

UFT – Utility Object



“Take the first step in faith. You don't have to see the whole staircase, just take the first step.”
- Martin Luther King, Jr. quotes

In this chapter we will cover the following topics

1. Record First Test in UFT
2. Understanding the Application Object and UFT
3. Object Identification Process by UFT
4. VBScript and UFT
5. Understanding UFT IDE
6. Enhance Test Script to interact at run-time
7. Solution in UFT
8. Shared Vs. Local Object Repository in UFT

In this chapter we will discuss the utility object. Utility object provide immense help to the user during automation test design. This chapter may have long list of utility objects but it will be great help during UFT test designing.

What is Utility Object?

The Utility Objects are UFT reserved objects. Reserved objects are not learned or stored in the object repository. Reserved objects can manage UFT behaviour during a run session. We have already discussed some of them e.g. Wait, Print and DataTable etc. Following is the snapshot of UFT utility objects:

<Revise : It have object and statement >

Crypt Object	QCUtil Object	CallServiceTest Statement
DataTable Object	RandomNumber Object	DescribeResult Statement
Description Object	Recovery Object	ExecuteFile Statement
DotNetFactory Object	Reporter Object	ExitAction Statement
DTParameter Object	RepositoriesCollection Object	ExitActionIteration Statement
DTSheet Object	Repository Object	ExitComponent Statement
Environment Object	Services Object	ExitComponentIteration Statement
Extern Object	Setting Object	ExitTest Statement
LocalParameter Object	SystemMonitor Object	ExitTestIteration Statement
MercuryTimers Object	TextUtil Object	GetLastError Statement
MercuryTimer Object	VisualRelation Object	InvokeApplication Statement
Parameter Object	VisualRelations Object	LoadAndRunAction Statement
PathFinder Object	VisualRelationsCollection Object	LoadFunctionLibrary Statement
Properties Object	XMLUtil Object	ManualStep Statement
Print Statement	RunAction Statement	UnregisterUserFunc Statement
RegisterUserFunc Statement	SetLastError Statement	Wait Statement

Utility of Utility Object

Let's discuss the all utility object one by one. We will focus on "What-Why- How" relation of each object.

1. What - What is the specific object
2. Why – Advantage of that object
3. How – How It works

Crypt Object

What - The object used to encrypt strings so its real value should not reveal in report (or test).

Why – Sensitive data (password, SSN, UDID etc.) can be protected in the script or reports

How – we can encrypt any value by directly taking in variable and encrypt it

```
pwd = "GetValue_From_Somewhere_at_Run_Time_Like_DataBase"  
encrypt_pwd = Crypt.Encrypt(pwd)
```

DataTable Object

What – Run-time datatable (not design time) provides data storage capabilities. As it is similar to MS Excel and supports various method and properties to manage the data. Few popular methods are ImportSheet, GetCurrentRow, Export, SetNextRow.

Why – Testing require user input and synthesize the data during run-time. We can insert as many records in datatable and iterate whole logic with the row of datatable. There is segregation between global and local datatable to provide data modularity for per action and whole test. We can also create external excel (.xls , .xlsx or csv) file and import into datatable.

How – We need to write DataTable.<Method> or DataTable.<PropertyName>

```
'Import excel sheet into Datatable  
DataTable.Import ("C:\QuickTestProTests.xls")
```

```
'Send excel sheet into Action1 Datatable in User column.  
DataTable.Value ("User", "Action1")="KrishanShukla"
```

```
'Take the value from Datatable in script  
myValue = DataTable.Value ("User", "Action1")
```

Description Object

Create method of Description create an empty object. We can create the properties for this object and assign values to those properties. Set use to assign any object to variable.

```
Set oMyObjectDesc = Description.Create
```

Now we can assign the property and value pair to this newly created object.

```
oMyObjectDesc ("property_1").Value = value_1  
oMyObjectDesc ("property_2").Value = value_2  
oMyObjectDesc ("property_n").Value = value_n  
OR  
oMyObjectDesc ("micclass").Value = "WebEdit"  
oMyObjectDesc ("name").Value = "q"
```



A Rhyme

Description object is a collection. Collection of Properties. Properties that have values. Values that can match the values of run time object. Run time object that can be used to perform any action. So effectively you may do not need to rely on Object repository to keep the description of object.

