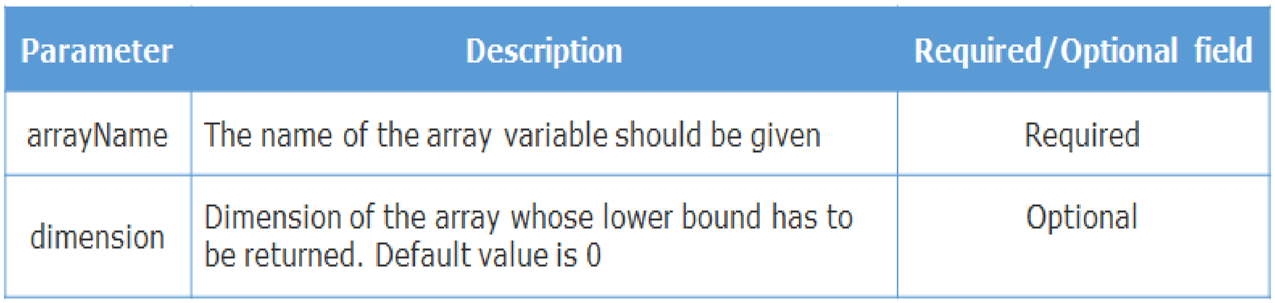
Let us see the syntax and examples of each functions

1. Lower Bound

This function returns the smallest subscript for the indicated dimension of an array.

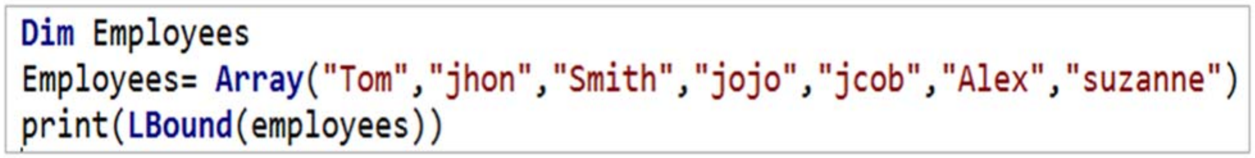
Syntax:

LBound(arrayName,[dimension])



Parameter	Description	Required/Optional field
arrayName	The name of the array variable should be given	Required
dimension	Dimension of the array whose lower bound has to be returned. Default value is 0	Optional

Example:



```
Dim Employees
Employees= Array("Tom","jhon","Smith","jojo","jcob","Alex","suzanne")
print(LBound(employees))
```

Output:

0

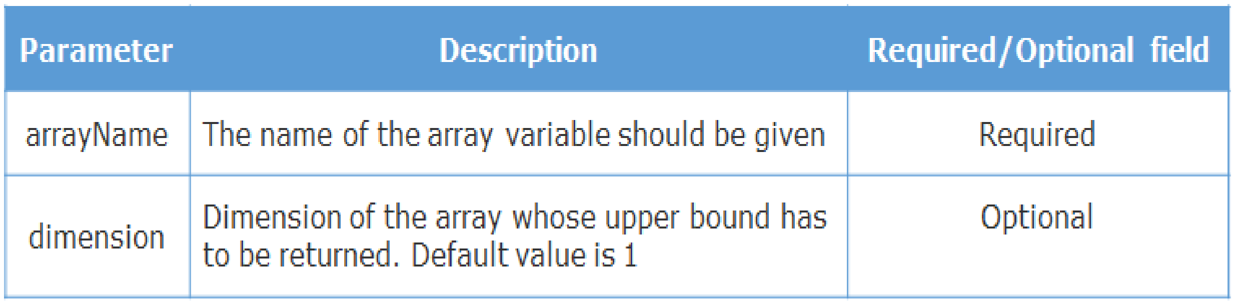
Note: Lbound for any dimension is always 0.

2. Upper Bound

This function returns the largest subscript for the indicated dimension of an array.

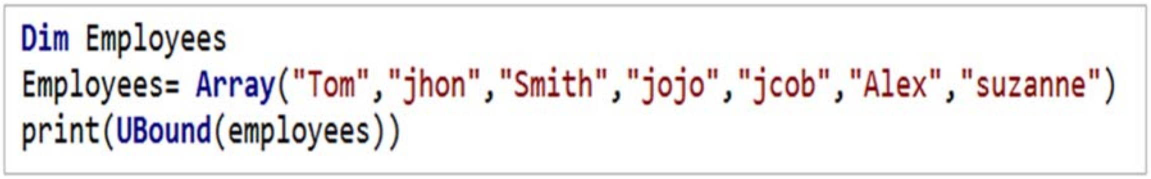
Syntax:

UBound(arrayName,[dimension])



Parameter	Description	Required/Optional field
arrayName	The name of the array variable should be given	Required
dimension	Dimension of the array whose upper bound has to be returned. Default value is 1	Optional

Example:



```
Dim Employees
Employees= Array("Tom","jhon","Smith","jojo","jcob","Alex","suzanne")
print(UBound(employees))
```

Output:

6

Note: In Order to find the length of an array, you can use UBound and LBound.

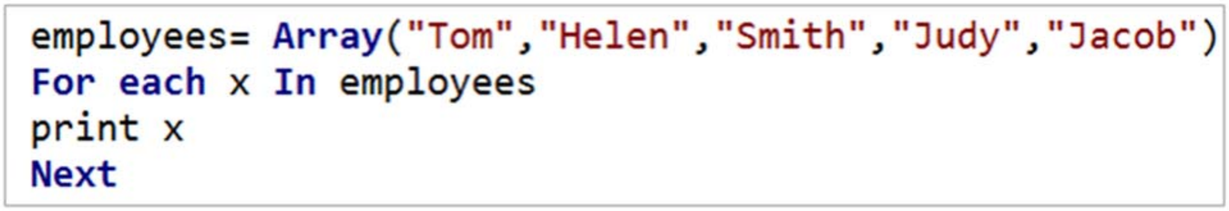
For Each….Next Loop

- For Each...Next statement is used when you want a statement or a block of statements to be executed for each element in an array or a collection.
- This loop does not have step keyword as it executes the statements for each element in the array/collection.

Syntax:

```
For Each element In Array/Collection
[statements]
[statements]
Next
```

Example:



```
employees= Array("Tom","Helen","Smith","Judy","Jacob")
For each x In employees
print x
Next
```

Output:

Tom

Helen

Smith

Judy

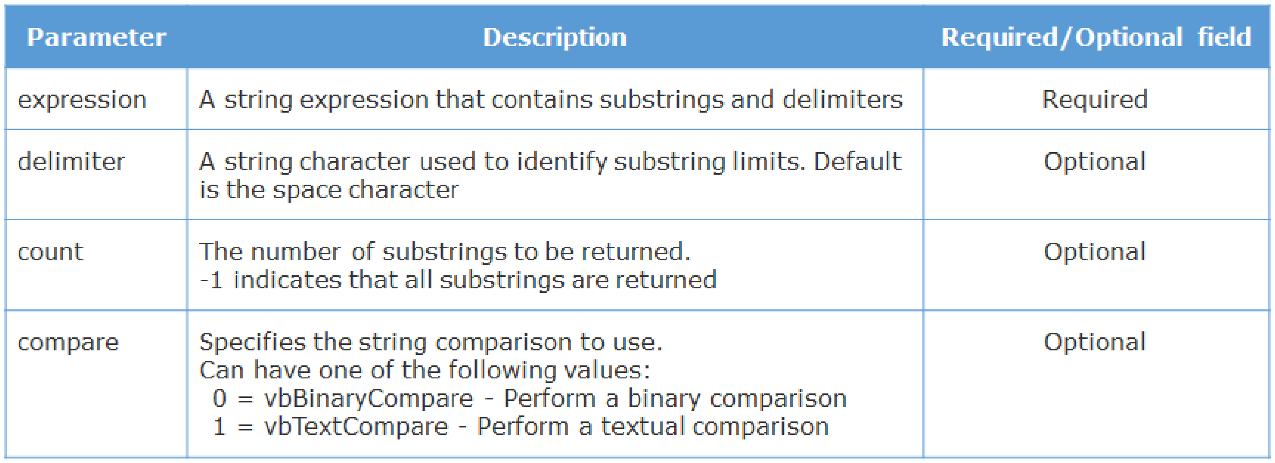
Jacob

3. Split

This Function returns an array that contains a specified number of values. Splitted based on a Delimiter.

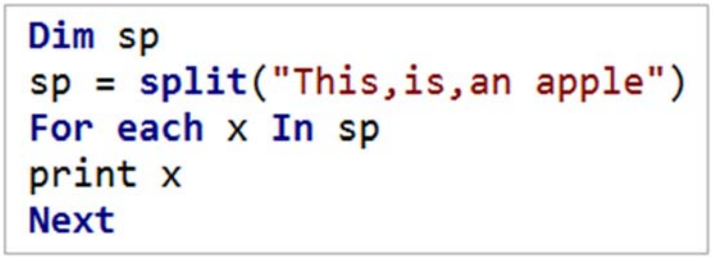
Syntax:

Split(expression, [delimiter], [count],[compare])



Parameter	Description	Required/Optional field
expression	A string expression that contains substrings and delimiters	Required
delimiter	A string character used to identify substring limits. Default is the space character	Optional
count	The number of substrings to be returned. -1 indicates that all substrings are returned	Optional
compare	Specifies the string comparison to use. Can have one of the following values: 0 = vbBinaryCompare - Perform a binary comparison 1 = vbTextCompare - Perform a textual comparison	Optional

Example 01:



```
Dim sp
sp = split("This,is,an apple")
For each x In sp
print x
Next
```

Output:

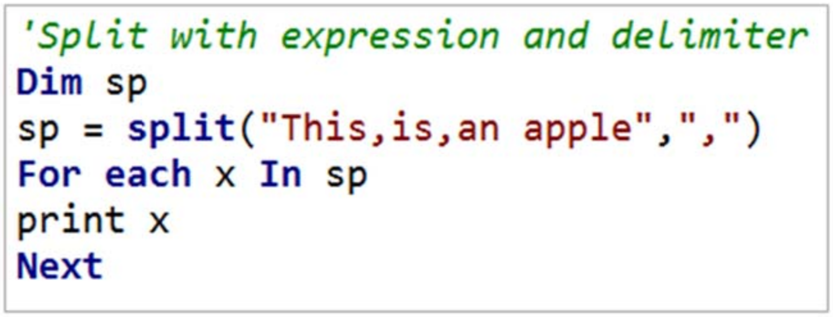
This

is

an

apple

Example 02:



```
'Split with expression and delimiter
Dim sp
sp = split("This,is,an apple",",")
For each x In sp
print x
Next
```

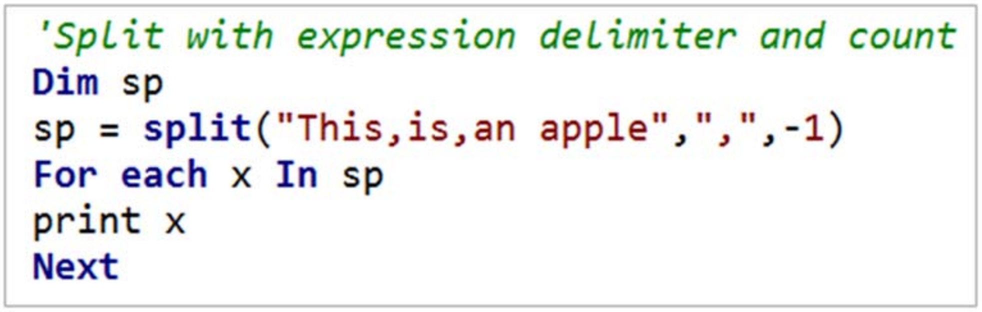
Output:

This

is

an apple

Example 03:



```
'Split with expression delimiter and count
Dim sp
sp = split("This,is,an apple",",",-1)
For each x In sp
print x
Next
```

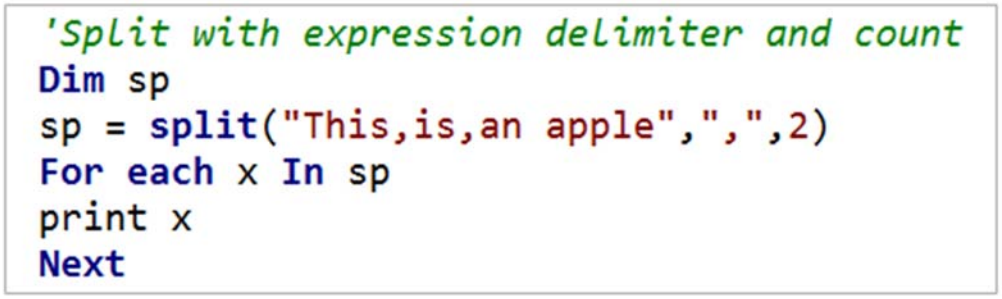
Output:

