Kanakarajan kandasamy

About the author

"The best way to start achieving a thing is to dream for it"
-Dr.A.P.J.Abdual kalam

Preface

”This moment is not permanent in life”

QuickTest Professional is a Test Automation tool and uses VBScript as its scripting language. QTP is a record and playback tool which can record events we perform on an application and replay them back.QTP is an object based tool which recognizes each element of the application as an object and provides various methods to work on them. All this makes look QTP an easy to use test tool. The myth about Record & Playback is that it makes people think that they do not need development skills for QTP, but to create effective Automation Frameworks one needs to view QTP as a development tool and not as a testing tool.

This book presents QTP as a development tool rather than a mere test tool. One of my problems while evaluating the tool led to me to join automation excerpts community in orkut, without knowing that I will specialize in the use of this tool in future. For sharing my articles on QTP with the larger group. Dealing with day to day automation problems faced by people on the QTP forums, I tried solving those problems for them and learnt a few new things on my own. Observing the patterns of queries being asked on the QTP forums, I thought what the QTP community was missing is a book which can guide the amateur automation engineers in becoming a professional in the use of this tool. I took up this responsibility and started writing this book in May 2009. I spent a less then 6 month.

Being a first time author, I had a very hard time getting this project completed. It was an additional responsibility, over and above my office work, QTP forum support, writing articles on Automation Anywhere, creating tools for the community. It required a lot of motivation to keep myself on the project.

Who This Book Is For

This book is for Test engineers, Test Analysts, Test Consultants and anyone who is interested in learning advanced techniques of problem solving in QTP. This book is also for beginners who have just started with QTP and want to be experts in its use. The book assumes that one has the basic knowledge of QTP and VBScript, if not than it is advised that one should go through the basic help first. As the main focus of this book is to view the tool from a developer's eye, the book does not teach how to record and replay script in QTP. Also the book does not discuss about the Keyword view of QTP, which is for non-technical people who don't want to code in QTP.

Contents

Recording modes in QTP
Data Tables
Actions
Environment Variables
Utility Objects
Checkpoints
Output Values
Descriptive Programming
Debugging in QTP
Recovery Scenario
Regular Expression
Synchronization
Test Results
Automation Object Model
Working with MS Excel
Working with XML
Designing Framework
QTP Methods and Properties
Working with Test Objects
Working with Filesystem Objects
Dictionary Object
Virtual Object
Object Repository
Working with Databases
What's New in QTP 9.2
VB Script Basics
Advanced and Most useful things
Sample script for web based application
About QTP Certification

1

Introduction

What is test Automation?

It is a process in which all the drawbacks of manual testing are addressed (over come) properly and provides speed and accuracy to the existing testing phase.

Note:

Automation Testing is not a replacement for manual testing it is just a continuation for a manual testing in order to provide speed and accuracy.

Drawbacks of Automation Testing

1. Too cost.
2. Cannot automat all the areas.
3. Lake of experience.
4.

AUTOMATION TOOL

Automated Tool is an Assistance of test engineers, which works based on the instructions and information.

General foam work to learn any automated tool.

A test engineer should learn the following to work with any automated tool.

1. How to give the instruction.
2. How to give the information.
3. How to use its recording facility.
4. How to use its play back facility.
5. How to analysis the Result.

When should test Automation be used?

U think it is tedious to apply the test manually.
U think it will be reasonably feasible to automate the test case.
U think it is better we execute the test case manually rather than breaking our head trying to automate it.
Lack of WILL POWER has caused more failure than Lack of INTELLIGENCE or ABILITY.
Basically we can start with Manual Testing; the situations to move to automate will be
1) when we need to test the same functionality with more set of input data.
2) Difficult to do manual testing (time wise, resources wise)
3) Where users work mostly on that particular page or section of that application
4) Pages/ sections very important to the users
5) where More no of mathematical calculations will be done

What is HP Quick Test Professional (QTP?)

- QTP – Quick Test Professional
- QTP is a Mercury Interactive Automated Testing Tool which provides the industry's best solution for Functional test and Regression test automation.
- Quick Test Professional enables you to test standard Windows applications, Web applications, ActiveX controls, and Visual Basic applications.
- You can also acquire additional Quick Test add-ins for a number of special environments (such as Java, Oracle, SAP Solutions, .NET Windows and Web Forms, Siebel, PeopleSoft, Web services, and Terminal Emulator applications).
-

Benefits of using QTP

Fast - Quick Test runs tests significantly faster than human users.

- **Reliable** - Tests perform precisely the same operations each time they are run, thereby eliminating human error.
- **Repeatable** - You can test how the Web site or application reacts after repeated execution of the same operations.
- **Programmable** - You can program sophisticated tests that bring out hidden information.
- **Comprehensive** - You can build a suite of tests that covers every feature in your Web site or application.
- **Reusable** - You can reuse tests on different versions of a Web site or application, even if the user interfaces changes.
- Quick Test Professional satisfies the needs of both Technical and Non-Technical (Business Analysts, Subject Matter Experts) users. It enables you to deploy high quality applications faster, cheaper, and with less risk.
- Empower the entire team to create sophisticated test suites with minimal training.
- Ensure correct functionality across all environments, data sets, and business processes.
- Fully document and replicate defects for developers, enabling them to fix defects faster and meet production deadlines.
- Easily regression-test ever-changing applications and environments.

Become a key player in enabling the organization to deliver quality products and services, and improve revenues and profitability.

QTP Testing Process

Create your test plan

Prior to automating there should be a detailed description of the test including the exact steps to follow data to be input and all items to be verified by the test. The verification information should include both data validations and existence or state verifications of objects in the application.

Recording a session on your application

As you navigate through your application QuickTest graphically displays each *step* you perform in the form of a collapsible icon-based *test tree*. A step is any user action that causes or makes a change in your site such as clicking a link or image or entering data in a form.

Enhancing your test

- Inserting *checkpoints* into your test lets you search for a specific value of a page object or text string which helps you identify whether or not your application is functioning correctly.

NOTE: Checkpoints can be added to a test as you record it or after the fact via the Active Screen. It is **much** easier and faster to add the checkpoints during the recording process.

- Broadening the scope of your test by replacing fixed values with *parameters* lets you check how your application performs the same operations with multiple sets of data.
- Adding logic and conditional statements to your test enables you to add sophisticated checks to your test.

Debugging your test

If changes were made to the script you need to debug it to check that it operates smoothly and without interruption.

Running your test on a new version of your application

You run a test to check the behavior of your application. While running QuickTest connects to your application and performs each step in your test.

Analyzing the test results

You examine the test results to pinpoint defects in your application.

Reporting defects

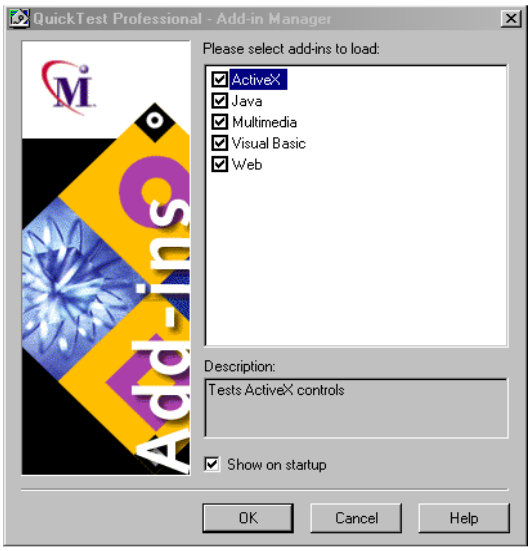
As you encounter failures in the application when analyzing test results you will create defect reports in Defect Reporting Tool.

Setting up the Environment

- Setting up the QTP environment for Record and Playback
  - Select the correct Add-ins to load from the Add-in Manager
  - Select the appropriate Active Screen capture level (Tools → Options)
  - Configure the Record and Run settings (Test → Record and Run Settings)
  - Configure the Editor options (Tools → Editor Options)
- Record a basic test from a manual test case.
- Play Back a test.
- Debug and Enhance the Test.
- Set the Initial and End conditions for a test.
- Save the test.

Ad-in Manager: - It is a feature provided by qtp used for making the qtp compatible with a specified environment by default the qtp provides 3 add-ins

1. Visual basic
2. Java
3. Multimedia
4. Visual Basis
5. Web



2

Recording modes in QTP

Recording and Running

Recording and Run Settings:

Recording and Run Setting is a feature provided by Q.T.P, which is used for making the Q.T.P understand on which applications will need to concentrate while Recording and Running. This setting has to be done at least once for every new test.

Operational Overview of Recording.

During recording Q.T.P will be during us following

1. It will generate the corresponding test script statement for every user action.
2. It will also store the required related information in the object repository.

Operational Overview of Running.

Q.T.P will be doing the following will be running

1. It will be read the script statement.
2. It will understand what action to be performed on which object.
3. When it is realizes it needs to identify that object for that it requires some information for that information it will go to the object repository and search.
4. Once the information is identified using that information it will try to identify the object.
5. Once the object is identify it will perform the action

Recording Modes

There are 3 types of recording modes.

1. Contact sensitive recording mode / normal recording mode.
2. Analog recording mode.
3. Low-level recording mode.

***1*) Normal recording**

It is used for recording the operations perform at different contacts on the standard GUI objects.

2) Analog Recording (Ctrl+Shift+F4)

It is used for recording the continuous operations. This mode is useful for the operation you can record at object level such as drawing a picture, recording signature. The steps recorded using analog mode is saved in separated data file. Analog recording divided into two types.

1. **Relative to screen**
   Quicktest inserts the Run Analog step under desktop parent item. For example
   Desktop, runAnalog "Track1"

2. **Relative to window**
   Quicktest inserts the Run Analog steps under a window parent item. For example
   Window ("Microsoft internet").Run Analog "track1"

The track file called by the run analog method contains all your data and is stored with action.
Note: track0 –Fast
Track1-Normal

3) Low-level Recording

It is special recording mode provided by Q.T.P, which is used for recording the minimum operations on the Non-Supported environments also. Low-level-recording for when you need to record the exact location of the operation on your application screen-L-R supports the following

Winobject -Click, doubleclick, drag, drop, type
Window -Click, double-clicks, drag, drop, type, activate, minimize, and restore, maxmize.

Normal Recording code for Agent name in flight reservation application.
Browser ("mercury").page ("mercury").webedit (Agent name).set "mercury"

Low-Level-Recording code
Wnd ("mercury").page ("mercury").click 564,263
Wnd ("mercury").page ("mercury").type "mercury"
Wnd ("mercury").page ("mercury").type MicTab

Anatomy or QTP

Apart from these 3 add-ins qtp is always compatible with standard windows environment QTP serene is divided in to 5 parts

1. Test Pane
2. Active Screen
3. Data Table
4. Debug Viewer Pane
5. Tool Options

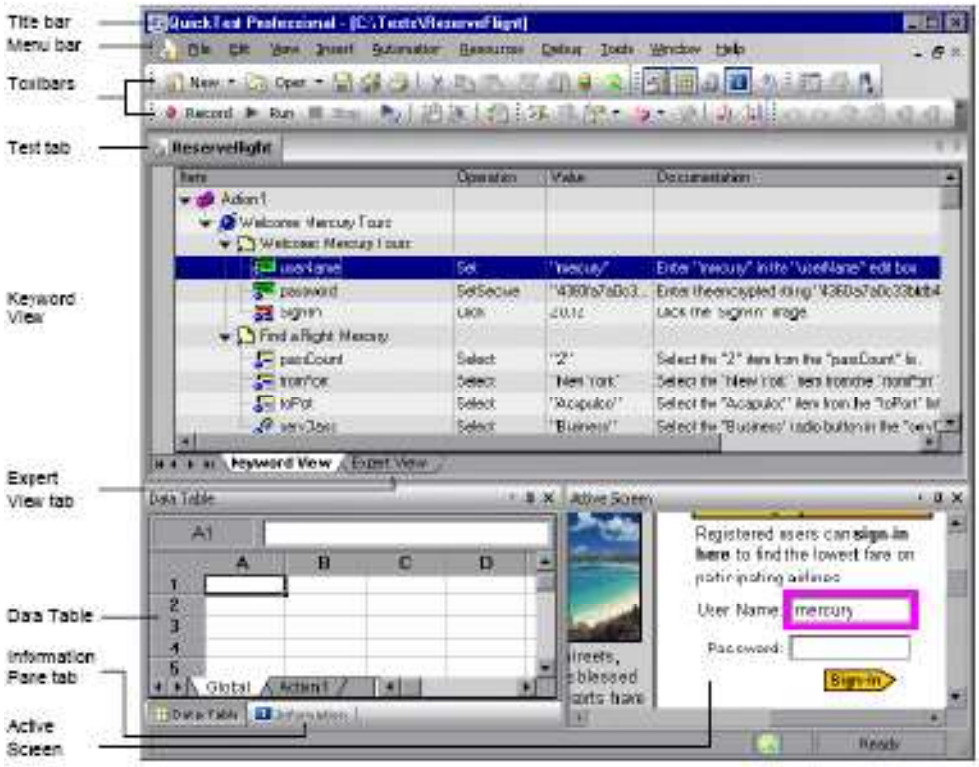
]1). Test Pane

Test pane is an area provided by Q.T.P, which is used for developing, viewing and modifying the test script.

It represents the Test script in 2 views.

1. Expert view
2. Keyboard view

QTP Main Window



Expert view

Expert view represents the script in VB script format.

Keyboard View

It represents the scripts using a graphical user interface, which is further divided, into 4 parts.

1. Item
2. Operation
3. Value
4. Documentation

2) Active Screen

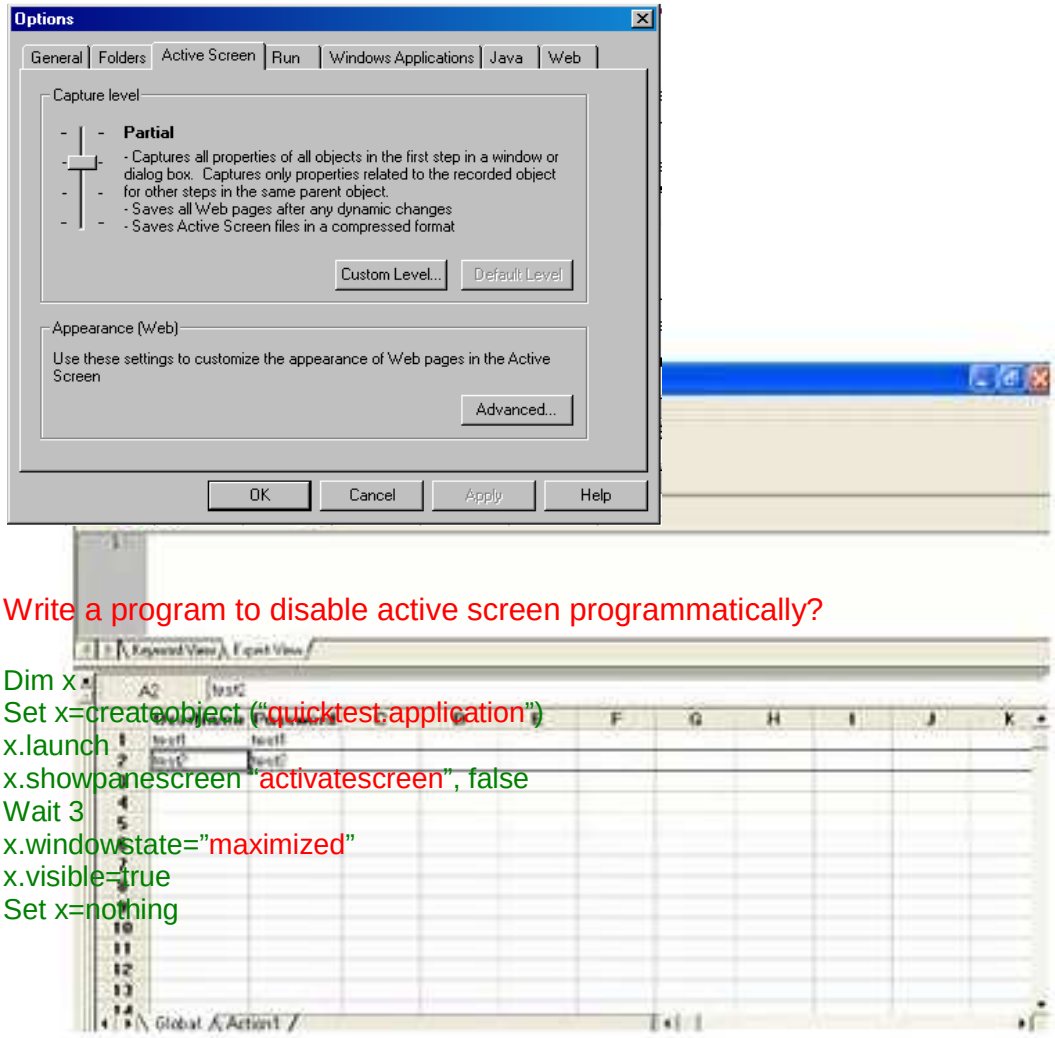
Active Screen is a feature provided by Q.T.P which holds the snap shots related to each and every script statement and used for understanding the script easily as well as enhancing the script easily.

Features:-

It is used for understand the script easily.

It is used for enhancing the script easily.

Active Screen capture level settings



Write a program to disable active screen programmatically?

```
Dim x
Set x=createobject (“quicktest.application”)
x.launch
x.showpanescreen “activatescreen”, false
Wait 3
x.windowstate=”maximized”
x.visible=true
Set x=nothing
```

3. Data Table

A Data Table provides a way to create data driven test cases. Data table is also called as formula1 sheet, which is developed by the third party and integrated with the Q.T.P. Each test case has one global data sheet which is accessible to all actions inside that test case and each action has its own private data table also known as local data table. The name local data table is somewhat misleading because it is in fact possible to access any action’s local data table from any other action, but the way of accessing the data becomes a bit different.

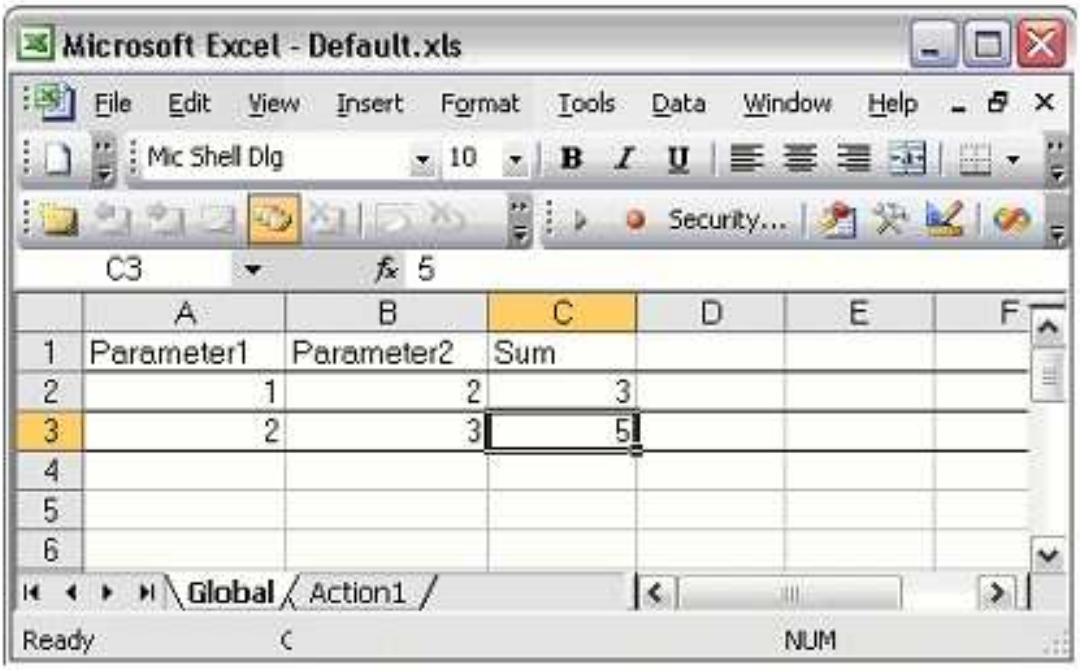
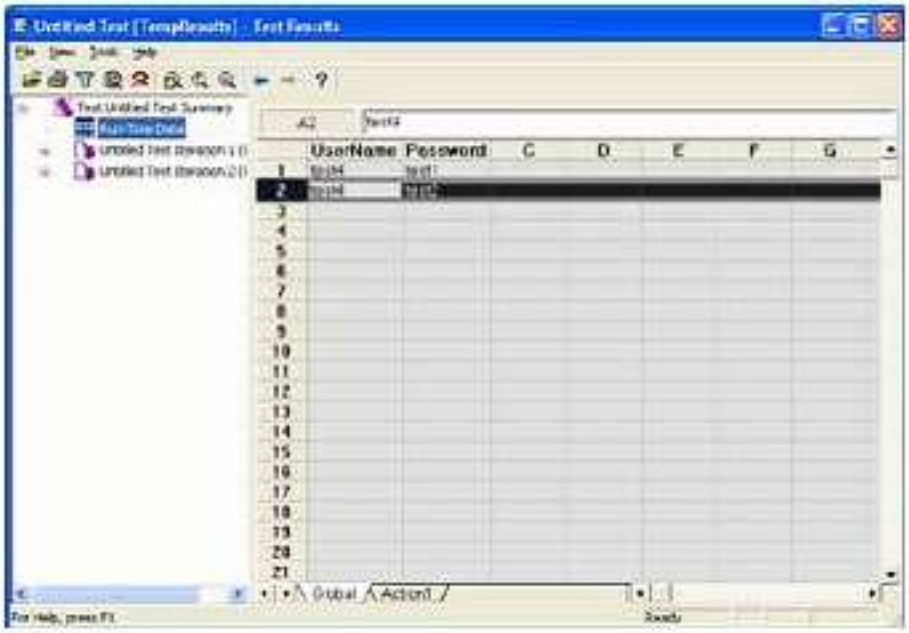


Figure 3.1 shows a sample DataTable with 2 parameters, Username and Password

We can use most of the formulas that work inside a typical Excel spreadsheet. But there are some
differences between a DataTable and an Excel spreadsheet. In fact a DataTable is wrapped around an Excel spreadsheet—which provides access functionality to the values but does not expose the Excel spreadsheet object model.

‘Gives the value of Parameter1 stored in
‘The Global data table.
Data Table (**“Parameter1”**, dtGlobalSheet)
‘Gives the value of Parameter1 stored in
‘The current’s action local data table.
Data Table (**“Parameter1”**, dtLocalSheet)

The same Data Table cannot have duplicate parameter names but we can use the same name Parameters in different sheets (Global Data Table and Local Data Table). Each Data Table has only 1row enabled even when it is blank and the other rows get enabled when data is entered into a new row. A Data Table is stored as “Default.xls” file in the test folder



When viewed in Excel, the first row of the sheet contains the parameter names, while QTP displays the parameter as the column titles. Therefore, when viewed using Excel, the 2nd row starts the 1st row of data in the DataTable. The DataTable shown above has only 2 data rows enabled. Note that QTP makes a data row enabled by marking the borders of the row in the actual spreadsheet. A row with no data but with marked borders is still considered as an enabled row by QTP. To delete an enabled row we must select the row and delete it from the context menu which appears on right clicking the row.

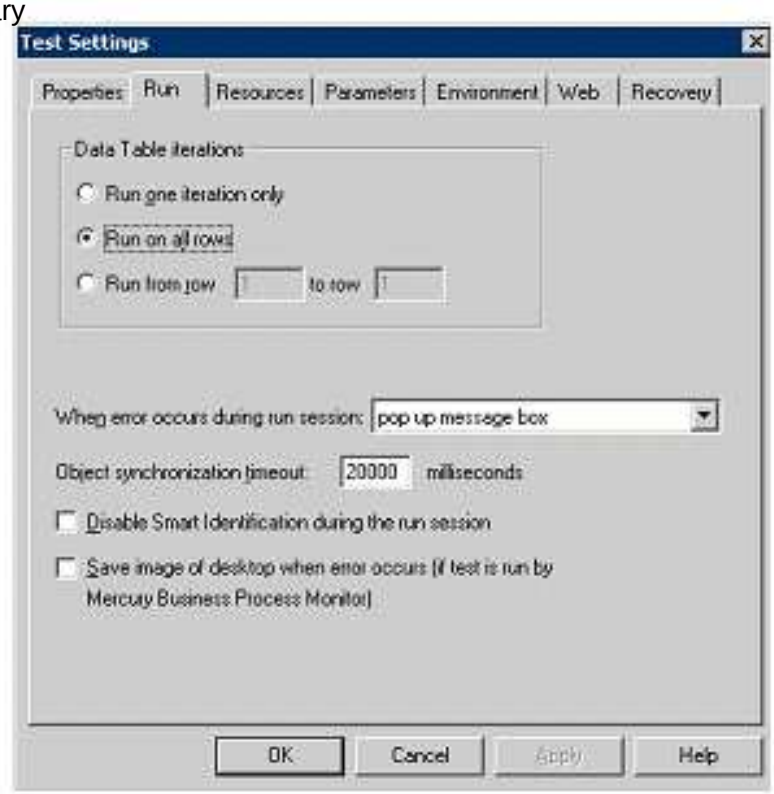
Design and run-time data table

Design time data table

As the name suggest the data table during the script design time is known as design time data table.
Any changes to this are saved when the script is saved.

Run-time data table

The run-time data table contains a copy of the design time data table when a script is executed. It may contain values that are changed during script execution and are presented in the test result summary. The changes made to the data table during run-time are not saved to design time data table. Figure 4-3 shows a run-time data table from the test results summary



When to use the global or a local data table

It is important to understand in what situations the global or a local data table should be used.

Consider the following two scenarios

Scenario 1 - Log into the application, book 1 ticket, log out. Repeat the scenario for many users

Scenario 2 - Log into the application, book 3 tickets, and log out

Scenario 1

The Global data table is better suited for this scenario where we have the user name, password
and tickets details as the parameters and we execute the scenario using a single action (which does
everything) or multiple actions (Login, booking and logout).

NOTE: We can use an external spreadsheet as a Data table by

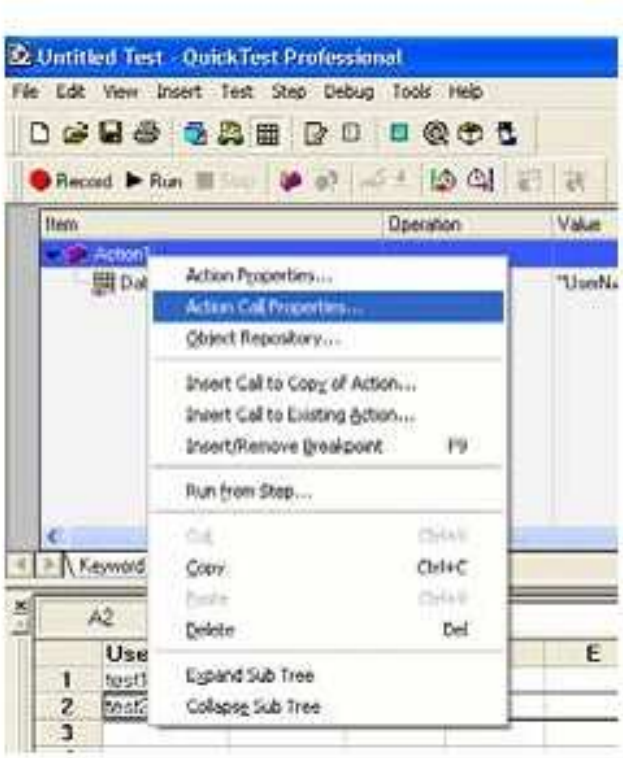
Specifying the location of the file in the *Resource (Tab)* as shown in the
Figure 4-4

Scenario 2

A Local data table is better suited for this scenario. Here a good approach would be to split the test
into three actions: login, booking and logout. Login and logout can use the username and password
parameters from the global data table and booking can use ticket detail parameters from its local data table and the action will be executed for all rows in its local data table.

Setting data table iterations

To run a test case for some number of iterations we need to set the iterations of global data table in
the Test Settings dialog, which is invoked using *Test→Settings…→Run (Tab)* Figure shows the
iteration settings for the global table. These settings are specific to script.



The Action call properties dialog can be used to set the iterations as shown in the Figure 4-6
We can set the iteration settings for an Action call by going into the keyword view and then right
clicking on the Action and selecting *Action Call Properties…*as shown in the below figure

Data table object model

QTP provides an object model to access various properties and methods in a data table:

There are three types of objects

- DataTable - Represents all the global and local data tables in the test
- DTSheet - Represents a single sheet in the test
- DTParameter - Represents a single column in a sheet.

Each object has certain functions available and certain properties associated with it. These are explained in detail in the QTP user manual.

Data table formatting

When data is entered into the data table it automatically formats the value using the best possible matching format. For example, if "12345678901" is entered into a cell then it would be auto formatted to "1.23456789E+010". In situations where the formats are important the data should be entered with care. If data entered in the cell start with a single quote (‘) then it is always treated as a text and no format conversion is performed. We can also define a specific format by right clicking the cell or an entire column and then picking a specific format from the popup context menu.

Problem 3.1 How to access a parameter from the global data sheet

There are a variety of ways to access a parameter from the global data table, most of which are
presented in the following code snippet:

```
'Methods of getting a Data Table value
Val = DataTable.Value ("ParamName", dtGlobalSheet)
Val = DataTable.Value ("ParamName","Global")

'By giving the sheet index starting from 1 for the global sheet
Val = DataTable.Value ("ParamName", 1)

' Sheet name or id is a optional parameter and is assumed
' to be as for global data sheet in case not provided
Val = DataTable.Value ("ParamName")

' Value property is the default property of the DataTable object
```

```
' So DataTable ("ParamName", dtGlobalSheet) is
' Equivalent to DataTable.Value ("ParamName", dtGlobalSheet)
Val = DataTable ("ParamName", dtGlobalSheet)
Val = DataTable ("ParamName")

'Using the data table object model
Val = DataTable.GlobalSheet.GetParameter ("ParamName").Value

'Using the data table object model
Val = DataTable.GlobalSheet.GetParameter ("ParamName").ValueByRow
(1)
```

Problem 3-2. How to access a parameter from a Local data sheet

```
'Various methods to get data table value
Val = DataTable.Value ("ParamName", dtLocalSheet)
Val = DataTable.Value ("ParamName","<LocalActionName>")
Val = DataTable ("ParamName", dtLocalSheet)
Val = DataTable ("ParamName","<LocalActionName>")

'The local sheet of the action which is executing this statement
Val = DataTable.LocalSheet.GetParameter ("ParamName").value
```

Problem 3-3. How to check if a Sheet exists

```
'Function to check if DataTable sheet exists
Function isSheetExists (sheetName)
   On error resume next
    isSheetExists = TRUE
    Err. Clear
    Set objSheet= DataTable.GetSheet (sheetName)
    'In case error occurred sheet does not exist
    If err. number<>0 then
          isSheetExists = FALSE
    End if
End Function
```

Problem 3-4. How to preserve format of data output to a data table

```
'This would be modified to 1.23456789E+010 due to auto formatting
DataTable ("ParamName") = "12345678901"
'This will not be auto formatted and will be treated as text
DataTable ("ParamName") = "'" & "12345678901"
```

Problem 3-5. How to check if a parameter exists in a specific sheet

```
'Check if a parameter exists in data table
Function isParameterExists (sheetName, paramName)
   On error resume next
    isParameterExists = TRUE
    Err. Clear
    ParamTotal = DataTable.GetSheet (sheetName).GetParameter (paramName)
    'In case of error the parameter does not exist
```

```
    If  err. number<>0  then
          isParameterExists  =  False
    End  if
End  Function
```

Problem 3-6. Current iteration number of QTP script:

```
STR = "Current QTP iteration: " & Environment ("TestIteration") & vbNewLine & _
'DataTable ("Param1", dtGlobalSheet) & vbNewLine & _
DataTable ("Param2", dtGlobalSheet)
MsgBox STR
```

Problem 3-7.How to export contents of a WebTable to a data sheet. Let's assume that the first row of the data table contains the columns heading. We then add those as parameters of the data table:

```
'Variable declaration
Dim i, j
Dim rowCount, colCount
Dim cellText, objTable

 'Get table object
Set objTable = Browser ("").Page ("" _____).WebTable ("")
'Get the row count of the webtable
rowCount = objTable.RowCount
'Get the column count of the webtable header row
ColCount = objTable.ColumnCount (1)
'Create an output sheet
Set outSheet = DataTable.AddSheet ("Output")
'Create Parameters based on the 1st row of the web table
For i = 2 to colCount
cellText = objTable.GetCellData (1,i)
'Note in case the CellText contains space in between
'then QTP will automatically convert it to a "_" character
outSheet.AddParameter cellText,""
Next
'Skip first row as we assumed it to be a header row
For i = 2 to rowCount
outSheet.SetCurrentRow i-1
'Re-calculate the column count as some rows
'Have different column sizes
colCount = objTable.ColumnCount (i)
For j = 2 to colCount
cellText = objTable.GetCellData (i, j)
'We are using index here to avoid the problem of
'the "_" issue if cell text has spaces or new line chars
'then we will get an error. to overcome that we can also use
'outSheet.GetParameter (Replace (cellText," ","_")).Value
```

```
outSheet.GetParameter (j-1).value = cellText
Next
Next
```

Problem 3-8. How to get value of a parameter from any specific row in the data table

We use the ValueByRow method to get value for any row

```
'Get a value by row
DataTable.GetSheet ("SheetName").GetParameter ("ParameterName").
ValueByRow (RowNumber)
```

Problem 3-9. How to execute a script for all Global Data Table iterations, when the script is set to run for only one iteration:

In case we want to manually repeat the code for each iteration, we need to write a bit code.

```
'Declare variable
Dim i, iCount
'Get the global sheet object
Set oGlobal = DataTable.GlobalSheet
'Get # of rows
iCount = oGlobal.GetRowCount
For i = 1 to iCount
'Set the current row
oGlobal.SetCurrentRow i
'Execute the code to be repeated here
Msgbox DataTable ("UserName")
Next
```

Problem 3-10. How to get the number of columns that contain data:

To solve this problem we need to utilize the excel formula COUNTA. We add a parameter to the data table with the formula and then read its value:

```
'Add a new parameter with the formula
'For Columns 1 of data table use A1:A65536
'For column 2 of data table use B1:B65536 and so on
DataTable.GlobalSheet.AddParameter "New","=COUNTA(A1:A65536)"
'Get the new value
Msgbox DataTable ("New")
```

Note: The above code won't work when there are no columns in the data table or all the columns have been used

Data table Parameterization

1. Record new script “*www.mail.yahoo.com*”
2. Start recording and provide user name and password
3. Stop recording
4. Now create 2 column in Global data table named “*user name”* and “*password*”
5. provide values for both column
6. With help of data table associated method properties the most popular is value which is the default data table property. Now let us see how we access values stored in data table.

```
Dim Uid, Pw
Uid =datatable.value (”user name”, dtGlobalSheet)
Pw =datatable.value (”password”, dtGlobalSheet)
‘After that changes it into
Browser (“mail yahoo”).page (“mail yahoo”).webebit (“username“ ).set “Uid”
Browser (“mail yahoo“).page (“mail yahoo”).webebit (“password”).set “Pw”
‘In order to move your pointer to next row, if we want to use second row just use the bellow code
  Datatable.SetNextRox
```

Working with the data table objects

Adds the specified sheet to the run-time Data Table

AddSheet Method

Syntax: DataTable.AddSheet (*Sheet Name*)
Example:
Variable=DataTable.AddSheet ("MySheet").Add Parameter ("Time", "8:00")

DeleteSheet Method

Deletes the specified sheet from the run-time Data Table.