You can create a description and UFT provide childobject method. This method will return all the child of the parent object. In this example we can get the all links on Google page.

```

Set oDesc = Description.Create
oDesc( "micclass" ).value = "Link"
Set oLinkObject = Browser("Google").Page("Google").ChildObjects(oDesc)

```

Descriptive programming is one of the most powerful aspect of UFT and we will discuss it later in descriptive programming chapter.

DotNetFactory Object

Microsoft provide comprehensive classes in .Net framework. These classes can be used to manage or manipulate system information. DotNetFactory can create an instance of a .NET object, access its methods and properties. Static methods and properties of the class can be access by this object directly, for example, System.Environment, System.Windows.Forms.

Example: Run the following code in UFT

```

Set oDotNetInstance = DotNetFactory.CreateInstance("System.Environment")
msgbox oDotNetInstance.CurrentDirectory

```

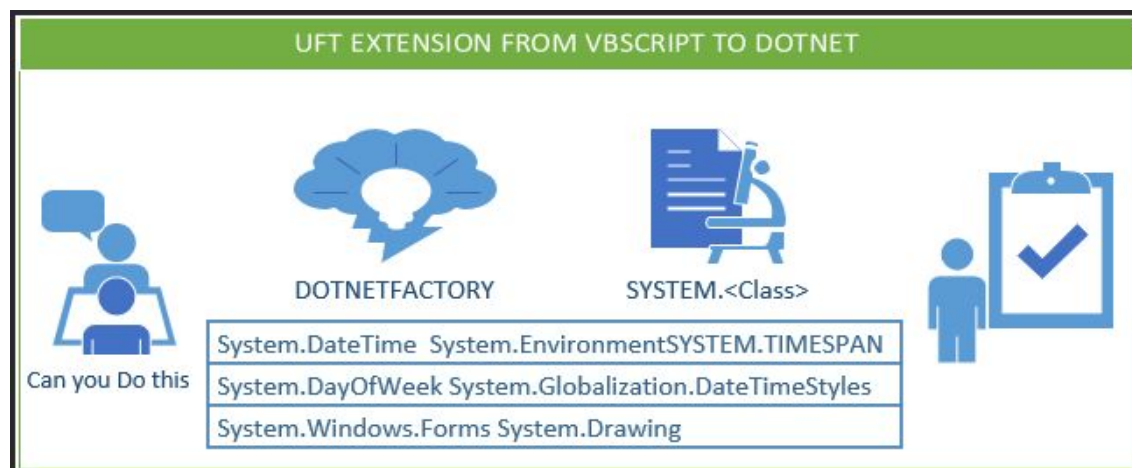
Here CreateInstance is creating a .Net Object (of COM interface) of Environment class [REF] from system namespace [REF]. So we can get crucial information by accessing the .Net framework classes, for example, MachineName, UserName, UserDomainName, ProcessorCount. These information is from one class only. We can work with many classes, for example, System.Math have 50 odd methods and we can call these method by using dotnetfactory. Refer MSDN for full list of methods and classes.

```

Set oDotNetInstance = DotNetFactory.CreateInstance("System.Math")
msgbox oDotNetInstance.Cosh(120)

```

So Logical DotNetfactory in UFT enables to extend our reach from VBScript functions to .NetFramework classes. We can create our own assembly (classes) and register it in System. Also this way we can work with 3rd party components as well.



The following code will create a blank Windows form by using Form class

```
Set oDotNetInstance2 = DotNetFactory.CreateInstance("System.Windows.Forms.Form", "System.Windows.Forms")
oDotNetInstance2.Show
```

Let's discuss the specification of CreateInstance method. It returns a COM interface for a .NET object.

```
Set dotnetInstance = DotNetFactory.CreateInstance (TypeName, [Assembly], [args])
```

<i>TypeName</i>	String	The full name of the object type, for example, System.Windows.Forms.Form.
<i>Assembly</i>	Any	Optional. If the assembly is preloaded in the registry, you do not need to enter it. Otherwise, you should enter the full path name, for example, System.Windows.Forms. If you want to create an instance of a type in the GAC (Global Assembly Cache), you can enter the short name, for example, Forms.
		Note: If you do not pass this argument, UFT assumes that the assembly is resident in memory. If UFT does not find the assembly, an error message is displayed during the run session.
		Some assemblies are designed to depend on specific configuration information. To load such an assembly using DOTNetFactory.CreateInstance , you must add the configuration required by the assembly to the <UFT installation>/bin/uft.exe.config file before opening
<i>args</i>	Any	Optional. The required arguments, if any, for the specified <i>TypeName</i> and/or <i>Assembly</i> .