This

is

an apple

Example 04



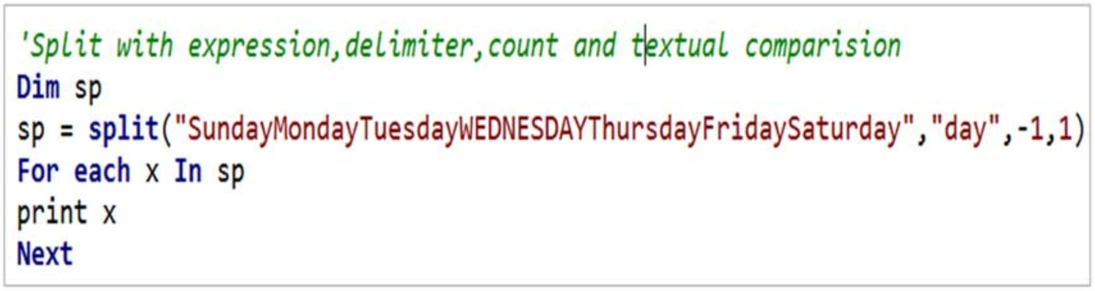
```
'Split with expression delimiter and count
Dim sp
sp = split("This,is,an apple",",",2)
For each x In sp
print x
Next
```

Output:

This

is, an apple

Example 05:

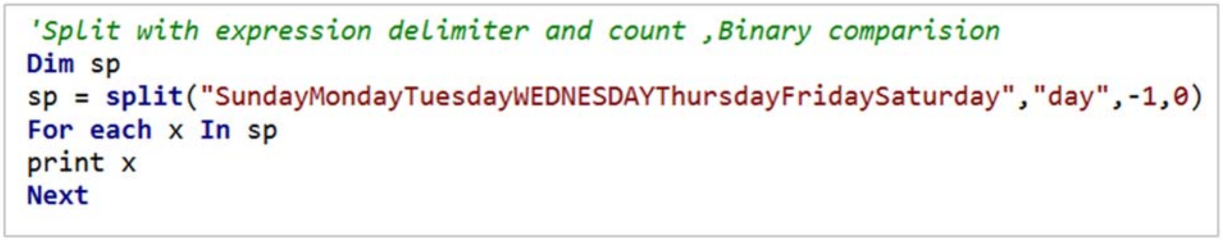


```
'Split with expression,delimiter,count and textual comparision
Dim sp
sp = split("SundayMondayTuesdayWEDNESDAYThursdayFridaySaturday","day",-1,1)
For each x In sp
print x
Next
```

Output:

```
Sun
Mon
Tues
WEDNES
Thurs
Fri
Satur
```

Example 06:



```
'Split with expression delimiter and count ,Binary comparision
Dim sp
sp = split("SundayMondayTuesdayWEDNESDAYThursdayFridaySaturday","day",-1,0)
For each x In sp
print x
Next
```

Output:

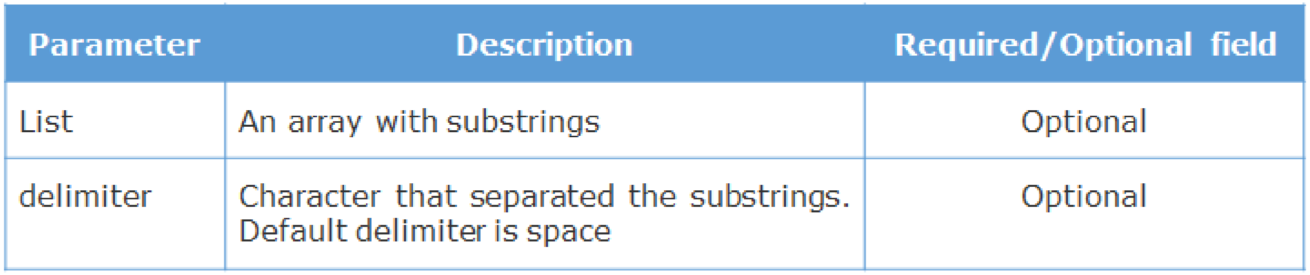
```
Sun
Mon
Tues
WEDNESDAYThurs
Fri
Satur
```

4. Join

This method is used to return a string that contains the various substrings in an array.

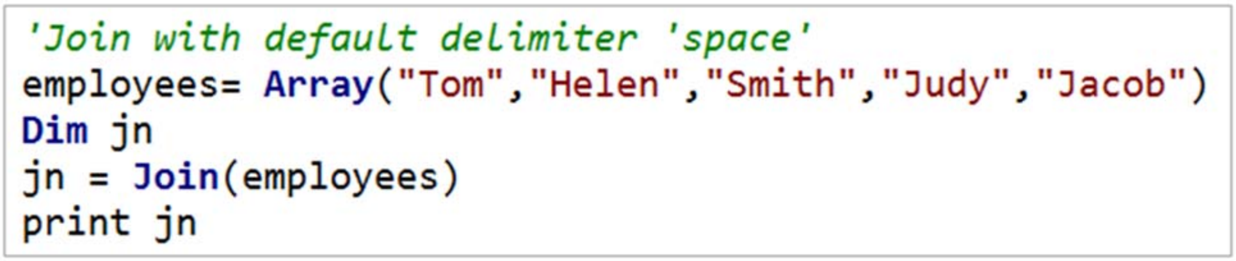
Syntax:

Join(list[,delimiter])



Parameter	Description	Required/Optional field
List	An array with substrings	Optional
delimiter	Character that separated the substrings. Default delimiter is space	Optional

Example 01:

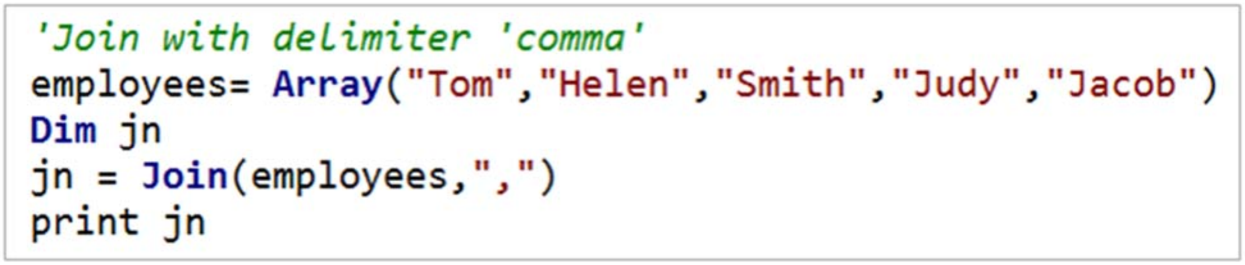


```
'Join with default delimiter 'space'
employees= Array("Tom","Helen","Smith","Judy","Jacob")
Dim jn
jn = Join(employees)
print jn
```

Output:

```
Tom Helen Smith Judy Jacob
```

Example 02:



```
'Join with delimiter 'comma'
employees= Array("Tom","Helen","Smith","Judy","Jacob")
Dim jn
jn = Join(employees,",")
print jn
```

Output:

Tom,Helen,Smith,Judy,Jacob

Demo 5 : Arrays

Highlights:

- Implementation of array functions
- We will do it by fetching the items in the combo box ‘Fly From’ and by joining the fetched elements in a variable and printing the value in the console.

Demo Steps:

Step 1: Declare a variable i_count and an array a_item

```
Dim i_count
Dim a_item()
```

Step 2: Fetch the number of items available in the ‘Fly To’ combo box. Store the value in i_count. Also, print the value in the console.

```
i_count = WpfWindow(myWindow).WpfComboBox("devname:=fromCity").GetItemsCount
Print i_count
```

Step 3: Resize the array a_item

```
Dim size
size = i_count-1
ReDim a_item(size)
```

Step 4: Fetch each item from the combo box and store in the array a_item. Also, print each item in the console.

```
For i=0 to i_count-1 Step 1
   a_item(i)= WpfWindow(myWindow).WpfComboBox("devname:=fromCity").GetItem(i)
   print(a_item(i))
Next
```

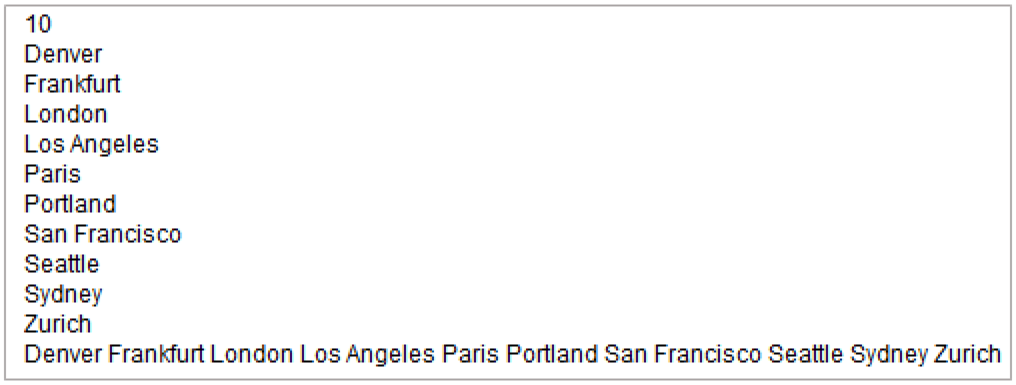
Step 5: Join the elements in the array and store in the variable var. Also, print the value in console.

```
Dim var
var = join(a_item," ")
print var
```

Step 6: The final script will look like this.

```
Dim i_count
Dim a_item()
i_count = WpfWindow(myWindow).WpfComboBox("devname:=fromCity").GetItemsCount
Print i_count
Dim size
size = i_count-1
ReDim a_item(size)
For i=0 to i_count-1 Step 1
a_item(i)= WpfWindow(myWindow).WpfComboBox("devname:=fromCity").GetItem(i)
print(a_item(i))
Next
Dim var
var = join(a_item," ")
print var
```

Step 7 : Upon executing the script, the output must be printed as shown below.



```
10
Denver
Frankfurt
London
Los Angeles
Paris
Portland
San Francisco
Seattle
Sydney
Zurich
Denver Frankfurt London Los Angeles Paris Portland San Francisco Seattle Sydney Zurich
```

Dictionary objects

What is a dictionary object

Most of the time we use variables, when we need to store a input value or a value retrieved from an outside source. When there are many such values, either we use individual variables or we can store in an array. Alternatively, information can also be stored in a Dictionary object. Dictionary object is ideally used when you need to access the data that is associated with unique named value.

The most important reason for using a dictionary instead of an array is that a dictionary is more flexible and richer in terms of built-in functionality. Dictionaries work better than arrays when you need to access random elements frequently. Dictionaries also work better when you want to locate items by their content rather than their position.

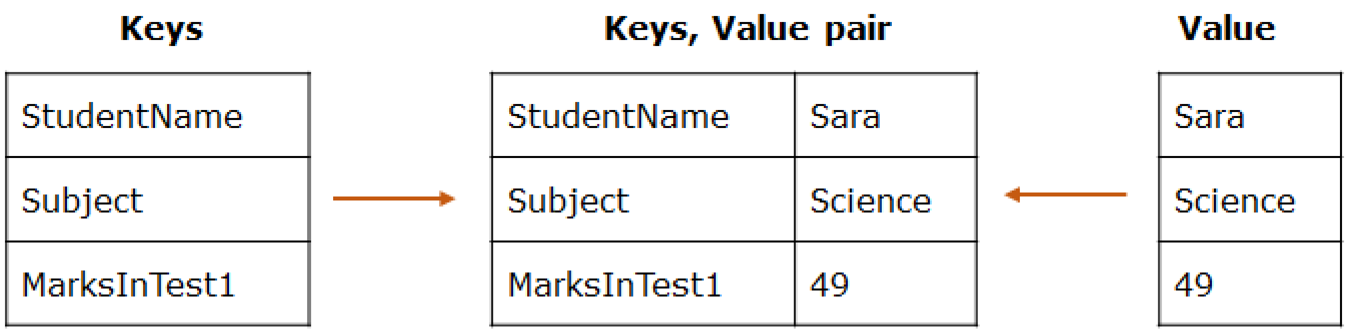
The Dictionary object is used to hold a set of data values in the form of (key, item) pairs.

The keys behave in a way similar to indices in an array, except that array indices are numeric and keys are arbitrary strings. Each key in a Dictionary object must be unique

Dictionary objects

The Dictionary object is used to store a set of data values in the form of (key, item) pairs.

The key is a unique identifier for the corresponding item and cannot be used for any other item in the same Dictionary object.



In this case, a Dictionary object can be created where StudentName, Subject and MarksInTest1 will be keys and corresponding values can be stored for the particular key.

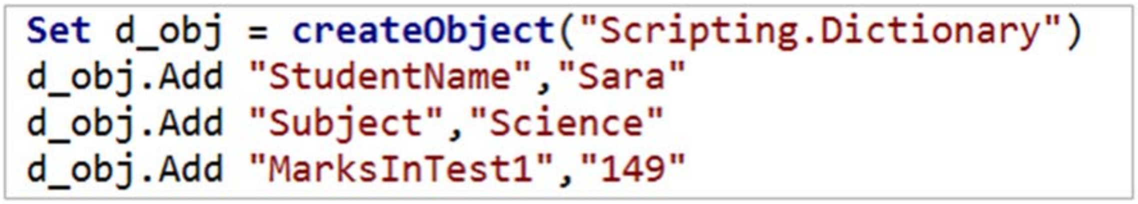
Syntax:

Set d_obj = CreateObject(“Scripting.Dictionary”)

This statement will create an object of Dictionary class named “d_obj”

Example:

Below code is used to create a Dictionary object d_obj and adding key/item pair for student Sara:



```
Set d_obj = createObject("Scripting.Dictionary")
d_obj.Add "StudentName","Sara"
d_obj.Add "Subject","Science"
d_obj.Add "MarksInTest1","149"
```

Dictionary objects – Properties

Dictionary objects have the below given properties.