Syntax: DataTable.DeleteSheet *SheetID*
Example: DataTable.DeleteSheet "MySheet“

Export Method

Saves a copy of the run-time Data Table in the specified location.

Syntax:
DataTable.Export (*FileName*)
Example:

DataTable.Export ("C:\flights.xls")

Uname=datatable.value (“uname”, dtGlobalSheet)

Pw=datatable.value (“pw”, dtGlobalSheet)

Name=”Agentname”

Dialog (“login”).winedit (name).set Uname

Dialog (“login”).winedit (“password”).set Pw

Datatable.export (“c:\login.xls”)

ExportSheet Method

Exports a specified sheet of the run-time Data Table to the specified file.

Syntax: DataTable.ExportSheet (*FileName*, DT*Sheet*)
Example: DataTable.ExportSheet "C:\name.xls”, 1

GetRowCount Method

Returns the total number of rows in the longest column in the global data sheet or in the specified data sheet of the run-time Data Table.

Example:
Rowcount = DataTable.GetSheet ("MySheet").GetRowCount

GetSheet Method

Returns the specified sheet from the run-time Data Table.

Example: MyParam=DataTable.GetSheet ("MySheet").Add Parameter ("Time", "8:00")

GetSheetCount Method

Returns the total number of sheets in the run-time Data Table.

Import Method

Imports the specified Microsoft Excel file to the run-time Data Table.

Syntax: DataTable.Import **(***FileName***)**
Example: DataTable.Import ("C:\flights.xls")

ImportSheet Method

Imports a sheet of a specified file to a specified sheet in the run-time Data Table.

Syntax: DataTable.ImportSheet (*FileName*, *SheetSource*,*SheetDest*)

Example: DataTable.ImportSheet "C:\name.xls", 1,"name"

SetCurrentRow Method

Sets the specified row as the current (active) row in the run-time Data Table.
Example: DataTable.SetCurrentRow (2)

SetNextRow Method

Sets the row after the current (active) row as the new current row in the run-time Data Table.

SetPrevRow Method

Sets the row above the current (active) row as the new current (active) row in the run-time Data Table**.**

GlobalSheet Property

Returns the Global sheet of the run-time Data Table.
Example:
DataTable.GlobalSheet.AddParameter "Time", "5:45"

LocalSheet Property

Returns the current (active) local sheet of the run-time Data Table.
Example:
MyParam=DataTable.LocalSheet.AddParameter("Time", "5:45")

RawValue Property

Retrieves the *raw value* of the cell in the specified parameter and the current row of the run-time
Data Table.

Syntax: DataTable.RawValue *ParameterID* [**,** *SheetID*]
SheetID can be the sheet name, index or dtLocalSheet, or
DtGlobalSheet.

Value Property

Retrieves or sets the value of the cell in the specified parameter and the current row of the run-time Data Table.

Syntax:

DataTable.Value**(***ParameterID* [**,** *SheetID*]**)**

1. .

Creating Tests with Multiple Actions

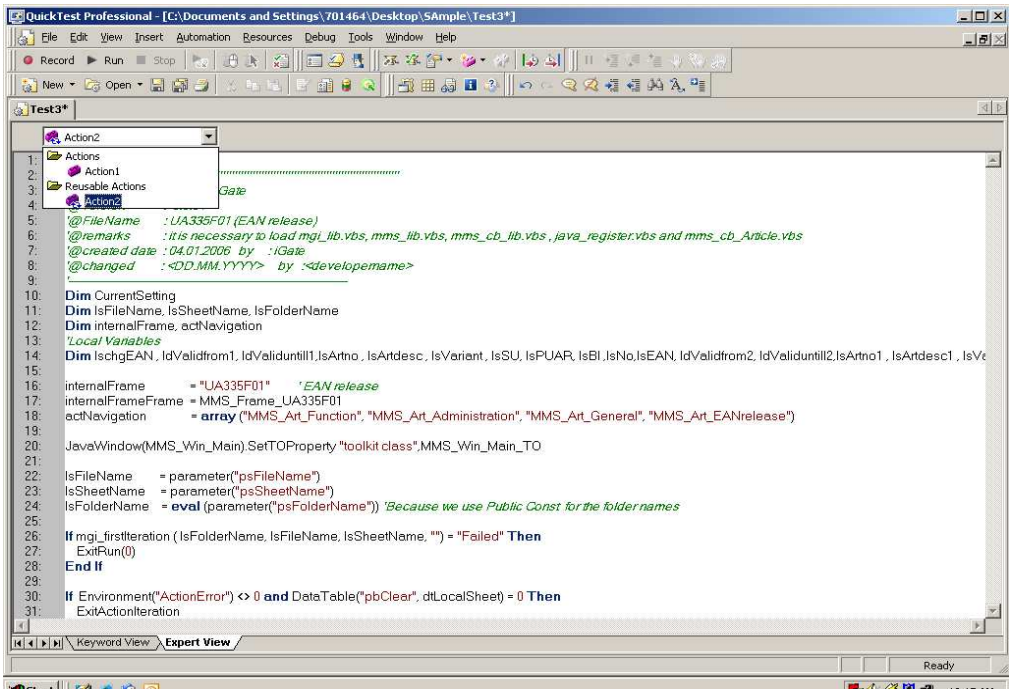
Actions divide your test into logical sections. When you create a new test, it contains a
call to one action. By dividing your tests into calls to multiple actions, you can design more modular and efficient tests. Use one of the following menu options or toolbar buttons to add actions to your test:

- Step → Split Action (or) use the Split Action button.
- Insert → Call to New Action (or) use the Insert Call to New Action button.
- Insert → Call to Copy of Action (or) right-click an action and choose Insert Call to Copy of Action.
- Insert → Call to Existing Action (or) right-click an action and choose Insert Call to Existing Action.

Steps to divide the Test into multiple Actions

- Save the Recorded application with an appropriate name.
- Select the page where you want the second action to begin.
- Choose Step → Split Action (or) click the Split Action button.
- Enter Names, Descriptions for the two actions and click 'OK' button.
- The two Actions are displayed in the Keyword View.

Different Types of Actions



Actions are classified into three types.

- Non-Reusable Action – An action that can be used within the test in which it is created.
- Reusable Action – An action that can be called multiples times by the test in which it was created as well as by other tests.
- External Action – Similar to Reusable action created in another test. These are in Read-Only format. The actions can be modified from its Original Test.

Creating New Actions

You can add new actions to your test during a recording session or while designing your test.
You can add the action as a top-level action, or you can add the action as a sub-action (or nested action) of an existing action in your test.

To create a new action in your test:
If you want to insert the action within an existing action, click the step after which you want to insert the new action.

- Choose Insert > New Action or click the New Action button. The Insert New Action dialog box opens.
- Type a new action name or accept the default name.
- If you wish, add a description of the action. You can also add an action description at a later time in the Action Properties dialog box.
- Select Reusable Action if you want to make the action reusable. You can also set or modify this setting at a later time in the Action Properties dialog box.
- Decide where to insert the action and select At the end of the test or After the current step.
- Click OK.

A new action is added to your test and is displayed at the bottom of the test tree or after the current step. You can move your action to another location in your test by dragging it to the desired location.

Inserting Existing Actions

You can insert an existing action by inserting a copy of the action into your test, or by inserting a call to the original action.

Inserting Copies Actions

When you insert a copy of an action into a test, the action is copied in its entirety, including checkpoints, parameterization, and the corresponding action tab in the Data Table. The action is inserted into the test as an independent, non-reusable action

Once the action is copied into your test, you can add to, delete from, or modify the action just as you would with any other recorded action. Any changes you make to this action after you insert it affect only this action, and changes you make to the original action do not affect the inserted action. You can insert copies of both reusable and non-reusable actions.

Steps to insert a copy of an action:

- Choose Insert > Copy of Action, right-click the action and select Insert Copy of Action, or right-click any step and select Action > Insert Copy. The Insert Copy of Action dialog box opens.
- Type a meaningful name for the action in the New action name box and give action description
- Specify where to insert the action: At the end of the test or after the current step.
- Click OK. The action is inserted into the test as an independent, non reusable action.

Inserting Call to Actions

You can insert a call (link) to a reusable action that resides in your current test (local action), or in any other test (external action).When you insert a call to an external action, the action is inserted in read-only format. You can view the components of the action in the action tree, but you cannot modify them.

Steps to insert a call to an action:

- Choose Insert > Call to Action, right-click the action and select Insert Call to Action, or right-click any step and select Action > Insert Call. The Insert Call to Action dialog box opens.
- In the Select an action box, select the action you want to insert from the list.
- Specify where to insert the action : At the end of the test or After the current step.
- Click OK. The action is inserted into the test as a call to the original action

Nesting Actions

Sometimes you may want to run an action within an action. This is called *nesting*. Nesting actions Help you maintain the modularity of your test. Enable you to run one action or another based on the results of a conditional statement.

Splitting Actions

You can split an existing action into two sibling actions or into parent-child nested actions. You cannot split an action and the option is disabled

- When an external action is selected
- When the first line of the action is selected
- While recording a test
- While running a test
- When you are working with a read-only test

Miscellaneous

Setting Action Properties

The Action Properties dialog box enables you to modify an action name, add or modify an action description, and set an action as reusable.

Sharing Action Information

There are several ways to share or pass values from one action to other actions:

- Store values from one action in the global Data Table and use these values as Data Table parameters in other actions.
- Set a value from one action as a user-defined environment variable and then use the environment variable in other actions.
- Add values to a Dictionary object in one action and retrieve the values in other actions.

Exiting an Action

You can add a line in your script in the Expert View to exit an action before it runs in its entirety.
There are four types of exit action statements you can use:

- ExitAction - Exits the current action, regardless of its iteration attributes.
- ExitActionIteration - Exits the current iteration of the action.
- ExitRun - Exits the test, regardless of its iteration attributes.
- ExitGlobalIteration - Exits the current global iteration.

Removing Actions from a Test

We can remove Non-reusable actions, External Actions, Reusable Actions, or Calls to External or Reusable actions.

Renaming Actions

You can rename actions from the Tree View or from the Expert View.

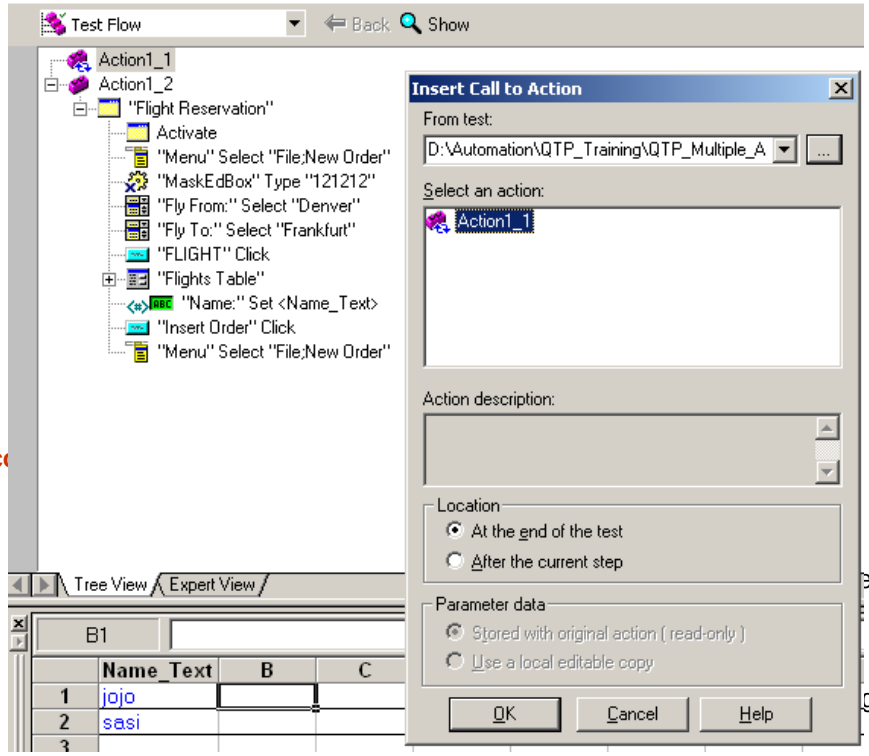
Action Template

If you want to include one or more statements in every new action in your test, you can create an action template.
Steps to create an action template:

- Create a text file containing the comments, function calls, and other statements that you want to include in your action template.

Save the text file as *ActionTemplate.mst* in your <*QuickTest Installation Folder*>*dat* folder

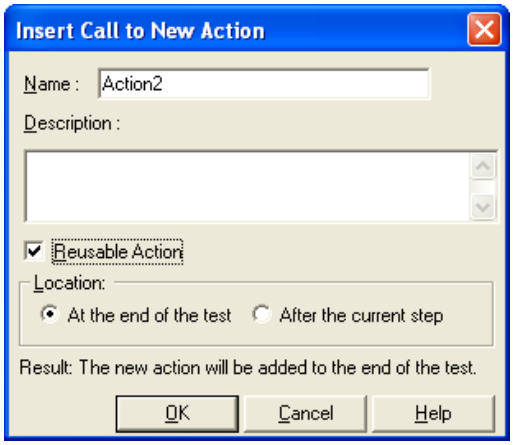
Call to copy of Action

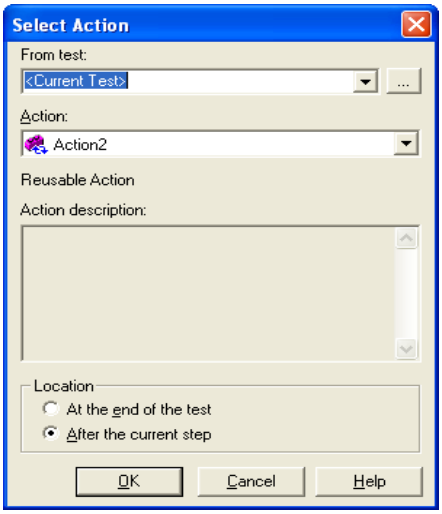


Pre-c

epository.

g action

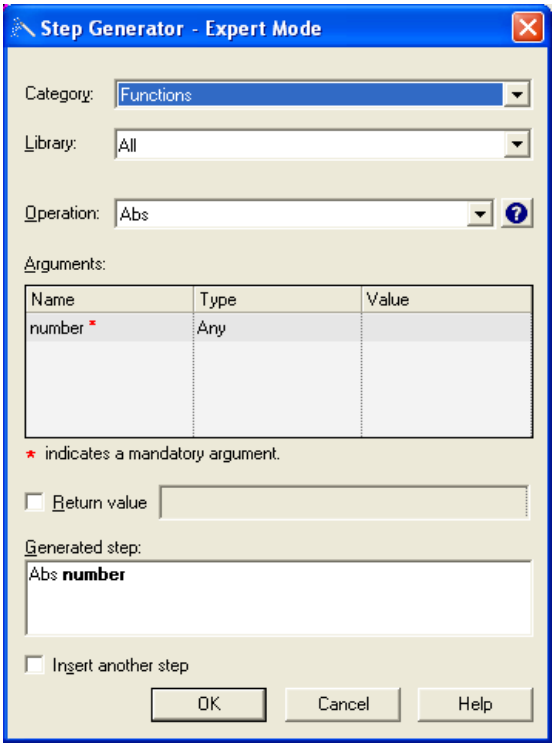




The Step Generator dialog box helps you quickly and easily add steps that use test object methods, utility object methods, and function calls, so that you do not need to memorize syntax or

to be proficient in high-level VBScript. You can use the Step Generator from the Keyword View and also from the Expert View.

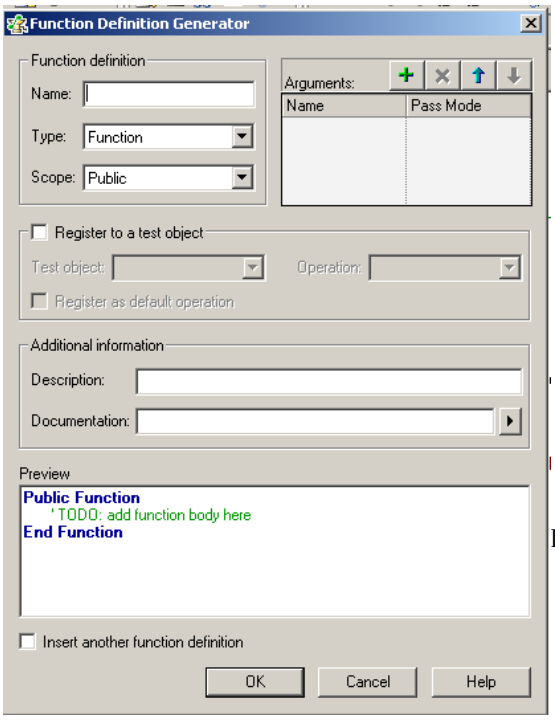
Choose Insert→ Function definition Generator, from the menu bar



Function Definition

The Function definition Generator which enables to generate definitions for new user defined functions & adds header information to them. You can register these functions to a test object if needed. You fill in the required information and function definition.

Choose Insert→ Function definition Generator, from the menu bar



hin you, you can do it

Pass the values from one action to another action?

```
**********************************************************************************
'Action 1
Environment .Value ("strname") =inputbox ("enter the name")
'Action 2
Set a = Environment. Value ('strname')
Msgbox a
'Web application
'Action 1 code
Browser ("Premium Tropicals").Page ("Premium Tropicals").WebEdit ("Cname").Set
DataTable ("username", dtGlobaSheet)
'Username name through DataTable
Environment. Value (Agent Name).DataTable.Value ("Cname"dtGlobalSheet)
'Another action
Username = Environment. Value (Agent Name).
Msgbox Username
**********************************************************************************
```

QTP Environment Variables

The variables that are commonly used across the environment in many tests by different resources are known as Environment Variables. There are two types of Environment variables

1. Built-in-variables
2. User Defined Variables

1. Built-in-variables:

These variables will be by default available in every test and can be directly used in any test with help of following syntax.

Syntax: Environment. Value ("Built-in-variables")

Example:

```
Var=environment. Value ("OS")
 'To display the Operating System
Msgbox var
```

2. User Defined Variables:

The variables which are required commonly in number of test apart from the Built-in-variables need to be created by the user which is known as User Defined Variables .User Defined Variables are created in environment file, any body in that environment can **Associate** this file and use the variables in it.

There are two types of User Defined Variables

1. **Internal User Defined Variables: -** which are used in the same file

Example:

- Open the Cal application
- Put the tool under recording mode
- Capture the objects properties of Cal application to Object Repository
- Stop recording
  - Declaring the Environment Variables
- Activate the menu item Test
- Go to Settings
- Select the Environment tab
- Select variable type as User-defined
- Click on New button
- Add new Environment window will appear
- Give the details of Name and value (type will be Internal)
- Click on OK
- Again Click on New button to add one more variable
- Add new Environment window will appear
- Give the details of Name and value (type will be Internal)
- Click on OK
- Click on Apply
- Click on OK
- If you want you can Export these data to a a file with .xml extention file in the Environment folder
  - Associating the Environment Variables (by parameterizing)
- Develop the script in test pane as below

```
**********************************************************************************
'Setting the declared environment value (a) to value1 edit button
VbWindow ("Form1").VbEdit ("val1").Set environment.Value ("a")
'Setting the declared environment value (b) to value2 edit button
VbWindow ("Form1").VbEdit ("val2").Set environment. Value ("b")
'Clicking on ADD button
VbWindow ("Form1").VbButton ("ADD").Click
**********************************************************************************
```

1. Run the test
2. Analyze the results

2. External User Defined Variables:-

Which are imported from other file:
Example:

- Open the Cal application
- Put the tool under recording mode
- Capture the objects properties of Cal application to Object Repository
- Stop recording

Declaring the Environment Variables

- Activate the menu item Test
- Go to Settings
- Select the Environment tab
- Select variable type as User-defined
- Select the check box of 'load variables and values from an external file'

If you want you can make use of Exported data or you can create your own data in a file with .xml extension file in the Environment folder

- Browse that file
- Click on Apply
- Click on OK

Associating the Environment Variables (by parameterizing)

- Develop the script in test pane as below

```
**********************************************************************************
Setting the declared environment value (a) to value1 edit button
VbWindow ("Form1").VbEdit ("val1").Set environment.Value ("a")
'Setting the declared environment value (b) to value2 edit button
VbWindow ("Form1").VbEdit ("val2").Set environment. Value ("b")
'Clicking on ADD button
VbWindow ("Form1").VbButton ("ADD").Click
**********************************************************************************
```

- Run the test
- Analyze the results

Environment object:

We have three types of environment object in QTP.The types are

1. LoadFromFile

Example:

```
If environment.loadfromfile (“C:\book.xml“).exist then
Msgbox “xml file exists”
Else
Msgbox “xml file does not exist”
```

2. Value

Example 1:

```
Environment. Value (“StrUserName”).inputbox (“enter the agent name”)
Username= Environment. Value (“StrUserName”)
Msgbox Username
```

Example 2:

```
Environment. Value (“my variable”) =10
My value= Environment. Value (“osversion”)
Msgbox My value
```

Utility Objects

1. TextUtil Object

GetText Method

Returns the text from the specified window handle area.
Syntax
TextUtil.GetText **(***hWnd* [**,** *Left***,** *Top***,** *Right***,** *Bottom*]**)**

GetTextLocation Method

Checks whether a specified text string is contained in a specified window area.
Syntax
TextUtil.GetTextLocation (*TextToFind*, *hWnd*, *Left*, *Top*, *Right*, *Bottom* [, *MatchWholeWordOnly*])

2. Reporter Objects

ReportEvent Method

Reports an event to the Test Report.

Syntax

Reporter.ReportEvent *EventStatus, ReportStepName, Details [, in]*
EventStatus – micPass, micFail, micDone, micWarning

Filter Property

Retrieves or sets the current mode for displaying events in the Test Results.
Syntax

To retrieve mode setting: *CurrentMode* = Reporter.Filter
To set the mode: Reporter.Filter = *NewMode*
Mode - 0 or rfEnableAll, 1 or rfEnableErrorsAndWarnings,
2 or rfEnableErrorsOnly, 3 or rfDisableAll

ReportPath Property

Retrieves the folder path in which the current test's Test Results are stored.
Syntax
Path = Reporter.ReportPath

3. Crypt

****

```
Set a = Inputbox (“enter the password”)
Dialog (“Login”).WinEdit (“Password”).SetSecure a
Set password = Crypt. Encrypt (a)
Msgbox password
```

****

4. Pathfinder

****

```
Set a = CreateObject (“Scripting.FileSystemObject”)
Set b = Pathfinder. Locate (“kanak.txt”)
Set c=a.OpentextFile (b)
Do while not c.AtEndofStream
Set d = readline ()
Browser (“name = G.*”).Page (“title = G.*”).WebEdit (“index = 2”).SetSecure d
Loop
```

****

5. SystemUtil

SystemUtil has the following methods that can be used to close processes, including browsers:

1.CloseProcessByName

```
SystemUtil.CloseProcessByName "iexplore.exe"
```

2.CloseProcessByHWND: Uses windows handle to close a window.

```
Dim HWND: HWND = Browser (“title: =Google”).GetROProperty (“HWND”)
        SystemUtil.CloseProcessByHWND HWND
```

CloseProcessByWndTitle: Uses window title to close it.

```
        SystemUtil.CloseProcessByWndTitle "Google", True
```

3. TSKill with SystemUtil

```
SystemUtil.Run "tskill", "iexplore"
```

4. Process ID with SystemUtil

```
PID = Browser ("title: =Google").GetROProperty ("process id")
        SystemUtil.CloseProcessByID PID
```

Checkpoints

A checkpoint is a verification point that compares a current value for a specified property with the expected value for that property. This enables you to identify whether you're Web site or application is functioning correctly.

Adding Checkpoints to a test

There are several ways to add checkpoints to your tests.

To add checkpoints while recording:

We can add checkpoints while recording the test. Use the commands on the **Insert** menu, or click the arrow beside the **Insert Checkpoint** button on the Test toolbar. This displays a menu of checkpoint options that are relevant to the selected step in the test tree.

From Menu bar

Use the commands on the Insert menu, or click the arrow beside the Insert Checkpoint button on the Test toolbar. This displays a menu of checkpoint options that are relevant to the selected step in the test tree.

To add a checkpoint while editing your test

From Test Tree

Right-click the step in the test tree where you want to add the checkpoint and choose Insert Standard Checkpoint.

From the Active Screen

Right-click any object in the Active Screen and choose Insert Standard Checkpoint. This option can be used to create checkpoints for any object in the Active Screen (even if the object is not part of any step in your test tree).

Types of Checkpoints

A checkpoint is a verification point that compares a current value for a specified property with the expected value for that property. This enables you to identify whether you're Web site or application is functioning correctly.

Checkpoint Type	Description
Standard Checkpoint	Checks values of an object's properties
Image Checkpoint	Checks the property values of an image
Table Checkpoint	Checks information in a table
Page checkpoint	Checks the characteristics of a Web page
Text / Text Area Checkpoint	Checks that a text string is displayed in the appropriate place in a Web page or application window
Bitmap Checkpoint	Checks an area of a Web page or application after capturing it as a bitmap
Database Checkpoint	Checks the contents of databases accessed by an application or Web site
Accessibility Checkpoint	Identifies areas of a Web site to check for Section 508 compliancy
XML Checkpoint	Checks the data content of XML documents

1. Standard Checkpoint: -

We can use this Checkpoint to verify Properties of Objects like as GUI check point in Winrunner.
Example: - Manual test case.
Test case id: TC-id
Test case Name: Verify delete button
Feature to be test: Flight Reservation
Test suite Id: TS-FR
Priority: Po
Pre-condition: Existing records to be deleted.
Test procedure:

Step No	Action	I/P required	Expected

1	Focus to Flight reservation window	-------	Delete button Disabled
2	Open Existing record	ValidOrder No	Delete button enabled

Automation Program:-
Window (“Flight Reservation”).Activate
Window (“Flight Reservation”).Winbutton (“Delete order). Check Checkpoint (“Delete order”).
Window (“Flight Reservation”).WinMenu (“Menu”). Select “File; open order….”
Window (“Flight Reservation”).Dialog (“Open order”).WincheckBox (“Order No”). Set “ON”
Window (“Flight Reservation”).Dialog (“Open order”).WinEdit (“Edit”).Set “1”
Window (“Flight Reservation”).Dialog (“Open order”).WinButton (“ok”).Click.
Window (“Flight Reservation”).Winbutton (“Delete order”).Check Checkpoint (“Delete Order-2”)

Check Point Syntax:-

Window (“WindowName”).Winobject (“ObjectName”).CheckCheckpoint (“CheckpointName”)

Check Point Insertion Navigation:-

Select Position in Script → Insert menu→ checkpoint

→ Standard check point→ select testable object → click “ok” after confirmation → Select required properties with expected values → click “OK”.

NOTE:-

(a) One check point allows one object at a time unlike WinRunner.

(b) Check point insertion is possible before click stop recording except database Checkpoint and XMI check point.

(c) Object selection is mandatory before insert any type of checkpoint.

(d) Whenever check point is pass (or) fail, but QTP continues test execution up to end.

(e) If any type of Checkpoint inserted into VB Script QTP is showing a common Syntax as above.

Example 2:- Manual test Case:-

Test Case id : Tc-2

Test case Name: verify “OK” button

Feature to be tested: verify employee recruitment

Test suite ID: Ts_ER

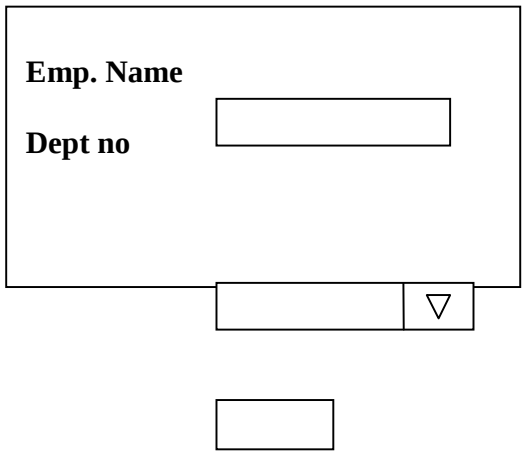
Priority: Po

Pre – condition I/P objects are taking values

Test Procedure

Step no	Action	I/P required	Expected
1	**Focus to Employee window**	------	**“OK” button disabled.**
2	**Enter Employee Name**	**Valid value**	**“OK” button disabled.**
3	**Select department No**	------	**“OK” button enabled.**

Build :-



Automation Program:-

Window (“Employee”).Activate

Window (“Employee”).Winbutton (“ok).check checkpoint (“ok”)

Window (“Employee”).Winedit (“Empname”).Set “xxx”

Window (“Employee”).Winbutton (“ok).check checkpoint (“ok.2”)

Window (“Employee”).Winbutton (“Department no”).select “xxx”

Window (“Employee”).Wincombobox (“ok).check checkpoint (“ok.3”)

Note1: -

Unlike WinRunner every QTP checkpoint is taking two types of expected values form test engineer’s, such as constant value and parameters (excel sheet column name).

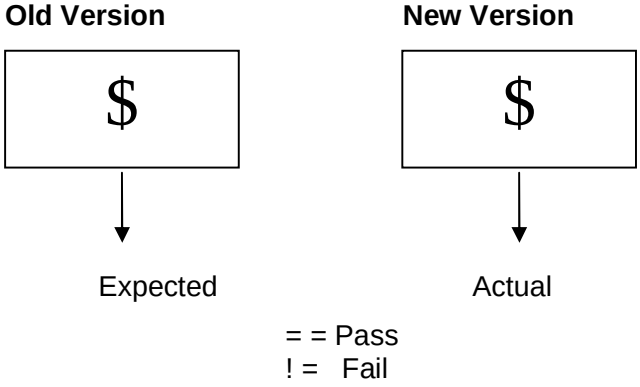
Note2: -

Our checkpoint is executing one tire when our checkpoint expected is constant. Our checkpoint is executing more than one time. Depending on no of rows in excel sheet.

(2) BITMAP CHEKPOINT:-

To compare images, we can use this check point. Unlike use QTP is supporting static and dynamic images for comparison. But test Engineers have to select multimedia option in adding manager to compare dynamic images. QTP supports 10 sec. As maximum play time of images.

Example: Logo Testing



Navigation:-

Open old Version of build → starting recording → open expected image in old version of build→ Insert menu → Check point → Bit map Checkpoint → show Expected image in old version in build -→ click "OK" after confirmation -→ Select area of image if required → Click "OK" → Close old build version → Open new build version → click run-→ Analyze result manually.

(3) Database Check Point:-

We can use this checkpoint to verify the impact front-end operation on backend tables content in terms data validation and data integrate. This checkpoint option is exactly equal to default checkpoint in database checkpoint of Win Runner.

Example:

- Create database check point (Current content of database tables selected as expected) Valid differences
- Perform front-end operation
- Run database check point (Current content of database tables selected Actual).

To apply data base testing, Test Engineer's are collecting below information from development team.→ Connectivity name in between front-end and backend in our application build → Names of tables and their column.→ Screens vs. Tables.

Above information is available Database design document development Team.

Navigation:-

Insert menu → Check point → DB checkpoint → Specified connect to DB using ODBC or Data Junction -→ Select SQL statement manually → click "Next"→ click create to select connectivity (Ex.Flight32)→ write select statement → click finish→ click "OK" after conformation of Database Table content → Perform Front-end Operation → run Database checkpoint → analysis results manually.

(4) Text Check Point: -

To verify content of an object in terms of match upper case, lower case ignore spaces exact match and text not displayed, we can use these check points.

Navigation: -

Select position Script→ Insert menu→ checkpoint → Text checkpoint → select Testable object-→ click configure if we have to apply testing on selected part object value (using text before and text after)-→ Select Test (match case, Ignore spaces, exact match , Text not displayed)-→ click "OK".