This topic can be discussed in great detail but it is out of scope of this book. You need to research and explore the MSDN library and implement all the classes and functions as per your automation requirements.

[REF] <http://msdn.microsoft.com/en-us/library/System.aspx>

[REF] <http://msdn.microsoft.com/en-us/library/system.environment.aspx>

MercuryTimers Object

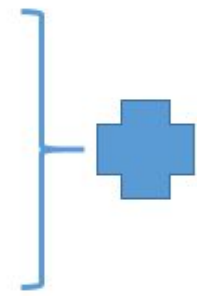
What – An internal timer object that measures the passage of time in milliseconds.

Why – We can get the time duration between steps. We can even pause and continue transaction at later steps during run time. Suppose we need to find the time duration of user search and pay option. These two steps can come in multiple test flows in various test.

So, we can set a same timer for only these steps in different work-flow in various test scripts.

How – MercuryTimer supports Continue Method, Reset Method, Start Method and Stop Method. ElapsedTime Property return the total accumulated time (in milliseconds).

```
1 MercuryTimers("SearchKrishan").Start
2 wait 2
3 MercuryTimers("SearchKrishan").Stop
4 'do something
5 wait 1
6 'will continue to measure time from
7 'the time it stopped
8 MercuryTimers("SearchKrishan").Continue
9 wait 3
10 MercuryTimers("SearchKrishan").stop
11 MsgBox MercuryTimers("SearchKrishan").ElapsedTime 'In millisecond
```



Parameter Object

What – Action can have input and output parameter. We can access action parameter by Parameter object.

Why – Parameterization is the backbone of automation. There are many ways to do so e.g. datatable, environment variable etc. Actions parameter are also very handy to provide value at run-time in the script.

How – Create a test and provide “Name” action property with value “Krishan” in Input parameter section.

```
Browser("Browser").Page("Google").WebEdit("q").Set "krishanshukla"
Browser("Browser").Page("Google").WebEdit("q").Set Parameter("Name")
```

PathFinder Object

What – Enables to find file paths by the help of PathFinder.Locate (<filename>). Locate property find the full path of the file from the folder specified in **Tools | Options | GUI Testing** tab | **Folders** pane.

Why – File use for various purpose e.g. library can contains reusable code in various file. We sometime need to find the file path to check whether it exist or there may be a file with same name at more than two location. PathFinder.Locate provides opportunity to verify the path of file before contemplating any action on that.

How – Pathfinder will search the file in the folder path provided in options | folder.

STEP1: Create test and write code to access pathfinder.



*Navigate to UFT | New | Test | PathFinder | **Msgbox** PathFinder.Locate ("MyFile.txt") | Save | Run*

STEP2: Create file and tell UFT in option | Folder



Navigate to "C:\Test" | Create "MyFile.txt" | Navigate to UFT | Tools | Options | GUI Test | Folder | Add "C:\Test" | OK | Run

STEP3: Create other location and find the value



Navigate to "C:\Test\Aric" | Create "MyFile.txt" | Navigate to UFT | Tools | Options | GUI Test | Folder | Remove "C:\Test" | Add "C:\Test\Aric" | OK | Run

Properties Object

QCUtil Object

What – The object used to access the ALM (formerly known as Quality Center – QC) Open Test Architecture (OTA) interface. This object paved the way to access ALM. Previously there was TDUtil object and QCUtil replaced that. IsConnected Property and QCConnection Property are two important properties in QCUtil object.

Why – We can manage test, test set, defect, requirement and release programmatically in ALM. It means that manipulating test artefacts in ALM is possible by writing code instead of manually performed actions on ALM.