1. Count

2. Item

3. Key

Let us see the syntax and examples of each property

1. Count

It returns the number of key/item pairs in a Dictionary object

Syntax:

dictionaryObject.Count

Example:

Print d_obj.count

Output:

3

2. Item

It sets or returns the value of an item in a Dictionary object

Syntax:

To return the value of item for a particular key,

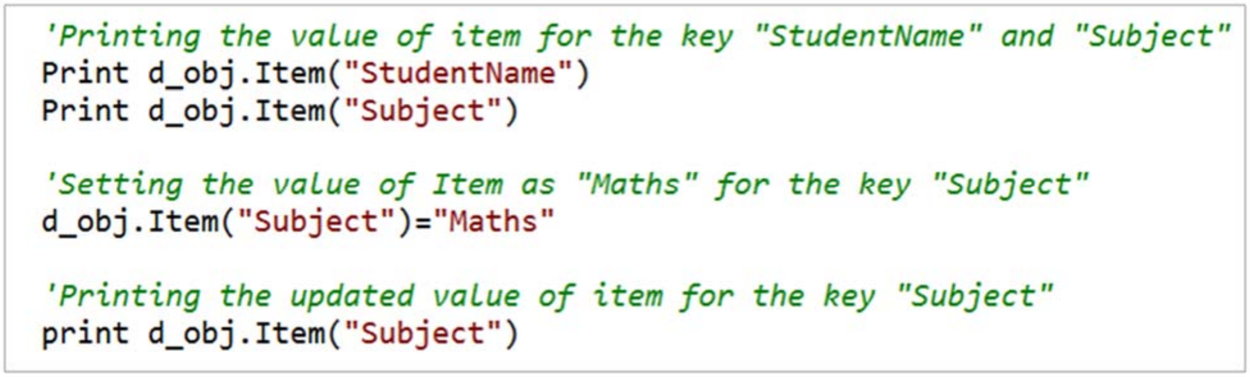
dictionaryObject.Item(key)

To set a new value for item for a particular key,

dictionaryObject.Item(key) = NewItem

If key is not found when attempting to return an existing item, a new key is created and its corresponding item is left empty.

Example:



```
'Printing the value of item for the key "StudentName" and "Subject"
Print d_obj.Item("StudentName")
Print d_obj.Item("Subject")

'Setting the value of Item as "Maths" for the key "Subject"
d_obj.Item("Subject")="Maths"

'Printing the updated value of item for the key "Subject"
print d_obj.Item("Subject")
```

Output:

Sara

Science

Maths

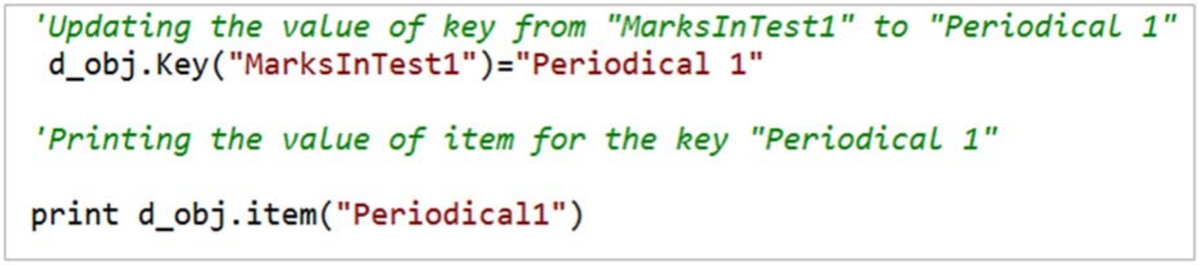
3. Key

It sets a new key value for an existing key value in a Dictionary object

Syntax:

dictionaryObject.Key(key) = NewKey

Example:



```
'Updating the value of key from "MarksInTest1" to "Periodical 1"
 d_obj.Key("MarksInTest1")="Periodical 1"

'Printing the value of item for the key "Periodical 1"

print d_obj.item("Periodical1")
```

Output:

49

Dictionary objects – Methods

Dictionary objects have the below given methods.

1. Add
2. Keys
3. Items
4. Remove
5. Removeall

Let us see the syntax and examples of each method

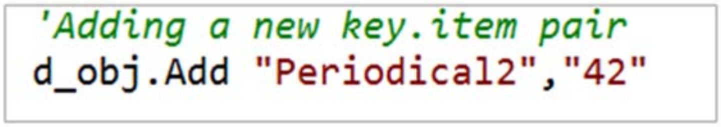
1. Add

It adds a key and item pair to a Dictionary object.

Syntax:

dictionaryObject.Add (key, item)

Example:



```
'Adding a new key.item pair
d_obj.Add "Periodical2","42"
```

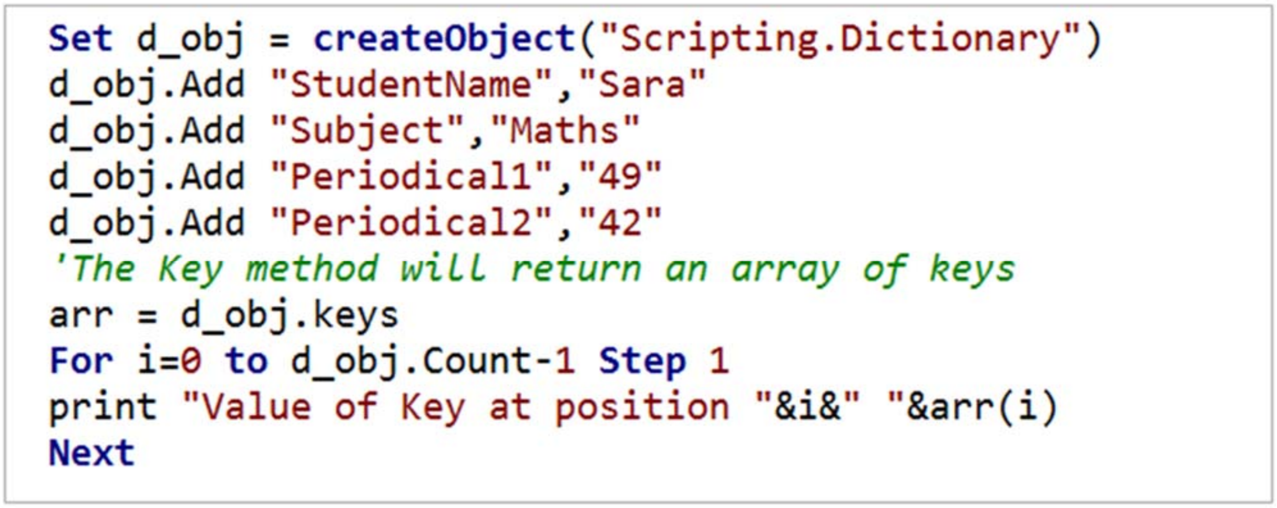
2. Keys

It returns an array of all the keys in a Dictionary object.

Syntax:

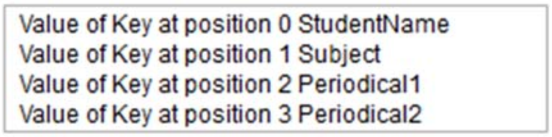
dictionaryObject.Keys

Example:



```
Set d_obj = createObject("Scripting.Dictionary")
d_obj.Add "StudentName","Sara"
d_obj.Add "Subject","Maths"
d_obj.Add "Periodical1","49"
d_obj.Add "Periodical2","42"
'The Key method will return an array of keys
arr = d_obj.keys
For i=0 to d_obj.Count-1 Step 1
print "Value of Key at position "&i&" "&arr(i)
Next
```

Output:



```
Value of Key at position 0 StudentName
Value of Key at position 1 Subject
Value of Key at position 2 Periodical1
Value of Key at position 3 Periodical2
```

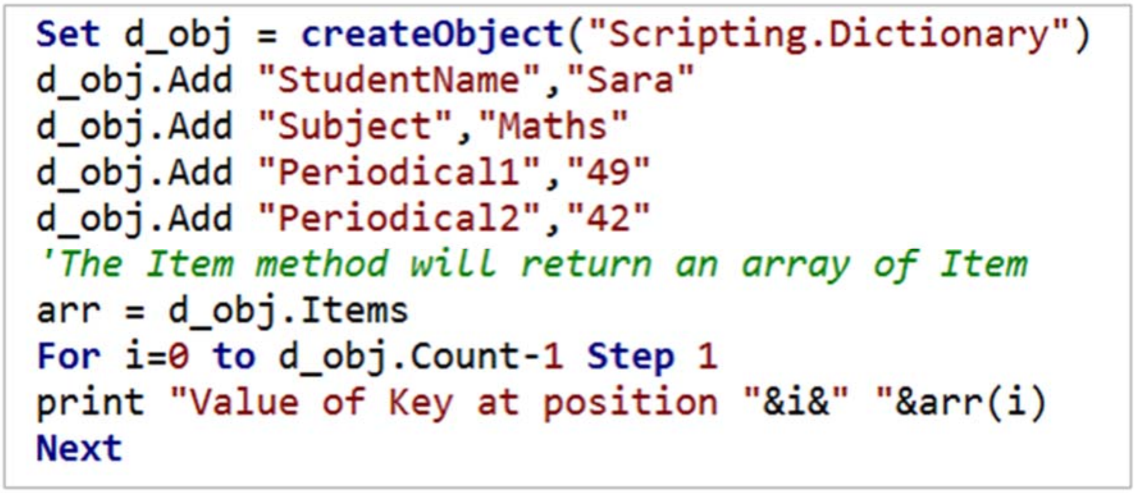
3. Items

It returns an array of all the items in a Dictionary object.

Syntax:

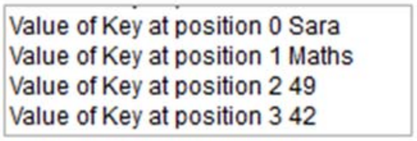
dictionaryObject.Items

Example:



```
Set d_obj = createObject("Scripting.Dictionary")
d_obj.Add "StudentName","Sara"
d_obj.Add "Subject","Maths"
d_obj.Add "Periodical1","49"
d_obj.Add "Periodical2","42"
'The Item method will return an array of Item
arr = d_obj.Items
For i=0 to d_obj.Count-1 Step 1
print "Value of Key at position "&i&" "&arr(i)
Next
```

Output:



```
Value of Key at position 0 Sara
Value of Key at position 1 Maths
Value of Key at position 2 49
Value of Key at position 3 42
```

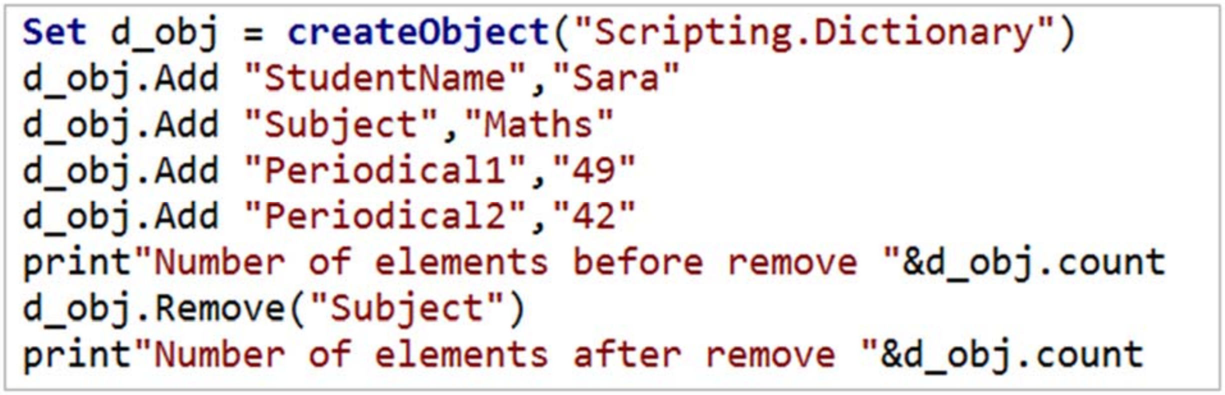
4. Remove

It removes one specified key/item pair from a Dictionary object.