(5) Text Area Check point: -

To Verify content of screen area value in terms of match case, ignore spaces, Exact match, Text not displayed, we can use these check point.

Navigation:-

Select position script→ Insert menu → checkpoint →Text area check point → select value region → click "OK" after conformation → click configure if we have to apply testing on selected part → Select test to be applied → click "OK".

Updating Checkpoints at run-time

GetRoProperty:

Example:

```
Uname=Window ("flightReservation").Winedit ("Agent Name").GetRoProperty ("text")
If Uname=exist then
Msgbox "Agent name exists"
Else
Msgbox "does not exists"
```

Counting and displaying the content of weblist

```
a=browser ("makemytrip').page ("makemytrip").weblist ("city").GetRoProperty ("itemscount")
Msgbox a
For i=1 to 10
b=browser ("makemytrip').page ("makemytrip").weblist ("city").select (i)
Msgbox i
Cname= browser ("makemytrip').page ("makemytrip").weblist ("city").GetItem (i)
Msgbox Cname
Next
```

Output value

An output value is a step in which one or more values are captured at a specific point in your test or component and store for the duration of the run session (runtime data table). The value can later be used as input at a different point in the run session.

Types of output value

- Standard output value.
- Text area output value
- Database output value
- Xml output value

Text area output value:

Retrieve the order number and price and give input to the open order by using text area output value?

- Put the tool into the recording mode.
- Open the flight reservation application.
- Records the script for create order.
- Stop recording.
- Go to insert menu and select text area output value and capture the order number and price.

Code:

```
*******************************************************************************
Window (“flight reservation”).WinEdit (“Order No”).Output Checkpoint (“140”)
Set o_no=Datatable.Value (“order_no”, dtGlobaSheet)
Window (“Flight reservation”).WinButton (“Button”).Click
Window (“Flight reservation”).Dialog (“Open Order”).WinCheckBox (“order no”).Set “ON”
Wait 2
Window (“Flight reservation”).Dialog (“Open Order”).WinEdit (“Edit’).Set (o_no)
*******************************************************************************
```

Descriptive Programming (DP)

Introduction:

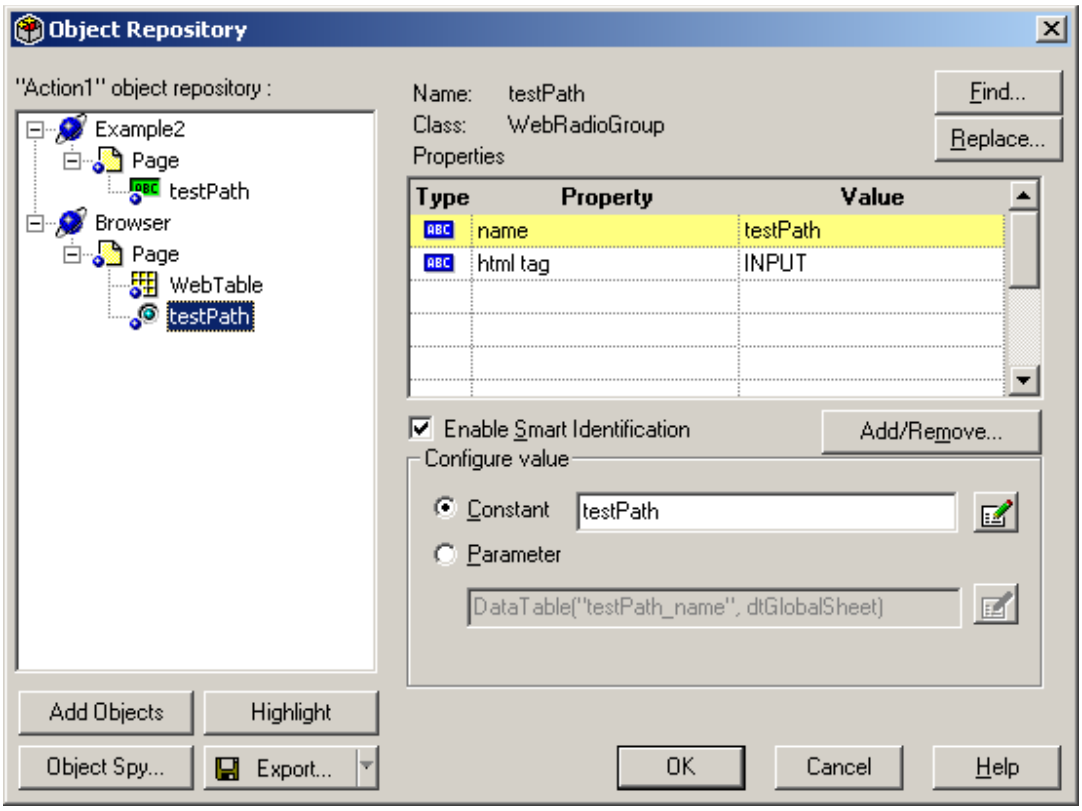
This document demonstrates the usage of Descriptive programming in QTP 8.20. It also discusses situations where Descriptive programming can be used. Using Descriptive Programming automation scripts can be created even if the application has not been developed.

Descriptive Programming:

Whenever QTP records any action on any object of an application, it adds some description on how to recognize that object to a repository of objects called object repository. QTP cannot take action on an object until unless its object description is in the Object Repository. But descriptive programming provides a way to perform action on objects which are not in Object repository

Object Identification:

To identify an object during the play back of the scripts QTP stores some properties which helps QTP to uniquely identify the object on a page. Below screen shots shows an example Object repository:



Now to recognize a radio button on a page QTP had added 2 properties the name of the radio button and the html tag for it. The name the left tree view is the logical name given by QTP for the object. This can be changed as per the convenience of the person writing the test case. QTP only allows UNIQUE logical name under same level of hierarchy. As we see in the snapshot the two objects in Browser->Page node are “WebTable” and “testPath”, they cannot have the same logical name. But an object under some other node can have the same name. Now with the current repository that we have, we can only write operation on objects which are in the repository. Some of the example operations are given below

```
Browser ("Browser").Page ("Page").WebRadioGroup ("testPath").Select "2"
CellData = Browser ("Browser").Page ("Page").WebTable ("WebTable").GetCellData (1, 1)
Browser ("Example2").Page ("Page").WebEdit ("testPath").Set "Test text"
```

When to use Descriptive Programming?

Here there are some situations

1. If the application is having Dynamic Objects
   OR: - Difficult to handle Dynamic Objects using Object Repository

2. When we have more objects to perform operations
OR: - The performance will decrease if object repository is having huge number of objects.

3. If the application is having objects that are adding in the Run Time
OR: - We can't add objects to Object Repository in run time.

4. If we need to start Automation before Build Release
OR: - There is no application to create Object Repository.

5. If Application is having similar type of objects or similar name objects
OR: - Object Repository will create multiple objects with same description unnecessarily.

6. Big Team Size
OR: - Shared Object Repository is not changeable by multiple persons at a time.
Maintenance becomes harder if all the team members have created their own object repositories.

Types of Programmatic Descriptions

1. Static: -

You list the set of properties and values that describe the object directly in a VBScript statement.

a. Direct specification of description in script

```
Browser("micclass:=Browser").Page(micclass:=page").Link("name:=Login").Click
```

b. Assigning description to the variables and use that variables in script

```
g_MainBrowser = "micclass:=Browser"
g_MainPage ="micclass:=Page"
g_Lnk_Login = "name:=Login"

Browser(g_MainBrowser).Page(g_MainPage).Link(g_Lnk_Login).Click
```

2. Dynamic: -

You add a collection of properties and values to a Description object, and then enter the Description object name in the statement.

```
Set oBrowser = Description. create
oBrowser ("micclass").value="Browser"
oBrowser ("name").value= "Google"

Set oPage = Description. create
oPage ("micclass").value="Page"
```

```
oPage (“name”).value= “Google”

Set oLink = Description. Create
oLink (“name”).value= “Login”
oLink (“index”).value= 1

Browser (oBrowser).Page (oPage).Link (oLink).click
```

Using the Static type to enter programmatic descriptions directly into your statements may be easier for basic object description needs. However, in most cases, using the Dynamic type provides more power, efficiency, and flexibility.

How to use Descriptive programming?

There are two ways in which descriptive programming can be used
By creating properties collection object for the description.
By giving the description in form of the string arguments.

By creating properties collection object for the description.

To use this method you need first to create an empty description

```
Dim obj_Desc ‘Not necessary to declare
Set obj_Desc = Description. Create
```

Now we have a blank description in “obj_Desc”. Each description has 3 properties “Name”, “Value” and “Regular Expression”.

```
obj_Desc (“html tag”).value= “INPUT”
```

When you use a property name for the first time the property is added to the collection and when you use it again the property is modified. By default each property that is defined is a regular expression. Suppose if we have the following description

```
obj_Desc (“html tag”).value= “INPUT”
obj_Desc (“name”).value= “txt.*”
```

This would mean an object with html tag as INPUT and name starting with txt. Now actually that “.*” was considered as regular expression. So, if you want the property “name” not to be recognized as a regular expression then you need to set the “regular expression” property as FALSE

```
obj_Desc (“html tag”).value= “INPUT”

obj_Desc (“name”).value= “txt.*”

obj_Desc (“name”).regular expression= “txt.*”
```

This is how of we create a description. Now below is the way we can use it

```
Browser (“Browser”).Page (“Page”).WebEdit (obj_Desc).set “Test”
```

When we say .WebEdit(obj_Desc) we define one more property for our description that was not earlier defined that is it’s a text box (because QTPs WebEdit boxes map to text boxes in a web page).

If we know that we have more than 1 element with same description on the page then we must define “index” property for the that description

Consider the HTML code given below

```
<INPUT type=”textbox” name=”txt_Name”>
```

<INPUT type=”textbox” name=”txt_Name”>

Now the html code has two objects with same description. So distinguish between these 2 objects we will use the “index” property. Here is the description for both the object

For 1st textbox:

obj_Desc (“html tag”).value= “INPUT”

obj_Desc (“name”).value= “txt_Name”

obj_Desc (“index”).value= “0”

For 2nd textbox:

obj_Desc (“html tag”).value= “INPUT”

obj_Desc (“name”).value= “txt_Name”

obj_Desc (“index”).value= “1”

Consider the HTML Code given below:

<INPUT type=”textbox” name=”txt_Name”>

<INPUT type=”radio” name=”txt_Name”>

We can use the same description for both the objects and still distinguish between both of them

obj_Desc (“html tag”).value= “INPUT”

obj_Desc (“name”).value= “txt_Name”

When I want to refer to the textbox then I will use the inside a WebEdit object and to refer to the radio button I will use the description object with the WebRadioGroup object.

Browser (“Browser”).Page (“Page”).WebEdit (obj_Desc).set “Test” ‘Refers to the text box

Browser (“Browser”).Page (“Page”).WebRadioGroup (obj_Desc).set “Test” ‘Refers to the radio button

But if we use WebElement object for the description then we must define the “index” property because for a webelement the current description would return two objects.

Hierarchy of test description:

When using programmatic descriptions from a specific point within a test object hierarchy, you must continue to use programmatic descriptions from that point onward within the same statement. If you specify a test object by its object repository name after other objects in the hierarchy have been described using programmatic

descriptions, QuickTest cannot identify the object. For example, you can use Browser (Desc1).Page (Desc1).Link (desc3), since it uses programmatic descriptions throughout the entire test object hierarchy. You can also use Browser ("Index").Page (Desc1).Link (desc3), since it uses programmatic descriptions from a certain point in the description (starting from the Page object description).

However, you cannot use Browser (Desc1).Page (Desc1).Link ("Example1"), since it uses programmatic descriptions for the Browser and Page objects but

Then attempts to use an object repository name for the Link test object (QuickTest tries to locate the Link object based on its name, but cannot

Locate it in the repository because the parent objects were specified using programmatic descriptions).

Different ways to write a statement

Example on how to click a button in 7 ways

```
'1st method
Window("Flight Reservation").WinButton("Update Order").Click    'Common Method

'2nd method
Set wndObject=Window("Flight Reservation")    ' Assigning window object to an object variable
wndObject.WinButton("Update Order").Click       ' Following normal syntax ( click on a button)

'          OR

Set btnObject=Window("Flight Reservation").WinButton("Update Order")    ' Assigning Button object to an object variable
btnObject.Click    ' Clicking on button using button object variable

'3rd method
With Window("Flight Reservation")       ' Using With statement
          .WinButton("Update Order").click
End with

'4th method
Window("text:=Flight Reservation").WinButton("text:=&Update Order").Click    ' Descriptive programming

'5th method
Set oDes=Description.Create           ' creating a description object
oDes("nativeclass").value="Button"    ' assigning description to the description object
oDes("text").value="&Update Order"
Window("text:=Flight Reservation").winbutton(oDes).click    ' clicking on button using the created description object

'6th method
Set oDes=Description.Create       '   creating a description object
```

```
set btnObjList=Window("text:=Flight Reservation").ChildObjects(oDes)    ' Filtering the objects
For objIndex=0 to btnObjList.count-1
      propVal=btnObjList(objIndex).getroproperty("text")    ' Get property value from object
      If propVal="&Update Order" Then       ' Compare property value
            btnObjList(objIndex).click       ' Click on identified object
            Exit for       ' Exit For loop after clicking on the button
      End If
Next

'7th method
Public const wndFlight="text:=Flight Reservation"    ' Assigning window object description to a constant
Public const btnUpdate="text:=&Update Order"      ' Assigning Button object description to a constant
Window(wndFlight).winbutton(btnUpdate).click       ' Click on a button using description constants
```

Getting Child Object:

We can use description object to get all the objects on the page that matches that specific description. Suppose we have to check all the checkboxes present on a web page. So we will first create an object description for a checkboxes and then get all the checkboxes from the page

```
Dim obj_ChkDesc
Set obj_ChkDesc=Description.Create
obj_ChkDesc (“html tag”).value = “INPUT”
obj_ChkDesc (“type”).value = “checkbox”
Dim allCheckboxes, single Checkbox
Set allCheckboxes = Browse (“Browser”).Page (“Page”).ChildObjects (obj_ChkDesc)
For each single Checkbox in allCheckboxes
    singleCheckBox.Set “ON”
Next
```

The above code will check all the check boxes present on the page. To get all the child objects we need to specify an object description i.e. we can’t use the string arguments that will be discussed later in the 2nd way of using the programming description.

Debugging in QTP

Establishing our Debugging Configuration

Using Breakpoints

To instruct Quicktest to pause a run session at a predetermined place in a test or function library.Quicktest pause the run when it reaches the breakpoint before executing the step. You can use breakpoint to suspend a run session and inspect the state of your site or application.

Navigation

Debug viewer pane

The debug viewer pane contains three tabs to assist you in debugging a test .the panes are

1. Watch

To view the current value of any variable or vbscript expression that you added to the watch tab

2. Variable

During a run session, the variable tab displays the current value of all variable that you have recognized up to the last step performed in the run session

3. Command

To run a line of script to set or modify the current value of a variable or vbscript object in your test of function library which you continue the run session Quicktest uses the new value that was set in the command

Information pane

The information pane provides a list of syntax errors in your test. When you switch from expert view to keyword view quick test automatically checks the syntax error in your script and shows them in the information pane. It the information pane is not currently disabled quick test automatically opens it when a syntax error is detected

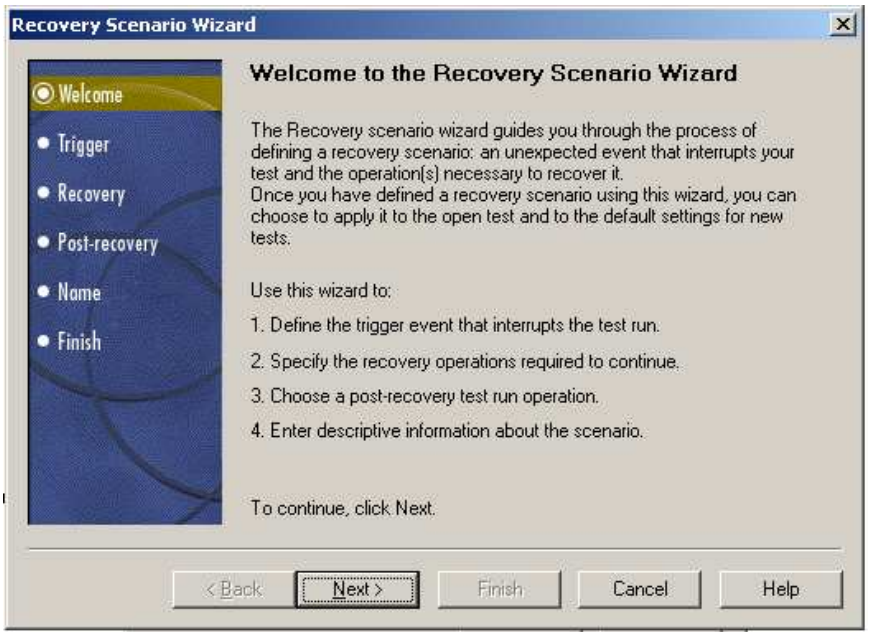
Recovery Scenarios

Unexpected events, errors, and application crashes during a run session can disrupt your run session and distort results. This is a problem particularly when running tests or components unattended - the test or component is suspended until you perform the operation needed to recover. You can instruct QuickTest to recover from unexpected events and errors that occur in your testing environment during a run session.

Handling Unexpected Events and Errors

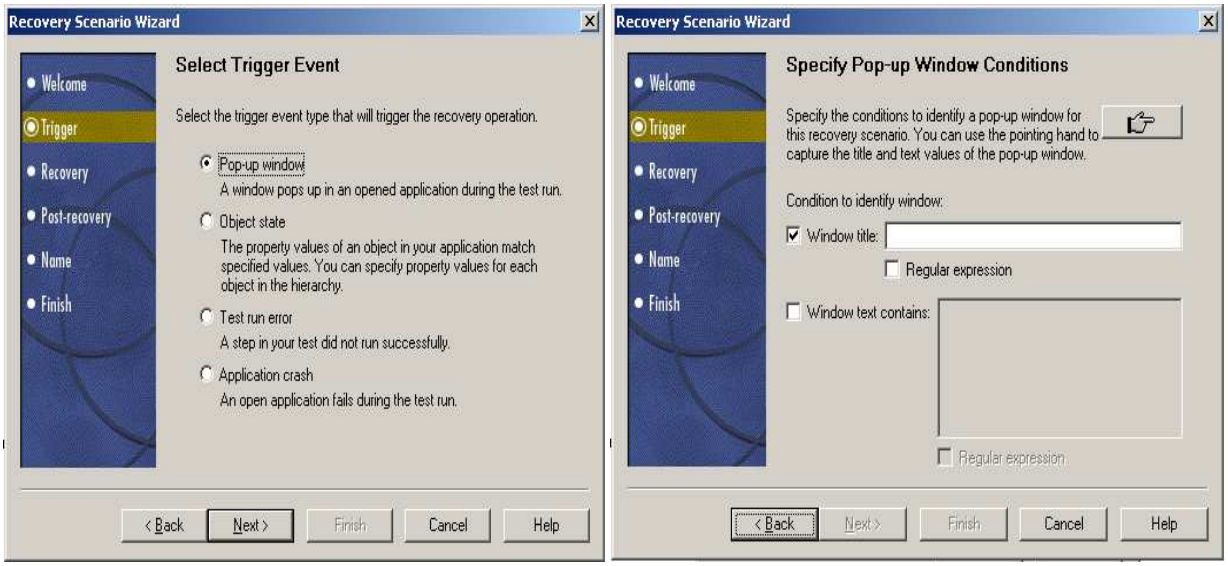
* Unexpected events and errors during a test run can disturb testing
  * Unattended tests require an action to recover.
* QTP Recovery scenario Manager can…
  * Detect and Handle the appearance of a specific error dialog.
  * Recover from an error and continue to the next step.
  * Guide you through creating a recovery scenario using the scenario Wizard.

Recovery Scenario Components



Choose, **Tools → Recovery Scenario Manager** from the Menu bar. The Recovery Scenario Manager Dialog box will be opened; click on the **New Scenario** icon, the Recovery Scenario Wizard will appear.

Trigger Event



Select the appropriate **Trigger Event** from the dialog box.

- **Pop-up window**—QuickTest 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 run session, indicating that the printer is out of paper. QuickTest can detect this window and activate a defined recovery scenario in order to continue the run session. Select this option and click **next** to continue to the Specify Pop-up Window Conditions screen.

- **Object state**—QuickTest detects a specific test object state and identifies it according to its property values and the property values of all its ancestors. **Part III** • Creating Tests 418 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. QuickTest 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 run session. Select this option and click **next** to continue to the Select Object screen.

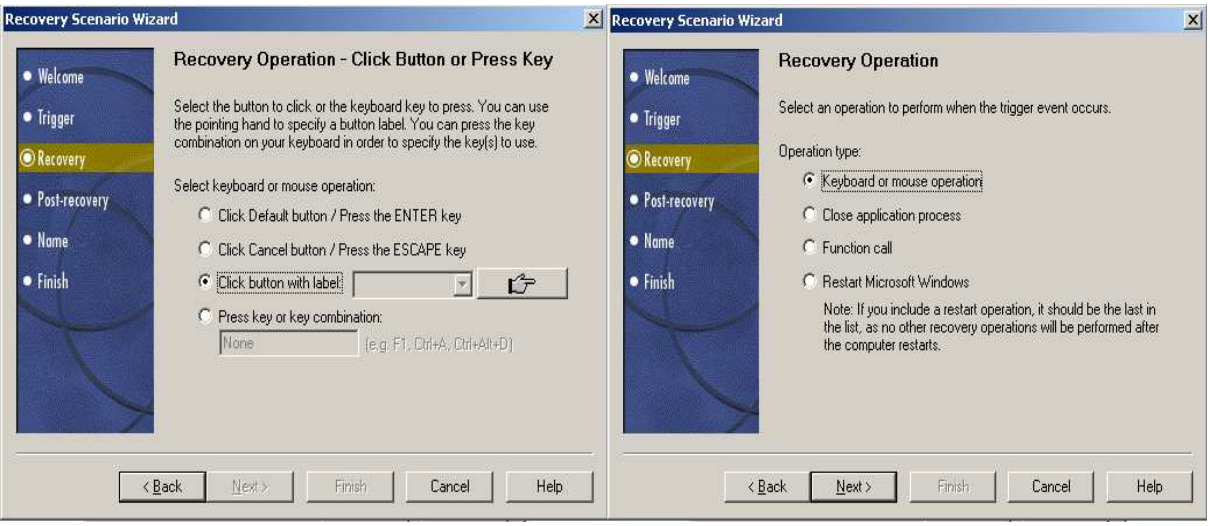
- **Test run error**—QuickTest detects a run error and identifies it by a failed return value from a method. For example, QuickTest may not be able to identify a menu item specified in the method argument, due to the fact that the menu item is not available at a specific point during the run session. QuickTest can detect this run error and activate a defined recovery scenario in order to continue the run session. Select this option and click **next** to continue to the Select Test Run Error screen.

- **Application crash**—QuickTest 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 run session. You want to be sure that the run session does not fail because of this crash, which may indicate a

different problem with your application. QuickTest can detect this application crash and activate a defined recovery scenario to continue the run session. Select this option and click **next** to continue to the Recovery Operations screen.

Notes: The set of recovery operations is performed for each occurrence of the trigger event criteria. For example, suppose you define a specific object state, and two objects match this state, the set of replay operations is performed two times, once for each object that matches the specified state. The recovery mechanism does not handle triggers that occur in the last step
of a test or component. If you need to recover from an unexpected event or error that may occur in the last step of a test or component, you can do this by adding an extra step to the end of your test or component.

Recovery Operation



Select the appropriate **Recovery Operation** from the dialog box.

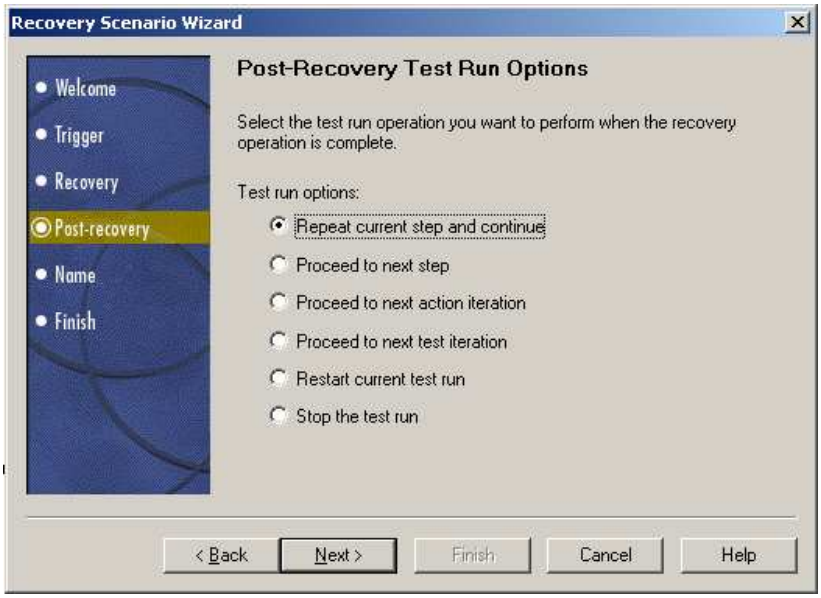
You can define the following types of recovery operations:

- **Keyboard or mouse operation**—QuickTest simulates a click on a button in a window or a press of a keyboard key. Select this option and click **next** to continue to the Recovery Operation – Click Button or Press Key screen.
- **Close application process**—QuickTest closes specified processes. Select this option and click **next** to continue to the Recovery Operation – Close Processes screen.
- **Function call**—QuickTest calls a VBScript function. Select this option and click **Next** to continue to the Recovery Operation – Function screen.
- **Restart Microsoft Windows**—QuickTest restarts Microsoft Windows. Select this option and click **next** to continue to the Recovery Operations screen.

Note: If you use the Restart Microsoft Windows recovery operation, you must ensure that any test or component associated with this recovery scenario is saved before you run it.

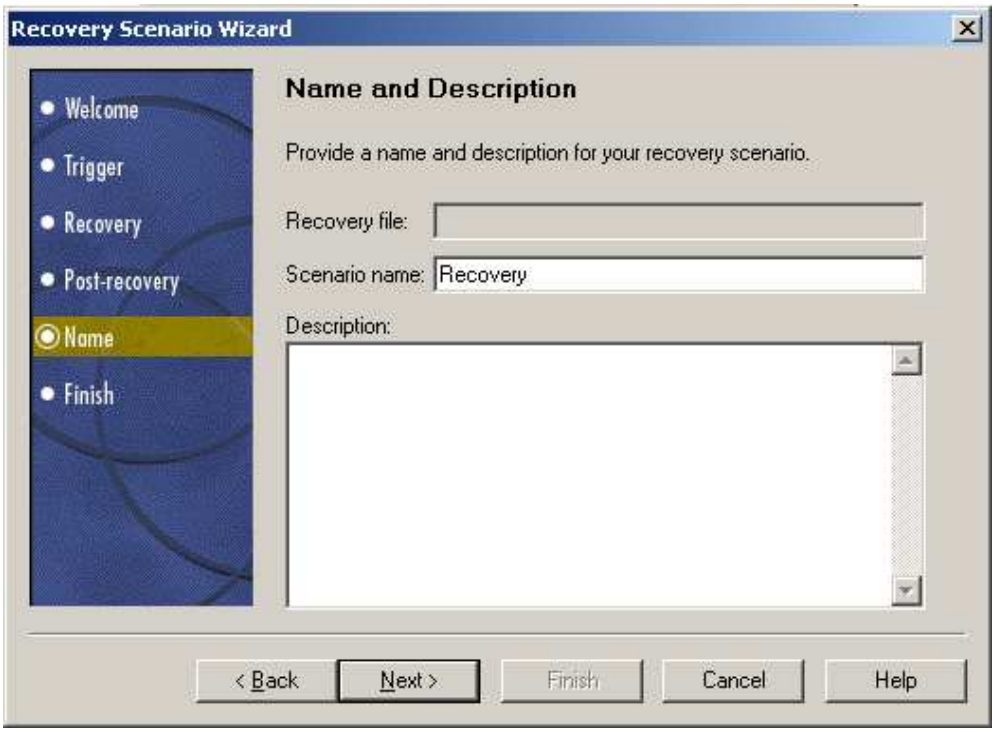
You must also configure the computer on which the test or component is run to auto login on restart.

Post-Recovery Test Run Option



Select the appropriate **Post-Recovery Operation** from the dialog box.

Save the Recovery Scenario file



Save the recovery scenario file in the specified location with an appropriate name and description (Recovery Scenario files will be saved with **.qrs** extension). And use this Recovery Scenario file wherever required.

When you want to recover from any problem, first, face the problem manually, Find the solution manually, and Recover from that problem manually.
And in the same way you implement for Automation (in QTP)

Recovery Scenario with Pop-Up window
(When a pop-window appears either thru +ve testing or –Ve testing of the application)

I. Pop-Up window (Manual)

- Put tool in Recording mode
- open the AUT (say flight Application)
- Enter User name (kanakadri)
- *Click on OK (wontedly for getting an error): see the relevant script and copy that line only.
- You will get a “pop-up” window saying with some message
- Enter OK on that pop-up window
- Enter password (mercury)
- Click OK
- Stop Recoding

Open a new test , Type Function () , you will get
Function
End Function
Modify the above code as below

Function popup recovery ()

* Copy that line here
End Function

Copy it and open a new Notepad and past it.
Save it with .vbs (VBSctipt) Extension as library file

2. Pop-Up window (thru QTP)

- Put tool under recording mode
- open the AUT (say same flight Application)
- Enter User name (kanak)
- Click on OK
- *A pop-up will appear , click on OK (see this script code, which should be removed later)
- Enter password
- Click on OK
- Stop recording

* Remove that code and keep cursor here it self.

Now Actual Recovery scenario for pop-up window starts …..

- open the AUT (say same flight Application)
- Enter User name (kanak)
- Click on OK
- *the same pop-up will appear, DON'T' click on it.
- Open tools -> Recovery Scenario Manager
- Select the new scenario (wizard will appear)
- Click NEXT
- Select trigger event option **as pop-up window**
- Click NEXT
- With help of HAND Button select * that pop-up's object (OK) of AUT
- Click NEXT
- Click NEXT to specify the recovery operation
- Select **Function Call** option
- Browse the library file which u has saved as with .vbs extension in the beginning.
- Click NEXT
- De-select the check box (of add an other recovery operation)
- Click NEXT
- Choose Post-Recovery operation as Repeat **Current Step and Continue.**
- Click NEXT
- Give the name and description

- Click NEXT
- Check the check-box of Add Scenario to Current test
- Click FINISH
- Save it as with .qrs (Quick Recovery Scenario) extension
- Click CLOSE

Run the Test which will pass the results of course with Warnings (may ignore it)

Recovery Scenario with Object State

(When the object is disabled)

I. Object State (Manual)

- Put tool in Recording mode
- open the AUT (say flight Application)
- Click on Open order icon
- A open order **Dialog box** will appear
- In which check the Order number check box
- See OK button is **Disabled** now
- Enter order number (say 9)
- *See the relevant script and copy that line only ie. where u entered 9
- Now OK button will be enabled
- Click on OK
- An order with that number (say 9) will be opened
- Stop recording

Open a new test, Type Function (), you will get

```
Function
End Function
```

Modify the above code as below

```
Function   popup recovery ()

* Copy that line here

End Function
```

Copy it and open a new Notepad and past it.
Save it with .vbs (VBSctipt) Extension as library file

]

2. Object State (thru QTP)

- Put tool in Recording mode
- open the AUT (say flight Application)
- Click on Open order icon
- A open order ***Dialog box** will appear
- In which check the Order number check box
- DON'T enter order number
- Stop recording

Now Actual Recovery scenario for Object state starts …..

- Open tools -> Recovery Scenario Manager
- Select the new scenario (wizard will appear)
- Click NEXT
- Select trigger event option **as Object state**
- Click NEXT
- With help of HAND Button select * that disabled object (OK) in the ***Dialog box** of AUT
- Click NEXT
- Set object properties and values
- Click on Add/Remove button
- Edit properties window will appear
  - **Check** property name **as Enabled** and value **as false**
  - **Check** property name **as Text** and value **as OK**
  - **Check** property name **as window Id** and value **as 1**
- Click on OK
- Click NEXT to specify the recovery operation
- Select **Function Call** option
- Browse the library file which u have saved as with .vbs extension in the beginning.
- Click NEXT
- De-select the check box (of add an other recovery operation)
- Click NEXT
- Choose Post-Recovery operation as Repeat **Current Step and Continue.**
- Click NEXT
- Give the name and description
- Click NEXT
- Check the check-box of Add Scenario to Current test
- Click FINISH
- Save it as with .qrs (Quick Recovery Scenario) extension
- Click CLOSE

Run the Test which will pass the results of course with Warnings (may ignore it)

==========

Recovery Scenario with Test Run Error

During execution **one step may not execute properly**, QTP will ignore it run next step on words.

For that **we just need to call an empty function ().**

Version 1.0 is released and came for testing

V1.0

Chennai
Hyderabad
Delhi

- Open the version 1.0 application
- put tool in recording mode
- Select all the cities names, **so that *script will be generated.**
- Stop recoding

After some time Version 2.0 is released with some changes

V2.0

Chennai
Delhi

- Open the version 2.0 application
- **Use the *same generated script only**

But when u runs this program,
It execute fist city name (Chennai)
When it comes to second city name (Hyderabad)
Tests will Stops and FAIL.
Though it's missed, in order to continue the execution **from next step onwards**

We just **call an empty function** which is stored in a library file

- **Keep the cursor where the city name seems to be missed**
- Activate tool menu item **Tools**
- Open recovery scenario manager wizard
- select trigger event as **Test Run error**
- choose the error type from drop-down box **as Item in list or menu not fund**
- Click NEXT
- Click NEXT to specify the recovery operation
- select operation type as Function call
- Click NEXT
- choose library file path where an empty function was stored
- Click NEXT
- De-select check box (of add an other recovery scenario)
- Click NEXT
- Select post-recovery as **Proceed to Next Step**
- Give name and description for this scenario
- Click NEXT
- Select add scenario to current test
- click on Finish
- Save it with .qrs extension

Run the Test

When you Run it, it execute normally till it finds an error, **when it found an error, It stops a while (means calling an empty function) and continue.**

The test will pass with warning (which u can ignore)

Recovery Scenario with Application error

This error may come when the application is missed.