How – QCUtil can enable automater to manipulate, retrieve or manage ALM. The following code creating a bug without going in to ALM.

```
Dim QCConnection
```

```
' ALM must connected to ALM for this code to work
```

```
Set QCConnection = QCUtil.QCConnection
```

```
'Create empty bugfactory to produce bugs
```

```
Set BugFactory = QCConnection.BugFactory
```

```
'Add a new, empty defect
```

```
Set Bug = BugFactory.AddItem (Nothing)
```

```
'Modify the bug values
```

```
Bug.Status = "New"
```

```
Bug.Summary = "New defect"
```

```
Bug.DetectedBy = "Krishan" ' user that must exist in the ALM
```

```
'Post the bug in ALM
```

```
Bug.Post
```

RandomNumber Object

What – Generates random number from the given range

Why – It is very helpful to generate new value every time for various purpose e.g. unique userid, different numbers for price, age etc.

How – The following example generates a random number between 0 and 1000.

```
intRandom = RandomNumber (0,1000)  
or intRandom = RandomNumber.Value(0,1000)
```

Recovery Object

What – It enables to control the recovery scenario mechanism programmatically during the run session. Recovery scenario are the fire-fighter when UFT test execution mired with some unexpected error. Methods supported by Recovery object are Activate Method, GetScenarioName Method, GetScenarioPosition Method, GetScenarioStatus Method, SetScenarioStatus Method. Properties supported by Recovery object are Count Property, Enabled Property (default property).

Why – Recovery scenario can be explored from Resource | Recovery Scenario Manager. It can covers unexpected following errors during run-time 1. Intermittent Pop-up window 2. Object state for example button remain disable 3. Test Run Result e.g. “Object not found” or “duplicate object found” or 4. Application crash during run-time.

How – The following example shows the recovery scenario status by checking enabled property. It is default property and it returns True or False.

```
CurrentStatusOfRecovery=Recovery.Enabled 'Returns true or false  
If Not (CurrentStatusOfRecovery) Then 'If recovery is disable then set it to True  
Recovery.Enabled=True  
End If  
Recovery.Activate  
Recovery = MyCurrentStatus
```

Reporter Object

What – Reporter object can be used to send information to UFT report during run-time.

Why – Reporter object can log the customize messages in the UFT report for example if user exist, order confirmation number, test step of interest pass during run-time. These are following methods and properties in supported by Reporter object.

Associated Methods	Associated Properties
ReportEvent Method	Filter Property
ReportNote Method	ReportPath Property
	RunStatus Property

ReportEvent log an event into the run results while ReportNote add note in the Executive Summary Notes section.

How – Reporter.<Method/Property> use to access reporter object. There are for status used with reporter object 1. micPass 2. micFail 3. micDone and 4. micWarning.

```
Reporter.ReportNote "Logo @Home Page at QuickTestPro.co.uk"
```

```
If Browser("X").Page("Y").Image("MyLogo").Exist(0) Then  
Reporter.ReportEvent micPass, "Display Logo", "This is my logo", "MyLogo.bmp"  
Else  
Reporter.ReportEvent micFail, "Display Logo", "This is my logo", "MyLogo.bmp"  
End If
```

```
If Reporter.RunStatus = micFail Then ExitAction
```

RepositoriesCollection Object

What – RepositoriesCollection object is a collection object that enables us to programmatically manage the shared object repositories associated with the specific action.

Why – Repository are the home where design-time object resides. In practical world, we create few shared object repository based on the application components e.g. repository for payment component or reservation component etc. We need to associate repository during run time for the specific action depending on the need and covering area. These are following methods and properties in supported by Reporter object.

Associated Methods	Associated Properties
Add Method	Count Property
Find Method	Item Property
MoveToPos Method	
Remove Method	
RemoveAll Method	

Add method adds the repository at run-time. Before adding or removing any repository, we can find the repository for the specified path. If we add more than one shared object repository then they can be priorities by MoveToPos method. Count property returns the number of associated object repositories.

How – Repositories can be add, remove, count and check for existence.

```
RepoPath = "\\Aric-PC\Krishan\SharedObjectRepository.tsr"  
RepositoriesCollection.RemoveAll()  
RepositoriesCollection.Add(RepoPath)
```

```
Pos = RepositoriesCollection.Find(RepPath)
```

```
RepositoriesCollection.Remove(Pos)
RepositoriesCollection.Add(RepPath)
MsgBox RepositoriesCollection.Count
Browser ("X").Page ("Y").Image ("MyLogo").Click
```

Repository Object

What – We can set or retrieve the value of repository parameters using the Repository object. This object is different from RepositoriesCollection object that handle the repository association, count, disassociation etc. while Repository object speak about the parameter inside a repository.