Syntax:

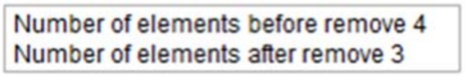
dictionaryObject.Remove(key)

Example:



```
Set d_obj = createObject("Scripting.Dictionary")
d_obj.Add "StudentName","Sara"
d_obj.Add "Subject","Maths"
d_obj.Add "Periodical1","49"
d_obj.Add "Periodical2","42"
print"Number of elements before remove "&d_obj.count
d_obj.Remove("Subject")
print"Number of elements after remove "&d_obj.count
```

Output:



```
Number of elements before remove 4
Number of elements after remove 3
```

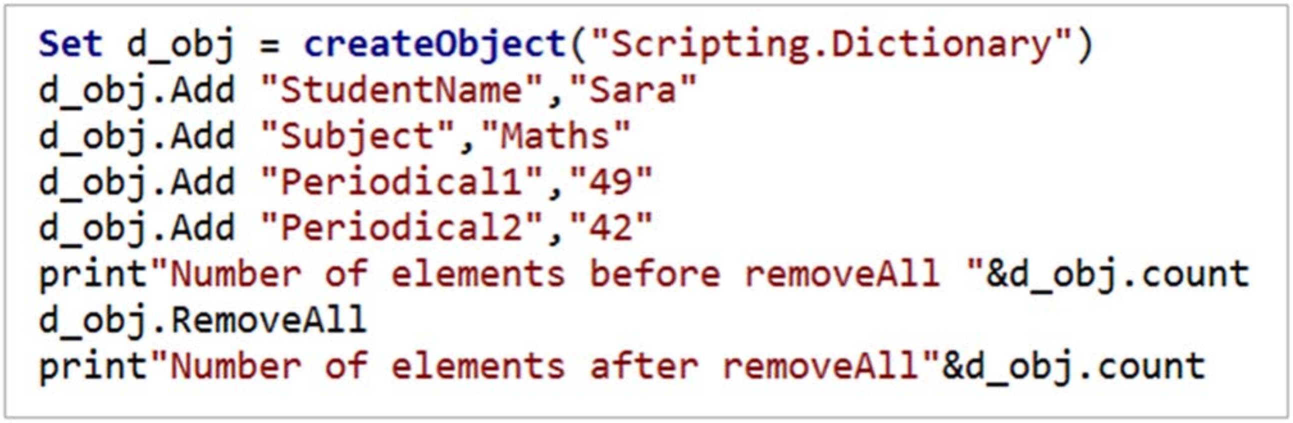
5. RemoveAll

It removes all the key/item pairs from a Dictionary object

Syntax:

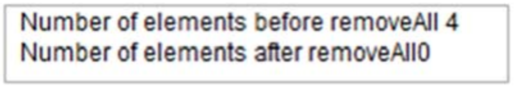
dictionaryObject.RemoveAll

Example:



```
Set d_obj = createObject("Scripting.Dictionary")
d_obj.Add "StudentName","Sara"
d_obj.Add "Subject","Maths"
d_obj.Add "Periodical1","49"
d_obj.Add "Periodical2","42"
print"Number of elements before removeAll "&d_obj.count
d_obj.RemoveAll
print"Number of elements after removeAll"&d_obj.count
```

Output:



```
Number of elements before removeAll 4
Number of elements after removeAll0
```

Demo 6 : Dictionary objects

Highlights:

- Implementation of dictionary object methods
- We will do it by creating a dictionary object with keys such as customername, age, location and height. We will determine the count of items present in the description object, create new key and will modifiy a key and item value

Demo Steps:

Step 1: Write the code as shown below.

```
'Creation of Dictionary Object d_obj
Set d_obj = createObject("Scripting.Dictionary")
'Add the keys and items to the Dictionary object d_obj
d_obj.Add "CustomerName","Jacob"
d_obj.Add "Age",49
d_obj.Add "Loc","Australia"
d_obj.Add "Height","5.10"
'print number of items present in dictionary object d_obj
print "Count of dictionary object :"& d_obj.count
```

Step 2: Determine the new count of items as given in the code

```
print "count of items in the dictionary object before adding the new key and
item:" & d_obj.count
d_obj.add "NewCustomerName","Alex"
Print "Item value for key 'NewCustomername':" d_obj.item("NewCustomerName")
print "count of items in the dictionary object after  adding the new key and
item:" & d_obj.count
```

Step 3: Modifiy of the item of ‘Loc’ key as given below

```
print "Item value of 'Loc' key before modification "&d_obj.Item("Loc")
d_obj.item("Loc")="paris"
```

```
print "Item value of 'Loc' key after modification "&d_obj.Item("Loc")
```

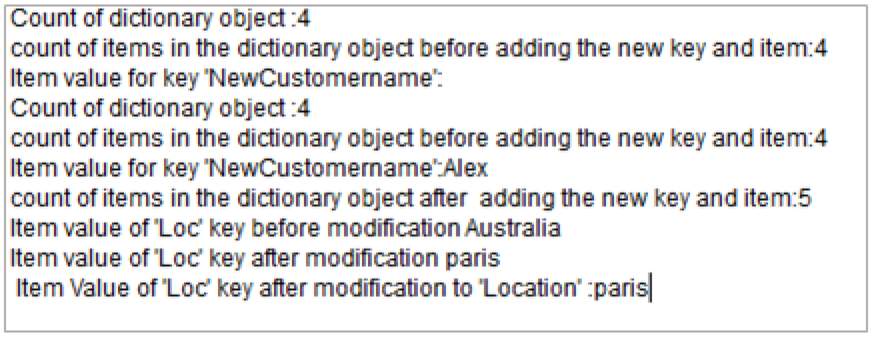
Step 4: Modifiy the key ‘Loc’ to ‘Location’ as given in the code

```
d_obj.key("Loc")="Location"

print " Item Value of 'Loc' key after modification to 'Location' :" &d_obj("L
ocation")
```

Step 5: The final script will look like this

```
'Creation of Dictionary Object d_obj

Set d_obj = createObject("Scripting.Dictionary")

'Add the keys and items to the Dictionary object d_obj

d_obj.Add "CustomerName","Jacob"

d_obj.Add "Age",49

d_obj.Add "Loc","Australia"

d_obj.Add "Height","5.10"

'print number of items present in dictionary object d_obj

print "Count of dictionary object :"& d_obj.count

print "count of items in the dictionary object before adding the new key and
item:" & d_obj.count

d_obj.add "NewCustomerName","Alex"

Print "Item value for key 'NewCustomername':" d_obj.item("NewCustomerName")

print "count of items in the dictionary object after  adding the new key and
item:" & d_obj.count

print "Item value of 'Loc' key before modification "&d_obj.Item("Loc")

d_obj.item("Loc")="paris"

print "Item value of 'Loc' key after modification "&d_obj.Item("Loc")

d_obj.key("Loc")="Location"

print " Item Value of 'Loc' key after modification to 'Location' :" &d_obj("L
ocation")
```

Step 6: Upon executing the script, the output must be printed as shown below.



```
Count of dictionary object :4
count of items in the dictionary object before adding the new key and item:4
Item value for key 'NewCustomername':
Count of dictionary object :4
count of items in the dictionary object before adding the new key and item:4
Item value for key 'NewCustomername':Alex
count of items in the dictionary object after  adding the new key and item:5
Item value of 'Loc' key before modification Australia
Item value of 'Loc' key after modification paris
 Item Value of 'Loc' key after modification to 'Location' :paris
```

Excel automation

Why excel automation

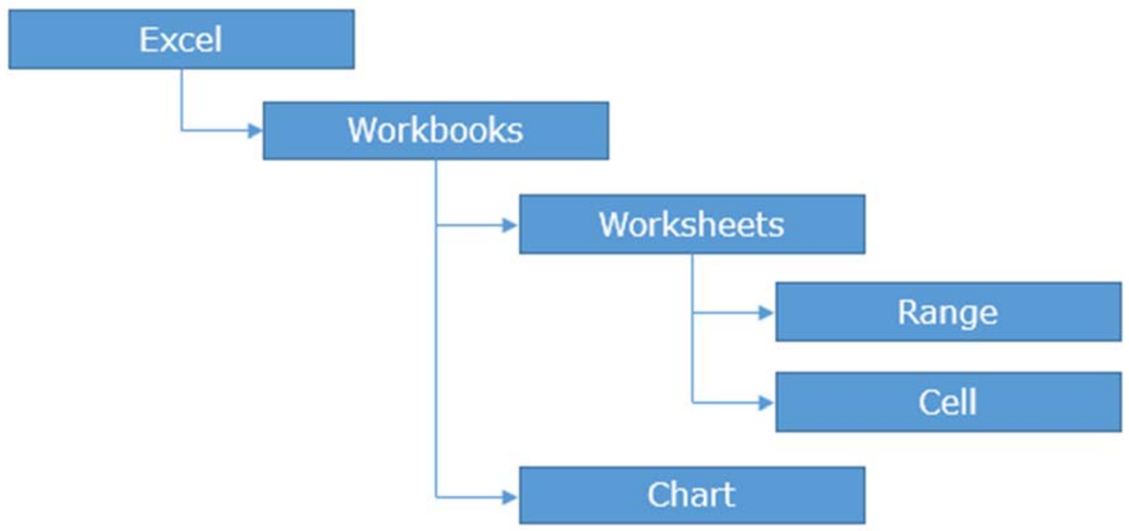
At times there is huge set of data which tester has to deal with while testing the application. There are multiple sources where that data can be kept like DBMS, .csv files, xml files, excel etc.

Any of the suitable option can be selected out of all these for creating a data pool. However, excel is considered as the one easy to operate and efficient from usage perspective. Here in this section we will look at the various ways of automating the excel which will help in performing the required operations.

Excel automation - Excel object model

Microsoft has developed the Excel application with hierarchy of object model. We can do excel operations using excel object model

Data present in the excel sheet can be accessed only if the hierarchy is properly defined in the script. So in order to perform any operation on the cell (where data is present), objects must be created for the components present above the hierarchy of the 'Cell' object and should be referenced properly. See the below diagram for understanding excel object hierarchy:



Demo 8 : Excel automation

Highlights:

- Implementation of excel automation methods
- We will do the creation and destruction of an excel object

Demo Steps:

Step 1: Declare object for excel application.

```
Dim objExcel
```

Step 2: Create object for excel application.

```
Set objExcel=createObject("Excel.Application")
```

Step 3: Make the excel application visible. By default it is set to 'false'. If the value is set to false all operations will be performed but we cannot see them.

```
objExcel.Visible=true
```

Demo 9 : Excel automation

Highlights:

- Implementation of excel automation methods
- We will do the creation of excel workbook,excel worksheet and how to read the data from a cell.

Demo Steps:

Step 1: Declare objects for Excel application, WorkBook, WorkSheet.

```
Dim objExcel
Dim objWorkBook
Dim objWorkSheet
```

Step 2: Create objects for Excel application and make the object visible.

```
Set objExcel=createObject("Excel.Application")
objExcel.Visible=true
```

Step 3: Use the below statement if you are creating a new excel workbook. Add a workbook to Excel application. Add method returns an object that refers to newly created workbook.

```
Set objWorkBook=objExcel.Workbooks.Add
```

Step 4: Use the below statement if you are accessing an existing excel workbook. Opens an existing workbook present in the specified path. 'objWorkBook' points to that workbook.

```
Set objWorkBook=objExcel.Workbooks.Open("Path of excel file")
```

Step 5: Refer the worksheet you want to work with.

```
Set objWorkSheet=objWorkBook.Worksheets("SheetName")
```

Step 6: Read data from cell at first row, first column position in the worksheet. cells() function contains parameters as rowid, columnid.

```
msgbox objWorkSheet.cells(1,1)
```

Step 7: Close the excel application instance.

```
objExcel.quit
```

Note: If you are following "Step03" to create a new workbook, then to save the workbook on filesystem use the below mentioned code.

```
objWorkBook.SaveAs "Path of the file to save with .xls extension."
```

Demo 10 : Excel automation

Highlights:

- Implementation of excel automation methods
- We will do adding,renaming and deleting a sheet in excel.

Demo Steps:

Step 1: Declare excel, workbook and worksheet object.

```
Dim oExcel
Dim oWB
Dim oSheet
```

Step 2: Create excel application object and make it visible.

```
set oExcel = createobject("Excel.application")
oExcel.visible = True
```

Step 3: Open external excel workbook.

```
set oWB = oExcel.Workbooks.Open("Path of the excel file.")
```

Step 4: Add a new sheet in the workbook.

```
set oSheet = oWB.Sheets.add
```

Step 5: Rename newly added sheet.

```
oSheet.Name = "Give some name to the sheet."
```

Step 6: Delete newly added sheet.

```
oSheet.delete
```

Step 7: Save workbook.

```
oWB.Save
```

Step 8: Close workbook.

```
oWB.close
```

Step 9: Destroy all object references.

```
Set oExcel = nothing
Set oWB = nothing
Set oSheet = nothing
```

Demo 11 : Excel automation

Highlights:

- Implementation of excel automation methods
- We will update a cell value in the worksheet and will save the workbook after updating

Demo Steps:

Step 1: Declare excel, workbook and worksheet objects.

```
Dim oExcel
Dim oWB
Dim oSheet
```

Step 2: Create excel application object and make it visible.

```
Set oExcel = createobject("Excel.application")
oExcel.visible = True
```

Step 3: Open workbook and refer to the worksheet.

```
set oWB = oExcel.Workbooks.open("Path of the file with extension")
Set oSheet = oExcel.Worksheets.Item("Demo_Sheet")
```

Step 4: Update value in the cell and save the workbook.

```
oSheet.cells(rowid,columnid).value = "Data"
oWB.Save
```

Step 5: Close workbook and destroy all object references.

```
oWB.close
Set oExcel = nothing
Set oWB = nothing
Set oSheet = nothing
```

Demo 12 : Excel automation

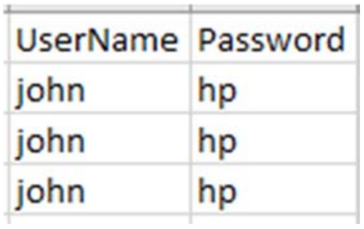
Highlights:

- Implementation of excel automation using windows application as AUT

- We will do it by parameterizing the login functionality using excel sheet

Demo Steps:

Create an excel sheet with below mentioned agent name and password details.