Working with Recovery Scenarios using Scripting

Precondition:-
To understand this topic you need to have knowledge on creating and using recovery scenarios.
In this document I am discussing about how to add, remove recovery scenario files (.QRS) to a test and after adding how to activate, deactivate and renumbering the order to execute recovery scenarios. For example I have a recovery scenario file with the name of "sample .qrs". Assume that in this recovery file I have two recoveries with the names login Pop, RunErr. (All of you know that one file can have multiple recoveries). Now if I want to use that file in my test then I have to use following script.

```
'*****************************************************************
' Create the Application object
Set qtApp = CreateObject ("QuickTest.Application")

'Return the Recovery object for the current test
Set qtTestRecovery = qtApp.Test.Settings.Recovery

' Add the " loginPop " scenario as the first scenario
qtTestRecovery.Add "E:\kanak\Recoveryfiles\sample.qrs", " loginPop ", 1

'Add the "RunErr" scenario as the second scenario
qtTestRecovery.Add "E:\kanak\Recoveryfiles\sample.qrs", "RunErr ", 2

'Iterate the scenarios
for intIndex = 1 to qtTestRecovery.Count

'Enable each Recovery Scenario (Note: the 'Item' property is default and can be omitted)
qtTestRecovery.Item(intIndex).Enabled = True
Next

'Enable the recovery mechanism (with default, on errors, setting)
qtTestRecovery.Enabled = true

'Ensure that the recovery mechanism is set to be activated only after errors
qtTestRecovery.SetActivationMode "OnError"
Set qtApp = Nothing ' Release the Application object
Set qtTestRecovery = Nothing ' Release the Recovery object
```

'**

with the above code automatically the recovery scenarios will be added to the specified test. After adding the scenarios if you want to control the scenario like changing the scenario status, to get the scenario name, to activate or deactivate we have to use recovery object (one of the Utility object).

Recovery Object

it is a utility object to control the recovery scenario mechanism programmatically during the run session. It's having some properties and methods to control the scenarios.

Associated Methods

1. Activate Method:
Explicitly activates the recovery scenario mechanism at a specific point in the run.
Note: The Activate method only works if the recovery mechanism is enabled, and only activates those recovery scenarios that are currently enabled.
If the recovery mechanism is currently disabled, the Activate method does not activate any recovery scenarios. You can use the Recovery object's Enabled property to change the status of the recovery mechanism.
Ex: - Recovery. Activate

2. GetScenarioName Method:
Retrieves the name and source file of a recovery scenario, according to the specified position in the list of recovery scenarios associated with the test.
Ex: - Recovery.GetScenarioName Position, out_ScenarioFile, out_ScenarioName
Msgbox (out_ScenarioFile)
Msgbox(out_ScenarioName)

3. GetScenarioPosition Method
Returns the index position of a recovery scenario in the list of recovery scenarios associated with the test, according to the specified name and source file.
Ex: - Recovery.GetScenarioPosition (ScenarioFile, ScenarioName)

4. GetScenarioStatus Method
Returns the status of a recovery scenario (True = enabled or False = disabled), according to the specified index position in the list of recovery scenarios associated with the test.
Ex: - Recovery.GetScenarioStatus Position

SetScenarioStatus Method
Enables or disables the specified recovery scenario, according to its index position in the list of recovery scenarios associated with the test.
Ex: - Recovery.SetScenarioStatus Position, Status

Associated Properties

Count Property
Returns the number of recovery scenarios associated with the current test.
Ex: - msgbox Recovery. Count

Enabled Property
Recovery default property. Retrieves or sets the status of the recovery scenario mechanism for the current test.
Ex: - Recovery. Enabled =Status

Sample Script

```
For Iter = 1 to Recovery. Count
Recovery.GetScenarioName Iter, ScenarioFile, ScenarioName
Position = Recovery.GetScenarioPosition (ScenarioFile, ScenarioName)
Status = Recovery.GetScenarioStatus (Position)
Scenario=scenario& ScenarioFile&":="& ScenarioName&","&position&", Status
Next
Msgbox Scenario
```

This code will show total scenarios in the QRS file, position and status of those scenarios.

Regular Expressions

Introduction

You have created a document and saved it to your hard disk. After few days again you want to update that document, but you forgot where you saved. Now you started searching for that document. Do you go to every folder in hard disk to search for it? … No. You will just use search window to search the document by using name. Unfortunately you didn't find any document on that name. So what will you do?

Here exactly the concept of **Regular Expression** will come in to the picture.

A Regular Expression is a string that provides a complex search phrase.

If you create a word document then you will search for *.doc. Here the * indicates any name which are there in specified disk.

As per the definition, **"*"** is a regular expression which provides a phrase to match any name of the document.

Phrase is an expression consisting of one or more words.

I have used above concept to tell you that "Regular Expressions are not new to us (Testers)". Some how we used it in regular activities but we don't know that these are Regular Expressions.

Use of Regular Expressions in Scripting

- Test for a pattern within a string.
  - *To check for existence of substring in a string.* For example, you can test an input string to see if a telephone number pattern or a credit card number pattern occurs within the string. This is called data validation.

- Replace text.
  - *To find and replace a string with another string.* You can use a regular expression to identify specific text in a document and either remove it completely or replace it with other text.

- Extract a substring from a string based upon a pattern match.

- *To get a string based on pattern match.* You want all the words starting with “A” from a document, In this case you will use regular expression which will create pattern match and will return all words starting with “A”.

Regular Expression Characters

The below table contains the complete list of regular expression characters and behavior of them.

Character	Description
****	Marks the next character as either a special character or a literal. For example, "n" matches the character "n". "\n" matches a newline character. The sequence "\\" matches "\" and "\(" matches "(".
^	Matches the beginning of input.
$	Matches the end of input.
*****	Matches the preceding character zero or more times. For example, "zo*" matches either "z" or "zoo".
+	Matches the preceding character one or more times. For example, "zo+" matches "zoo" but not "z".
?	Matches the preceding character zero or one time. For example, "a?ve?" matches the "ve" in "never".
.	Matches any single character except a newline character.
(pattern**)**	Matches *pattern* and remembers the match. The matched substring can be retrieved from the resulting **Matches** collection, using Item **[0]...[n]**. To match parentheses characters ( ), use "\(" or "\)".
x**|**y	Matches either *x* or *y*. For example, "z|wood" matches "z" or "wood". "(z|w)oo" matches "zoo" or "wood".
{*n*}	*n* is a nonnegative integer. Matches exactly *n* times. For example, "o{2}" does not match the "o" in "Bob," but matches the first two o's in "foooood".
{*n*,}	*n* is a nonnegative integer. Matches at least *n* times. For example, "o{2,}" does not match the "o" in "Bob" and matches all the o's in "foooood." "o{1,}" is equivalent to "o+". "o{0,}" is equivalent to "o*".
{*n***,***m***}**	*m* and *n* are nonnegative integers. Matches at least *n* and at most *m* times. For example, "o{1,3}" matches the first three o's in "fooooood." "o{0,1}" is equivalent to "o?".
[*xyz***]**	A character set. Matches any one of the enclosed characters. For example, "[abc]" matches the "a" in "plain".
[^*xyz***]**	A negative character set. Matches any character not enclosed. For example, "[^abc]" matches the "p" in "plain".
[*a-z***]**	A range of characters. Matches any character in the specified range. For example, "[a-z]" matches any lowercase alphabetic character in the range "a" through "z".

[^*m-z***]**	A negative range characters. Matches any character not in the specified range. For example, "[m-z]" matches any character not in the range "m" through "z".
\b	Matches a word boundary, that is, the position between a word and a space. For example, "er\b" matches the "er" in "never" but not the "er" in "verb".
\B	Matches a non-word boundary. "ea*r\B" matches the "ear" in "never early".
\d	Matches a digit character. Equivalent to [0-9].
\D	Matches a non-digit character. Equivalent to [^0-9].
\f	Matches a form-feed character.
\n	Matches a newline character.
\r	Matches a carriage return character.
\s	Matches any white space including space, tab, form-feed, etc. Equivalent to "[\f\n\r\t\v]".
\S	Matches any nonwhite space character. Equivalent to "[^ \f\n\r\t\v]".
\t	Matches a tab character.
\v	Matches a vertical tab character.
\w	Matches any word character including underscore. Equivalent to "[A-Za-z0-9_]".
\W	Matches any non-word character. Equivalent to "[^A-Za-z0-9_]".
*****num	Matches *num*, where *num* is a positive integer. A reference back to remembered matches. For example, "(.)\1" matches two consecutive identical characters.
*****n	Matches *n*, where *n* is an octal escape value. Octal escape values must be 1, 2, or 3 digits long. For example, "\11" and "\011" both match a tab character. "\0011" is the equivalent of "\001" & "1". Octal escape values must not exceed 256. If they do, only the first two digits comprise the expression. Allows ASCII codes to be used in regular expressions.
\x*n*	Matches *n*, where *n* is a hexadecimal escape value. Hexadecimal escape values must be exactly two digits long. For example, "\x41" matches "A". "\x041" is equivalent to "\x04" & "1". Allows ASCII codes to be used in regular expressions.

We can extend a regular expression by combining or grouping multiple regular expression operators. In this case we should follow the order of precedence.

Order of Precedence

Regular expressions are interpreted from left to right. The order of precedence when building a Regular Expressions is

Order	Operator(s)	Description
1	\	Escape
2	(), (?:), (?=), []	Parentheses and Brackets
3	*, +, ?, {n}, {n,}, {n,m}	Quantifiers

4	^, $, *anymetacharacter*	Anchors and Sequences
5	|	Alternation

Escape (\)

There are so many special characters in regular expressions. I have to verify “2*2=4” is available in the main text. For that I have to specify regular expression pattern as “2*2=4”. But “*” will work like a regular expression and the verification will get fail. In this case the “*” should be considered as a literal character instead regular expression.

Back Slash (\) character is useful to treat a special character as a literal character. Provide the Back Slash (\) character in precede of special characters which you want to treat as literal character.

In the above situation we should use “2*2=4” in the pattern.

List of Special Characters in Regular Expressions

“ **$** ”, “ **(** ” , “ **)** ” , “ ***** ”, “ **+** ” , “ **.** ”, “ **[** ” , “ **]** ”, “ **?** ”, “ **** ”, “ **^** ”, “ **{** ”, “ **|** ”

Parentheses (())

Parentheses used to group the matches.

Brackets ([])

You can create a list of matching characters by placing one or more individual characters within square brackets (**[]**). When characters are enclosed in brackets, the list is called a bracket expression. Within brackets, as anywhere else, ordinary characters represent themselves, that is, they match an occurrence of themselves in the input text. Most special characters lose their meaning when they occur inside a bracket expression.

Parentheses and **Brackets** will be explained detailed in **Alternation**.

Quantifiers

Quantifiers are used to specify the number of occurrences to match against or when we don’t have the quantity of the characters are there to match.

Ex:

If we need to match a word “Zoooo” then we should write regular expression like Zo{4}. 4 indicate the number of o’s in the word “Zoooo”.

Suppose we don’t know how many times “**o**” exist in the word, but we expect at least two **o**’s should available in the word. Then the regular expression will be like this **Zo{2,}**

Here **{2,}** tells that at least two times the character should exist.

List of Quantifiers

“ *****”, “ **+** ” , “ **?** ”, “ **{n}** ”, “ **{n,}** ”, “ **{n,m}** ”

Anchors

Anchors do not match any characters. They match a position. These are used to specify which part of the string should be matched. The part is either beginning or end of a line or word.

Ex:

If we are verifying the word “QTP” is starting with **Q** or not then we use regular expression like “**^Q**”.
Here carot **(^)** is not matching the character “Q” but it is matching the position of “Q”.
That’s what **Anchors** do.

List of Anchors

“ ^ ”, “ $ ”, “ \b ”, “ \B ”

Alternation (|)

Alternation allows us to use a choice between two or more matches. It can be used to match a single regular expression out of several possible regular expressions.

Ex:

The below Regular Expression is to match a Date.

Format: MM/DD/YYYY

MM: (0[1-9]|1[0-2])

Min month number is 1 and Max Month number is 12

DD: (0[1-9]|1[0-9]|2[0-9]|3[0-1])

Min Date number is 1 and Max Month number is 31

YYYY: ([0-9][0-9][0-9][1-9]|[1-9]000|[1-9][1-9]00|[1-9][1-9][1-9]0)

Min Year number is 1 and Max Year number is 9999 (Assume)

In the above regular expression we have used **Parentheses**, **Brackets** and **Alternation**.

Brackets used to match values between the specified ranges. **0[1-9]** means, this expression should match numbers from **01** to **09.**

Alternation used to match a single regular expression from the specified regular expression matches. **0[1-9]|1[0-2]** means, use any one of the regular expression to match.

Parentheses used to group all regular expression matches. **(0[1-9]|1[0-2])** means, use any one of the regular expression to match from this Group.

Scripting Regular Expressions

From VBScript 5.0 Microsoft provided facility to use Regular Expressions in Scripting Techniques.

By using this we can write scripts to Test for a pattern within a string, to replace text and to extract a substring from a string based upon a pattern match.

Using Regular Expressions in Scripting Techniques

1. To use Regular Expressions in scripting first we should create Instance of Regular Expression Class.

   Set SampleRegExP = New RegExp

2. Set the Search Pattern (Regular Expression)

   SampleRegExP.Pattern= “H.*”

3. Specify the Case Sensitivity is true or false

   SampleRegExP.IgnoreCase= False

4. Specify required search results (True is for all search Results, False is for only one)

   SampleRegExP.Global=True

5. Execute Search conditions on a main string

   Set Matches = SampleRegExP.Execute(“Hi How Are You”)

6. Get the results by using a For Loop

   For Each Match in Matches

   Msgbox Match.FirstIndex

```
        Msgbox Match.Value

    Next
```

‘Script to extract a substring from a string based upon a pattern match.

```
‘*********************************************

    rExpression="H."
    MainString="Hi How Are You"

    Set SampleRegExP = New RegExp
    SampleRegExP.Pattern= rExpression
    SampleRegExP.IgnoreCase= False
    SampleRegExP.Global=True

    Set Matches = SampleRegExP.Execute(MainString)

    For Each Match in Matches

        Msgbox Match.FirstIndex
        Msgbox Match.Value

    Next
```

‘Script to Replace string

```
‘*********************************************

    rExpression="H."
    MainString="Hi How Are You"
    ReplacedString= "Hello"

    Set SampleRegExP = New RegExp
    SampleRegExP.Pattern= rExpression
    SampleRegExP.IgnoreCase= False
    SampleRegExP.Global=True

    Msgbox SampleRegExP.Replace (MainString,ReplacedString)
```

```
'***********************************************
'Script to Test a string existence
'***********************************************

        rExpression="H."
        MainString="Hi How Are You"

        Set SampleRegExP = New RegExp
        SampleRegExP.Pattern= rExpression
        SampleRegExP.IgnoreCase= False
        SampleRegExP.Global=True

        retVal = SampleRegExP.Test(MainString)
          If retVal Then
            Msgbox  "One or more matches were found."
          Else
            Msgbox "No match was found."
          End If

'***********************************************
```

Synchronization

When you run tests, your application may not always respond with the same speed. For example, it might take a few seconds:

- For a progress bar to reach 100%
- For a status message to appear
- For a button to become enabled
- For a window or pop-up message to open

You can handle these anticipated timing problems by synchronizing your test to ensure that QuickTest waits until your application is ready before performing a certain step.

There are several options that you can use to synchronize your test:

- You can insert a *synchronization point*, which instructs QuickTest to pause the test until an object property achieves the value you specify. When you insert a synchronization point into your test, QuickTest generates a **WaitProperty** statement in the Expert View.
- You can insert **Exist** or **Wait** statements that instruct QuickTest to wait until an object exists or to wait a specified amount of time before continuing the test.
- You can also increase the default timeout settings in the Test Settings and Options dialog boxes in order

Inserting synchronization point

First find where you want to insert synchronization point and click on record insert synchronization point a select the object a select the property and value an OK.

Syntax: -

Object.waitproperty "property name", "property value", time out

Ex: If you're inserting a synchronization point on insert order in flight reservation application then the statement will be like this.

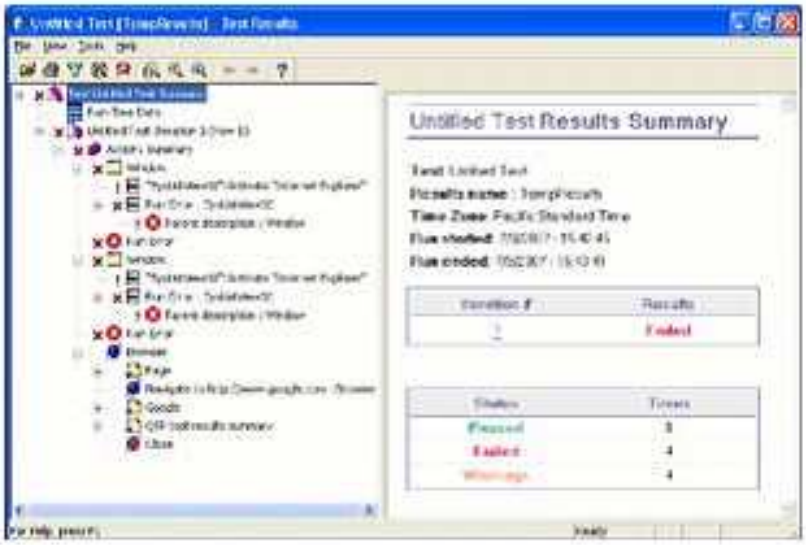
Window ("Flight Reservation").Winbutton ("Insert Order").WaitProperty "enabled", 1, 10000

Difference between wait and synchronization point

Wait (20) a waits for 20 seconds. It's mandatory to wait for 20 seconds.

At same place if you're giving synchronization point and mention 20 seconds to wait and that wait is not mandatory. When ever the given condition becomes true below 20 seconds then QTP immediately goes to the next step without wait for 20 seconds.

```
'*************************************************************************

'Writing own synchronization Point

oTimeout=100

For oTime=1 to oTimeout

	oPropval=window ("Flight Reservation").WinButton ("Delete Order").GetROProperty ("enabled")

	If oPropval=true Then

		Exit for

	else

		Wait (1)

	End If

Next

'*************************************************************************
```



Test Results

Test Results provides a summary of a Quicktest run session. The results are stored in an XML file
format. QTP provides a Test Results viewer tool used to review and print these results. Test Results
Optionally contain image snapshots, based on QTP configuration settings. Below Figure shows a typical Test result summary

The left hand pane provides summary information about each step performed during the test run. Icons to the left of each step provide the following information as well:

- A Cross (X) icon denotes a failed step
- An Exclamation (!) icon denotes a warning step
- A Tick (□) icon denotes a passed step
- A step without any of these symbols denotes an information step

QTP assigns status to a step based on one of the follow situations:

- Checkpoint: Checkpoint can cause a step to pass or fail
- Smart Identification: If Smart identification is used to identify an object then that step is assigned the warning icon
- Error: If a step encounters any error, it will be assigned the failed icon
- Custom events: Custom events are used to directly assign a step an explicit status

Filtering Steps in a Report

It is possible to control what types of steps are written to the test results using the following Statement:

Reporter. Filter = <Filter Value>

The <Filter Value> must use one of the following QTP built-in variables:

- rfEnableAll – Report all steps. This is the default setting
- rfEnableErrorsAndWarnings – Only report error (failed) and warning steps
- rfEnableErrorsOnly – Only report error steps
- rfDisableAll – Does not report any steps
-

The following code shows how to suppress a single checkpoint's pass/fail status:

```
'Store the old filter value
oldFilter = Reporter.Filter
'Disable reporting of all events
Reporter.Filter = rfDisableAll
Set oPg = Browser ("Browser").Page ("Page")
```

```
chkStatus = oPg.WebEdit (“username”).Check (Checkpoint (“username”))
If chkStatus Then
MsgBox “Passed”
Else
MsgBox “Failed”
End If
‘Restrore the old filter value
Reporter.Filter = oldFilter
```

Reporting Custom Steps

We can insert our own steps in the Test Results using the following statement:

```
Reporter.ReportEvent <EventStatus>, <ReportStepName>, <Details>
```

The <EventStatus> should use one of the following QTP built-in variables:

- micPass – Reports a step with passed status
- micFail – Reports a step with failed status
- micWarning – Reports a step with warning status
- micDone – Reports a step with no status

```
‘Get the actual link href
actualLink = Browser (“Browser”).Page (“Page”).Link(“Login”).
GetROProperty (“href”)

If actualLink = “http://mywebsite.com/login.do” Then
Reporter.ReportEvent micPass, “Validate Link - Login”, “Correct Link”
Else
Reporter.ReportEvent micFail, “Validate Login”, “Wrong Link - “& actualLink
End if
```

While the <ReportStepName> and <Details> parameters are plain text strings, it is possible to embed

HTML tags into these strings as follows:

```
‘HTML text to be entered
sHTML = “&lt;<A target=_New href=””http://www.mywebsite.com””>Click
Me</A>&gt;”

‘Add to reporter
Reporter.ReportEvent micDone, “Link”, sHTML
```

QTP also supports one more undocumented EventStatus, micInfo. Using micInfo creates a step with an “i” icon for the step. This is useful to report just information in the report, which we may want to visually segregate from the similar micDone entries.

Inserting Files in Test Results

Consider the following code:

```
'Create the html file path
'Store it in a Test Results folder
sFile = Reporter.ReportPath & "\StepsToRecreate.html"

'Create the HTML file
Set FSO = CreateObject ("Scripting.FileSystemObject")
Set file = FSO.CreateTextFile (sFile, True)
file.Write "<I><B>Step 1</B></I>. Launch www.mywebsite.com"
file.Write "<P><B>Step 2</B>. Click on the Login link"
file.Write "<P><B>Step 3</B>. Enter the username kanak"
file.Write "<P><B>Step 4</B>. Enter password as kanak"
file.Write "<P><B>Step 5</B>. Click on the Login button"
file.Write "<P><B><I>Step 6</I></B>. Close the browser"
file.close

'Insert the above file as a IFRAME in the report
sHTML = "&lt;<IFRAME width=""100%"" height=250 src=""file:///" &
sFile & ""><"IFRAME>&gt;"
Reporter.ReportEvent micPass, "Steps To Recreate", sHTML

'Clean up
Set file = Nothing
Set FSO = Nothing
```

Running the above code will display the file as shown below:

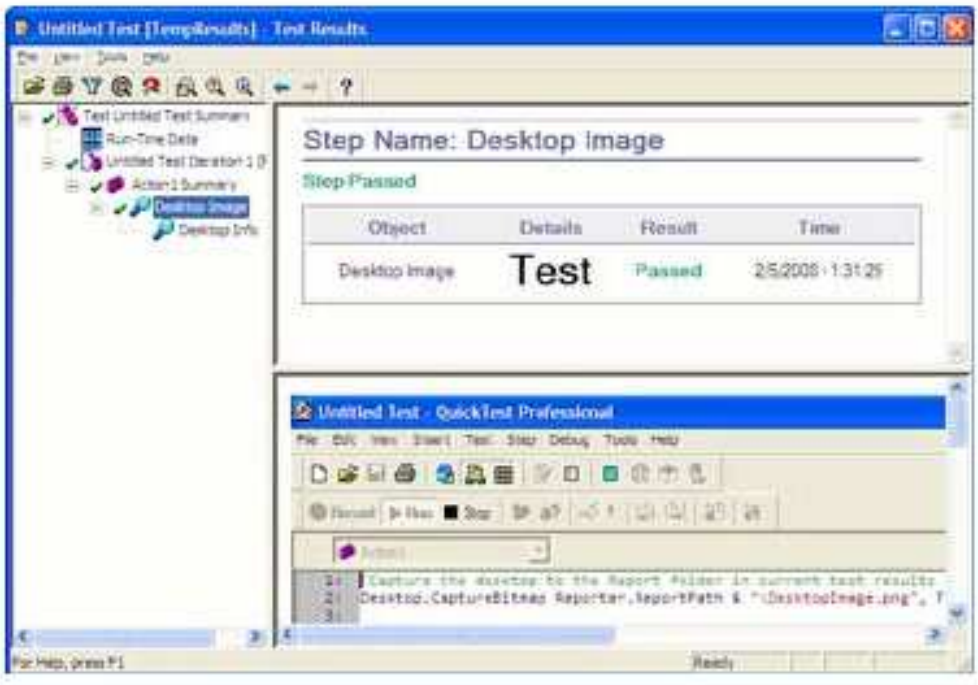
Inserting Snapshots in Test Results

This section describes various ways of inserting screen snapshots into the Test Results.
Method 1
Configure QTP to save a screen snapshot for every step. Go to *Tools→Options…→Run (Tab)* and set the option for *"Save step screen capture to results:"* **to** *"Always"* as shown in the Figure

Test Results >> Inserting Snapshots in Test Results

These techniques have the following advantages over the ReportEvent method:

- Support for relative paths
- Inserts HTML step information without having to use "<" and ">" etc. HTML tags.



Accessing Test Results at the end

QTP creates the results in the test results folder. We may want to access these results at the end of the test script to save them to another location or to send them through email. Let's say we want to copy the Results.xml file which is created in the Report folder. We can write the following code to perform this task in a QTP script:

```
'Report a pass event
Reporter.ReportEvent micPass, "Testing Report", "Testing Exporting of Report"

'Get the result directory
sResultDir = Environment ("ResultDir")

'Copy the file to destination
Set fso = CreateObject ("Scripting.FileSystemObject")
fso.CopyFile sResultDir & "\Report\Results.xml", "C:\Copy_Results.xml", True
Set fso = Nothing
```

QTP Automation Object Model

What is the QuickTest Automation Object Model?

An object model is a structural representation of software objects (classes) that comprise the implementation of a system or application. An object model defines a set of classes and interfaces, together with their properties, methods and events, and their relationships.

Essentially all configuration and run functionality provided via the QuickTest interface is in some way represented in the QuickTest automation object model via objects, methods, and properties. Although a one-on-one comparison cannot always be made, most dialog boxes in QuickTest have a corresponding automation object, most options in dialog boxes can be set and/or retrieved using the corresponding object property, and most menu commands and other operations have corresponding automation methods.

You can use the objects, methods, and properties exposed by the QuickTest automation object model, along with standard programming elements such as loops and conditional statements to design your script.

Automation scripts are especially useful for performing the same tasks multiple times or on multiple tests or components, or quickly configuring QuickTest according to your needs for a particular environment or application.

The QuickTest automation object model exposes the objects shown in the diagram below. You can use these objects, and their associated methods and properties, to write programs that automatically configure QuickTest options and run tests.

Some places where we can use AOM

This is a small list of places (but not limited to) where we can use AOM. Thumb Rule - Use it at any place where you find yourself doing repetitive tasks while using QTP.

- AOM can come handy when you have a large no of scripts to be uploaded to QC. A simple script can save you hours of manual work!
- Use AOM to initialize QTP options and settings like add-ins etc.
- You can use AOM to call QTP from other application: For ex: You can write a macro for calling QTP from excel.

Caution: *AOM should be used outside of QTP and not within the script (during playback). Though there is no harm using it inside but some of the AOM statements might fail.*

Creating automation programs:

The Properties tab of the Test Settings dialog box, the General tab of the Options dialog box, and the Object Identification dialog box each contain a “Generate Script” button. Clicking this button generates a automation script file (.vbs) containing the current settings from the corresponding dialog box.

You can run the generated script as is to open QuickTest with the exact configuration of the QuickTest application that generated the script, or you can copy and paste selected lines from the generated files into your own automation script.

Generating an automation script for QuickTest Professional options:

1. Go to Tools -> Options.
2. Select the General tab.
3. Click <Generate Script>.
4. Save the script to the desired location.
5. Click <OK> to close the Options dialog.

Generating an automation script for test settings:

1. Go to Test -> Settings.
2. Select the Properties tab.
3. Click <Generate Script>.
4. Save the script to the desired location.
5. Click <OK> to close the Test Settings dialog.

Generating an automation script for object identification settings:

1. Go to Tools -> Object Identification.
2. Click <Generate Script>.
3. Save the script to the desired location.
4. Click <OK> to close the Object Identification dialog.

The QuickTest Automation Object Model Reference file is a help file that provides detailed descriptions, syntax information, and examples for the objects, methods, and properties in the QuickTest Automation Object Model.

How to write AOM scripts?

You need to understand that the very root of QT AOM is Application Object. Every automation script begins with the creation of the QuickTest "**Application"** object. Creating this object does not start QuickTest. It simply provides an object from which you can access all other objects, methods and properties of the QuickTest automation object model. You can create only one instance of the Application object. You do not need to recreate the QuickTest Application object even if you start and exit QuickTest several times during your script.

Once you have defined this object you can then successfully work and perform operations on other objects given in Quick Test Pro > Documentation > QuickTest Automation Reference.

For ex: Let us connect to TD QC using AOM and open a script "qtp_demo"

```
Dim qt_obj 'Define a Quick Test object
qt_obj = CreateObject ("Quick Test. Application") ' Instantiate a QT Object. It does not start QTP.
qt_obj.launch ' Launch QT
qt_obj.visible ' Make QT visible
qt_obj.TDConnection.Connect "http://tdserver/tdbin", _ 'Referencing TDConnection Object
"TEST_DOMAIN", "TEST_Project", "kanak", "Testing", False ' Connect to Quality Center
If qt_obj.TDConnection.IsConnected Then ' If connection is successful
   qt_obj.Open "[QualityCenter] Subject\tests\qtp_demo", False ' Open the test
Else
   MsgBox "Cannot connect to Quality Center" ' If connection is not successful, display an error message.
End If
```

To quickly generate an AOM script with the current QTP settings. Use the Properties tab of the Test Settings dialog box (File > Settings) OR the General tab of the Options dialog box (Tools > Options) OR the Object Identification dialog box (Tools > Object Identification). Each contains a "Generate Script" button. Clicking this button generates a automation script file (.vbs) containing the current settings from the corresponding dialog box. You can run the generated script as is to open QuickTest with the exact configuration of the QuickTest application that generated the script, or you can copy and paste selected lines from the generated files into your own automation script.