Why – We can map the object property with parameter in Object Repository. These parameter can map with Datatable, Environment variable or random number. Recovery object can directly set or get the value from the parameter.

How – Repository object can manage object parameter.



Navigate to "UFT" | New | Test | Record | IE | Google.co.uk | Search "Krishan" | Close IE | Stop | Create a column in Datatable "editbox" | Resources | Object Repository | Go to object "q" editbox | Click on "name:q" row | Click configuration | Parameter "Datatable" | Select "editbox" | OK | Close OR | Modify script with following line

```
Repository.Value ("editbox") = DataTable("editbox", dtGlobalSheet)
MsgBox Repository.Value("editbox")
Repository.Value("editbox")= "NewValueforParmeter"
```

Services Object

What – Service object mainly used for monitoring and control transaction related activity. Performance tools e.g. Loadrunner measure transaction time, think time, waste time etc to monitor application performance under load. Service object provide similar functionality to measure the transaction.

Why – The transaction is the point where any business process submit. To measure the time between steps, we can use Service.start and Service.end transactions. There are other methods for Service object but mostly related in conjunction of loadrunner or Business Process Monitoring (BPM).

How – Following example count the total time taken between "Krish" transaction and log in report.

```
Services.StartTransaction "Krish"
'Do some steps
Wait 4
Services.EndTransaction "Krish" 'End transaction must same as start transaction
```

Method	Detail	Method	Detail
AddWastedTime	Increments the wasted time for all open transactions.	SetTransaction	Reports a completed transaction using manually entered transaction data.
EndTransaction	Marks the end of a transaction and records the amount of time it took to perform the transaction.	StartTransaction	Marks the beginning of a transaction for performance analysis.
GetEnvironmentAttribute	Retrieves the test's environment properties.	ThinkTime	Loadrunner Specific - user wait time
LogMessage	Loadrunner Specific - send log message	Rendezvous	Loadrunner Specific - meeting point of all vusers

Setting Object

What – Setting object enables test setting modification during run-time. It is like dictionary with key-value pair. Some valid keys are “IterNum”, “**TestsDirectory**” etc.

Why – Testing require data input. Setting provide key-value pair where automater can decide the logic based on the key-value during run-time.

It is similar to VBScript dictionary object and supports Add, Exists and Remove methods.

A special note for webpackage keys. Sometime UFT find it difficult to perform operation on tricky objects. We can set the operation to mimic real mouse to handle the mouse event instead rely on browser events. We should reset it to browser event after passing the tricky section of application.

Key	Value	Description
ReplayType	1 or 2	Indicates how mouse operations should be run. The value can be one of the following: 1 - Runs mouse operations using browser events. 2 - Runs mouse operations using the mouse. Example: Setting.WebPackage("ReplayType") = 2

How – This code find the current test directory. It search for the “username” key and if it doesn’t exist then add value “Krishan” and one more key-value (website-website name) pair.

```

MsgBox Setting("TestsDirectory")
'It is not Exist
if Not Setting.Exists("UserName") Then
    Setting("UserName") = "Krishan"
    Setting("WebSite") = "QuickTestPro.co.uk"
end if
MsgBox Setting.Item("UserName")
MsgBox Setting.Item("WebSite")

```

SystemMonitor Object

What – SystemMonitor object enables us to retrieve information about a system counter (related to disk, memory, CPU etc.) during a test run.

Why – Testing is the process to monitor all the checks on application. These checks includes hardware and software performance as well. Doctor test temperature, blood-pressure in same way UFT can test various application and system counters. These result will log in report. SystemMonitor object supports two methods:

1. SystemMonitor.**IsCounterExist**("ApplicationName","CounterName")
2. SystemMonitor.**GetValue**("ApplicationName","CounterName")

How – The following code will capture memory and processor counter.