UserName	Password
john	hp
john	hp
john	hp

Login to the flight reservation application, with the help of above mentioned credentials using either recording feature of UFT or using Descriptive Programming.

Step 1: Create objects and references to fetch data from the excel sheet.

```
Dim objExcel
Dim objWorkBook
Dim objWorkSheet
Dim username, password, i, rowcount
Set objExcel=createObject("Excel.Application")
objExcel.Visible=true
Set objWorkBook=objExcel.Workbooks.Open("Path of the file.")
Set objWorkSheet=objWorkBook.Worksheets("Sheet Name")
```

Step 2: Count the number of rows that needs to be used for fetching details from the sheet.

```
'Below statement retrieves number of rows in the worksheet
rowCount=objWorkSheet.usedRange.rows.Count
```

Step 3: Use the details to login to the application.

```
'Loop through all rows starting from row 2.
```

```
For i=2 to rowCount
   ‘Retrieve username and password
   userName=objWorkSheet.Cells(i,1)
   password=objWorkSheet.Cells(i,2)

   ‘Invoke flight reservation application
    SystemUtil.Run "C:\Program Files\HP\Unified Functional Testing\samples\fl
ight\app\flight4a.exe"

   ‘Set userName, Password retrieved from excel sheet
   WpfWindow(myWindow).WpfEdit("devname:=agentName").Set username
   WpfWindow(mywindow).WpfEdit("devnamepath:=password;;").set "hp"
   WpfWindow(myWindow).WpfButton("devname:=okButton").Click
   ‘Exit flight reservation application
   WpfWindow(myWindow).Close
Next

‘Close excel sheet
objExcel.Quit
'Destroy all objects references
Set objWorkSheet=Nothing
Set objWorkBook=Nothing
Set objExcel=Nothing
```

File automation

Why File automation

Files play a major role in the automation testing process. In many scenarios you have to read the data from the provided files and have to fetch into your application. Same way you have to extract the data from the application and have to write into a file.

For example, there is a business requirement to verify the 'About Us' detail of a web application. The expected data is given in a text file by the customer and you are supposed to verify whether this data is matching with the application data. File automation helps you in this scenario to read the data from file through scripting.

FileSystemObject model provides a rich set of properties and methods that you can use to process them.

Many times you need to read input from a file or write output to a file. FSO model is used for handling files using UFT. Here in this course we will be dealing with file handling part only for demonstration purposes.

Before using the methods and properties of FileSystemObject, the FSO object should be created.

```
Dim fso

Set fso = Createobject("Scripting.filesystemobject")
```

File automation - Methods of filesystemobject

Below mentioned are few of the methods used with filesystemobject while dealing with files.

CreateTextFile

Description:

It creates a specified file name and returns a **TextStream** object that can be used to read from or write to the file.

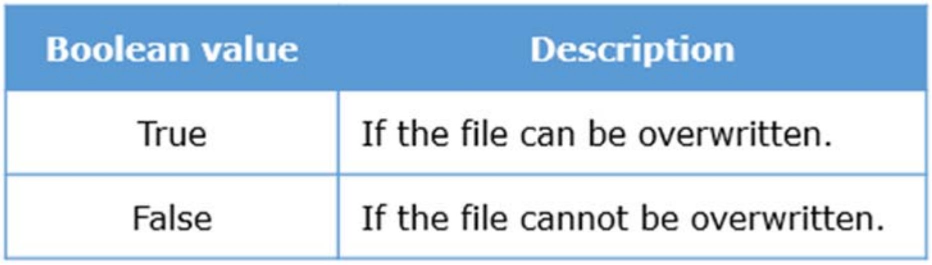
The **TextStream** object provides sequential access to the contents of any file where the contents are in text-readable form.

Syntax:

```
FileSystemObject.CreateTextFile(file_name,overwrite)
```

`file_name` - This argument is used for passing the file name with complete path.

`overwrite` - This argument is optional and indicates whether an existing file can be overwritten or not. It can have any of the following values:-



Boolean value	Description
True	If the file can be overwritten.
False	If the file cannot be overwritten.

Note:- If the overwrite argument is false, or is not provided, for a filename that already exists, an error is thrown.

OpenTextFile

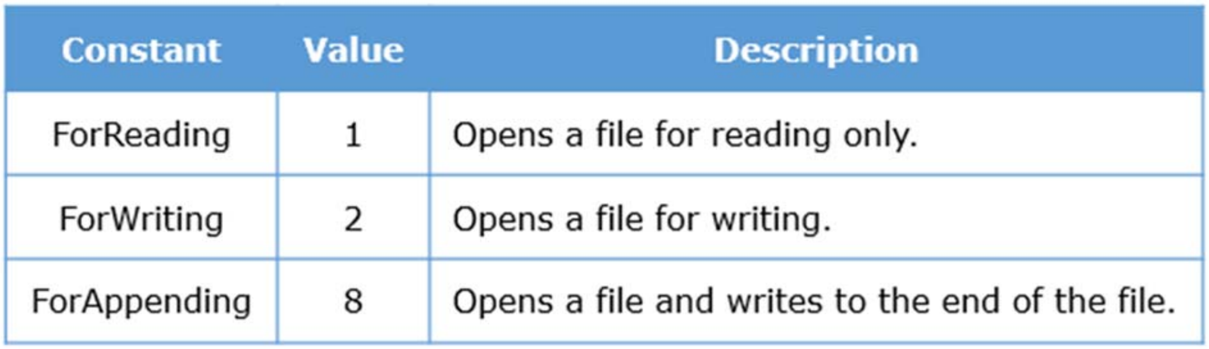
Description:

It opens a specified file and returns a **TextStream** object that can be used to read from, write to or append to the file.

Syntax:

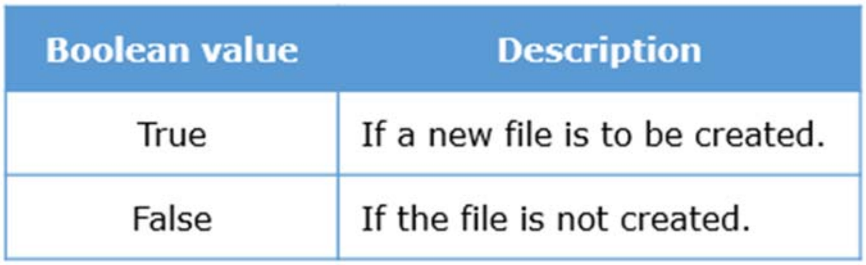
```
FileSystemObject.OpenTextFile (file_name,iomode,create)
```

`iomode` - This argument can have any of the following values:



Constant	Value	Description
ForReading	1	Opens a file for reading only.
ForWriting	2	Opens a file for writing.
ForAppending	8	Opens a file and writes to the end of the file.

`create` - This argument is optional and indicates whether a new file can be created if the specified filename doesn't exist. It can have any of the following values.



Boolean value	Description
True	If a new file is to be created.
False	If the file is not created.

If the create argument is not provided, a new file is not created. By default, value is taken as false.

CopyFile

Description:

It copies a file from one location to another.

Syntax:

```
FileSystemObject.CopyFile source, destination
```

DeleteFile

Description:

It deletes a specified file.

Syntax:

```
FileSystemObject.DeleteFile (file_name)
```

Demo 13 : File automation - Methods of filesystemobject

Highlights:

- Implementation of CreateTextFile method
- We will do it be creating a file named "Employee.txt"

Demo Steps:

Given below is the sample code to create a text file located in "C:\Users\Public\Pictures\Sample Pictures" location of the computer system.

```
Dim fso, file_location
Set fso = CreateObject("Scripting.FileSystemObject")
```

```
file_location = "C:\Users\Public\Pictures\Sample Pictures\Employee.txt"
Set file = fso.CreateTextFile(file_location,true)
```

Demo 14 : File automation - Methods of filesystemobject

Highlights:

- Implementation of OpenTextFile method.
- We will do it by writing data in "Employee.txt" file created using CreateTextFile method

Demo Steps:

Given below is the sample code to write inside the file "Employee.txt" .

```
Dim fso, file_location, file
Set fso = CreateObject("Scripting.FileSystemObject")
file_location = "C:\Users\Public\Pictures\Sample Pictures\Employee.txt"
' 2 is the value for writing into the file.
set file = fso.OpenTextFile(file_location, 2, False)
file.Write "Hello"
file.close
```

Alternative Code:

If the user wants to use the constants **ForWriting**, **ForReading** and **ForAppending** declare the constants.

```
Dim fso, file_location, file
Set fso = CreateObject("Scripting.FileSystemObject")
Const ForWriting = 2
file_location = "C:\Users\Public\Pictures\Sample Pictures\employee.txt"
Set file = fso.OpenTextFile(file_location, ForWriting, False)
file.Write "Hello"
file.Close
```

Demo 15 : File automation - Methods of filesystemobject

Highlights:

- Implementation of CopyFile method.
- We will do it by copying "Employee.txt" file from one location to another

Demo Steps:

Given below is the sample code to copy file "Employee.txt" from one location to another.

```
Dim fso, sourceFileLocation, destinationFileLocation
Set fso = CreateObject("Scripting.FileSystemObject")
sourceFileLocation = "C:\Users\Public\Pictures\Sample Pictures\Employee.txt"
destinationFileLocation = "C:\Users\Public\Pictures\Sample Music\"
fso.CopyFile sourceFileLocation, destinationFileLocation
```

Demo 16 : File automation - Methods of filesystemobject

Highlights:

- Implementation of DeleteFile method
- We will do it by deleting Employee.txt file.

Demo Steps:

Given below is the sample code to delete file "Employee.txt" from system.

```
Dim fso, file_location
Set fso = CreateObject("Scripting.FileSystemObject")
file_location = "C:\Users\Public\Pictures\Sample Pictures\Employee.txt"
fso.DeleteFile(file_location)
```

File automation - Methods of TextStream object

Below mentioned are few of the methods used with filesystemobject for reading and writing the data.

ReadLine

Description:

It reads an entire line (up to the newline character) from a **TextStream** file and returns the resulting string.

Syntax:

```
TextStreamObject.ReadLine()
```

ReadAll

Description:

It reads an entire **TextStream** file and returns the resulting string.

Syntax:

```
TextStreamObject.ReadAll()
```

Write

Description:

It writes a specified string to a **TextStream** file.

Syntax:

```
TextStreamObject.Write(string)
```

WriteLine

Description:

It writes a specified string and a newline character to a **TextStream** file.

Syntax:

```
TextStreamObject.WriteLine(string)
```

String argument is optional. If omitted, a newline character is added.

Close

It closes an open **TextStream** file.

Syntax:

```
TextStreamObject.Close()
```

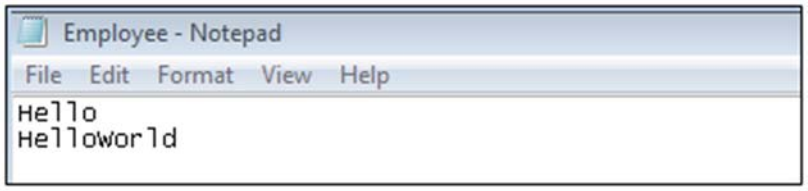
Demo 17 : File automation - Methods of TextStream object

Highlights:

- Implementation of ReadLine method
- We will do it by reading data from Employee.txt file.

Demo Steps:

Suppose the Employee.txt file contains values as:



```
Hello
Helloworld
```

Given below is the sample code to read data from file "Employee.txt".

```
Dim fso, file_location, file
Set fso = CreateObject("Scripting.FileSystemObject")
Const ForReading = 1
file_location = "C:\Users\Public\Pictures\Sample Pictures\Employee.txt"
Set file = fso.OpenTextFile(file_location, ForReading)
content = file.ReadLine()
print content
```

Output:

Hello

Note:- The ReadLine() method will read the first line of the file till new line character.

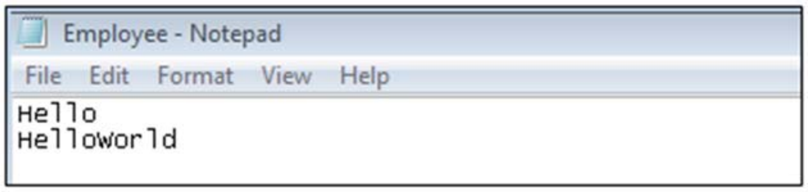
Demo 18 : File automation - Methods of TextStream object

Highlights:

- Implementation of ReadAll method.
- We will do it by reading all the data from Employee.txt file.