AOM Examples

```
'*******************************************************************************
'Open QTP and Connect to Quality Center
'*******************************************************************************
Dim qtApp ' Declare the Application object variable
Set qtApp = CreateObject("QuickTest.Application") ' Create the Application object
qtApp.Launch ' Start QuickTest
qtApp.Visible = True ' Make the QuickTest application visible
' Make changes in a test on Quality Center with version control
qtApp.TDConnection.Connect "QC URL", "DOMAIN Name", "Project Name", "User Name", "Password", False ' Connect to Quality Center
```

```
If qtApp.TDConnection.IsConnected Then ' If connection is successful
 MsgBox "Succesfully connected to Quality Center"
Else
   MsgBox "Cannot connect to Quality Center"
End If

qtApp.Quit ' Exit QuickTest
Set qtApp = Nothing ' Release the Application object


'*******************************************************************************
'Start  QTP and open New test
'*******************************************************************************
Dim qtApp ' Declare the application object variable
Set qtApp = CreateObject("QuickTest.Application") ' Create the application object
qtApp.Launch ' Start QuickTest
qtApp.Visible = True ' Make the QuickTest application visible
qtApp.New ' Open a new test
Set qtApp = Nothing ' Release the Application object
'*******************************************************************************
'Start  QTP, open an existing test and Run the Test
'*******************************************************************************

Dim qtApp
Dim qtTest
Set qtApp = CreateObject("QuickTest.Application") ' Create the Application object
qtApp.Launch ' Start QuickTest
qtApp.Visible = True ' Make the QuickTest application visible
' Set QuickTest run options
qtApp.Options.Run.ImageCaptureForTestResults = "OnError"
qtApp.Options.Run.RunMode = "Fast"
qtApp.Options.Run.ViewResults = False
qtApp.Open "C:\Tests\Test1", True ' Open the test in read-only mode
' set run settings for the test
Set qtTest = qtApp.Test
qtTest.Settings.Run.OnError = "NextStep" ' Instruct QuickTest to perform next step when
error occurs
qtTest.Run ' Run the test
MsgBox qtTest.LastRunResults.Status ' Check the results of the test run
qtTest.Close ' Close the test
Set qtTest = Nothing ' Release the Test object
Set qtApp = Nothing ' Release the Application object

'*******************************************************************************
'Start  QTP, open an existing test and Run the Test with configured Run options
' And Store Run Results in Specified Folder

'*******************************************************************************
Dim qtApp
```

```
Dim qtTest
Dim qtResultsOpt
Set qtApp = CreateObject("QuickTest.Application") ' Create the Application object
qtApp.Launch ' Start QuickTest
qtApp.Visible = True ' Make the QuickTest application visible
' Set QuickTest run options
qtApp.Options.Run.ImageCaptureForTestResults = "OnError"
qtApp.Options.Run.RunMode = "Fast"
qtApp.Options.Run.ViewResults = False
qtApp.Open "C:\Tests\Test1", True ' Open the test in read-only mode
' set run settings for the test
Set qtTest = qtApp.Test
qtTest.Settings.Run.IterationMode = "rngIterations" ' Run only iterations 2 to 4
qtTest.Settings.Run.StartIteration = 2
qtTest.Settings.Run.EndIteration = 4
qtTest.Settings.Run.OnError = "NextStep" ' Instruct QuickTest to perform next step when
error occurs
Set qtResultsOpt = CreateObject("QuickTest.RunResultsOptions") ' Create the Run
Results Options object
qtResultsOpt.ResultsLocation = "C:\Tests\Test1\Res1" ' Set the results location
qtTest.Run qtResultsOpt ' Run the test
MsgBox qtTest.LastRunResults.Status ' Check the results of the test run
qtTest.Close ' Close the test
Set qtResultsOpt = Nothing ' Release the Run Results Options object
Set qtTest = Nothing ' Release the Test object
Set qtApp = Nothing ' Release the Application object
'********************************************************************************
```

'Start QTP, open an existing test, associate libraries and save the test

```
'********************************************************************************

Dim qtApp
Dim qtLibraries
Dim lngPosition
' Open QuickTest
Set qtApp = CreateObject("QuickTest.Application") ' Create the Application object
qtApp.Launch ' Launch QuickTest
qtApp.Visible = True ' Set QuickTest to be visible
' Open a test and get its libraries collection
qtApp.Open "C:\Tests\Test1", False, False ' Open a test
Set qtLibraries = qtApp.Test.Settings.Resources.Libraries ' Get the libraries collection
object
' Add Utilities.vbs if it's not in the collection
If qtLibraries.Find("C:\Utilities.vbs") = -1 Then ' If the library cannot be found in the
collection
    qtLibraries.Add "C:\Utilities.vbs", 1 ' Add the library to the collection
End If
'Save the test and close QuickTest
qtApp.Test.Save ' Save the test
qtApp.Quit ' Quit QuickTest
```

```
Set qtLibraries = Nothing ' Release the test's libraries collection
Set qtApp = Nothing ' Release the Application object
'*****************************************************************************
```

'Start QTP, open an existing test, associate Object Repositories and save the test

```
'*****************************************************************************

Dim qtApp
Dim qtRepositories
Dim lngPosition
' Open QuickTest
Set qtApp = CreateObject("QuickTest.Application") ' Create the Application object
qtApp.Launch ' Launch QuickTest
qtApp.Visible = True ' Set QuickTest to be visible
' Open a test and get the "Login" action's object repositories collection
qtApp.Open "C:\Tests\Test1", False, False ' Open a test
Set qtRepositories = qtApp.Test.Actions("Login").ObjectRepositories ' Get the object
repositories collection object of the "Login" action
' Add MainApp.tsr if it's not already in the collection
If qtRepositories.Find("C:\MainApp.tsr") = -1 Then ' If the repository cannot be found in the
collection
    qtRepositories.Add "C:\MainApp.tsr", 1 ' Add the repository to the collection
End If
'Save the test and close QuickTest
qtApp.Test.Save ' Save the test
qtApp.Quit ' Quit QuickTest
Set qtRepositories = Nothing ' Release the action's shared repositories collection
Set qtApp = Nothing ' Release the Application object
'*****************************************************************************
```

'Open and minimize QTP Window

```
'*****************************************************************************

Dim qtApp
Set qtApp = CreateObject("QuickTest.Application") ' Create the Application object
qtApp.Launch ' Start QuickTest
qtApp.Visible = True ' Make the QuickTest window visible
qtApp.WindowState = "Minimized" ' Maximize the QuickTest window
Set qtApp = Nothing ' Release the Application object
'*****************************************************************************
```

'Start QTP, Open an Existing Test and Define Environment Variables

```
'*****************************************************************************

Dim qtApp
```

```
Set qtApp = CreateObject("QuickTest.Application") ' Create the Application object
qtApp.Launch ' Start QuickTest
qtApp.Visible = True ' Make the QuickTest application visible
' Open the test
qtApp.Open "C:\Tests\Test1", False ' Open a test named "Test1"
' Set some environment variables
qtApp.Test.Environment.Value("Root") = "C:\"
qtApp.Test.Environment.Value("Password") = "QuickTest"
qtApp.Test.Environment.Value("Days") = 14
qtApp.Test.Save ' Save the test
qtApp.Quit ' Exit QuickTest
Set qtApp = Nothing ' Release the Application object
'*****************************************************************************
```

'Start QTP, Open an Existing Test and Get All Available Action Names From the Test

```
'*****************************************************************************

Dim qtApp
' Open QuickTest
Set qtApp = CreateObject("QuickTest.Application") ' Create the Application object
qtApp.Launch ' Launch QuickTest
qtApp.Visible = True ' Set QuickTest to be visible
qtApp.Open "C:\Tests\Test1", False, False ' Open a test
oActCount=qtApp.Test.Actions.Count
For iCounter=1 to oActCount
 ' Get the first action in the test by index (start from 1)
 MsgBox qtApp.Test.Actions(iCounter).Name
Next
'Close QuickTest
qtApp.Quit ' Quit QuickTest
Set qtApp = Nothing ' Release the Application object
'*****************************************************************************
```

'Start QTP with specified views

```
'*****************************************************************************
Dim qtApp
Set qtApp = CreateObject("QuickTest.Application") ' Create the Application object
qtApp.Launch ' Start QuickTest
qtApp.ActivateView "ExpertView" ' Display the Expert View
qtApp.ShowPaneScreen "ActiveScreen", True ' Display the Active Screen pane
qtApp.ShowPaneScreen "DataTable", False ' Hide the Data Table pane
qtApp.ShowPaneScreen "DebugViewer", True ' Display the Debug Viewer pane
qtApp.WindowState = "Maximized" ' Maximize the QuickTest window
qtApp.Visible = True ' Make the QuickTest window visible
Set qtApp = Nothing ' Release the Application object

'*****************************************************************************
```

Working with Microsoft Excel

Introduction

We create framework for automating the application. For this we used the independent structure for reporting and data. Excel plays a very important role in this approach.

QTP has its own test result displaying mechanism in the predefined format. Once the test is run, the result sheet is generated which gives you insight of the script-stating the point of failures, warnings and passes

We create customized in the script and it is possible to customize the result file also depending upon the checkpoint created will be passed or failed.

In most of the cases we want to create summarized or detailed report of the entire test in excels. The reason to create customized report is that one is able to keep the file in central location and to create the report in our format

Now we are going to learn the interaction of Excel with VBScript.the whole mechanism goes in the following steps.

1. Understanding the hierarchy of Excel Application
2. Creating the Excel Object
3. Opening an existing workbook or creating new one
4. Setting the objects for various sheets in workbook
5. Writing and fetching the data values in the result
6. Saving the and closing the workbook
7. Closing the application and releasing the memory

We will go through each of the above stated steps with a suitable example to understand the approach properly

Understanding the hierarchy of Excel Application

We will not go into the details of the complete hierarchy of the Excel Application but to the extend what is required

Excel Application
  Workbooks
    Sheets
      Cells

Creating the Excel Object

The first step onwards the processes of reporting via excel is to create object of Excel. Reporting in Excel can either be done in backend, without making the application visible or you can make it appear to user once the process of writing or fetching the data is going. In either way creating of the Excel Application object is required. it goes as

```
Dim XL
Set XL=CreateObject ("Excel. Application")
```

Opening an existing workbook or creating the new one

Once the excel object has been created, it means the excel application has been invoked but is not visible. So either one can perform the operation like that or make the application visible and then perform the operations.

```
'To make the application visible
XL.visible=true

'To open a new book
XL.workbooks.Add

'To open an existing Workbook
XL.workbooks.Open ("file name of the complete path")
```

Setting and accessing the objects of sheets in workbook

Once the workbook has been opened, either existing or new one, we need to write some data in various cells in various sheets of that workbook.

By default there are 3 sheets in a workbook and various operations can be performed on. So one need create the object to reference these sheets as it becomes easy to access them and you don't have to mention the complete hierarchy over and over again.

```
'Say one has to create a reference for sheet with index I, which starts from 1
Set sht1=XL.activeworkbook.sheet (1)

'One can add or delete n number of sheets from the activeworkbook
'To add a sheet in workbook-
XL.activeworkbook.sheets.add

'To delete a particular sheet where I represent the index which starts from 1-
Xl.activeworkbook.sheets (1).delete

'To change the name of the sheet-
XL.activeworkbook.sheets (1).name="Name of your choice"
```

```
'To count the total number of sheets in the workbook
Cnt= XL.activeworkbook.sheets.count
```

Writing and deleting the data value in the cells

To write the data in Excel sheet, one should know the cell address in which the data has to be written. Same thing goes for accessing the data from the cells

To write the data in sheet1 cell as D8, we write the following command.Cell address here is represented by row number followed by column number-

```
XL.activeworkbook.sheets (2).cells (8, 4) ="hemajothi"

'To fetch the data from sheet3 cell address A7-
Val= XL.activeworkbook.sheets (3).cells (7, 1)
Msgbox Val
```

If one has already created the object of the particular sheet, you don't have to write the complete hierarchy but simply-

```
Object. cells (row, col) =value
```

Saving and closing the workbook

Once the work completed you can save the newly created workbook to a specified location or save the changes made to already existing opened workbook.

```
'To save as if case of new workbook
XL.activeworkbook.saveas "path with file name.xls"

'To save in case of existing workbook
XL.activeworkbook.save

'To close the workbook
XL.activeworkbook.close
```

Closing the application and releasing the memory

```
'To close the application
XL.quit

'To release the memory of all the objects
Set XL=nothing
```

Searching word in excel sheet

```
 Set appExcel = CreateObject("Excel.Application")
appExcel.visible=true
 Set objWorkBook = appExcel.Workbooks.Open ("E:\exe.xls")
 'opens the sheet
 Set objSheet = appExcel.Sheets("Sheet1")
 'To select particular sheet
With objSheet.UsedRange
' select the used range in particular sheet
    Set c = .Find ("Denver")
 ' data to find
For each c in objSheet.UsedRange
 ' Loop through the used range
 If c="Denver" then
  ' compare with the expected data
          c.Interior.ColorIndex = 20
   ' make the gary color if it finds the data
End If
          Set c = .FindNext(c)
   ' next search
next
End With
objWorkBook.save
objWorkBook.close
set appExcel=nothing
```

Copy an excel sheet to another excel

```
Set objExcel = CreateObject ("Excel.Application")
objExcel.Visible = True
Set objWorkbook1= objExcel.Workbooks.Open ("E:\exe.xls")
Set objWorkbook2= objExcel.Workbooks.Open("E:\exe2.xls")
objWorkbook1.Worksheets("Sheet1").UsedRange.Copy
objWorkbook2.Worksheets("Sheet1").Range("A1").PasteSpecial Paste =xlValues
objWorkbook1.save
objWorkbook2.save
objWorkbook1.close
objWorkbook2.close
set objExcel=nothing
```

Compare two excel sheet:

```
Set objExcel = CreateObject ("Excel.Application")
objExcel.Visible = True
Set objWorkbook1= objExcel.Workbooks.Open ("E:\exe.xls")
Set objWorkbook2= objExcel.Workbooks.Open ("E:\exe1.xls")

Set objWorksheet1= objWorkbook1.Worksheets (1)

Set objWorksheet2= objWorkbook2.Worksheets (1)
```

```
  For Each cell in objWorksheet1.UsedRange
     If cell. Value <> objWorksheet2.Range (cell. Address).Value Then
        cell.Interior.ColorIndex = 3
   ''Highlights in red color if any changes in cells
     Else
        cell.Interior.ColorIndex = 0
     End If
  Next

Set objExcel=nothing
```

Display result in different excel sheet

```
Dialog ("Login").WinEdit ("Agent Name :") .Set
DataTable ("Agent Name", dtGlobalSheet)
Dialog ("Login").WinEdit ("Password :") .Set
DataTable ("Pword", dtGlobalSheet)
Dialog ("Login").WinButton ("OK").Click

If Window ("Flight Reservation").Exist then
Window ("Flight Reservation").WinMenu ("Menu").Select "File; Exit"
DataTable.Value ("Act_Res") ="Pass"
Reporter.ReportEvent 0,"Login Details", "Valid Values"
Else
Dialog ("Login").Activate
Dialog ("Login").WinButton ("Cancel").Click
DataTable.Value ("Act_Res") ="Fail"
Reporter.ReportEvent 1,"Login Details", "Invalid Values"
End If

Er=DataTable ("Exp_Res", dtGlobalSheet)
Ar=DataTable ("Act_Res", dtGlobalSheet)
If Strcomp(Er, Ar) =0 Then
        DataTable.Value ("Remarks") ="OK"
        Else
        DataTable.Value ("Remarks") ="Defect"
End If
DataTable.Export ("e:\\Results.xls")
```

Working with XML

What is Document object Model?

A platform- and language-independent standard object model for representing HTML or XML and related formats. DOM is a method for QTP engineers to access the source (IE –> View –> Source) of any webpage direct through VB Scripting.

When can we use DOM?

One of the very important places you can use it is when a QTP web table checkpoint doesn't show you the desired content in a cell. I mean the content in the cell is in a format that is not supported by the web table checkpoint. Another use can be when you want to access all the HTML tags used in a webpage. You can get the names, the innertext, and innerHTML property of all these tags. The possibilities are endless.

How can we use DOM to access the source page?

We can access the source page of any webpage using **.object** notation.

Example:

```
Set img = Browser ("Browser").Page ("Page").WebTable ("Happy").Object.all.tags ("img")
Msgbox img.length
For i=0 to img.length-1
Msgbox img (i).src
Next
```

The loading of XML file in QTP is simple enough:

```
Const XMLDataFile = "C:\TestData.xml"
Set xmlDoc = CreateObject ("Microsoft.XMLDOM")
xmlDoc.Async = False
xmlDoc.Load(XMLDataFile)
```

Note:

1. **Microsoft.XMLDOM** is a name of COM object of Microsoft's XML parser
2. **Async** is a property of **Microsoft.XMLDOM.**
   The property specifies whether asynchronous download of the document is permitted.
   Processing of asynchronous operations is more complex, that's why I've disabled it (**xmlDoc.Async = False**).
3. **Load** method loads an XML document from the specified file.
   Also, you can use **LoadXML** method, which loads an XML document using the supplied string.

After that we can use SelectSingleNode or SelectNodes of Microsoft's XML parser to execute **XPath** query. You can use this approach in QTP to get data from XML file.

How to get number of books in a bookstore?

```
Set nodes = xmlDoc.SelectNodes ("/bookstore/book")
MsgBox "Total books: " & nodes. Length
```

How to get titles of all books in a bookstore?

```
‘Get all titles
Set nodes = xmlDoc.SelectNodes ("/bookstore/book/title/text ()")
‘Get their values
for i = 0 to (nodes. Length - 1)
   Title = nodes (i).Node Value
MsgBox "Title #" & (i + 1) & ": " & Title
Next
```

How to get title of the first book?

```
Set node = xmlDoc.SelectSingleNode ("/bookstore/book [0]/title/text ()")
MsgBox "Title of 1st book: " & node.NodeValue
How to get titles of all John Smith's books?
‘Get all titles
Set nodes = xmlDoc.SelectNodes ("/bookstore/book/title/text ()")
' get their values
for i = 0 to (nodes. Length - 1)
   Title = nodes (i).Node Value
   MsgBox "Title #" & (i + 1) & ": " & Title
Next
```

How to get title of the first book?

```
Set node = xmlDoc.SelectSingleNode ("/bookstore/book [0]/title/text ()")
MsgBox "Title of 1st book: " & node.NodeValue
```

How to get titles of all John Smith's books?

```
‘Get list of John Smith's
Set nodes = xmlDoc.SelectNodes ("/bookstore/book/title [. /author = 'John Smith']/text ()")
' get their titles
for i = 0 to (nodes. Length - 1)
   Title = nodes (i).NodeValue
   MsgBox "Title #" & (i + 1) & ": " & Title
Next
```

Note: We use square brackets to apply a filter. So, [.../author = 'John Smith'] means *'to get only those books whose author is John Smith'*.

How to get titles of all books published after 2003?

```
'get list of books published after 2003
Set nodes = xmlDoc.SelectNodes ("/bookstore/book/title [@published > 2003]/text ()")
' get their titles
for i = 0 to (nodes. Length - 1)
    Title = nodes (i).NodeValue
    MsgBox "Title #" & (i + 1) & ": " & Title
Next
```

How to rename the title of first book?

```
Const XMLDataFile = "C:\TestData.xml"
Const XMLNewFile = "C:\TestData2.xml"

Set xmlDoc = CreateObject ("Microsoft.XMLDOM")
xmlDoc.Async = False
xmlDoc.Load(XMLDataFile)

' update the title of the first book
Set node = xmlDoc.SelectSingleNode("/bookstore/book[0]/title")
node. Text = "Romeo and Juliet - Salvation"

' save changes
xmlDoc.Save (XMLNewFile)
```

How to change the year of second book?

```
Update the attribute of the second book
Set node = xmlDoc.SelectSingleNode ("/bookstore/book [1]/title/@published")
node. Text = "2009"
```

How to add new author add its new attribute?

```
Select a parent node
Set parent Node = xmlDoc.SelectSingleNode ("/bookstore/book [2]")

' add a new author
Set newNode = xmlDoc.CreateElement ("author")
newNode.Text = "Mr. Noname"
parentNode.AppendChild (newNode)
```

How to add new attribute for author (XML node)?

```
'select a parent node
Set parentNode = xmlDoc.SelectSingleNode ("/bookstore/book [2]")
' add its attribute
Set newAttrib = xmlDoc.CreateAttribute ("bestseller")
newAttrib.Text = "yes"
parentNode.Attributes.SetNamedItem(newAttrib)
```

How to rename the title of first book?

```
Const XMLDataFile = "C:\TestData.xml"
Const XMLNewFile = "C:\TestData2.xml"
Set xmlDoc = CreateObject ("Microsoft.XMLDOM")
xmlDoc.Async = False
xmlDoc.Load(XMLDataFile)
' update the title of the first book
Set node = xmlDoc.SelectSingleNode("/bookstore/book[0]/title")
node. Text = "Romeo and Juliet - Salvation"
' save changes
xmlDoc.Save (XMLNewFile)
```

Note: The numeration begins from zero. That's why I use book [0] to access first item.

How to change the year of second book?

I skip the opening and saving of XML file (see above QTP script). I show only the essence:

```
‘Update the attribute of the second book
Set node = xmlDoc.SelectSingleNode ("/bookstore/book [1]/title/@published")
node. Text = "2009"
```

Note*:* Use @ to access an attribute of XML node.

How to add new author add its new attribute?

```
‘select a parent node
Set parentNode = xmlDoc.SelectSingleNode ("/bookstore/book [2]")
' add a new author
Set newNode = xmlDoc.CreateElement ("author")
newNode.Text = "Mr. Noname"
parentNode.AppendChild (newNode)
```

How to add new attribute for author (XML node)?

```
‘select a parent node
Set parentNode = xmlDoc.SelectSingleNode ("/bookstore/book [2]")
' add its attribute
Set newAttrib = xmlDoc.CreateAttribute ("bestseller")
newAttrib.Text = "yes"
parentNode.Attributes.SetNamedItem(newAttrib)
```

Designing framework

Frame Work:

Frame work is a Generic work designed by an expert and followed by many people to perform a particular task in an effective, efficient and optimized way.

Test Settings for Keyword-driven Scripting

In the keyword-driven approach the entire script is developed with keywords. The script is developed in a spreadsheet that is interpreted by the main driver script, which then uses the function
Library to execute the complete script.

The QTP Settings.vbs file can be used to easily perform the test settings that are needed before proceeding with the Keyword-driven Scripting. This file associates the Function libraries, recovery scenarios, Environment Variables and the Object Repository files that are needed for a script.

The QTP Settings file needs to be customized before usage. Edit the .vbs file in notepad and make the necessary changes (mentioned below)
in the 'Input Data' section of the file.

1. Function Libraries

Specify the path where the Framework Files, Common Functions and the User Defined Functions are Stored.

2. Recovery Scenario File

Specify the path of the file where the Recovery scenarios are placed.

3. Recovery Scenarios Name

Specify the names of the Recovery scenarios that need to be associated to the test script.

4. Environment File

Specify the path of the xml file for the Global Environment Variables.

5. Object Repository files

Specify the path of the Object Repository

.Open the test script in QTP for which the settings are needed and double click on the QTP Settings.vbs file. This file will perform the preferred settings automatically.

Note

If there are multiple items of libraries, object repositories or global variables file, they can be specified as an array separated by ','

Set the Flag envi_flag, recover_flag, repos_flag, library flag to "Yes" if the corresponding files need to be associated to test, else set it to "no"

Given below is a sample "QTP Settings".

```
libraries= array ("C:\WebFramework.vbs","C:\common functions.vbs")
recovery_file= "C:\recover.qrs"
recovery_name=array ("Scenario1","Scenario2")
```

```
environment_file= "C:\Environment.xml"
repository_name= array ("C:\repository1.tsr","C:\repository2.tsr")

'NOTE: Please set the Flag to "Yes" if the files are to be associated to
Test, otherwise set it to "no"

envi_flag="yes"
recover_flag=" yes"
repos_flag=" yes"
library_flag=" yes"
```

There should be just one Recovery file which holds all the Recovery scenarios. Multiple
Recovery files cannot be used.

There should be no unsaved tests currently open in QTP. The test script for which the test
settings need to be done should be open in QTP.

Run Settings

In the Run tab of the Test Settings dialog,
1. The Run one iteration only radio button will be selected.
2. The Disable Smart Identification during the run session check box will be checked.
3. The Object Synchronization timeout value will be set as 20 seconds.

Resources Settings

After the run settings are completed, the QTP Settings file associates the framework with the test script. Here, the path and the names of the framework files need to be specified in the QTP Settings
file. The framework will be taken from the location specified and associated with the test as shown below.
Figure 2: Associating Framework File

NOTE: The Common functions file and the User defined functions file should be associated with the test only if needed by the test script.

Environment Tab Settings

QTP can insert a value from the environment variable list, which is a list of variables and corresponding values that can be accessed from the test. Throughout the test run, the value of an environment variable remains the same, regardless of the number of iterations, unless the value of the variable is changed programmatically in the script.

Associating the user-defined Environment Variables file with the test is also handled by the QTP Settings file. The Environment Variables file with an .xml file type will be taken from the path specified in the QTP Settings file and associated with the test.

Managing Object Repository

After the test settings are completed, the QTP Settings file continues to associate the specified shared object repositories with the test. The objects from the shared repository will be uploaded and
made available for the tests.

Figure 4: Associate Repositories Dialog

Call to Framework

The call to Keyword_Driver () needs to be specified in the Expert View as shown below. This will call the Framework file associated with the test and perform the actions by interpreting the keywords specified in the data table.

Figure 5: Call to Framework

Usage of Keywords

The keywords should be entered in the global sheet of the data table of the test according to the syntax. To access the data table, choose View > Data Table or click on the toolbar button. Below is an *example of Keyword-driven Scripting*.

Figure 6: Using the keyword in a Data Table

Types of Frame work:-

- Linear / Sequential Frame work
- Modular frame work
- Keyword Driven frame work
- Hybrid frame work

Linear Frame work:

This is a general and old frame work that can be used by many people. Steps to follow ….

a). Generate the Basic Test
b). Enhance the test
c). Debug the test
d). Execute the test
e). Analyze the result

Example:

Tasks: **Login**
Insert order
Open existing order
Logout

Note*:* *Here* all the **tasks** are put together in one test pane *and done the job*

AUT

Login
……
……..
Insert order
……
……..
Open order
……
……
Logout

- ✓ Put the tool under recoding mode
- ✓ Open flight application
- ✓ **Login** with username and password
- ✓ Click on OK
- ✓ **Insert** an order by keying all the required info therein
- ✓ Click on insert order button
- ✓ That order will be inserted successfully. After inserting the order
- ✓ **Open existing order** by clicking on open folder icon
- ✓ A open order window will appear, check the order number check box
- ✓ Input the existing order number (say 9)
- ✓ Click on OK
- ✓ The order will open, if necessary you may update / delete the opened order
- ✓ **Logout** will be done by going to **menu bar** of the application and select **file**, select **exit.**
- ✓ The application will close
- ✓ Stop recording
- ✓ Run the test
- ✓ Analyze the result

Modular Frame work:

This is also a general frame work that can be used by some people.Steps to follow ….

a). Prepare the Individual **Components** for different tasks
b). Make the require Components as **Re-Usable**
c). Prepare the desired **Driver** based on end-to-end scenario
d). Execute the Driver
e). Analyze the results

! A script file can be called as a Driver
! Test : means a script file
! An action: Is a set of instructions to perform a task
! Component can be called as an Action
!! Ways of action:

- Prepare complete action and Split it to each action or
- Prepare a single action and Insert each action into it.

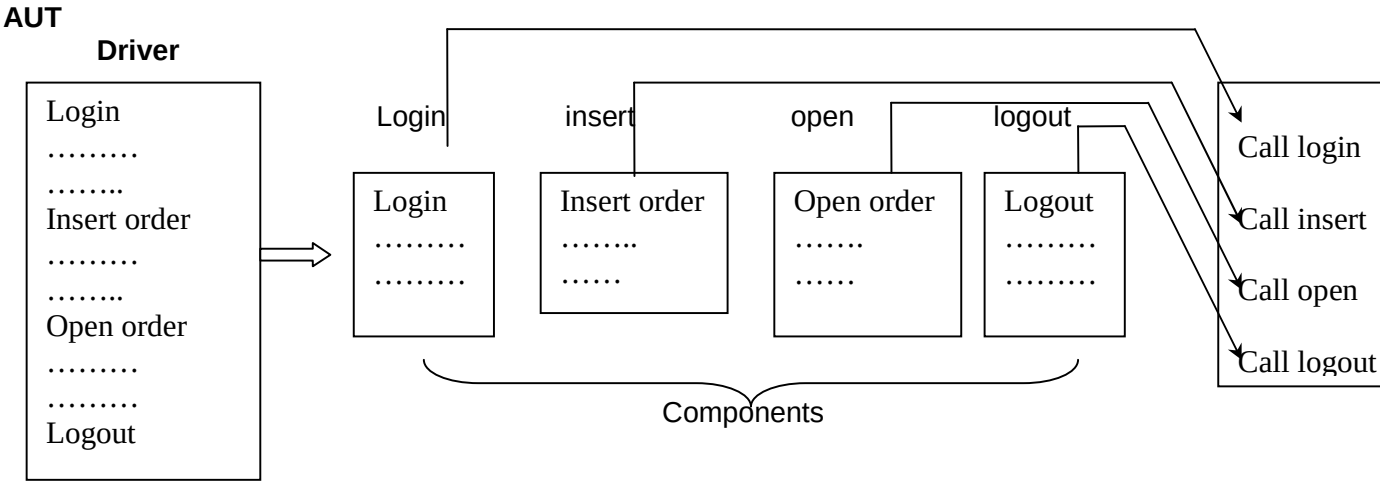
* One action can be spited into two actions a time only and so on.

Example:

Tasks : **Login**
Insert order
Open existing order
Logout

Note 1*:* ***A***fter preparing complete action, split it *into different individual* ***actions*** *and* call them in a driver.

Note 2*:* *Other words in this frame work,* **we are calling the action/s (Ex. Call login)**



- ✓ Put the tool under recoding mode
- ✓ Open flight application
- ✓ **Login** with username and password
- ✓ Click on OK
- ✓ **Insert Order** by keying all the required info therein
- ✓ Click on insert order button
- ✓ That order will be inserted successfully. After inserting the order
- ✓ **Open order** by clicking on open folder icon
- ✓ A open order window will appear, check the order number check box
- ✓ Input the existing order number (say 9)
- ✓ Click on OK
- ✓ The order will open, if necessary you may update / delete the opened order
- ✓ **Logout** will be done by closing the window/application
- ✓ Stop recording
- ✓ **Save the Script** (say fl_application , No extention is required)
- ✓ **Split the script into 4 tasks** (login, insert order, open order and logout) Keep the cursor at the beginning of 1st line of 2nd part (i.e. starting of insert order line)
- ✓ Go to **menu** bar, click on **Step** , select **Split Action**
- ✓ The split action window will appear
- ✓ choose action type as **independent of each other,** give the **1st action name** (say **login**) and **leave the 2nd action name** as it is (because, again we are going to split the 2nd part)
- ✓ Click OK
- ✓ Save the changes. Next,

- ✓ Keep the cursor at the beginning of 1st line of existing 2nd part (i.e. starting of insert order line)
- ✓ Go to **menu** bar, click on **Step**, and select **Split Action**
- ✓ The split action window will appear
- ✓ choose action type as **independent of each other,** give the **1st action name** (say **insert order**) and **leave the 2nd action name** as it is (because, again we are going to split the 2nd part)
- ✓ Click OK
- ✓ Save the changes. Next,

- ✓ Keep the cursor at the beginning of 1st line of existing 2nd part (i.e. starting of open order line)
- ✓ Go to **menu** bar, click on **Step** , select **Split Action**
- ✓ The split action window will appear
- ✓ choose action type as **independent of each other,** give the **1st action name** (say **open_order**) and **give the 2nd action name** as **logout** (because, its end of splits)
- ✓ Click OK
- ✓ Save the changes.
- ✓ So, we have splited all 4 tasks/actions successfully.
- ✓ **Now make them as re-usable components**
- ✓ Open the just created action i.e. login from drop-down box on the tool
- ✓ Go to **menu** bar , click on **Step** , select **Action Properties**
- ✓ Action properties window will appear
- ✓ Select General tab
- ✓ Check the **Reusable action** check box
- ✓ Click on OK
- ✓ **Do the same for other actions too i.e. insert order, open order and logout. Next**
- ✓ Open new Test
- ✓ Re-name the action as **Driver,** Go to **menu** bar, click on **Step**, and select **Action Properties.** Action properties window will appear, Select General tab, change the action name as Driver.
- ✓ Click on OK
- ✓ **Here we can call any or all those re-usable actions**
- ✓ Go to **menu** bar, click on **Insert**, select **Call to Existing Action**
- ✓ Select action window will appear ,
- ✓ Browse the saved application (i.e. fl_application)
- ✓ Select the one reusable action (say login)
- ✓ Select the **Location** option as **After the current step**
- ✓ Click on OK
- ✓ An existing action related to that login action will be added to this test
- ✓ **Do the same for other actions too i.e. insert order, open order and logout.**
- ✓ Now the Driver script is ready for login, inserting the order, open the order and logout.
- ✓ Run the test
- ✓ Analyze the results.

Note: in Driver test we can call any existing actions as we wish.

Type of actions: There are two types of actions

i). Normal actions

ii). Re-usable actions

Re-usable actions called in the other test are called External Actions

External actions are Non-Editable

Batch Testing :

Batch testing is a process of executing a group of tests at a time.

To do the same QTP has provided a separate tool by name “Test Batch Runner” and we have to configure the tool settings as below…..

QTP -> Tools -> Options -> Run -> Check Allow other mercury products to run tests and components.

Add the desired testAdd different script files each at a time to it
Save it as **.mtb** (Mercury Test Batch file) extension
Whenever require open the batch file, execute and analyze the results.

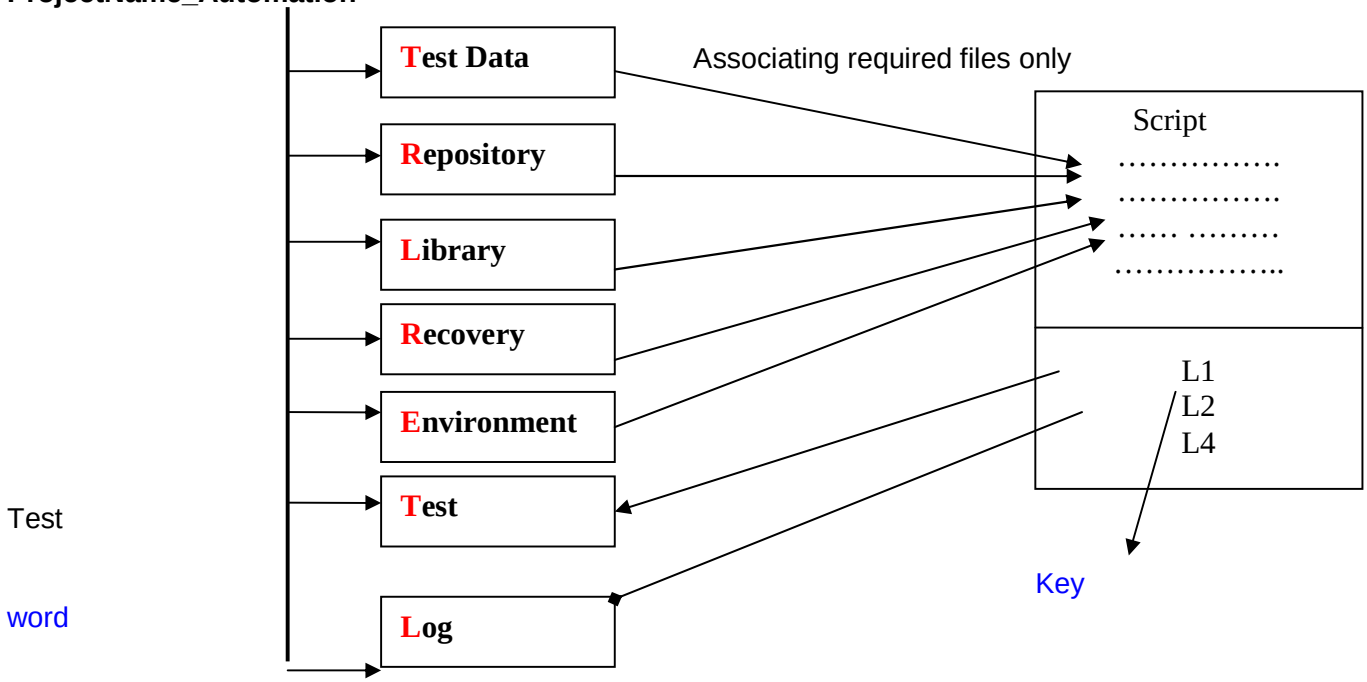
* We go for batch testing when we do Regression testing.