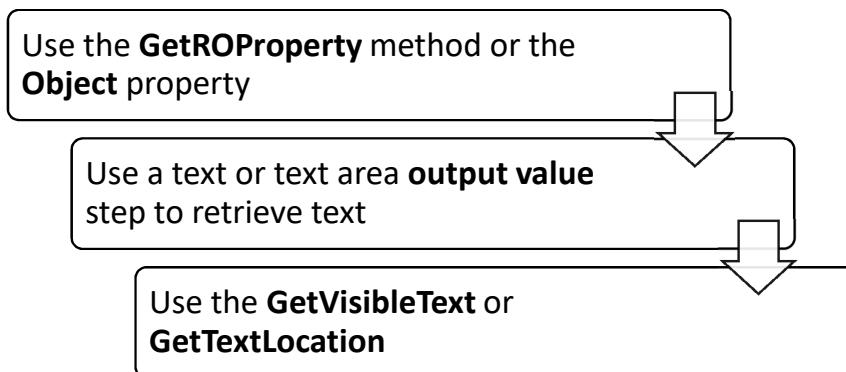
```
'Check the memory usage
print SystemMonitor.GetValue("QTPro","Memory Usage (in MB)")
' Processor usage of an application
print SystemMonitor.GetValue("tomcat","% Processor Time")
```

TextUtil Object

What – TextUtil Object used to recognize text within a specified window handle.

Why – It is very helpful utility to find the textual information from the window. Sometime we get numbers, symbols in window and need to capture them for verification. TextUtil just there to help you in those kind of situation but it is not the most recommended method.

When possible, it is recommended to use alternative methods of capturing text in following order.



TextUtil supports GetText and GetTextLocation methods.

How – We need to pass window

```
'Get Window Handler
windowText = TextUtil.GetText(hWindow)
```

MsgBox windowText

The following example retrieves the text from the screen coordinate.

'Get text from the screen coordinate

MsgBox TextUtil.GetText(0, 20, 20, 200, 200)

VisualRelation Object

What – VisualRelation Object defines the details of a single visual relationship between a related object and the test object. Confuse? Have you ever bought any house or searched any address without navigators? We try to find out house with adjacent houses or from the benchmark place(axis). In same manner, UFT can register neighbor or other object in relation to test object, so identification become more robust.

Why – Suppose you have encounter objects which are not unique in nature, object appearance depend on other object or the system is changing with different settings. In these scenario, we can specify other objects in relation to our test object. The relation can be establish with relative object in following manner

1. Horizontal - is object at right or Left
2. Vertical – is object is below our object
3. Closeness – is object close to x or y-axis

VisualRelation Object supports following method and properties:

Methods	Description
GetArgument	Find left or right, Above or below
SetArgument	Set Object to left or right, Above or below
Properties	Description
RelatedObjectPath	The full object hierarchy path of the related object
RelativePosition	The position of the related object relative to the test object. E.g. Left, Right etc.

How – The following example show how to define the relation



Open UFT | New | GUI Test with Name "ExploreUFT" | Create | Press {F6} or Record | Click on Internet explorer | Google.co.uk | Type "Krishna Wikipedia" in Edit Box | Enter | click on wikipedia link | Close Browser | Stop recording by clicking  stop | Save | Open IE | Go to Google | navigate to Resource in UFT | Object Repository | Go to "q" object | click on row of visual relation identifier settings | Click on  | Click + | click  | click "I'm Feeling Lucky" | OK | Select Left | Select "Closest on y-axis" | OK | Close | Run test

XMLUtil Object

What – XMLUtil Object used to access and return XML objects.

Why – XML files are very good way to create independent data across various platform. UFT can understand .xml file by XMLUtil.

How – Following code shows the CreateXML method create xmlData object from understanding of raw xml file.

- A. The following example creates an XMLData object and uses it to load an XML file.
Set XMLObj = XMLUtil.CreateXML()
XMLObj.LoadFile("C:\QuickTestPro_CO_UK\Training_Catlogue.xml")
- B. The following example creates an blank XML object and uses it to load an XML file from existing file.
Set XMLObj = XMLUtil.CreateXML()
XMLObj.LoadFile("C:\QuickTestPro_CO_UK\Training_Catlogue.xml ")

Utility Statements

DescribeResult Statement

What – DescribeResult Statement returns a text description of the specified error code.

Why – Error and automation goes hand in hand. If we know the error number (integer) and pass in to DescribeResult then it will return the more comprehensible error description.

How – Following code shows the the error description of error 1 to 300.