Demo Steps:

Suppose the Employee.txt file contains values as:



```
Hello
Helloworld
```

Given below is the sample code to read data from file "Employee.txt".

```
Dim fso, file_location, file
Set fso = CreateObject("Scripting.FileSystemObject")
Const ForReading = 1
file_location = "C:\Users\Public\Pictures\Sample Pictures\Employee.txt"
Set file = fso.OpenTextFile(file_location, ForReading)
content = file.ReadAll()
print content
```

Output:

Hello

HelloWorld

Demo 19 : File automation - Methods of TextStream object

Highlights:

- Implementation of write method.
- We will do it by writing data in a blank Employee.txt file.

Demo Steps:

Assume Employee.txt is a blank file.

Given below is the sample code to write data into "Employee.txt" file.

```
Dim fso, file_location, file
Set fso = CreateObject("Scripting.FileSystemObject")
file_location = "C:\Users\Public\Pictures\Sample Pictures\Employee.txt"
Set file = fso.OpenTextFile(file_location, 2, False)
file.Write "Hello"
file.Close
```

Demo 20 : File automation - Methods of TextStream object

Highlights:

- Implementation of WriteLine method.
- We will do it by writing data in Employee.txt file using WriteLine.

Demo Steps:

Assume Employee.txt is a blank file.

Given below is the sample code to write data into "Employee.txt" file.

```
Dim fso, file_location, file
Set fso = CreateObject("Scripting.FileSystemObject")
Const ForWriting = 2
file_location = "C:\Users\Public\Pictures\Sample Pictures\Employee.txt"
Set file = fso.OpenTextFile(file_location, ForWriting, false)
file.Write("Hello")
file.WriteLine("Hello World")
file.WriteLine("Hi")
file.WriteLine()
file.Write("Hi World")
file.Close
Set fileForReading = fso.OpenTextFile(file_location,1)
content = fileForReading.ReadAll()
print content
```

Output:

```
HelloHello World
Hi

Hi World
```

Demo 21 : File automation

Highlights:

- Implementation of various file automation methods.
- We will do it by fetching details of login page and writing inside a file.

Demo Steps:

Open the login page of flight reservation application. Count the number of childobjects present on the login page. Highlight the objects. Put their class names in a file and print those values from that file in console/output window in UFT. Delete the text file created after reading all the values. Click the Cancel button on login page and close the application.

Step 1: Launch the flight reservation application.

```
systemutil.Run "C:\Program Files\HP\Unified Functional Testing\samples\flight
\app\flight4a.exe"
```

Step 2: Count the number of childobjects on the page and display it on screen.

```
Set demo = dialog("regexpwndtitle:=Login").ChildObjects()
```

```
msgbox "total objects are:" & demo.count
```

Step 3: Highlight all the objects present on the login page.

```
For i = 0 To demo.count-1 Step 1
       demo(i).highlight
Nex
```

Step 4: Create the text file and put all the object's class names in that file.

```
Set fso = createobject("Scripting.filesystemobject")
Set file = fso.OpenTextFile("C:\Users\vishal_mangal\Desktop\Demo scripts\resu
lt.txt", 2, true)
For i = 0 To demo.count-1 Step 1
       file.WriteLine(demo(i).getroproperty("class name"))
Next
file.Close
```

Step 5: Print all the values in console from the file and delete the file from system.

```
Set file = fso.OpenTextFile("C:\Users\vishal_mangal\Desktop\Demo scripts\resu
lt.txt", 1, true)
print file.ReadAll
file.Close
fso.DeleteFile("C:\Users\vishal_mangal\Desktop\Demo scripts\result.txt")
Set file = nothing
Set demo = nothing
```

Step 6: Close the application.

```
dialog("regexpwndtitle:=Login").winbutton("regexpwndtitle:=Cancel").click
```

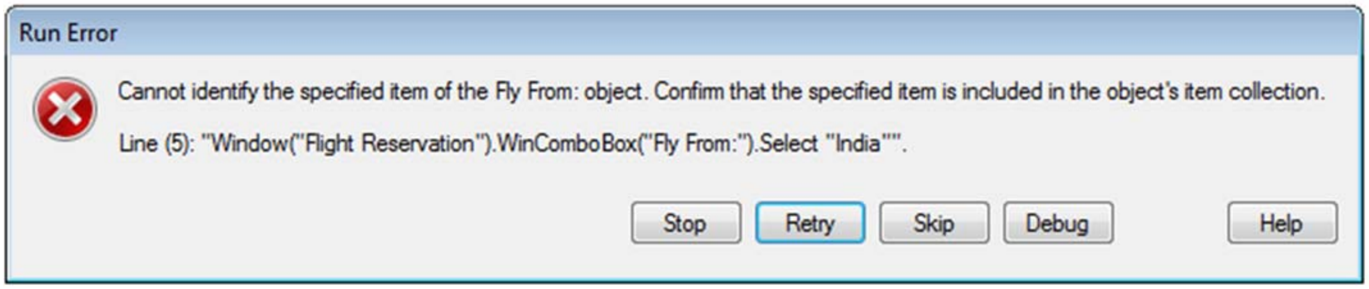
Recovery scenario

What is recovery scenario

A test run can be disrupted due to unexpected events, errors, or application crashes. This is a serious problem particularly when tests are running unattended. The test gets suspended until an action is taken to recover from the unexpected event.

Recovery scenarios are crucial for large tests and useful to resume the execution when it is disturbed by any unexpected event. When exception occurs, the information is collected by UFT recovery mechanism and a predefined recovery scenario is executed to overcome the error and to resume the execution.

For example, consider an insert order script for flight reservation application for a flight from 'India' to 'Denver'. Due to developer's negligence he forget to add 'India' in the drop down list and when you execute your script it will throw you the below error at the corresponding line and the script will never proceed with the execution.


Run Error

Cannot identify the specified item of the Fly From: object. Confirm that the specified item is included in the object's item collection.

Line (5): "Window("Flight Reservation").WinComboBox("Fly From:").Select "India"".

These kind of unexpected scenarios can be effectively handled using the recovery scenario option.

Recovery scenario

A recovery scenario have three main components.

1. Trigger event
2. Recovery operations
3. Post recovery run options

1.Trigger event

Trigger event is the event that has caused interruption in the test run.Four types of trigger events are available in UFT.

a. Popup window

UFT detects a pop-up window and identifies it according to the window title and textual content. For example, a message box may open during a test run indicating that the printer is out of paper. UFT can detect this window and activate a defined recovery scenario in order to continue the test run.

b. Object state

UFT detects a specific test object state and identifies it according to its property values and the property values of all its ancestors. Note that an object is identified only by its property values, and not by its class.For example, a specific button in a dialog box may be disabled when a specific process is open. UFT can detect the object property state of the button that occurs when this problematic process is open and activate a defined recovery scenario to close the process and continue the test run.

c. Test run error

UFT detects a test run error and identifies it by a failed return value from a method. For example, UFT may not be able to identify a menu item specified in the method argument, due to the fact that the menu item is not available at a specific point during the test run. UFT can detect this test run error and activate a defined recovery scenario in order to continue the test run.

d. Application crash

UFT detects an application crash and identifies it according to a predefined list of applications. For example, a secondary application may crash when a certain step is performed in the test run. We want to be sure that the test run does not fail because of this crash, which may indicate a different problem with your application. UFT can detect this application crash and activate a defined recovery scenario to continue the test run.

2. Recovery operations

Recovery operations are those operations which have to be performed when the specified trigger event get activated. Four types of recovery operations are available in UFT.

a. Keyboard or mouse operation

A Keyboard stroke/ mouse click will be performed. It will be used when there is only one step operation required.

Example: Click a button

b. Close application process

The specified process will be closed. It will be used when any other application is interrupting the execution.

Example: An antivirus system

c. Function call

A library function will be called. It will be used when there are multiple steps to be executed.All the needed steps to be executed will be saved inside a function and this function will be pointed out as the one which have to be executed when the trigger event occurs.

d. Restart Microsoft Windows

This option can be used for UFT to restart the Operating System. This will be used when the execution is blocked because of a pending restart.

3. Post recovery run options

Once the recovery operations are completed you have to instruct UFT what to do next regarding the execution process.Six types of post recovery run options are available in UFT.

a. Repeat Current Step and Continue

By default, recovery scenarios will be executed when execution got interrupted due to an error. There is a facility to execute recovery scenarios for each step. UFT understands that there is an error only when a step got failed. If that is important step and recovery operation is covering it, then we choose **repeat current step and continue** option.

Example: Login button got disabled. UFT tries to click on it and failed. Recovery operation did something to enable Login button. UFT already executed clicking on Login button and it will try to execute next step which will become fail. In this case repeat current step and continue will make UFT to re-execute the clicking on login button step.

b. Proceed to Next Step

If the recovery operation completely clears the way for execution then this option will be useful.

c. Proceed to next Action or Component Iteration

Exit from current action iteration and starts executing next Iteration. This will be used when a test is using actions concept and you want to skip only that action iteration execution. This option is useful to execute the same action with next set of data that is available in data table.

d. Proceed to next Test Iteration

Exit from current test iteration and starts executing next iteration. This option is useful to execute the same test with next set of data that is available in data table.

e. Restart current test run

This option is used when the functional flow gets disturbed and is not able to continue the execution.

f. Stop the test run

This will be used when application is down or an open defect is still there and there is no need to continue the execution of the test.

Debugger

What is debugger

A debugger is a tool to help you follow the logic of your test code as it runs. Debugging is attempting to figure out the cause of a problem in your test script, and then taking action to fix the problem.

Using the Debug option in UFT you can run your test line by line. This is helpful when developing your script when you're having issues and you need to troubleshoot a problematic area in the script.

The following are the various ways to debug your scripts

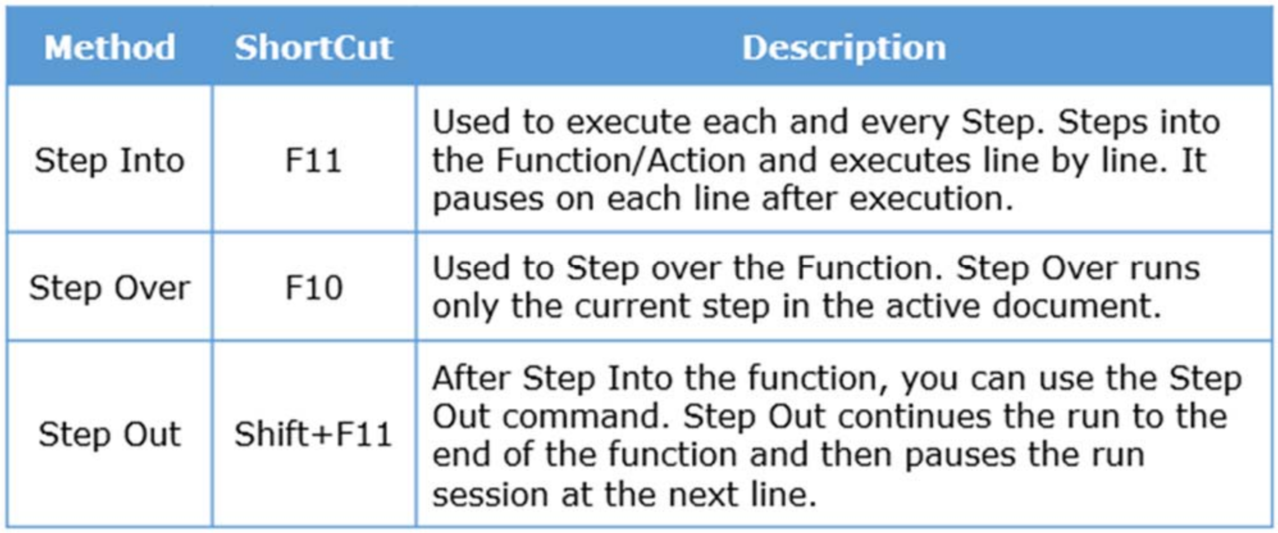
1. Stepping
2. Break points
3. Call stack
4. Local variables
5. Console
6. Watch

Let us see the above options in detail

Debugger – Stepping

1. Stepping

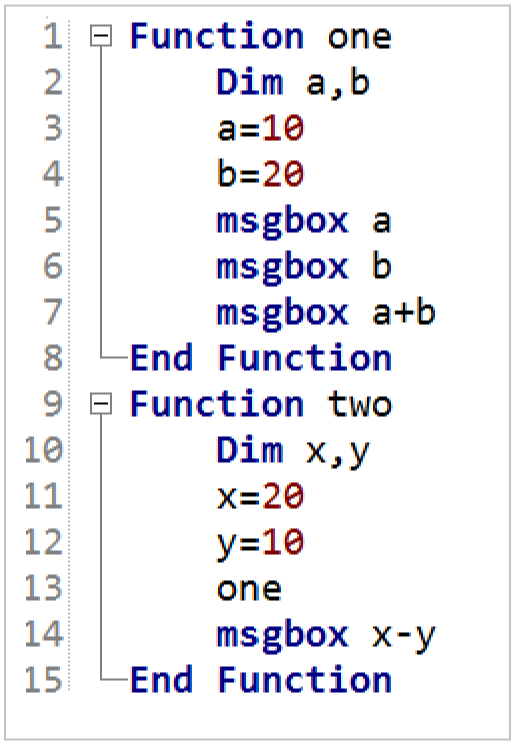
While debugging your script you can navigate through your script in a few different ways. The UFT's debugger has three kinds of navigation options, called "stepping"



Method	ShortCut	Description
Step Into	F11	Used to execute each and every Step. Steps into the Function/Action and executes line by line. It pauses on each line after execution.
Step Over	F10	Used to Step over the Function. Step Over runs only the current step in the active document.
Step Out	Shift+F11	After Step Into the function, you can use the Step Out command. Step Out continues the run to the end of the function and then pauses the run session at the next line.