Regression testing: Testing the functionalities of function and all its related functionalities at a time is called Regression testing. Or it is the process of executing number of test a time.

3. Keyword Driven frame work: This is also a general frame work that can be used by **most of people**.
Steps to follow ….

1. First of all create the **folder stricture** as follows short form: **TRL RET L**

ProjectName_Automation



1. Create the required Test Data files and save them in the corresponding folder (Test Data folder).
2. Create the required Shared Repository files and save them in corresponding folder (Repository folder)
3. Create the required Library files and save them in corresponding folder (Library folder)

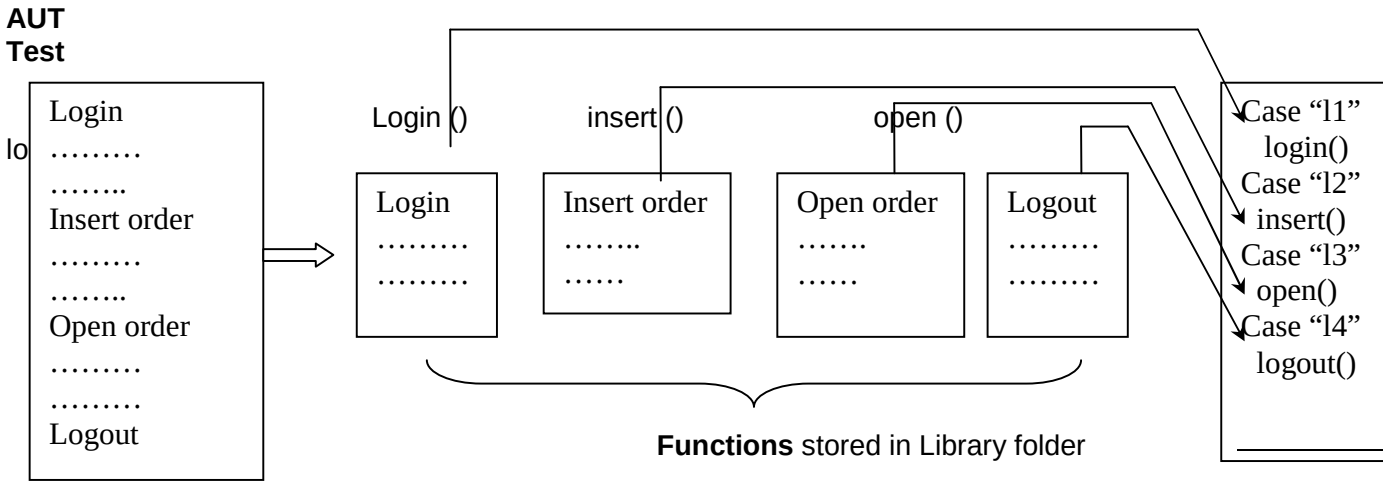
4. Create the required Recovery files and save them in corresponding folder (Recovery folder)
5. Create the required Environment files and save them in corresponding folder (Environment folder)
6. **Open the Main Test and associate all the required resources** like Test Data files, Repository files, library files, recovery files and environment files.
7. Develop the script in such way that **it executes based on the keyword given in the data table.**
8. Save the Test in the corresponding folder (Test folder)
9. Whenever require Open it and execute and analyze the results.

Example:

Tasks : **Login**
Insert order
Open existing order
Logout

Note : *Here,* after preparing complete action, split it *into different individual* ***Function****s and* call them in a **Test***.*

Note 2 : *Other words in this frame work***, we are calling the Function/s (Ex. Call login ())**



- ✓ Create a folder by any name like ProjectName_Automation (say fl_automation)
- ✓ Create all those 7 sub-folders in it (i.e. Test Data, Repository, Library, Recovery, Environment, Test and Log)
- ✓ Open the tool
- ✓ Open the flight application to do the following tasks
- ✓ Login
- ✓ Insert order
- ✓ Open order
- ✓ Logout

So, for above 4 tasks, we need to create 4 functions and store them in a library life under library folder. For that …

Open a new notepad and write down the below script in it

```
Function login ()
        * Here, Paste the login code from Test script
    End function
Function inserts order ()
        * Here, Paste the insert order code from Test script
    End function
Function open order ()
        * Here, Paste the open order code from Test script
    End function
Function logout ()
        * Here, Paste the logout code from Test script
    End function
```

Recording properties into object repository

- ✓ Put the tool under recoding mode
- ✓ Activate flight application
- ✓ **Login** with username and password
- ✓ Click on OK
- ✓ **Insert Order** by keying all the required info therein
- ✓ Click on insert order button
- ✓ That order will be inserted successfully. After inserting the order
- ✓ **Open order** by clicking on open folder icon
- ✓ A open order window will appear, check the order number check box
- ✓ Input the existing order number (say 9)
- ✓ Click on OK
- ✓ The order will open, if necessary you may update / delete the opened order
- ✓ **Logout** will be done by closing the window/application
- ✓ Stop recording

Creating Shared Repository

- ✓ Open **Object Repository**, where in which you can see all the properties of objects
- ✓ Click on **Export**
- ✓ Browse to Repository folder and save it with **.tsr** (Test Script Repository) extension.
- ✓ Click on OK

Copying and Pasting the corresponding scripts to library file from a Test Script

- ✓ Copy only login script from Test Script and past into opened notepad under Function login () ……*…. End Function.

Do the same for other functions too (i.e. Insert order, open order, and logout)

- ✓ Now save that file with **.vbs** extension under Library folder
- ✓ With this your currently open test area will be empty.
- ✓ Now , open a new Test
- ✓ Associate the required files to new test (into Repository) as below
- ✓ Menu -> Test -> settings -> Resource Tab -> select object repository type as **Shared**
- ✓ Browse the saved repository file from Repository folder
- ✓ Click on **Apply** and **OK**
- ✓ In the same manner associate the required library files to new test
- ✓ Associate the required files to new test (into Repository) as below

- ✓ Menu -> Test -> settings -> Resource Tab -> click on **+ (add) button**
- ✓ Browse the saved library file from Library folder
- ✓ Click on **Apply** and **OK**

Creating data in data table

- ✓ Activate data table of the test
- ✓ Rename the 1st column name as "**keys**" (by double clicking on it and type keys).
- ✓ Enter data like l1, l2, and l3 for each row in the table (**specifying the key value in the script so as to pick-up the relevant keyword from the data table).**
- ✓ Now Develop / write the script in test area in such way that **it uses some or all resources** and **execute based on key values** given in the data table.
- ✓ For example : to login- insert order – open order and logout

```
var =datatable ("keys", 1)      :pick-up 1st value in data table and
                                  Assigned to var
Select Case var
Case "l1"
        Login ()
Case "l2"
        Open ()
Case "l4"
        Logout ()
End Select
```

- ✓ Open the flight application
- ✓ Run the test
- ✓ Analyze the results.

4. Hybrid Frame work:

Hybrid frame work is a mixer of two or more frame works.

* Library file is a collection of one or more functions

QTP Methods and Properties

1. CaptureBitmap

Saves the screen capture of the objects as a .png or .bmp image using the specified filename

Syntax

Object.CaptureBitmap FullName,[Override Existing]

Example

The following example uses the capturebitmap method to capture a screen shot of the internet options dialog box. The file is automatically saved to a different folder (the test results folder) in each run

Browser (“Mercury Tours”).page (“Mercury Tours”).CaptureBitmap “internet_options.bmp”

2. ChildObjects

Return the collection of child objects contained within the object. This method accepts the Description object as input and returns a Collection object. The collection object includes both static and dynamic objects that satisfy the description of the object

Syntax

Object.ChildObjects (pDescription)

Example

The following examples uses the ChildObjects method to retrieve a set of child objects matching the description listed in the function call and uses the method to display a message indicating how many objects are found with the specified description:non,one(unique)or several (not unique)

```
Set a=description. Create
a ("html tag").value="A"
Set cnt=browser ("Yellow Pages").Page ("Yellow Pages").ChildObjects (a)
‘Use the count method to retrieve the number of child objects
Msgbox cnt.count
```

```
For i=0 to cnt.count-1
        c=cnt (i).getroproperty ("name")
Print c
Next
```

The above examples to display all links in the web page

```
'Get all child objects with the given description
Set children=parent.ChildObjects (oDesc)
If children. Count=1 then
checkObjectDescription="Object Unique"
Elseif children. Count=0 then
checkObjectDescription="Object not found"
Else
checkObjectDescription="Object not found"
End if
```

3. Close

Close the dialog box

Syntax

Object. Close

Example

The following examples use the close method to close the internet options dialog boxes.

```
Browser ("Mercury Tours").Dialog ("internet options").Close
```

4. Exists

Checks the object exists

Syntax

Object. Exist ([Timeout])

Example

The following examples use Exist method to determine the existence of the internet options dialog box. If the dialog box exists a message box appears conforming its appearance

```
If Browser ("Mercury Tours"). Dialog ("internet options").Exists then
Msgbox ("the object exists")
End if
```

5. GetRoProperty

Returns the current value of the runtime object property from the object in the application

RO (Runtime Object): Runtime object is the original object that was presented in the application (AUT).

Syntax

Objects.GetRoProperty (property, [PropData])

Example

The following examples print the URL name of web page

```
a=browser ("Yellow Pages").WinEdit ("Edit").GetROProperty ("text")
Print a
```

OR

```
a=browser ("Yellow Pages").page ("Yellow Pages").GetROProperty ("url")
```

Print a

6. GetTextLocation

Checks whether the specified text string is contained in the specified window area.

Syntax

Object. GetTextLocation (TextToFind, To, Top, Right, Bottom, [MatchWholeWordOnly])

Example

The following example uses the GetTextLocation method to retrieve all of the text within the object.

```
l = -1
t = -1
r = -1
b = -1
Result = Dialog ("text: =Login").Winedit ("attached text: =Agent Name :") .GetTextLocation ("2002", l, t, r, b)
If result Then
 MsgBox "Text found. Coordinates:" & l & "," & t & "," & r & "," & b
End If
```

7. GetToProperty

Returns the collection of properties and values used to identify the object.

TO (Test Object): Test object the Reference of original object stored in the object repository and used for identifying the original object in the AUT during the execution

Syntax

Object.GetToProperties

Example

```
Dim a
'Dim b
For i=0 to 6
a=dialog ("Login").Winedit ("Agent Name :") .GetToProperties (i)
Msgbox a
Next
```

8. GetToProperty

Returns the value of specified property from the test object description.

Syntax

Object. GetToProperty (property)

Example

The following example uses the GetToProperty method to retrieve the RegExpWndClass property from the Object Repository

```
Dim objectname
RegExpWndClass=Window (“Text”).GetToProperty (“RegExpWndClass”)
Msgbox RegExpWndClass
```

9. GetVisibleText
Returns the text from specified area
Syntax
Object.GetVisibleText ([Left], [To], [Right], [Bottom])
Example
The following example uses the GetVisibleText method to retrieve the from the telnet window. If the returned string contains in the sun-string, the type method is used to type the guest string in the window
Telnettext=Window ("Telnet").GetVisibleText
If instr (1, Telnettext,"login:" 1)>then
Window ("Telnet").Type "guest"
End if

10. WaitProperty
This method is used for make the tool to wait based on the object property's value or up to maximum time.
Syntax
Object Hierarchy. Waitproperty "propertyName", Property value, extra time in milliseconds
Example

- ✓ Open the flight application and put tool in recoding mode
- ✓ Open the order by clicking on open order, it will displays the open order window
- ✓ Enter an existing order number and click on Ok. That order will be opened.
- ✓ Stop recording.

Now, if you want to wait the tool even after clicking on OK button

- ✓ Take the property name (as text), value (as OK) from object repository and put extra time in milliseconds.

OH.WaitProperty "text", OK, 10000

- ✓ And put the above code after the OK button clicked statement in the script
- ✓ Run the test
- ✓ Analyze the results

10. Wait
This method is used for making the **tool to wait till the maximum time is elapsed**
Syntax
Object Hierarchy. Wait (Time in seconds)
Example

- ✓ Open the flight application and put tool in recoding mode
- ✓ Open the order by clicking on open order, it will displays the open order window
- ✓ Enter an existing order number and click on Ok. That order will be opened.
- ✓ Stop recording.

Now, if you want to wait the tool at any point of time

- ✓ Put the code any where in between the script
  Wait (10)
- ✓ Run the test
- ✓ Analyze the results

11. Check Property

Contains whether the specified object property achieves the specified value within the specified area

Syntax

Example

The following examples use the check property method to check whether the “check box1” check box is selected after setting it to “ON”

Dialog (“ac”).Activate (“ac windowarea”).ActiveCheckbox (B1).set “on”
Dialog (“ac”).Activate (“ac windowarea”).ActiveCheckbox (B1).CheckProperty “value”, true

12. GetContent

Returns all of the item in the combo box list

Syntax

Object.GetContent

Example

Set city=window (“Flight Reservation”).wincombobox (“fly from”).GetContent
Msgbox city

13. GetItem

Returns the value of the item specified by the index

Syntax

Object.GetItem (variable)

14. GetItermsCount

Returns the number of items in the combo box list

Syntax

Object.GetItemsCount

Example

Set city=window (“Flight Reservation”).wincombobox (“fly from”).GetItemsCount
Msgbox city
For I =0 to city-1
Set fcity=window (“Flight Reservation”).wincombobox (“fly from”).GetContent (I)
Msgbox fcity

15. GetSelection

Returns all of the selected items in the combo box list

Syntax

Object.GetSelection

Example

Set city=window (“Flight Reservation”).wincombobox (“fly from”).GetSelection
Msgbox city

Simple and Regular used Methods

1. Click Method: This is used for clicking on a specified object

Syntax: ***Object Hierarchy. Click [x, y [button]]***

0 for Left Click
1 for Right Click
2 for Middle Click

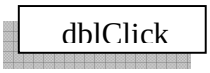
Example: VbWindow ("Emp").VbButton (Submit).Click

2. dbl Click : which is used for double clicking on a specified object

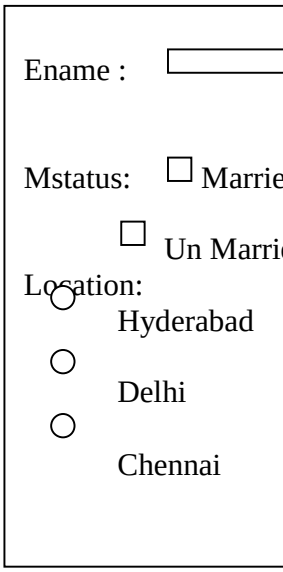
Syntax: ***Object Hierarchy. Click [x, y [button]]***

0 for Left Click
1 for Right Click
2 for Middle Click

Example: VbWindow ("Emp").VbButton (Submit).dblClick



3. Set Method : Set method is used mainly to perform on 3 objects i.e.
a). Edit box
b). Check Box
Emp
c). Radio Button



a) Edit Box : Set method is used for entering any value into an edit box
Syntax: ***Object Hierarchy. Set "value"***

Example: VbWindow ("Emp").VbEdit (Ename).Set "ak"

b) Check Box : Set method is used for Selecting/de-selecting the check box
Syntax: ***Object Hierarchy. Set "ON/OFF"***

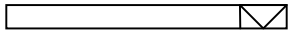
Example: VbWindow ("Emp").VbCheckBox (Mstatus).Set "ON"

c) Radio Button : Set method is used for selecting a Radio Button in a group
Syntax: ***Object Hierarchy. Set***

Example: VbWindow ("Emp").VbRadioButton (Location).Set

4. Select Method: This is used for selecting an item in a ComboBox or List
Syntax: ***Object Hierarchy. Select "item"***

Example:



5. Set Secure Method: This is used for setting the encrypted data into the edit box.

* Encrypted string can be generated with help of a tool "password encoder".

Navigation for password encoder: Start -> Programs -> QTP -> Tools -> Password encoder

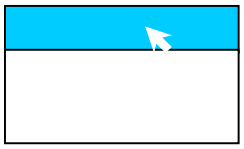
Syntax: ***Object Hierarchy. SetSecure "encrypted String"***

Example: **********

6. Activate Method: Which is used for activating a window or a dialog box

Syntax : ***Object Hierarchy. Activate***

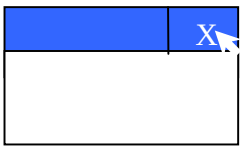
Example:



7. Close Method: This is used for Closing a window or browser

Syntax: ***Object Hierarchy. Close***

Example:



8. Type Method: This is used for performing any kind of keyboard related operations.

Syntax: ***Object Hierarchy. Type keyvalue***

Example: ggggg

Working with test objects

ListBox
The default method for List box is "Select".
Ex:-
Browser ("Mercury Tours").Page ("Find Flights").WebList ("depart").Select "London"

Can we do anything more with ListBox?
1. Selecting more items in a list
2. Get all items from a list

'Selecting more items in a list

Browser("Mercury Tours").Page("Find Flights").WebList("depart").Select "London"
Browser("Mercury Tours").Page("Find Flights").WebList("depart"). ExtendSelect "Paris"
'Select is to select single item in a list box
'ExtendSelect is to select more than one item after selecting an item in list box using select method

---

'Get all items from a list

```
Dim iCount
Dim iIndex

iCount=Browser("Countrywide").Page("Countrywide").WebList("LoanType").GetROPropert
y("items count")
For iIndex=1 to iCount
 msgbox
Browser("Countrywide").Page("Countrywide").WebList("LoanType").GetItem(iIndex)
Next
```

Menu Object

Menu object applies only window based applications. Default Method for Menu is "Select"

Example:

```
Window ("Notepad").WinMenu ("Menu").Select "File; New Ctrl+N"
```

Can we do anything more with menu's?

1. Get Menu list from Application
2. Get Menu Items from Menu

```
'Get menu list from application
Dim cnt
Dim n
Dim iPath
cnt = Window("Notepad").WinMenu("Menu").GetItemProperty("", "SubMenuCount") 'Get
total Menus count From application
For n = 1 To cnt
   iPath = Window("Notepad").WinMenu("Menu").BuildMenuPath(itemPath, n)
   msgbox Window("Notepad").WinMenu("Menu").GetItemProperty(iPath, "Label")
Next
```

```
'Get menu Items from menu
Dim cnt
Dim n
Dim iPath
Cnt = Window ("Notepad").WinMenu ("Menu").GetItemProperty ("File", "SubMenuCount")
'Get menu items count in "File" Menu
For n = 1 To Cnt
   iPath = Window ("Notepad").WinMenu ("Menu").BuildMenuPath ("File", n)
   msgbox Window ("Notepad").WinMenu ("Menu").GetItemProperty (iPath, "Label")
Next

'Get submenu count from application

Window ("Flight Reservation").Activate
Cnt=window ("Flight Reservation").WinMenu ("Menu").GetItemProperty
("File","submenucount")
```

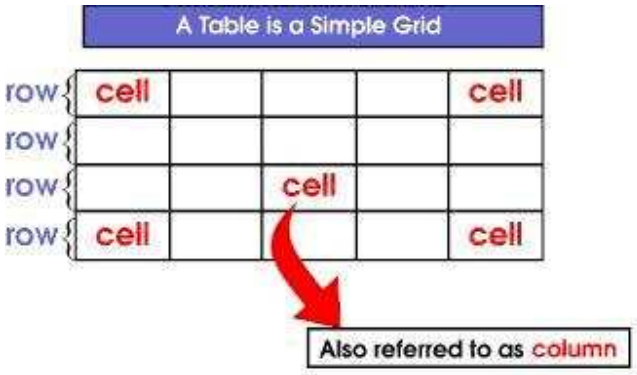
```
Print ("submenucount:"&Cnt)
For i=1 to cnt
   path1=window ("Flight Reservation").WinMenu ("Menu").BuildMenuPath ("File",i)
   b=window ("Flight Reservation").WinMenu ("Menu").GetItemProperty (path1,"Label")
   Print b
Next
```

Working with Webtable Object

What is a web table object?

Tables are one of the primary design tools for HTML documents. Tables allow for greater control over page layout, allowing creation of more visually interesting pages. Tables are also used to set apart sections of documents, such as in sidebars, navigation bars, or framing images and their titles and captions. Tables have literally changed the look of the Web page. Originally, tables let people present data in a column format. Designers quickly figured out ways to improve the layout of their pages using tables.

A Sample Web Table Object



Every web table contains data (text data) and child objects in specified cells. When we are working with descriptive programming using QTP, web table object is very useful to get data or to get object information for further activities.

Retrieving data from web table

```
'*********************************************************************
RCount=Browser ("Yahoo! Mail: The best").Page ("Yahoo! Mail: The best").WebTable
("Yahoo! ID :") .RowCount
```

```
For r=1 to RCount
CCount=Browser ("Yahoo! Mail: The best").Page ("Yahoo! Mail: The best").WebTable
("Yahoo! ID :") .ColumnCount(r)
For c=1 to CCount
CData=Browser ("Yahoo! Mail: The best").Page ("Yahoo! Mail: The best").WebTable
("Yahoo! ID :") .GetCellData(r, c)
Msgbox cData
Next
 Next

'**********************************************************************
```

Step1:- Get **row count** from table
Step2:-Using "**For Loop**" Get **column count** for every row.
Using this row number and column number we can able to identify the cells in a table. To get data from a cell we have to provide row and column numbers of a cell.
Step3:-Get cell data from every cell by providing row and column.

Accessing child objects from web table

```
'**********************************************************************
```

Description object- to return a property collection objects containing a set of property object. A property object consists of a property name and value. The Description object enables you to specify multiple properties to uniquely identify a dynamic object.

1. Creating a Description object by using the following syntax:

```
Dim objDescription
Set objDescription = Description. Create ()
```

Note – Use the *Dim* statement to declare a variable. Use the Create method to create a new and empty Description object. Use the Set statement to set the value of a variable and then assign the variable to an object.

2. Set property and value pairs in the Description object by using the following syntax:
<description_object>. (<property1>).value=<value1>….
<description_object>. (<PropertyX>).value=<valueX>

3. Refer to the dynamic object with the following syntax:
<object_hierarchy>.<object_class> (<description_object >)

```
'**********************************************************************
Set oWebEdit=Description. Create
oWebEdit("micclass").value="WebEdit"
Set objList=Browser ("Yahoo! Mail: The best").Page ("Yahoo! Mail: The best").WebTable
("Yahoo! ID :") .ChildObjects (oWebEdit)
For objIndex=0 to objlist.count-1
                        Msgbox objlist (objIndex).getroproperty ("html id")
Next
'**********************************************************************
```

This example is to access web table child objects using description object. This is a common method to get child objects from any parent level object. Not like other objects web table object is having a special method to get child items from web table cell without using description object.

Accessing child objects from web table cell

```
'*********************************************************************

RCount=Browser ("Yahoo! Mail: The best").Page ("Yahoo! Mail: The best").WebTable
("Yahoo! ID :") .RowCount
For r=1 to RCount
CCount=Browser ("Yahoo! Mail: The best").Page ("Yahoo! Mail: The best").WebTable
("Yahoo! ID :") .ColumnCount(r)
For c=1 to CCount
OCount=Browser ("Yahoo! Mail: The best").Page ("Yahoo! Mail: The best").WebTable
("Yahoo! ID :") .ChildItemCount(r, c,"WebEdit")
For obj=0 to OCount -1
Set cObject=Browser ("Yahoo! Mail: The best").Page ("Yahoo! Mail: The best").WebTable
("Yahoo! ID :") .ChildItem(r, c,"WebEdit", obj)
 Msgbox cObject.getROproperty ("html id")
Next
Next
Next

'*********************************************************************
```

This example to access child objects from web table cell. This method doesn't require description object support to access child objects from web table. For this we are using **Childitem** method.

Difference between childobject, childitem and Getcelldata methods

ChildObjects method is to access total child objects from web table object using description object.
Syntax: - *object*.ChildObjects ([*Description*])

ChildItem method is to access child objects from a web table cell in web table object without using description object.
Syntax:-*object*.ChildItem (*Row*, *Column*, *MicClass*, *Index*)

Getcelldata method is to retrieve data from a web table cell in web table object.
Syntax:-*object*.GetCellData (*Row*, *Column*)

Working with File System Object

VB Script does not recognize or read the flat files like notepad, etc.We create an object which is called **filesystemobject**,which contains flat files from that file system objective you can retrieve value and can apply on text. And we can retrieve, write, add the data's from the external environment to apply those value into the application

Description

Provides access to a computer's file system.

Syntax

Scripting.FileSystemObject

Remarks

The following code illustrates how the FileSystemObject is used to return a TextStream object that can be read from or written to:

```
**********************************************************************
Set fso = CreateObject ("Scripting.FileSystemObject")
Set a = fso.CreateTextFile ("c:\testfile.txt", True)
a.WriteLine ("This is a test.")
a. Close

**********************************************************************
```

In the preceding code, the CreateObject function returns the FileSystemObject (fso). The CreateTextFile method then creates the file as a TextStream object (a) and the WriteLine method writes a line of text to the created text file. The Close method flushes the buffer and closes the file.

Working with drives

Description

Provides access to the properties of a particular disk drive or network share.

```
**********************************************************************
Dim fso, d
Set fso = CreateObject ("Scripting.FileSystemObject")
Set d = fso.GetDrive(fso.GetDriveName(drvPath))
 **********************************************************************
```

```
'Volume Name
msgbox d.VolumeName

'The total size of the drive in bytes (TotalSize property)
msgbox "Total Space: " & FormatNumber (drv.TotalSize / 1024, 0)

'How much space is available on the drive in bytes (Available Space or FreeSpace properties)
msgbox "Available Space: " & FormatNumber (d.AvailableSpace/1024, 0)

'What letter is assigned to the drive (DriveLetter property)
msgbox "Drive” & d.DriveLetter

'What type of drive it is, such as removable, fixed, network, CD-ROM, or RAM disk (DriveType property)

'0:"Unknown"
'1:"Removable"
```

```
'2:"Fixed"
'3:"Network"
'4:"CD-ROM"
'5:"RAM Disk"
msgbox d.DriveType

'The drive's serial number (SerialNumber property)
msgbox d.SerialNumber
```

Working with folders

Description

Provides access to all the properties of a folder.

Examples

The following code illustrates how to obtain a **Folder** object and how to return one of its properties:

```
Function ShowDateCreated (folderspec)

**********************************************************************

  Dim fso, f
  Set fso = CreateObject ("Scripting.FileSystemObject")
  Set f = fso.GetFolder (folderspec)
  ShowDateCreated = f.DateCreated
End Function

**********************************************************************

'Create Folder
CreateObject ("scripting.filesystemobject").CreateFolder (FolderPath)

'Delete Folder
CreateObject ("scripting.filesystemobject").DeleteFolder (FolderPath)

'Move Folder
CreateObject ("scripting.filesystemobject").MoveFolder SourceFolderPath,
DestinationFolderPath

'Copy Folder
CreateObject ("scripting.filesystemobject").CopyFolder
SourceFolderPath,DestinationFolderPath

'Get Name of a Folder
msgbox CreateObject("scripting.filesystemobject").GetFolder(FolderPath).Name
```

```
'Check for Folder Existence
msgbox CreateObject("scripting.filesystemobject").FolderExists(FolderPath)

'Get Parent Folder Name
Msgbox CreateObject ("scripting.filesystemobject").GetFolder
(FolderPath).ParentFolder.Name

'Get Sub Folders from a Folder
Dim fso, fld, sfolders, sFld
Set fso = CreateObject ("Scripting.FileSystemObject")
Set fld = fso.GetFolder(FolderPath)
Set sfolders = fld.SubFolders
For Each sFld in sfolders
 msgbox sFld.name
Next

'Get Files From a Folder
Dim fso, fld, sFiles, sFile
Set fso = CreateObject ("Scripting.FileSystemObject")
Set fld = fso.GetFolder(FolderPath)
Set sFiles = fld.Files
For Each sFile in sFiles
 msgbox sFile.name
Next
```

Working with files

Description

Provides access to all the properties of a file

Creating Text Files

Description

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

Syntax

object.**CreateTextFile (***filename* [**,** *overwrite* [**,** *unicode*]]**)**

Examples

The following code illustrates how to use the **CreateTextFile** method to create and open a text file:

****

```
Set fso = CreateObject ("Scripting.FileSystemObject")
Set a = fso.CreateTextFile ("c:\testfile.txt", True)
a.WriteLine ("This is a test.")
a. Close
```

****

If the *overwrite* argument is **False**, or is not provided, for a *filename* that already exists, an error occurs.

Opening Text Files

Description

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

Syntax

*object***.OpenTextFile (***filename* [**,** *iomode* [**,** *create* [**,** *format*]]]**)**

Examples

The following code illustrates the use of the **OpenTextFile** method to open a file for writing text:

For reading

****

```
Dim fso, f
Set fso = CreateObject ("Scripting.FileSystemObject")
Set f = fso.OpenTextFile ("c:\kanak.txt", 1, True)
```

****

Note

The *iomode* argument can have either of the following settings:

Constant	Value	Description
ForReading	1	Open a file for reading only. You can't write to this file.
ForWriting	2	Open a file for writing only. You can't read from this file.

ForAppending	8	Open a file and write to the end of the file.

```
'Reading Specified number of Characters
 MsgBox f.Read (5)

 'Reading Complete Data from File
 MsgBox f.ReadAll

 'Read Data Line by Line
While Not f.AtEndOfStream
  MsgBox f.ReadLine
Wend
```

For writing

```
**********************************************************************
  Dim fso, fl
  Set fso = CreateObject ("Scripting.FileSystemObject")
  Set fl = fso.OpenTextFile ("c:\kanak.txt", 1, True)

**********************************************************************

'Write characters
 fl.Write ("hello")

 'Write blank lines
 fl.WriteBlankLines (2)

 'Write data line by line
 fl.WriteLine ("A New Line")
```

Update/Append Data to a Text File

```
**********************************************************************

Set fso=CreateObject ("scripting.filesystemobject")
Set fl=fso.OpenTextFile(FilePath,8)

**********************************************************************

 'Write characters
 fl.Write ("hello")

 'Write blank lines
 fl.WriteBlankLines (2)

 'Write data line by line
 fl.WriteLine ("A New Line")
```

'Remove Data from a text File

****

```
Set fso=CreateObject ("scripting.filesystemobject")
Set fl=fso.OpenTextFile(FilePath,2)
 'Write nothing
 fl.Write("")
```

****

Copy file

Description

Copies one or more files from one location to another.

Syntax

Object.CopyFile source, *destination* [, *overwrite*]

Remarks

Wildcard characters can only be used in the last path component of the *source* argument. For example, you can use:

****

```
FileSystemObject.CopyFile "c:\mydocuments\letters\*.doc", "c:\tempfolder\"
‘But you can't use:
FileSystemObject.CopyFile "c:\mydocuments\*\R1???97. xls", "c:\tempfolder"
```

****

If *source* contains wildcard characters or *destination* ends with a path separator (\), it is assumed that *destination* is an existing folder in which to copy matching files. Otherwise, *destination* is assumed to be the name of a file to create. In either case, three things can happen when an individual file is copied.

- If *destination* does not exist, *source* gets copied. This is the usual case.
- If *destination* is an existing file, an error occurs if *overwrite* is **False**. Otherwise, an attempt is made to copy *source* over the existing file.
- If *destination* is a directory, an error occurs.

An error also occurs if a *source* using wildcard characters doesn't match any files. The **CopyFile** method stops on the first error it encounters. No attempt is made to roll back or undo any changes made before an error occurs

'Moving a File

```
CreateObject ("scripting.filesystemobject").MoveFile SourceFilePath, DestinationFilePath
```

```
'*****************************************************************************
'Modify a Specific Line in a TextFile
'*****************************************************************************

Dim txtFilePath, txtRowNumber, TextToModify
Dim fso, f, txtData, strTxtData,cnt
txtFilePath= "C:\abc.txt"
txtRowNumber=3
TextToModify="three"
Set fso= CreateObject (“scripting. FilesystemObject ")
Set f=fso.OpenTextFile( txtFilePath, 1)
txtData=f.readall
f.close
Set f=fso.OpenTextFile ( txtFilePath, 2)
strTxtData=split (txtData, vbnewline)
For cnt=0 to ubound (strTxtData)
 If cnt<>txtRowNumber- 1 Then
  f.writeline (strTxtData (cnt))
 else
  f.writeline (TextToModify)
 End If
Next
f.close
'*****************************************************************************
'*****************************************************************************
```

.

Dictionary Object

Arrays are the first construct that VBScript instructors introduce when they discuss how to group data. With arrays, you can store columns of data in one place, and then access the data later through one variable. However, years of real-world use have revealed that arrays aren't always the most desirable solution to gather and maintain

related data. Fortunately, a new type of array has emerged: the dictionary. Here's a look at what dictionaries are and how you manipulate them with the methods and properties.

The Dictionary Object's Methods and Properties

Method or Property	Description
Methods	
Add	Adds a new item to the dictionary
Exists	Verifies whether a given key exists in the dictionary
Items	Returns an array with all the values in a dictionary
Keys	Returns an array with all the keys in a dictionary
Remove	Removes the item identified by the specified key
RemoveAll	Removes all the items in the dictionary
Properties	
Count	Returns the number of items in a dictionary
Item	Sets and returns an item for the specified key
Key	Changes an item's key

Comparing Dictionaries and Arrays

A dictionary is a general-purpose data structure that looks like a linked list but acts like a "super array." Like VBScript arrays, dictionaries store data and make that data available through one variable. However, dictionaries differ from arrays in many ways, including

- A dictionary has additional methods to add new items and check for existing items.
- You don't need to call ReDim to extend the dictionary's size.
- When you delete a particular item from a dictionary, all the subsequent items automatically shift up. For example, if you delete the second item in a three-item dictionary, the original third item automatically shifts up into the second-item slot.
- You use keys to identify dictionary items. Keys can be any data subtype, except an array or dictionary.
- A dictionary can't be multidimensional.
- You can store an array or another dictionary object in to a dictionary object.