<pre>On error resume next Dim a() a=array(1,2,3) print a(5) 'Array out of Boundary MsgBox DescribeResult(err.Number)</pre>	<pre>For i = 1 To 300 Step 1 Print DescribeResult(i) Next</pre>
---	---

ExecuteFile Statement

What – ExecuteFile statement executes VBScript code written in .vbs files. After the file runs, the definitions in the file (functions, subroutines, classes, and so forth) are available in the global scope of the action.

Why – we need to write functions and modular code in functional library e.g. database connections, generic functions etc. Execute file statement enable UFT to peek into vbs file. File will only associated in Action. We cannot debug a file that is called using an ExecuteFile statement. Also debug marker will not display correctly during debugging. You should use LoadFunctionLibrary Statement.

How – The following code will associate KrishanLibrary.vbs file in UFT action.

```
ExecuteFile "C:\Test\KrishanLibrary.vbs" 'or can use relative path
```

ExitAction Statement

What – ExitAction Statement exits the current action. The pass or fail status of the action remains as it was in the step prior to the ExitAction statement.

Why – ExitAction help a lot in many situations. Suppose we want to terminate our action premature due to error or finding any business condition (e.g. finding user already exist in NewUserCreate action). The status of Exit statement will reflect in UFT report.

How – The following code will search for “Home” link otherwise come out from Action.

```
Browser("Browser").Navigate "http://quicktestpro.co.uk/"  
If Browser("Browser").Page(Quicktestpro).link("Home").Exist(10) Then  
Browser("Browser").Page(Quicktestpro).link("Home").click  
Else  
ExitAction  
End If
```

ExitActionIteration Statement

What – Action can have one or more iterations. ExitActionIteration Statement exits the current iteration of the action but not the complete action. The pass or fail status of the action iteration remains as it was in the step prior to the ExitActionIteration statement.

Why – ExitActionIteration, like ExitAction, helps a lot in many situations where we want to exit the current iteration but continue from same action with different iteration.

How – The following code will match UFT site to skip the action iteration.

```
UFTSite = DataTable("SiteName", dtLocalSheet)  
If UFTSite = "QuickTestPro.co.uk" Then ExitActionIteration("Skipping UFT Site")
```

ExitTest Statement

What – ExitTest Statement exits the entire UFT test, regardless of the run-time iteration settings. The pass or fail status of the test remains as it was in the step prior to the ExitTest statement. It is superset of ExitAction statement.

Why – ExitTest helps a lot in many situations where we want to exit the current test e.g. system unresponsive, some known error occurs in test or checkpoint fails etc. that require to abandoned the test, ExistTest will throw out that test from UFT suite at run-time and keep continue to run other tests.

How – The following code will match UFT site to skip the Test.

```
result = Browser("X").Page("Y").WebEdit("Z").Check ( CheckPoint("KrishanShukla") )  
If result = False Then  
ExitTest  
End If
```

ExitTestIteration Statement

What – ExitTestIteration statements exits the current iteration of the UFT test and proceeds to the next iteration, or exits the test run if there are no additional run-time parameter iterations. File | Test Setting | Run specify the iterations.

Why – ExitTestIteration, like ExitTest, helps a lot in many situations where we want to exit the current test iteration e.g. system unresponsive, some known error occurs in test or checkpoint fails etc. that require to abandoned the specific test iteration.

How – The following code will match UFT site to skip the test iteration.

```
result = Browser("X").Page("Y").WebEdit("Z").Check ( CheckPoint("KrishanShukla") )
If result = False Then
    ExitTestIteration
End If
```

InvokeApplication Statement

What – InvokeApplication statement Invokes an executable application.

Why – Once upon a time there was a famous InvokeApplication statement used to invoke any application (.exe) but that place claimed by SystemUtil. Now InvokeApplication only supported by HP for backward compatibility

How – The following code will invoke Internet Explorer.

```
InvokeApplication "C:\Program Files\Microsoft Internet\EXPLORE.EXE"
```

LoadAndRunAction Statement

What – LoadAndRunAction Statement loads the specified action.

Why – Suppose we are not sure which action is going to be called during run-time, for example, we may use conditional statements that call external actions, and we do not want to load actions each time you open the test, since these actions may not be necessary during the run session. In this situation LoadAndRunAction statements prove very beneficial.