Example:

Create a function library having the below two functions



```
Function one
    Dim a,b
    a=10
    b=20
    msgbox a
    msgbox b
    msgbox a+b
End Function
Function two
    Dim x,y
    x=20
    y=10
    one
    msgbox x-y
End Function
```

Call the above functions one and two using the below syntax in your main test

```
one
two
```

1. Let us run the calling functions using **step into method by pressing F11**

Keep on pressing F11 and you can notice the below points

- Step into will execute line by line in your main test
- If the line is a function call, control will get into the function and each line inside the function will be executed line by line

2. Let us run the calling functions using **step over method by pressing F10**

For doing step over first you have to start with step into.Press F11 one time and then keep on pressing F11 and you can notice the below points

- Step into will execute line by line in your main test

- If the line is a function call, control will get into the function and the whole function will be executed together and the control will come to the next line of your test and will wait.

3. Let us run the calling functions using **step out method by pressing Shift+F11**

Step out will be used if your control is inside a function.Press F11 and get into a function.

Press Shift+F11 and you can notice the below points

- If you are inside a function,step out will execute the remaining lines of your function together and control will come out of the function
- Once you are inside a function stepinto and stepover will execute the lines one by one.

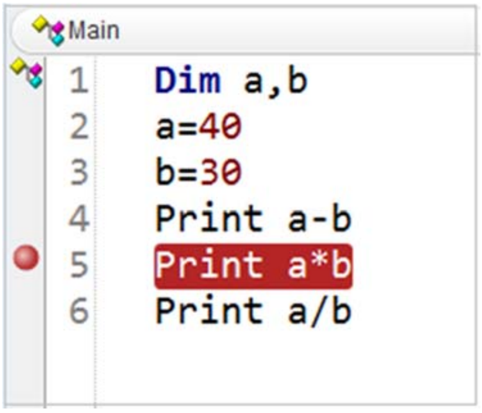
Debugger - Breakpoints,Callstack

2. Breakpoints

Breakpoints allow you to stop at a particular line of code in your test script. This is a really handy feature if you've got a long script that is failing near the end of the test run and you need to pause the running to see why an issue is occurring.

Example:

Create the below script. Place your cusrson at the starting of line 5 and press F9. You can see a breakpoint will be created on line number 5. Break point can be inserted/removed also through Run->Insert/Remove breakpoint.



```
1   Dim a,b
2   a=40
3   b=30
4   Print a-b
5   Print a*b
6   Print a/b
```

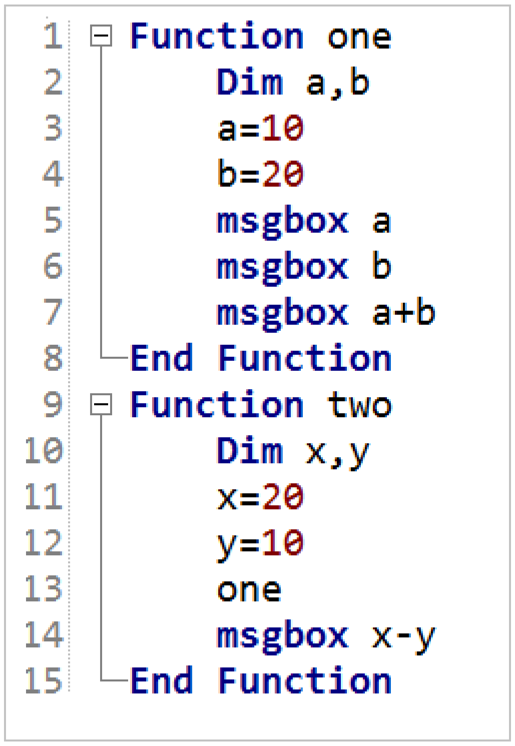
Execute your script and observe the output. You script will stop executing after line number four and the output will be 10.

3. Call stack

A call stack is a view of the nested sequence of functions that have led to a certain point in your test code. This can be helpful if you’re debugging and are paused inside a function and are not exactly sure how the path of execution got you to that point. It can help you when debugging long, nested functions.

Example:

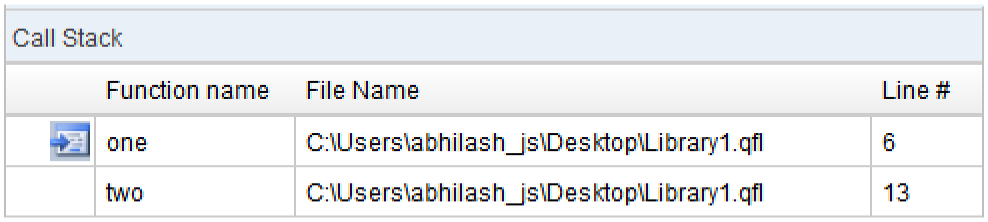
Create the below functions and call the function 'two' by step into process.



```
 1  Function one
        Dim a,b
 3      a=10
 4      b=20
 5      msgbox a
 6      msgbox b
 7      msgbox a+b
 8  End Function
 9  Function two
10      Dim x,y
11      x=20
12      y=10
13      one
14      msgbox x-y
15  End Function
```

Keep on pressing F11 until the control reaches line number six. At this point you are in line number 6 inside function one and you reached function one through line number 13 inside function two

Select View->Debug->Call stack and it will show you in which function you are and through which function you reached there.



Call Stack

	Function name	File Name	Line #
	one	C:\Users\abhilash_js\Desktop\Library1.qfl	6
	two	C:\Users\abhilash_js\Desktop\Library1.qfl	13

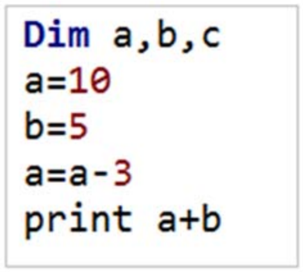
Debugger - Local variables,Console,Watch

4. Local variables

Local variables shows you the value of all the variables used in the script.

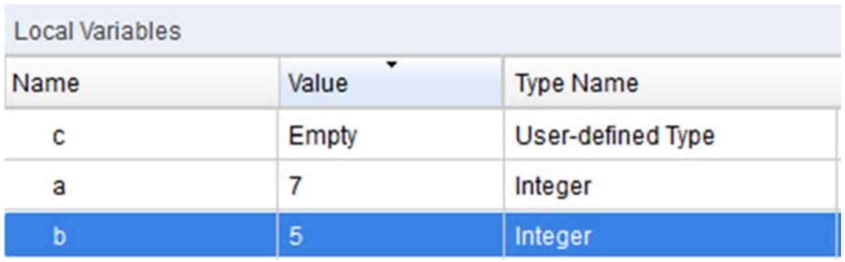
Example:

Create the below script and execute it using step into.



```
Dim a,b,c
a=10
b=5
a=a-3
print a+b
```

Once you reaches the print statement, open the local variables window through View->Debug->Local variable. You can see the value of all the variables at that point.



Local Variables

Name	Value	Type Name
c	Empty	User-defined Type
a	7	Integer
b	5	Integer

5. Console

The console can be a pretty powerful debugging tool. While you are debugging and UFT is paused on a line of code, you can perform multiple activities like

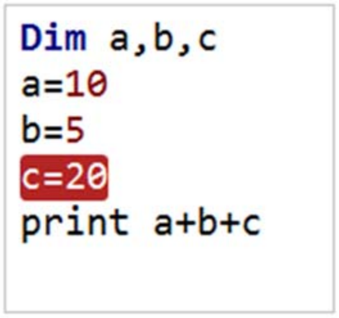
- Obtain info from the application under test
- Run a test method and display the return value

- Set or modify variables
- Call a native run time object method in your app

You can interact with the console by typing a line of code to be run in the context of your suspended run session.

Example:

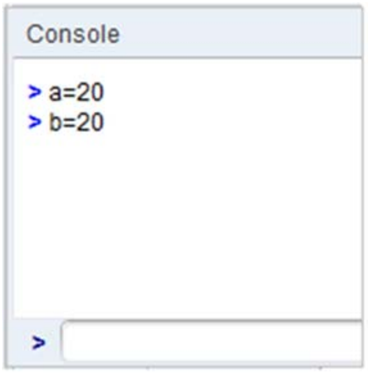
Create the below script. Put a break point at fourth line and execute the script.



```
Dim a,b,c
a=10
b=5
c=20
print a+b+c
```

Your script will stop at line number four. Open console by selecting View->Debug->Console.

Enter a=20 and b=20 inside the console like the below figure and stepover your script.



```
> a=20
> b=20
```

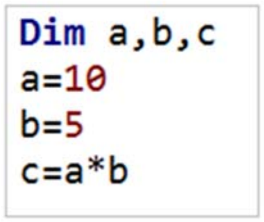
The output of your script will be 60 since the values of a and b got manipulated through the console.

6. Watch

Watch allows you to set specific values to watch for selected variables and expressions in your test's suspended run session. In a real world test script, you might have many variables but you may only be interested in 1 or 2 or them for your debugging purposes. This pane allows you to specify only the variables that you would like to watch during the execution of your tests.

Example

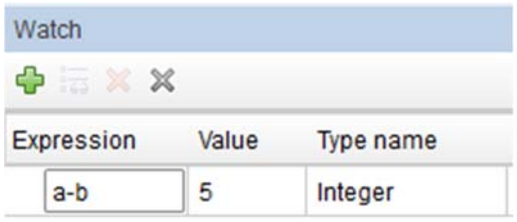
Create the below script and execute it using step into.



```
Dim a,b,c
a=10
b=5
c=a*b
```

Open the watch window through View->Debug->Watch.Select add watch new expression and

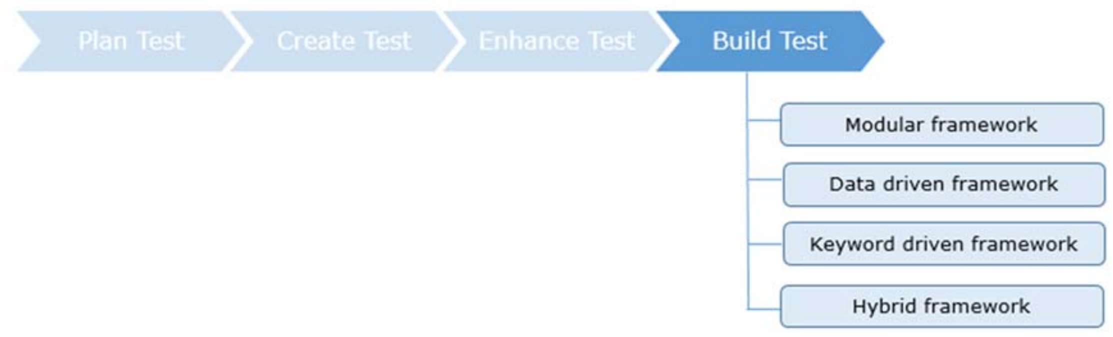
give 'a-b'. Execute through your script by pressing F11 and you can observe the watch value at each line.



Watch

Expression	Value	Type name
a-b	5	Integer

Build test - Automation framework



Till now you have learn how to write UFT scripts in a highly efficient and maintainable manner with the help of advanced scripting techniques. A complete UFT test script can be build with the help of automation frameworks.

Moving on wards you will see different available frameworks which is the best approach to create any projects.

Automation framework

An automation framework is a combination of various guidelines, coding standards, project hierarchies, modularity, reporting mechanism, test data injections etc. to pillar automation testing. A user can follow these guidelines while automating application to take advantages of various productive results.

The advantages in using a framework are as follows.

1. Frameworks reduces the manual efforts to a limit, works can be completed with minimal user intervention.
2. It will reduce maintenance cost with effective project repositories.
3. In case of any changes in the web application, instead of changing test scripts, only the test case file needs to be updated.

4. Test data can be separately kept in excel files, no need to include them inside the test scripts. So in case of any modifications, you can simply manipulate the data in the excel file.
5. Common library files can be reused when required, no need to develop them every time. Thus it saves time of the team and improves the productivity.
6. It also provides us the benefits of code re-usability, role-based access, easy reporting, consistency and maximum test coverage.

The four most popular testing frameworks are:

1. Modular
2. Data-Driven
3. Keyword
4. Hybrid

Automation framework - Modular framework

Modular Framework is based on the object oriented programming concept of abstraction. The framework divides the entire "Application Under Test" into a number of logical and isolated modules. For each module, you have to create a separate and independent test script. These test scripts taken together builds a larger test script representing more than one modules.

Advantages

- The framework introduces high level of modularization which leads to easier maintenance.
- If the changes are implemented in one part of the application, only the test script representing that part of the application needs to be fixed leaving all the other parts untouched.