The most important reason for using a dictionary instead of an array is that a dictionary is more flexible and is 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.

Use this object to support the creation, storage, and retrieval of name/value pairs in memory. Every value in a Dictionary object is a Variant, which means you can create a Dictionary object that consists of almost any kind of value (including other Dictionary objects and arrays), and that you can store any combination of Variant types in the same Dictionary object.

```
'****************************************************************
```

'Creation of a Dictionary Object

```
Dim d   ' Create a variable.
Set d = CreateObject("Scripting.Dictionary")
```

Here 'd' is a variable which is converted into a dictionary object.

```
'****************************************************************
'****************************************************************
```

'Adding items to Dictionary Object

Syntax:

Object.Add Item,Value

```
 Dim d   ' Create a variable.
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens"   ' Add some keys and items.
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
```

```
'****************************************************************
'****************************************************************
```

'To check for Key Exist or not

```
 Dim d   ' Create a variable.
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens"   ' Add some keys and items.
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
 If d.Exists("c") Then
msgbox "key exists"
```

```
Else
msgbox "key doesn't exist"
End If
 '*****************************************************************
'*****************************************************************
'To get values of Items
Dim a, d, i, iList
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens"
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
iList = d.Items
For i = 0 To d.Count -1
        msgbox iList(i)
Next
'*****************************************************************
'*****************************************************************
'To get names of the Keys
Dim a, d, i, iList
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens"
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
iList = d.Keys
For i = 0 To d.Count -1
```

```
        msgbox iList(i)
Next
'*****************************************************************
'*****************************************************************
'To Remove a key from Dictionary Object
Dim a, d
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens"
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
d.Remove("b")   ' Remove second pair.
'*****************************************************************
'*****************************************************************
'To Remove all keys from Dictionary
Dim d
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens"
d.Add "b", "Belgrade"
d.Add "c", "Cairo"
d.RemoveAll   ' Clear the dictionary.
'*****************************************************************
'*****************************************************************
'To get value of Single Key
Dim d
Set d = CreateObject("Scripting.Dictionary")
d.Add "a", "Athens"
```

```
d.Add "b", "Belgrade"

d.Add "c", "Cairo"

msgbox d("a") or msgbox d.Item("a")

'*************************************************************
'*************************************************************
```

Using Dictionary object in functions

```
Set UDetails=CreateObject("Scripting.Dictionary")

UDetails.add "UserName","Sudhakar"

UDetails.add "Password","qtp"
```

Here is a dictionary object with user name and password. To use these details in a function we should develop functions in this format.

```
 Function Login(UserDetails)

      Browser(bName).Page(pName).webedit(uName).set  UserDetails("UserName")

      Browser(bName).Page(pName).webedit(pwd).set  UserDetails("Password")

Browser(bName).Page(pName).webbutton(bName).click

 End Function
```

Calling the created function

```
 call Login (Udetails)

'*************************************************************
'*************************************************************
```

Virtual Objects

What is Virtual Object?

A virtual object is defined by user in QuickTest to recognize any area of your application as an object. Virtual objects enable you to record and run tests on objects that are not normally recognized by QuickTest.

What is virtual Object Collection?

A virtual object collection is a group of virtual objects that is stored in the Virtual Object Manager under a descriptive name.

How to disable virtual Objects while recording?

Go to **Tools-->Options--> General Tab--> Disable Recognition of virtual objects while recording**

Check and uncheck this option to disable or enable virtual objects while recording.

What is the storage location of Virtual Objects?

If you create any virtual objects automatically those objects will be stored in

<QuickTest installation folder>\ dat \ VoTemplate

What is extension of virtual objects file?

.VOT

How to use virtual objects on different machines?

After creation of virtual objects copy **<QuickTest installation folder>\ dat \ VoTemplate** Folder to other machines on which you want to use virtual objects.

What are the limitations and drawbacks of Virtual Objects?

- QuickTest does not support virtual objects for analog or low-level recording.
- Not possible to apply a checkpoint on a virtual object
- Only by recording we can add virtual objects
- Not possible to add virtual objects using Object Repository
- Not possible to spy on a virtual object using object spy
- Virtual Objects doesn't support all objects and methods.
- May not run perfectly on different screen resolutions if a test using Virtual Objects.
- Virtual object uses the properties Name, Height, Width, X, Y which the properties are having maximum possibilities for frequent change.

Object Repository

- Object Repository is a place where QTP stores learned objects
- QTP uses default Object Identification properties: mandatory and assistive to learn objects into OR

Script playback using OR

- QTP finds the Object in Object Repository using object Logical Name and Object Hierarchy
- QTP retrieves Test Object properties from OR
- QTP searches actual application for the Object with the same properties as the OR Test Object and performs user action

Object Repository Types

Test objects can be stored in two types of object repositories—a shared object repository and a local object repository.

A **shared object repository** stores test objects in a file that can be accessed by multiple tests (in read-only mode).

A **local object repository** stores objects in a file that is associated with one specific action, so that only that action can access the stored objects.

Local Object Repository

When you use a local object repository, QuickTest uses a separate object repository for each action.

- QuickTest creates a new (empty) object repository for each action.
- As you record operations on objects in your application, QuickTest automatically stores the information about those objects in the corresponding local object repository (if the objects do not already exist in an associated shared object repository).
  QuickTest adds all new objects to the local object repository even if one or more shared object repositories are already associated with the action. (This assumes that an object with the same description does not already exist in one of the associated shared object repositories).
- If a child object is added to a local object repository, and its parents are in a shared object repository, its parents are automatically moved to the local object repository.
- Every time you create a new action, QuickTest creates a new, corresponding local object repository and begins adding test objects to the local object repository as you record or learn objects.
- If you learn or record on the same object in your application in two different actions, the object is stored as a separate test object in each of the local object repositories.
- When you save your test, all of the local object repositories are automatically saved with the test (as part of each action within the test). The local object repository is not accessible as a separate file (unlike the shared object repository).

Shared Object Repository

When you use shared object repositories, QuickTest uses the shared object repositories you specify for the selected action. You can use one or more shared object repositories.

- If you record operations on an object that already exists in either the shared or local object repository, QuickTest uses the existing information and does not add the object to the object repository.
- If a child object is added to a local object repository, and its parents are in a shared object repository, its parents are automatically moved to the local object repository.
- QuickTest does not add an object to the shared object repository as you record operations on it. Instead, it adds new objects to the local object repository (not the shared object repository) as you learn objects or record steps on them (unless those same objects already exist in an associated shared object repository).
- You can export the local objects to a shared object repository.

You can also merge the local objects directly to a shared object repository that is associated with the same action. This can reduce maintenance since you can maintain the objects in a single shared location, instead of multiple locations.

Creating and managing shared object repositories will do using **Object Repository Manager.** This concept will come in **Managing object repositories.**

When to use Local object repository

- You have only one, or very few, tests that correspond to a given application interface, or set of objects.
- You do not expect to frequently modify test object properties.
- You generally create single-action tests.

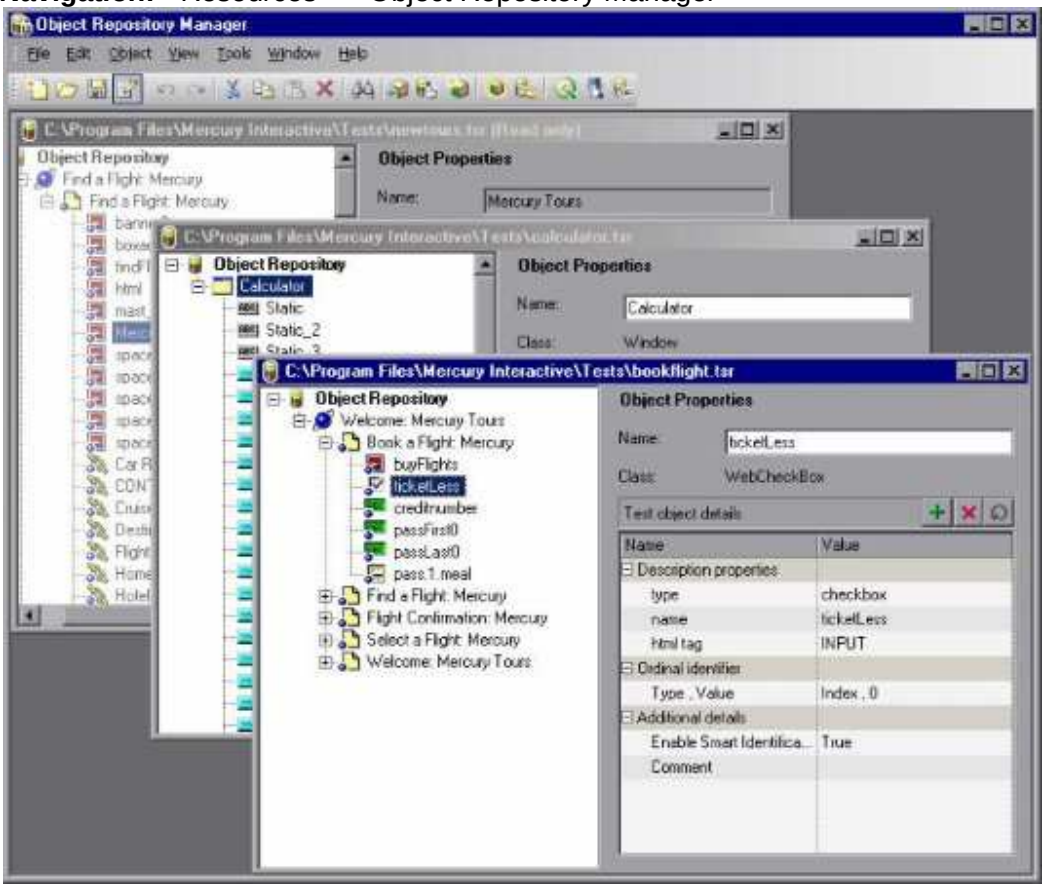
When to use shared object Repository

- You are creating tests using keyword-driven methodologies (not using record).
- You have several tests that test elements of the same application, interface, or set of objects.
- You expect the object properties in your application to change from time to time and/or you regularly need to update or modify test object properties.
- You often work with multi-action tests and regularly use the **Insert Copy of Action** and **Insert Call to Action** options.

Managing Object Repositories Using Object Repository Manager

The Object Repository Manager enables you to manage all of the shared object repositories used in your organization from a single, central location, including adding and defining objects, modifying objects and their descriptions, parameterizing repositories to make them more generic, maintaining and organizing repositories, merging repositories, and importing and exporting repositories in XML format.

Navigation: - Resources-->>Object Repository Manager



Using Object Repository Manager we can

1. Creating Shared Object Repositories
2. Managing objects in Shared object repositories
3. Modifying Test Object Details
4. Comparing object repositories
5. Merging Object Repositories

Creating Shared object repositories we can do in two ways

1. From object repository dialog box directly we can export local objects to shared
2. From Object repository Manager

But modifying, deleting or changing objects in shared object repositories is possible only with Object repository manager.

Creating Shared object Repositories

In the Object Repository Manager, choose File --> new or click the new button. A new object repository opens. You can now add objects to it, modify it, and save it.

Editing Shared object Repositories

Navigation: - In object repository manager select an Object Repository and then **File-->Enable Editing**

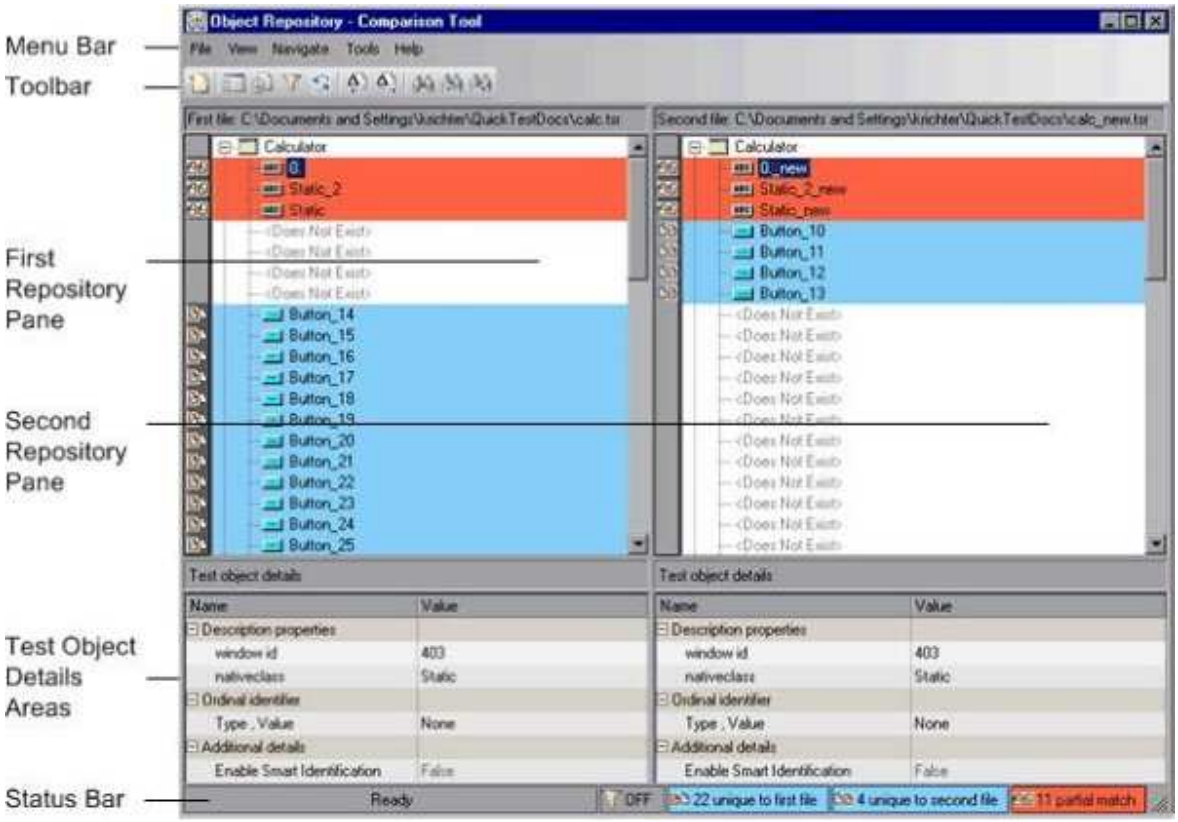
When you open an object repository, it is opened in read-only mode by default. You can open it in editable format by clearing the Open in read-only mode check box in the Open Shared Object Repository dialog box when you open it.

If you opened the object repository in read-only mode, you must enable editing for the object repository before you can modify it. You do not need to enable editing for an object repository if you only want to view it or copy objects from it to another object repository. When you enable editing for an object repository, it locks the object repository so that it cannot be modified by other users. To enable other users to modify the object repository, you must first unlock it (by disabling edit mode, or by closing it). If an object repository is already locked by another user, if it is saved in read-only format, or if you do not have the permissions required to open it, you cannot enable editing for it.

Object Repository comparison Tool

Navigation: - Tools-->Object Repository Comparison Tool

Object repository comparison tool enables you to compare two shared object repositories and to view the differences in their objects, such as different object names, different object descriptions, and so on.

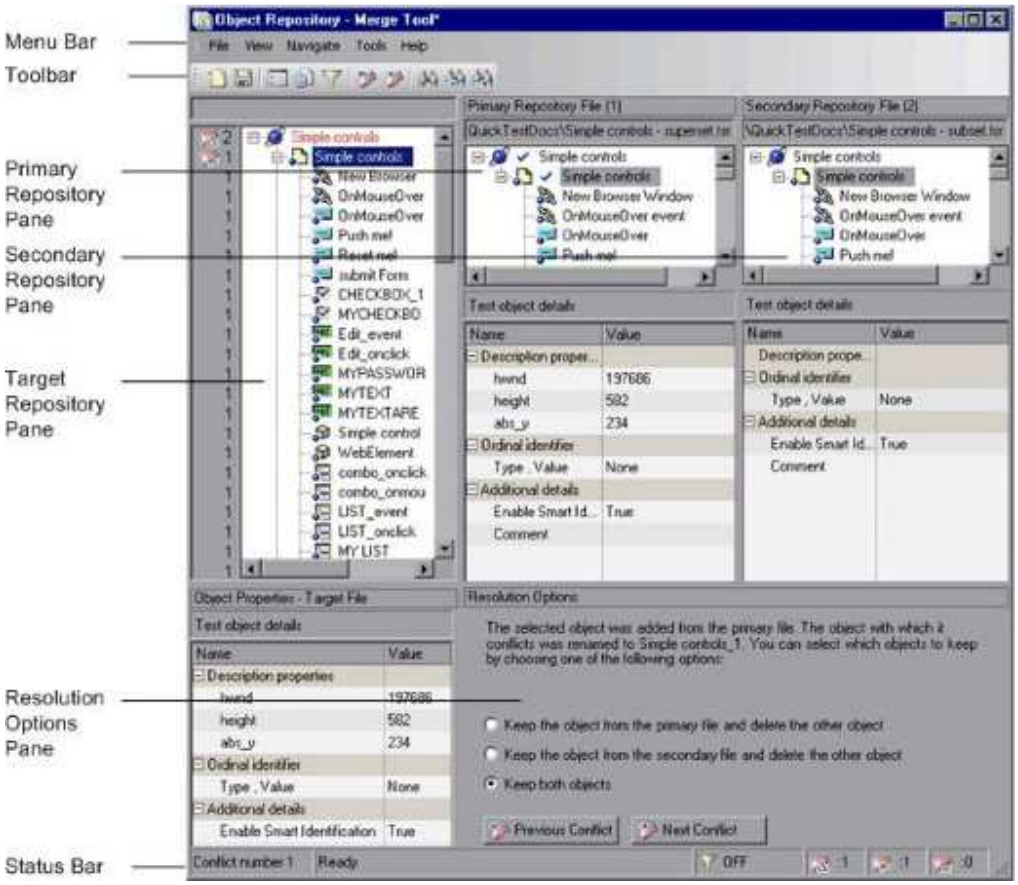


After opening the comparison tool, to compare the object repositories you need to **provide two object repositories file paths** and **click on OK**. Then the Comparison Tool provides a graphic presentation of the objects in the object repositories, which are shown as nodes in a hierarchy. Objects that have differences, as well as unique objects that are included in one object repository only, can be identified according to a color configuration that you can select. Objects that are included in one object repository only are identified in the other object repository by the text "Does not exist". You can also view the properties and values of each object that you select in either object repository.

Object Repository Merge Tool

Navigation: - Tools →Object Repository Merge Tool

Object repository merge tool enables you to merge two object repositories into a single shared object repository. You can also use this tool to merge objects from the local object repository of one or more actions into a shared object repository.



This tool enables you to merge two shared object repositories (called the **primary** object repository and the **secondary** object repository), into a new third object repository, called the **target** object repository. Objects in the primary and secondary object repositories are automatically compared and then added to the target object repository according to preconfigured rules that define how conflicts between objects are resolved.

After opening the merge tool, to merge the object repositories you need to **provide two object repositories file paths** and **click on OK**. After the merge process, the Object Repository Merge Tool provides a graphic presentation of the original objects in the primary and secondary object repositories, which remain unchanged, as well as the objects in the merged target object repository. Objects that had conflicts are highlighted. The conflict of each object that you select in the target object repository is described in detail. The Object Repository Merge Tool provides specific options that enable you to keep the suggested resolution for each conflict, or modify each conflict resolution individually, according to your requirements.

The Object Repository Merge Tool also enables you to merge objects from the local object repository of one or more actions into a shared object repository. For example, if QuickTest learned objects locally in a specific action in your test, you may want to add the objects to the shared object repository, so that they are available to all actions in different tests that use that object repository.

Working with Databases

Test data may be in Database, so that we should connect our test to database and retrieve the data and use the same in test. We should know 3 things while us dealing with database connections viz.

1. How to Connect
2. How to establish connection
3. How to Retrieve and use the data

Connecting to a Database:

For connection, we need to provide two things

1. Driver / Provider : A third party software used for establishing the connection between front end and back end of the application.

2. Location : Location of database

Record Set: It is a temporarily location where we can store the retrieve data from data base at time.
From that temporarily location we can use the data one by one or as per requirement in our testing.

Connection: Connects the application and database.

- We need to create **Object Instances** for both Record Set and Connection.

Connecting to a database

To connect a database from a Quicktest Professional script.
1. Create a connection object
2. Call the open method of the connection object

You use the *connection string* property of a *connection* object provide information about a database. Using this *connection string* property, the *open* method connects to a database

```
**********************************************************************************
'Create a connection object
Set objDB=CreateObject ("ADODB.Connection")

'Open a session to a db
objDB.ConnectionString="DSN=Flight32_testdata"
objDB.Open

**********************************************************************************
```

Executing a SQL Query

After the database connection is established, run an SQL query against the database. Use the execute method of the connection object to retrieve data from a database
The execute method accepts an SQL statements as an input and returns a RecordSet object when the run completes.

```
*******************************************************************************
'Create a connection object
Set objDB=CreateObject ("ADODB.Connection")

'Open a session to a db
objDB.ConnectionString="DSN=Flight32_testdata"
objDB.Open

*******************************************************************************
```

Examining the Query Result

After executing an SQL query, use the RecordSet object to examine the query results. Use the following properties and methods to examine the query results

- **BOF**(Beginning of file) and **EOF**(End of file)- These properties determine if you are at the boundaries of the RecordSet object
- **MoveNext, MovePrevious** and **Move -** The *MoveNext* methods moves one record forward and *Move Previous* method moves one record backward in the RecordSet object. The *Move* method moves multiple records forwards or backward at a time
- **Fields. Count –** This method indicates the number of columns of data returned by the SQL query
- **Fields ("MyColumn") or** *Fields. Item* **("MyColumn").***value* – These methods return the value saved in the specified column of the current record of the RecordSet object.

```
*******************************************************************************
'Execute a query
StrQuery="select * from flights where departure= San Francisco"
Set objResults=objDB.Execute (StrQuery)
'Examine query results
Do Until objResults.Fields ("flight_Number")
strDestination= objResults.fields ("Arrival")
'Do something with data
objResults.MoveNext
Loop

*******************************************************************************
```

In the above example, the code steps through all the flights rows with SAN FRANCISCO as the departure city .The code retrieve the flight number and arrival information from each row.

Closing the database session

After examining the output of an SQL query, close the database session by using the *Close* method. *Close* methods are provided in the *RecordSet* and *Connection* objects

Note –Closing the *Connection* object will automatically close any active *RecordSet* object associated with the *connection* object.

After the *RecordSet* or *Connection* objects is closed, you can set their variables to Nothing.

```
**********************************************************************************
'Clean up
objResults.Close
objDB.Close
Set objResults=Nothing
Set objDB=Nothing

**********************************************************************************
```

Print data from oracle database

```
**********************************************************************************
Set db=createobject ("ADODB.connection")
db.open ("DSN=QT_Flight32; UID=scott; FWD=tiger; SERVER=oracle")
If db.state=1 Then
   Msgbox "connection is opened"
   Else
   Msgbox "connection is not opened"
   End If
Set rs=db.execute ("select * from orders")
While not rs.eof
   Print rs (0) &vbtab&rs (1) &vbtab&rs (2)
   rs.movenext
Wend
**********************************************************************************
```

In the above example, the code steps through all the data are from the order table.

Insert the new rows into the database

```
**********************************************************************************
Set db=createobject ("ADODB.connection")
db.open ("DSN=QT_Flight32; UID=scott; FWD=tiger; SERVER=oracle")
If db.state=1 Then
Msgbox "connection is opened"
Else
Msgbox "connection is not opened"
```

```
End If
rs=db.execute ("insert into pbcatvld values ('kanak','Ness')")

**********************************************************************************
```

Update the specific field in the database

```
**********************************************************************************
Set db=createobject ("ADODB.connection")
db.open ("DSN=QT_Flight32; UID=scott; FWD=tiger; SERVER=oracle")
If db.state=1 Then
   Msgbox "connection is opened"
   Else
   Msgbox "connection is not opened"
End If
rs=db.execute ("update orders set customer_name=kanak where order_no=20")

**********************************************************************************
```

Delete the specific field in the database

```
**********************************************************************************
Set db=createobject ("ADODB.connection")
db.open ("DSN=QT_Flight32; UID=scott; FWD=tiger; SERVER=oracle")
If db.stste=1 Then
   Msgbox "connection is opened"
   Else
   Msgbox "connection is not opened"
End If
db.execute ("delete from orders where username like ‘demo %’")

**********************************************************************************
```

Example: For example, test data is stored as below; do write the database connection script.

Testdata.mdb

v1	v2	res
10	20	30
30	30	60
30	20	50
90	90	180
2	8	10

For MSACCESS :

```
‘Dimensioning connection (con) and recordset (rs)
```

Establishing the Connection

```
Dim con, rs
'Creating object instanced for both above con and rs
'Adodb = ActiveX Data Object Database
Set con = CreateObject ("adodb.connection")
Set rs = CreateObject ("adodb.recordset")        } these two lines never changed

'Assigning the connection with 3rd party provider i.e. with Microsoft
con.provider = "Microsoft.jet.oledb.4.0"
'Opening the database by specifying the location
*
con.open "d:/automation/testdat.mdb"
```

Retrieving the data

```
'Retrieving the data from data table
  rs.open "select * from info", on
```

Using the Data

```
'By using the retrieved data, checking all the rows
'Eof: eng of file
'Not = if the record is not end of file then go into the loop
'Else come out from the loop
Do while not rs.eof
        'Inserting the retrieved data (v1) into val1 edit field
        Vbwindow ("form1").vbEdit ("val1").set rs.field ("v1")
                'Inserting the retrieved data (v2) into val1 edit field
        Vbwindow ("form1").vbEdit ("val2").set rs.field ("v2")
        'Clicking on ADD button
        Vbwindow ("form1").vbButton ("ADD").click
        'Changing the focus to next row
        rs.moveNext
'Continuing the loop till eof
Loop

        ============================
```

*** For Oracle and SQL we will write both Provider name and Connection in one line and rest is same.**

For Oracle

con.open "provider=oraoledb.1; server=locahost; uid=userID; pwd=password; database=database name"

For SQL

con.open "provider=sqloledb.1; server=locahost; uid=userID; pwd=password; database=database name"

Word Object Model Overview

To develop solutions that use Microsoft Office Word, you can interact with the objects provided by the Word object model. Word objects are arranged in a hierarchical order, and the two main classes at the top of the hierarchy are the Application and Document classes. These two classes are important because most of the time you either work with the Word application itself, or manipulate Word documents in some way.

The Word object model closely follows the user interface. The Application object represents the entire application, each Document object represents a single Word document, the Paragraph object corresponds to a single paragraph, and so on. Each of these objects has many methods and properties that allow you to manipulate and interact with it.

Microsoft Visual Studio 2005 Tools for the Microsoft Office System (VSTO 2005) extends many of these native objects into host items and host controls that can be used in document-level customizations. These controls have additional functionality such as data-binding capabilities and events. For example, a native Word Microsoft.Office.Interop.Word.Bookmark object is extended into a Microsoft.Office.Tools.Word.Bookmark control, which can be bound to data and exposes events. For more information about host items and host controls, see Host Items and Host Controls Overview.

Accessing Objects in a Word Project

When you create a new application-level project for Word by using Microsoft Visual Studio 2005 Tools for the 2007 Microsoft Office System (VSTO 2005 SE), Visual Studio automatically creates a ThisAddIn.vb or ThisAddIn.cs code file. You can access the Application object by using Me.Application or this. Application.

When you create a new document-level project for Word by using VSTO 2005, you have the option of creating a new Word Application or Word Template project. VSTO 2005 automatically creates a ThisDocument.vb or ThisDocument.cs code file in your new Word project for both Document and Template projects. You can access the Application and Document objects by using the Me or this object reference.

At first glance, there appears to be a lot of overlap in the Word object model. For example, the Document and Selection objects are both members of the Application object, but the Document object is also a member of the Selection object. Both the Document and Selection objects contain Bookmark and Range objects. The overlap exists because there

are multiple ways you can access the same type of object. For example, you apply formatting to a Range object; but you may want to access the range of the current selection, a particular paragraph, section or the entire document.

Word Object Model Abstract

The Application object contains the Document, Selection, Bookmark, and Range objects.

Word provides hundreds of objects with which you can interact. The following sections briefly describe the top-level objects and how they interact with each other. These include:

Application object

Document object

Selection object

Range object

Bookmark object

Application Object

The Application object represents the Word application, and is the parent of all of the other objects. Its members usually apply to Word as a whole. You can use its properties and methods to control the Word environment.

Document Object

The Microsoft.Office.Interop.Word.Document object is central to programming Word. When you open a document or create a new document, you create a new Microsoft.Office.Interop.Word.Document object, which is added to the Documents collection in Word. The document that has the focus is called the active document and is represented by the Active Document property of the Application object.

Visual Studio Tools for Office extends the Microsoft.Office.Interop.Word.Document object by providing the Microsoft.Office.Tools.Word.Document object, which gives you access to all members of the Documents collection, as well as data-binding capabilities and additional events. For more information, see Host Items and Host Controls Overview. Since the majority of your code will be written in the This Document class, you can access members of This Document with the me or this object reference.

Selection Object

The Selection object represents the area that is currently selected. When you perform an operation in the Word user interface, such as bolding text, you select, or highlight the text and then apply the formatting. The Selection object is always present in a document. If nothing is selected, then it represents the insertion point. In addition, it can also be multiple blocks of text that are not contiguous.

Range Object

The Range object represents a contiguous area in a document, and is defined by a starting character position and an ending character position. You are not limited to a single Range object. You can define multiple Range objects in the same document. A Range object has the following characteristics:

- It can consist of the insertion point alone, a range of text, or the entire document.
- It includes non-printing characters such as spaces, tab characters, and paragraph marks.
- It can be the area represented by the current selection, or it can represent a different area than the current selection.
- It is not visible in a document, unlike a selection which is always visible.
- It is not saved with a document and exists only while the code is running.
- When you insert text at the end of a range, Word automatically expands the range to include the inserted text.

Bookmark Object

A Microsoft.Office.Interop.Word.Bookmark in a document is similar to a text box control on a Windows Form in that it is the easiest way to control text within a document. The Microsoft.Office.Interop.Word.Bookmark object represents a contiguous area in a document, with both a starting position and an ending position. You can use bookmarks to mark a location in a document, or as a container for text in a document. A Microsoft.Office.Interop.Word.Bookmark object can consist of the insertion point, or be as large as the entire document. A Microsoft.Office.Interop.Word.Bookmark has the following characteristics that set it apart from the Range object:

- You can name the bookmark at design-time.
- Microsoft.Office.Interop.Word.Bookmark objects are saved with the document, and thus do not get deleted when the code stops running or your document is closed.
- Bookmarks can be hidden or made visible by setting the Show Bookmarks property of the View object to True or False.

Visual Studio Tools for Office extends the Bookmark object into a host control. The Microsoft.Office.Tools.Word.Bookmark control behaves like a native Microsoft.Office.Interop.Word.Bookmark, but has additional events and data-binding

capabilities. You can now bind data to a bookmark control on a document in the same way that you bind data to a text box control on a Windows Form. For more information, see Host Items and Host Controls Overview.

Note

Microsoft.Office.Tools.Word.Bookmark controls that are added to a document programmatically at run time do not get persisted with the document. Only the underlying Microsoft.Office.Interop.Word.Bookmark object is saved. For more information, see Adding Controls to Office Documents at Run Time.

Extended Objects in Document-Level Projects

It is important to understand the differences between the native objects provided by the Word object model and the extended objects (host items and host controls) provided by VSTO 2005, because both types of objects are available to document-level projects. For more information, see Host Items and Host Controls Overview.

Design time. When you add any of the extended Word objects at design time, they are automatically created as host items and host controls. For example, if you add a bookmark to a document in the Designer, code is automatically generated to extend the bookmark into a Microsoft.Office.Tools.Word.Bookmark control.

Run time. Host items are not automatically created at run time. If you add documents at run time using the Add method, they are Microsoft.Office.Interop.Word.Document objects and do not have the additional capabilities that Microsoft.Office.Tools.Word.Document host items provide. You can programmatically add Microsoft.Office.Tools.Word.Bookmark controls to your document using the helper methods provided by VSTO 2005. For more information, see Host Items and Host Controls Overview.

Data binding and events. Host items and host controls have data-binding capabilities and events, which are not available to the native objects. Types. The native Word objects use the types defined in the Microsoft.Office.Interop.Word namespace, whereas host items and host controls use the aggregated types defined in the Microsoft.Office.Tools.Word namespace.