How – The following code will load and run action as decided during run-time.

```
user= DataTable.Value("user_Type","dtGlobalSheet")
If user="Krishan" Then
    LoadAndRunAction "c:\Test\MentorTest"
ElseIf user="Aric" Then
    LoadAndRunAction "c:\Test\SolverTest"
ElseIf user="Aadhya" Then
    LoadAndRunAction "c:\Test\UFTSuperStarTest"
Else
    LoadAndRunAction "c:\Test\GodOnlyKnowsTest"
End If
```

LoadFunctionLibrary Statement

What – LoadFunctionLibrary Statement Loads the specified function library. After the function library is loaded, the definitions in the library (functions, subroutines, classes, and so forth) are available. Library can be .vbs or .qfl file.

Why – It is sibling of ExecuteFile. LoadFunctionLibrary statement will associate library file. It enables to debug the library file (a big disadvantage in ExecuteFile).

How – Following is specification of LoadFunctionLibrary.

```
LoadFunctionLibrary("C:\Test\AricTest")
```

Delta	LoadFunctionLibrary	ExecuteFile
Concept	LoadFunctionLibrary Lets the debugging feature such as breakpoint in vbs file.	Using Execute file we cannot debug within the vbs files by using "breakpoints". If we try to use breakpoint in the vbsfiles by opening them in QTP, the QTP debugger will not stop at the break point specified.
Call	MultipleFiles	Single File
Scope	By associating functional library, all actions in the test can access those functions	By calling executefile, all functions are loaded in the calling action only
Update	Possible	Not Possible
Version	QTP11 Onward	VBS
Variable Name	If a dynamically loaded function library defines and initializes a global variable or a class, the value remains in effect till end of the run session and hence local variable and variable in Function Library matches and causes problem, means functions, variable sre available locally to edit	ExecuteFile does not load variable in script's local namespaces so there is no problem if same variable name is being found. ExecuteFile will only affect the namespace of the calling function.

Print Statement

What – Print statement displays information in the Output pane during the run session

Why – MsgBox is VBS statement and pause the execution flow until user take some action. Print statement log the statement in output pane. These statements are very helpful for debugging purpose. Print is UFT specific and will not work in functional library.

How – Following code will print the statement.

```
Print "QuickTestPro.co" & CountryDomain
```

RegisterUserFunc Statement

What – RegisterUserFunc statement enables to add new methods to test object classes or change the behavior of an existing test object method during a run session.

Why – We do not have overriding feature in vbscript. One function can behave in one manner. We cannot override the same method name with different behaviour. RegisterUserFunction can change the behaviour of the method until we unregister that function. So suppose if Set is method applicable on WebEdit box and we override set to "select index" behaviour then Set will behave to select index instead setting text in editbox until we unregister set method.

How – Following code change set behaviour.

```
'Override set with NewSet
RegisterUserFunc "WebEdit", "Set", "NewSet" 'Overridden Set by MySet

Browser("QuickTestProCOUK"). WebEdit ("Register").Set "Tutorial"
UnRegisterUserFunc "WebEdit", "Set"

Function NewSet (obj, x)
    Dim y
    y = obj.GetROProperty("innertext")
    Reporter.ReportEvent micDone, "set change to newest, object innertext : ", y
    MySet=obj.Set(x)
End Function
```

RunAction Statement

What – RunAction Statement Runs the specified action in the test.

Why – So why we need LoadAndRunAction? The RunAction statement can run only actions that are already associated with test as part of the test flow. So, test first need to associate and then this statement can be utilize. We should use LoadAndRunAction to call action from functional library. Also this statement is not supported by command tab of the Debug pane.

How – Following code shows the runaction statement. The Iteration can be oneliteration or allIterations.

```
RunAction ActionName, [Iteration ,Parameters]
RunAction "PayBalance", oneliteration, "Myinputparam", MyOutputparam
```

Wait Statement

What – Wait statement pause for specified duration during a run session.

Why – The most over-subscribed statement in UFT world. Wait statement is “easy to use statement” to synchronize application with UFT.

How – This code depicts global wait.

```
Dim x
Duration = 12
Wait (2.1)
MsgBox Wait (Duration) 'All wait statement can manage by one
variable
```

Summary