Disadvantages

While implementing test scripts for each module separately, you embed the test data (Data with which we are supposed to perform testing) into the test scripts. Thus, whenever you are supposed to test with a different set of test data, it requires the manipulations to be made in the test scripts.

Automation framework - Data driven framework

Data Driven Testing Framework helps the user separate the business logic and the test data from each other. It lets the user store the test data in an external data source. The external data source can be a property file or xml file or excel sheet or text file or ODBC repository or a combination of multiple types. The data is conventionally stored as a “Key-Value” pair and hence the key can be used to access and populate the data within the test scripts.

Advantages

- It considerably reduces the total number of test scripts required to cover all the possible combinations of test scenarios and hence lesser amount of code is required to test a complete set of scenarios.
- Any change in the test data would not affect the automation scripts created.
- Increases flexibility and maintainability

Disadvantage

The process requires good knowledge of the programming language as we need to develop the generalized logic to read data from any data sets.

Automation framework - Keyword driven framework

Keyword driven framework is a type of functional automation testing framework which is also known as table-driven testing or action word based testing. In this framework certain set of code belonging to the test script is kept in an external data file.

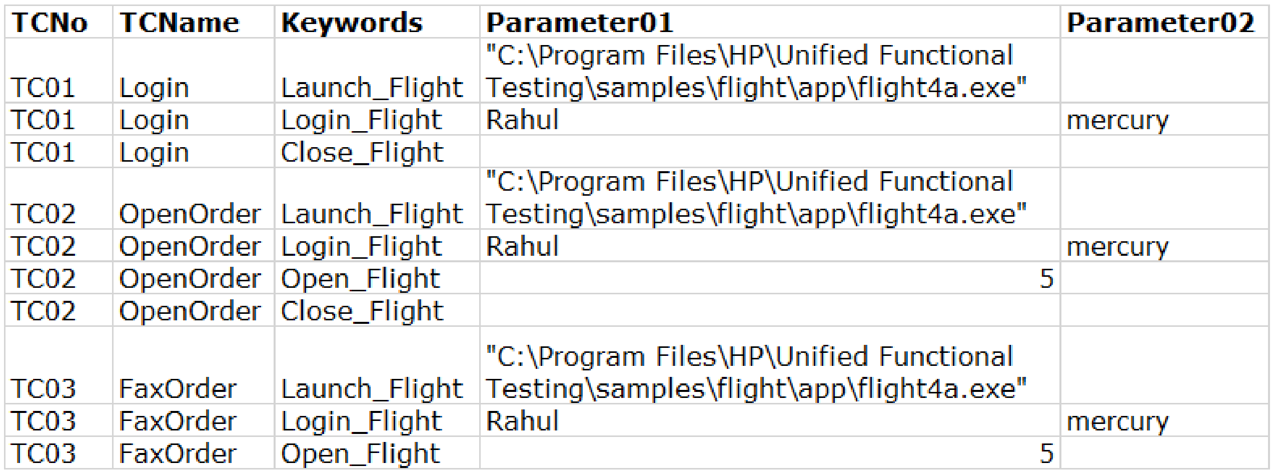
These set of code are known as keywords. These will be generalized functions written in a class or in a function to perform certain sets of actions.

Keywords can be created in two ways

1. Application dependent
2. Application independent

1.Application dependent

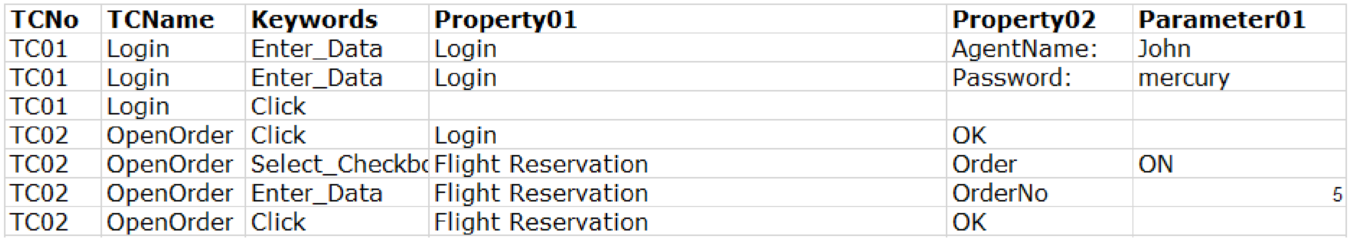
Application dependent keywords can be used for any scripts for that particular application.In these type of keywords, only test data will be passed as arguments and the whole data will be saved in an excel sheet like the one given below



TCNo	TCName	Keywords	Parameter01	Parameter02
TC01	Login	Launch_Flight	"C:\Program Files\HP\Unified Functional Testing\samples\flight\app\flight4a.exe"	
TC01	Login	Login_Flight	Rahul	mercury
TC01	Login	Close_Flight		
TC02	OpenOrder	Launch_Flight	"C:\Program Files\HP\Unified Functional Testing\samples\flight\app\flight4a.exe"	
TC02	OpenOrder	Login_Flight	Rahul	mercury
TC02	OpenOrder	Open_Flight	5	
TC02	OpenOrder	Close_Flight		
TC03	FaxOrder	Launch_Flight	"C:\Program Files\HP\Unified Functional Testing\samples\flight\app\flight4a.exe"	
TC03	FaxOrder	Login_Flight	Rahul	mercury
TC03	FaxOrder	Open_Flight	5	

2. Application independent

Application independent keywords can be used for any scripts in any application.In these type of keywords, object properties and test data will be passed as arguments and the whole data will be saved in an excel sheet like the one given below



TCNo	TCName	Keywords	Property01	Property02	Parameter01
TC01	Login	Enter_Data	Login	AgentName:	John
TC01	Login	Enter_Data	Login	Password:	mercury
TC01	Login	Click			
TC02	OpenOrder	Click	Login	OK	
TC02	OpenOrder	Select_Checkbo	Flight Reservation	Order	ON
TC02	OpenOrder	Enter_Data	Flight Reservation	OrderNo	5
TC02	OpenOrder	Click	Flight Reservation	OK	

In the above examples keywords like **Login_Flight**, **Click**, **Enter_Data** etc are defined within the code. All the keywords can be reused multiple times in test cases.

All the required keywords are designed and placed in base code of the framework.

Advantages

- A single keyword can be used across multiple test scripts.

Disadvantage

- The user should be well versed with the Keyword creation strategy.
- The framework becomes complicated as it grows and the number of keyword increases.

Automation framework - Hybrid framework

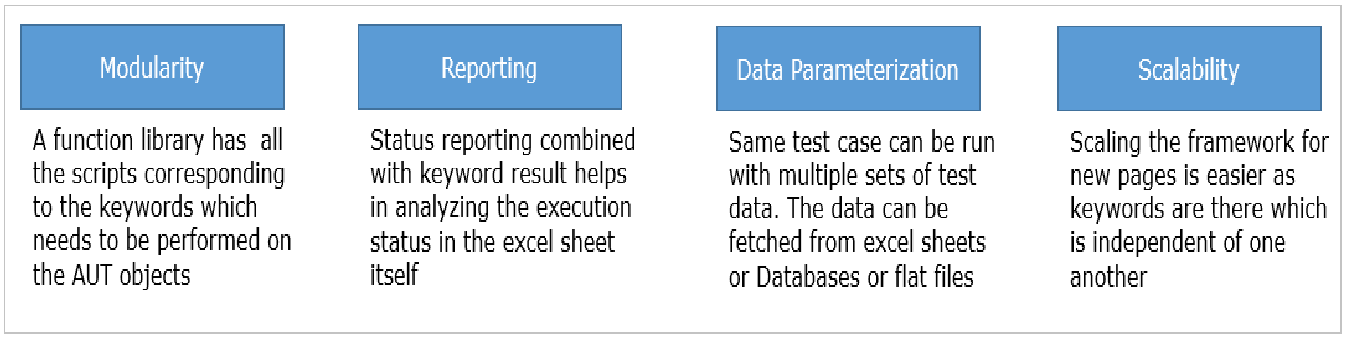
Hybrid test framework is a concept with the advantages of both keyword and data driven framework.

Features expected from hybrid framework

Please note that hybrid framework can be implemented by combining the features from the following:

1. Modular and Data-Driven
2. Data-Driven and Keyword-Driven
3. Combination of all three

You will be combining modular and data-driven concepts to create the hybrid framework. The below mentioned method is one of many ways to create a hybrid framework and the usage of the below mentioned way solely depends upon the project requirement.



Modularity

A function library has all the scripts corresponding to the keywords which needs to be performed on the AUT objects

Reporting

Status reporting combined with keyword result helps in analyzing the execution status in the excel sheet itself

Data Parameterization

Same test case can be run with multiple sets of test data. The data can be fetched from excel sheets or Databases or flat files

Scalability

Scaling the framework for new pages is easier as keywords are there which is independent of one another

In the previous demos and exercises, you have implemented the scripts inside an action or in a function library. Till now you have integrated the test data required for test execution along with the test script in many of the exercises. Integrating the code to fetch the test data and run the script multiple times with various test data sets, you are satisfying the conditions for creating a hybrid framework.

Case study - UFT implementation in an insurance project

Here is a case study of one of Infosys's client who achieved significant reduction, in the effort and cost involved in testing, with the implementation of test automation using UFT.

About the client

The client was a leading provider of insurance in United States. Their services consisted of individual and group insurances for life, dental, disability.

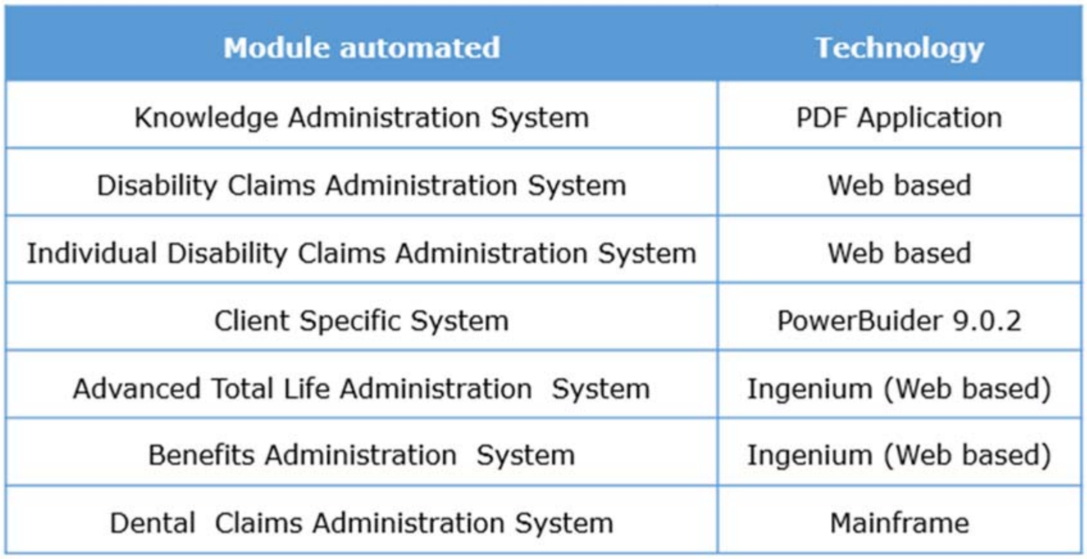
The project involved testing of the following applications.

- Life track
  - Web based applications involved in policy administration, claim generation, processing and record keeping of Group Life (GL), Critical Illness (CI) and Long Term Care(LTC) policies.
  - PowerBuilder based application involved in Statement Of Health(SOH) and record keeping.
- Dental track
  - Mainframe application for dental claims filing, processing and verification.
- Disability track
  - Web based applications used by disability policy holders for claim adjudication.
  - Individual disability income policy administration.
  - PowerBuilder based application for claim generation and processing.
  - Web services based testing for web based absence management application.

About the automation project

After one year of successful client engagement, Infosys identified the need to cut down QC costs, improve testing quality and productivity by implementing test automation. This was also an opportunity to build customer’s trust.

The following applications were automated using QTP 9.5 (earlier version of UFT).



Module automated	Technology
Knowledge Administration System	PDF Application
Disability Claims Administration System	Web based
Individual Disability Claims Administration System	Web based
Client Specific System	PowerBuider 9.0.2
Advanced Total Life Administration System	Ingenium (Web based)
Benefits Administration System	Ingenium (Web based)
Dental Claims Administration System	Mainframe

A hybrid framework was used for the automation process in which UFT was integrated with ALM for end-to-end test process automation. The following components were built into that framework for the most efficient way of test execution.

1. Driver script

Driver script was responsible for calling and executing the needed keywords for a specific test case .

2. Function libraries

A total of three sets of function libraries were used for the following purpose.

- Keyword
- Result generating functions
- Report generating functions

3. Object repository

Shared object repository was used, where the testers associated the .tsr file, for generating the test scripts.

4. Recovery scenario and error handling methods

Issues from the applications and errors inside the script were successfully handled using the recovery scenario mechanism and the error handling methods.

5. Excel automation

Test data were fetched using excel automation methods resulting in the faster executing of test data handling process.

Customer engagement feedback directly from the client on this automation project:

"The work that has been done around automation has really been impressive. There have been 12 applications out of 40 automated covering about 38% of our regression test suite, with $200K in annual savings."