Creating a Microsoft Word document

```
Dim objWD

'Create the Word Object
Set objWD = CreateObject ("Word.Application")
```

```
'Create a new document
objWD.Documents.Add

'Add text to the document
objWD.Selection.TypeText "This is some text." & Chr (13) & "This is some more text"

'Save the document
objWD.ActiveDocument.SaveAs "c:\temp\mydoc.doc"

'Quit Word
objWD.Quit
```

How to search for a specific string

The following example uses Word object methods to open a Microsoft Word Document and to use the Find object (the Find and Replace functionality) to search for the word "apple.

```
Dim wrdApp
Dim wrdDoc
Dim tString, tRange
Dim p, startRange, endRange
Dim searchString

'Create the Word Object
Set wrdApp = CreateObject ("Word.Application")
Set wrdDoc = wrdApp.Documents.Open ("C:\Temp\SampleWord.doc") 'replace the file
with your MSDoc
searchString = "apple" 'replace this with the text you're searching for

With wrdDoc
for p = 1 to .Paragraphs. Count
startRange = .Paragraphs (p).Range. Start
endRange = .Paragraphs (p).Range. End
Set tRange = .Range (startRange, endRange)
'tString = tRange.Text
tRange.Find.Text = searchString
tRange.Find.Execute

If tRange.Find.Found Then
msgbox "Yes!" & searchString &" is present"
End If
```

```
Next‘close the document
Close
End With
wrdApp.Quit ‘close the Word application
Set wrdDoc = Nothing
Set wrdApp = Nothing
```

The following example uses Word object methods to open a Microsoft Word Document and retrieve paragraphs from it. Then the InStr VBScript method is used to check for the word “apple.”

```
Dim wrdApp
Dim wrdDoc
Dim tString, tRange
Dim p, startRange, endRange
Dim searchString

Create the Word Object
Set wrdApp = CreateObject (”Word.Application”)
Set wrdDoc = wrdApp.Documents.Open(”C:\Temp\Text.doc”) ‘replace the file with your MSDoc
searchString = “apple” ‘replace this with the text you’re searching for

With wrdDoc
for p = 1 to .Paragraphs. Count
startRange = .Paragraphs (p).Range. Start
endRange = .Paragraphs (p).Range. End
Set tRange = .Range (startRange, endRange)
tString = tRange.Text
tString = Left (tString, Len (tString) - 1) ‘exclude the paragraph-mark
If InStr (1, tString, searchString) > 0 Then ‘check if the text has the content you want
‘some other processing here
msgbox “Yes!” & searchString &” is present”
End If
Next
.Close ‘close the document
End With
```

```
wrdApp.Quit ‘close the Word application
Set wrdDoc = Nothing
```

What’s New in QTP 9.2

Object Identification

This concept is tells about how QTP is identifying objects while running and recording session.

Object Identification in running session

If you record in Normal or Low level recording, object information will store in object repository. In script you can see class, object name, operations, and values.

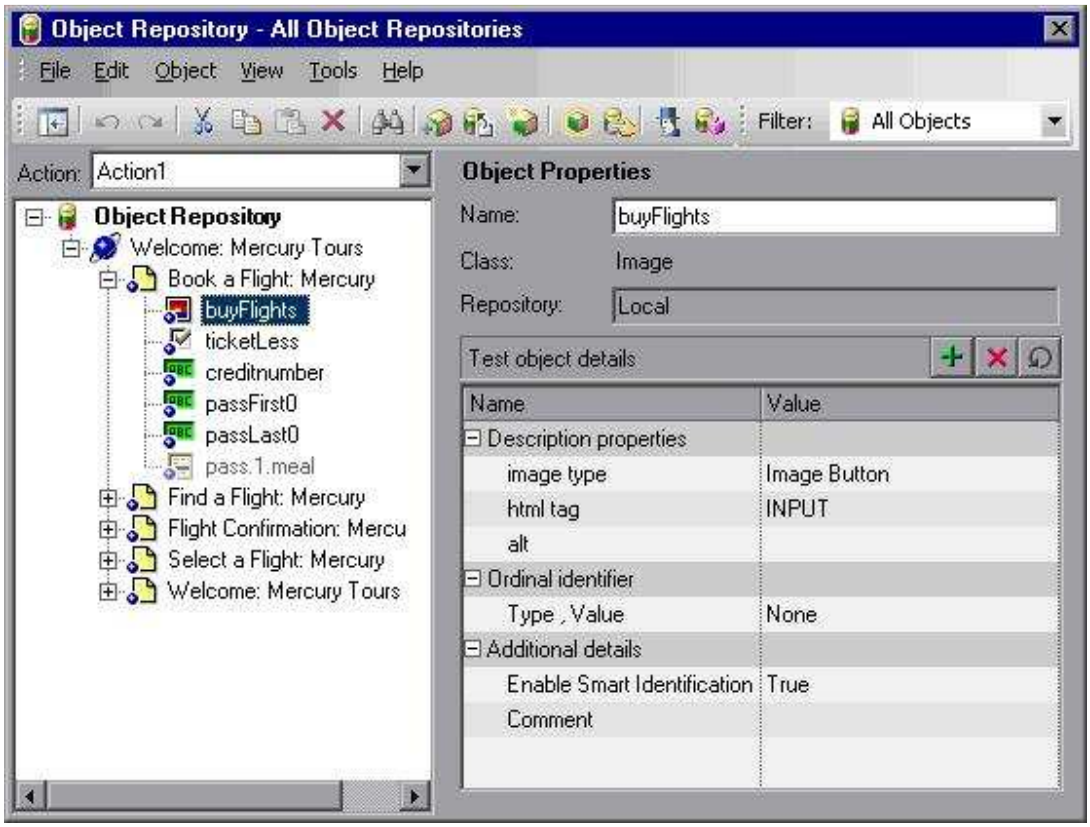
After starting running first QTP will see the class, object name in script and it searches for the same object name in object repository. If that name exists in object repository then QTP will identify the object in application using recorded properties in object repository.

Object repository

It is the place where the Recorded object information will store. In this you can see all objects properties and you can edit, modify and delete properties of the objects.

The objects and properties stored in object repository are called as **Test Objects** and **TOproperties**.

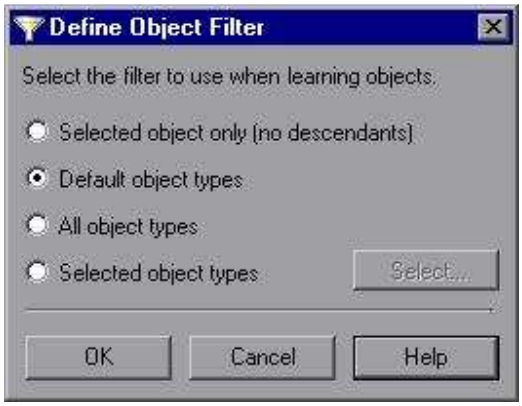
Object Repository Dialog box



Adding objects

In the Object Repository window, choose **Object--> Add Objects to Local.** Click the object you want to add to your object repository.

If the object is associated to any other object then you will this dialog box.



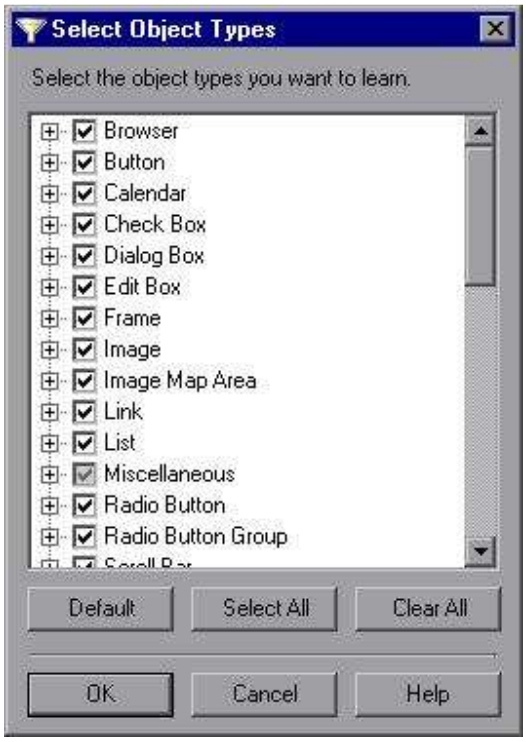
Selected object only: - Adds to the object repository the previously selected object's properties and values, without its descendant objects.

Default object types: - Adds to the object repository the previously selected object's properties and values, with the properties and values of its descendant objects according to the object types specified by the default filter. You can see which objects are in the default filter by clicking the **Select** button and then clicking the **Default** button.

All object types: - Adds to the object repository the previously selected object's properties and values, together with the properties and values of all of its descendant objects.

Selected object types: - Adds to the object repository the previously selected object's properties and values, as well as the properties and values of its descendant objects according to the object types and classes you specify in the object filter. You specify the objects and classes in the filter by clicking the **Select** button and selecting the required items in the Select Object Types dialog box.

Click on the select button to select the objects. After clicking on select this dialog box will open.



Select the filter type and add the objects to the repository.

Modifying properties of the objects

Select the object which you want to modify properties and go to description properties there select the property value. Automatically the property value will be edit; you can change that value to any value.

Changing name of the object

Select object in repository, right click**-->**Rename and change the name.

Deleting objects from repository

Select object in repository, right click**-->**Delete.

Highlighting Object

Select object in repository **View --> Highlight in Application**. This will highlight object in application.

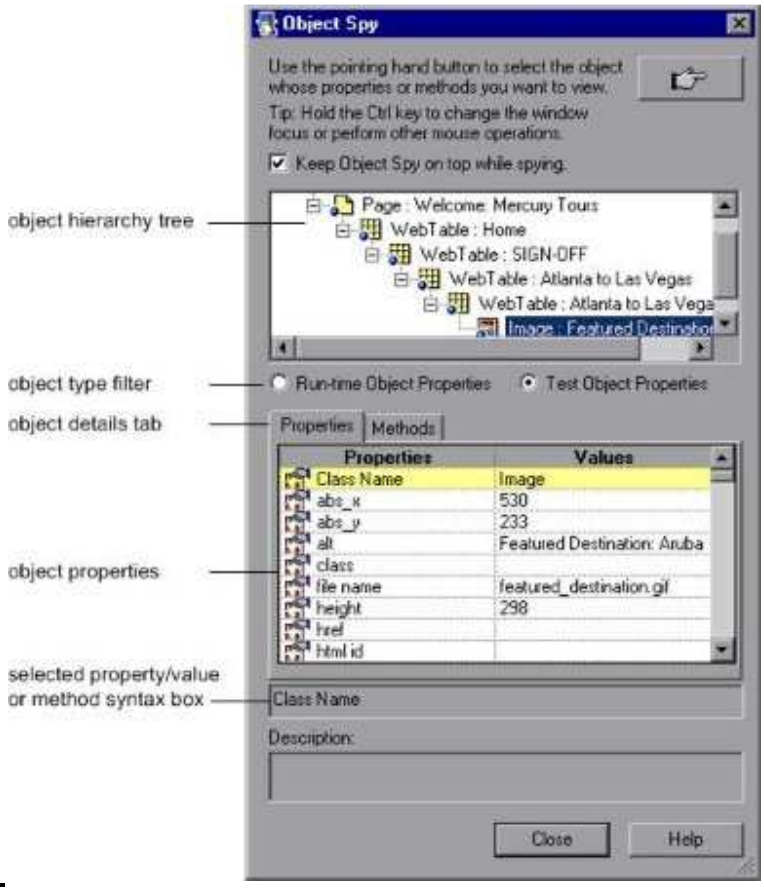
Locate in Repository

Go to View **-->** Locate in Repository and select object in application if object is there in repository then that object will be selected.

Object Spy

Navigation: - Tools-->object spy

Using the Object Spy, you can view the run-time or test object properties and methods of any object in an open application. You use the Object Spy pointer to point to an object. The Object Spy dialog box displays the selected object's hierarchy tree. It displays the run-time or test object properties and values of the selected object in the Properties tab. It displays the run-time or test object methods associated with the selected object in the Methods tab.



To use object spy Click on the hand button showing in up side and show the object to that hand button Quicktest will automatically displays properties of that object.

Tip: - Hold **Left ctrl key** for navigating on the application after clicking on the hand button.

By default object spy will show the runtime object properties for web application but not for window applications.

Object Identification Dialog box

Navigation: - Tools**-->** Object Identification

It is mainly deals with how QTP is recording objects and properties to object repository.

Using Object Identification dialog box we can

- Configure the properties for each class
- Selecting the Ordinal Identifier
- Configuring smart identification
- Creating user defined classes

In object Identification Dialog box you can find list of classes, for every class you can find pre-configured Mandatory and assistive properties.

Mandatory properties are properties that QuickTest always learns for a particular test object class.

Assistive properties are properties that QuickTest learns only if the mandatory properties that QuickTest learns for a particular object in your application are not sufficient to create a unique description. If several assistive properties are defined for an object class, then QuickTest learns one assistive property at a time and stops as soon as it creates a unique description for the object. If QuickTest does learn assistive properties, those properties are added to the test object description.

If the combination of all defined mandatory and assistive properties is not sufficient to create a unique test object description, QuickTest also records the value for the selected **ordinal identifier**.

Ordinal Identifier

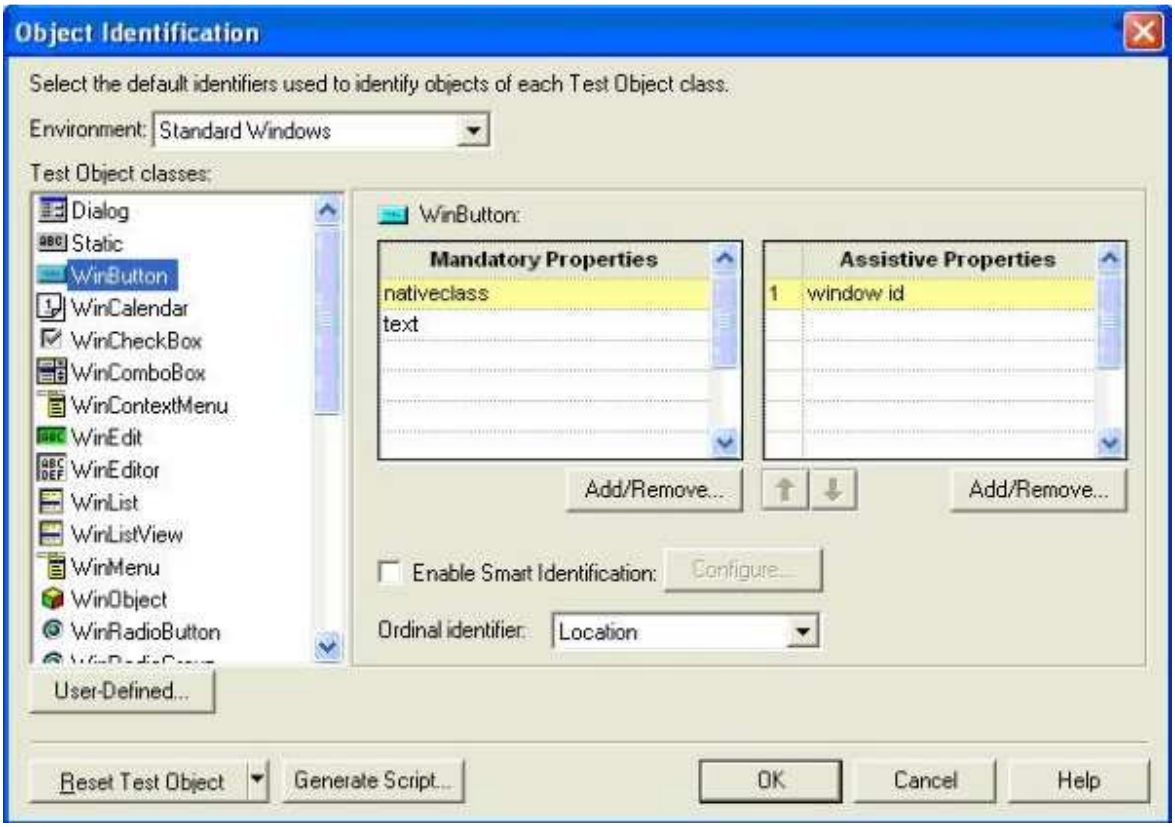
In general, there are two types of ordinal identifiers:

Index—indicates the order in which the object appears in the application code relative to other objects with an otherwise identical description.

Location—indicates the order in which the object appears within the parent window, frame, or dialog box relative to other objects with an otherwise identical description. Values are assigned from top to bottom, and then left to right.

The **Web Browser object** has a third ordinal identifier type:

CreationTime—indicates the order in which the browser was opened relative to other open browsers with an otherwise identical description. Each test object class has a default ordinal identifier selected.



Configuring Mandatory properties

In object identification dialog box select the object class from test object classes list and go to Mandatory properties for that class.

Click on **Add/Remove** button and select or deselect the properties you want to configure.

Configuring Assistive Properties

In object identification dialog box select the object class from test object classes list and go to Assistive properties for that class.

Click on **Add/Remove** button and select or deselect the properties you want to configure.

Configuring Ordinal Identifiers

To modify the selected ordinal identifier, select the desired type from the **Ordinal identifier** box.

Smart Identification

When QuickTest uses the recorded description to identify an object, it searches for an object that matches every one of the property values in the description. In most cases, this description is the simplest way to identify the object and unless the main properties of the object change, this method will work.

If QuickTest is unable to find any object that matches the recorded object description, or if it finds more than one object that fits the description, then QuickTest ignores the recorded description, and uses the Smart Identification mechanism to try to identify the object.

The Smart Identification mechanism uses two types of properties:

Base filter properties—The most fundamental properties of a particular test object class; those whose values cannot be changed without changing the essence of the original object

Optional filter properties—Other properties that can help identify objects of a particular class as they are unlikely to change on a regular

Understanding the Smart Identification Process

1. QuickTest "forgets" the recorded test object description and creates a new *object candidate* list containing the objects (within the object's parent object) that match all of the properties defined in the base filter property list.

2. From that list of objects, QuickTest filters out any object that does not match the first property listed in the Optional Filter Properties list. The remaining objects become the new object candidate list.

3. QuickTest evaluates the new object candidate list:

If the new object candidate list still has more than one object, QuickTest uses the new (smaller) object candidate list to repeat step 2 for the next optional filter property in the list.

If the new object candidate list is empty, QuickTest ignores this optional filter property, returns to the previous object candidate list, and repeats step 2 for the next optional filter property in the list.

If the object candidate list contains exactly one object, then QuickTest concludes that it has identified the object and performs the statement containing the object.

4. QuickTest continues the process described in steps 2 and 3 until it either identifies one object, or runs out of optional filter properties to use.

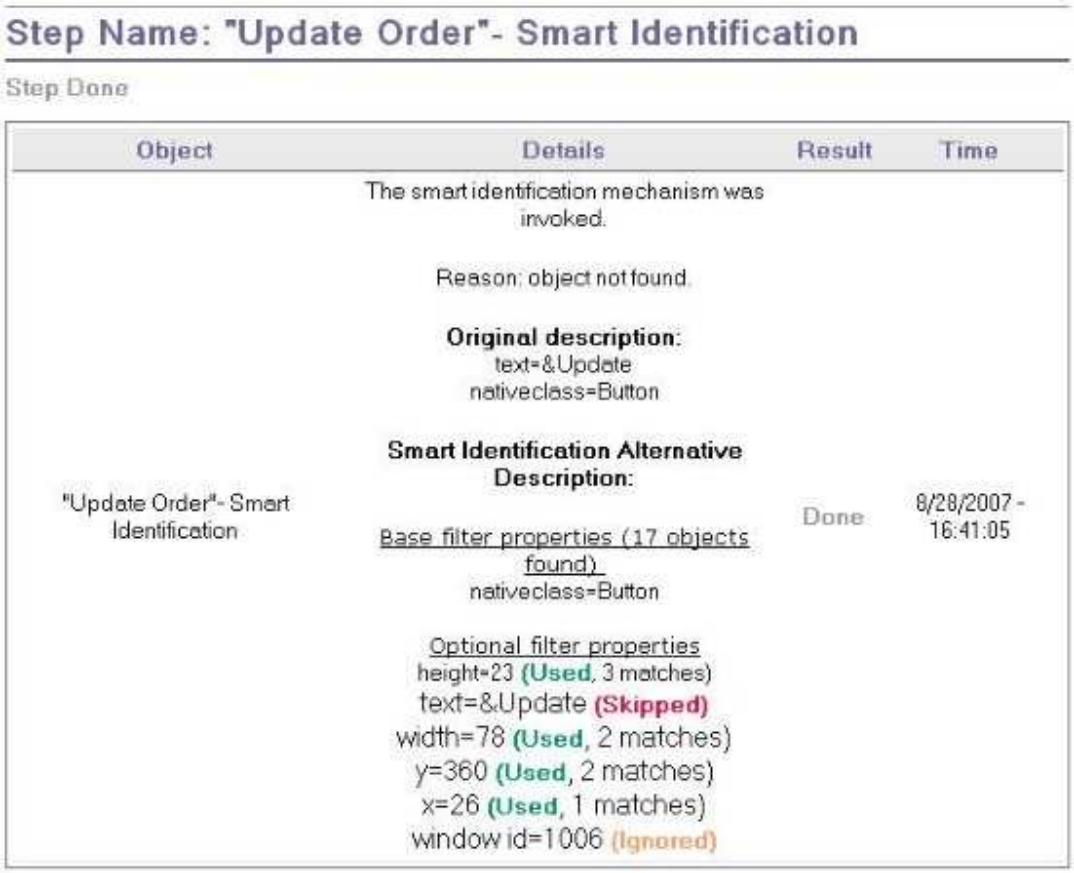
If, after completing the Smart Identification elimination process, QuickTest still cannot identify the object, then QuickTest uses the recorded description plus the ordinal identifier to identify the object.

If the combined recorded description and ordinal identifier are not sufficient to identify the object, then QuickTest stops the test run and displays a Run Error message.

If QuickTest successfully uses Smart Identification to find an object after no object matches the recorded description, the test results receive a warning status and include the following information:

Normal Identification fails to finds the object so that the warning message is showing.

If Quicktest uses Smart identification to find an object, a **cap symbol** will be appear in results where ever it is using.
And some description will show in results saying that how Quicktest identifies the object using smart identification.



Step Name: "Update Order"- Smart Identification

Step Done

Object	Details	Result	Time
"Update Order"- Smart Identification	The smart identification mechanism was invoked. Reason: object not found. **Original description:** text=&Update nativeclass=Button **Smart Identification Alternative Description:** Base filter properties (17 objects found) nativeclass=Button Optional filter properties height=23 **(Used**, 3 matches) text=&Update **(Skipped)** width=78 **(Used**, 2 matches) y=360 **(Used**, 2 matches) x=26 **(Used**, 1 matches) window id=1006 **(Ignored)**	Done	8/28/2007 - 16:41:05

In this example, Base filter properties prepares an objects list. (17 objects)

And optional filter properties filtered those object list using available properties.

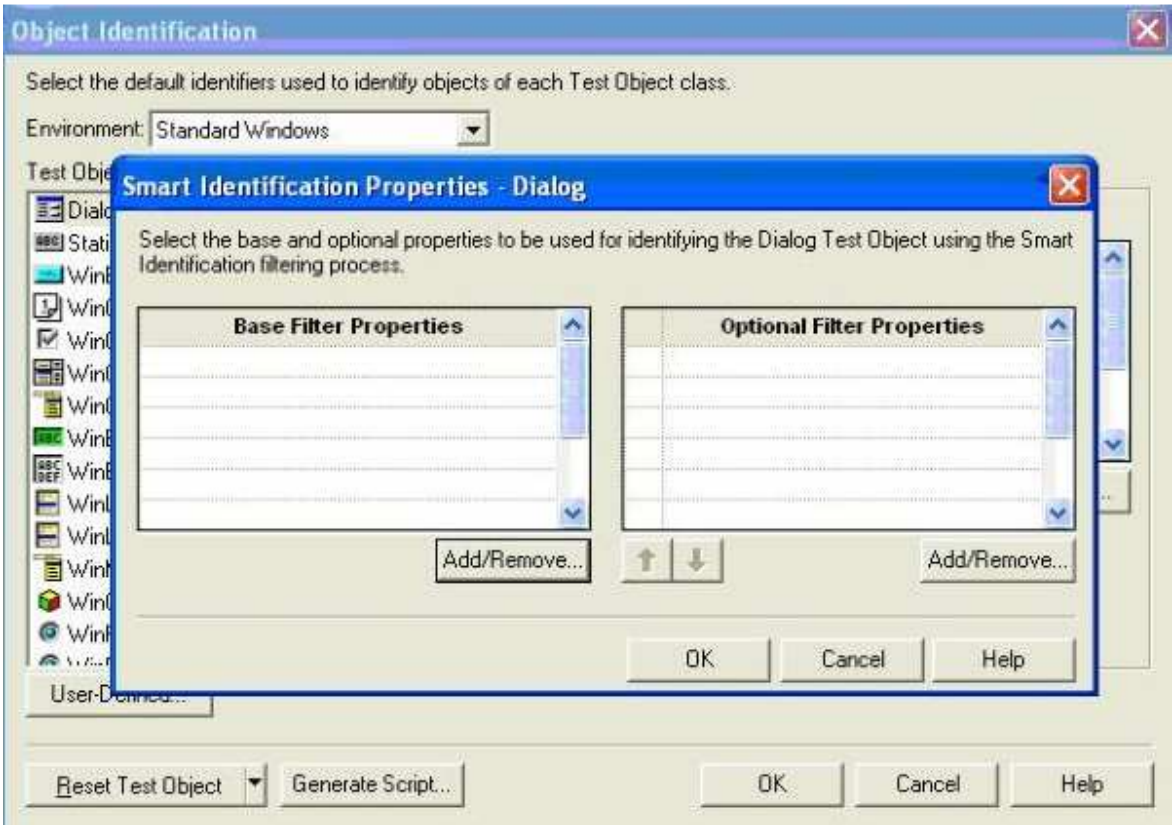
If one property fails it's using another property to find an object. If any property is finding one object QTP is ignoring other properties.

Where as in Normal Identification process all properties in object description should satisfied. If one property fails all object identification process will fail.

Configuring Smart Identification

In object identification window select a class in test object class list**-->** select option "Enable Smart Identification"**-->** Click on Configure

Then "Smart Identification Properties – class name" will be appearing…



Click **Add/Remove** to add or remove properties in Base and optional filter properties.

Note: - By default smart identification is configured and activated for web environment.

Mapping user defined test object classes

If your application has a button that cannot be identified, this button is recorded as a generic WinObject. You can teach QuickTest to identify your object as if it belonged to a standard Windows button class. Then, when you click the button while recording a test, QuickTest records the operation in the same way as a click on a standard Windows button. When you map an unidentified or custom object to a standard object, your object is added to the list of Standard Windows test object classes as a user-defined test object. You can configure the object identification settings for a user defined object class just as you would any other object class.

Note that an object that cannot be identified should be mapped only to a standard Windows class with comparable behavior.

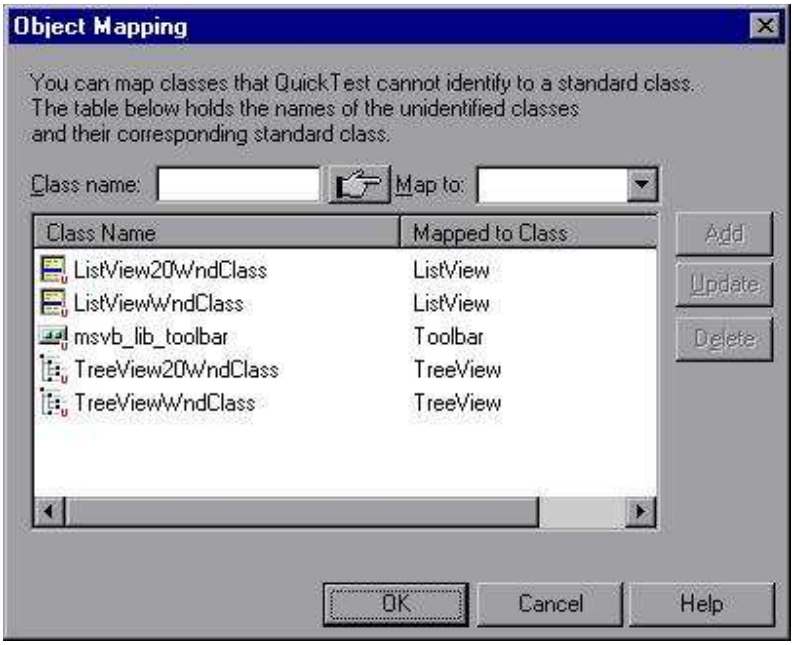
For example, do not map an object that behaves like a button to the edit class.

Note: You can define user-defined classes only when **Standard Windows** is selected in the **Environment** box.

The Object Mapping dialog box enables you to map an object of an unidentified or custom class to a Standard Windows class.

To map an unidentified or custom class to a standard Windows class:

1. Choose **Tools > Object Identification**. The Object Identification dialog box opens.
2. Select **Standard Windows** in the **Environment** box.
3. The **User-Defined** button becomes enabled. Click on **User-Defined** button. The Object Mapping dialog box opens.



4. Click the pointing hand and then click the object whose class you want to add as a user-defined class. The name of the user-defined object is displayed in the **Class Name** box.
5. In the **Map to** box, select the standard object class to which you want to map your user-defined object class and click **Add**. The class name and mapping is added to the object mapping list.
6. If you want to map additional objects to standard classes, repeat steps 4-5 for each object.
7. Click **OK**. The Object Mapping dialog box closes and your object is added to the list of Standard Windows test object classes as a user-defined test object.
8. Note that your object has an icon with a red U in the corner, identifying it as a user-defined class.
9. Configure the object identification settings for your user defined object class just as you would any other object class.

From here after that mapped user defined class objects records like standard class objects.

Object Identification - Conclusion

When you start recording on the application, QuickTest "looks" at the object on which you are recording and stores it as a **test object**, determining in which **test object class** it fits. QuickTest might classify the test object as a standard Windows dialog box (Dialog), a Web button (WebButton), or a Visual Basic scroll bar object (VbScrollBar), for example.

Then, for each test object class, QuickTest has a list of **mandatory** properties that it always learns. When you record on an object, QuickTest always learns these default property values, and then "looks" at the rest of the objects on the page, dialog box, or other parent object to check whether this **description** is enough to uniquely identify the object. If it is not, QuickTest adds **assistive** properties, one by one, to the description, until it has compiled a unique description. If no assistive properties are available, or if those available are not sufficient to create a unique description, QuickTest adds a special **ordinal identifier**, such as the object's location on the page or in the source code, to create a unique description.

Similarly, during a run session, QuickTest searches for a **run-time object** that exactly matches the description of the test object it learned while recording. It expects to find a perfect match for both the mandatory and any assistive properties it used to create a unique description while recording. As long as the object in the application does not change significantly, the description learned during recording is almost always sufficient for QuickTest to uniquely identify the object.

If QuickTest is unable to find any object that matches the learned object description, or if it finds more than one object that fits the description, then QuickTest ignores the learned description, and uses the Smart Identification mechanism to try to identify the object.

If QuickTest is unable to find any object using smart identification then QuickTest uses Combination of Learned description and ordinal identifiers to find the object uniquely.

.

VBScript Basics

VBScript has only one data type called a Variant. A Variant is a special kind of data type that can contain different kinds of information, depending on how it is used.
If you use a variable for assigning a numeric value, that variable behaves like a numeric data type. If you assign string value, that variable behaves like a string. However VBSCRIPT is having sub data types in Variant.

1. Empty
2. Null
3. Boolean
4. Byte
5. Integer
6. Currency
7. Long
8. Single
9. Double
10. Date (Time)
11. String
12. Object
13. Error.

We can use conversion functions to convert data from one subdatatype to another type. To find subdatatype of a variant we need to use **vartype** function.

Variables

A variable is a convenient placeholder to store program information. You can change variable value in script running time. In VBScript variables are always of one fundamental data type Variant.

Use of variables in script:

Variable is very useful for carrying a value. For example if your script is using a value 10 in five places (3rd, 7th, 12th, 17th, 20th lines). Suppose if that value is changed from 10 to 20 then you need to change that value in all the places where ever it is used. But if you have used variable in place of value (x=10) you need to change in only one place if that value is changed from 10 to 20(x=20). Variables are having flexibility to change value in run time.

Declaring Variables:

Because of vbscript is having only one data type no need to declare any variable in the script. By default all variables are comes under variant datatype. But it is not the good practice because you could misspell the variable name in one or more places, causing unexpected results when your script is run.

For that reason, the Option Explicit statement is available to require explicit declaration of all variables. Option Explicit statement will enforce you to declare all the variables.

We can declare the variables using Dim statement.

```
**************************************************************************
Dim x

    X=10   'Normal Declaration

Optional Explicit

Dim x

X=10   'When working with optional explicit

**************************************************************************
```

Naming Restrictions to Variables:

Variable names follow the standard rules for naming anything in VBScript.

A variable name:

- Must begin with an alphabetic character.
- Cannot contain an embedded period.
- Must not exceed 255 characters.
- Must be unique in the scope in which it is declared.

Scope of a Variable:

If you declare a variable with in a Function then it is local to that function only. You can access that variable only with in that function. Now It has local scope and is a procedure-level variable.

If you declare a variable with in a Script then it can be used by entire script and can be accessed by all functions. Now It has local scope and is a procedure-level variable. This is a script-level variable, and it has script-level scope.

Array Variables

A variable containing a single value is a scalar variable. You can create a variable that can contain a series of values using an index number. This is called an array variable. Arrays are useful when you're storing sets of similar data. You can store any kind of data in an array. The array can hold a combination of data types.

Creating Arrays:

Using Dim Statement we can create an array. We can convert a variable in to an array using array function.

Types of Arrays:

1. Fixed Length Arrays
2. Dynamic Arrays

Fixed arrays have a specific number of elements in them, whereas dynamic arrays can vary in the number of elements depending on how many are stored in the array.

Creating Fixed Length Arrays:

Dim a(10)

Here 'a' is an array and is having 11 elements (Array count starts from 0). Here it's a fixed size array with size 10.

Creating Dynamic Arrays:

A dynamic array is created in the same way as a fixed array, but you don't put any bounds in the declaration.

Dim x ()

Here 'x' is the dynamic array and we can store n number of elements in it. The benefit of a dynamic array is that if you don't know how large the array will be when you write the code, you can create code that sets or changes the size while the VBScript code is running.

We can store more values in dynamic array by redeclaring the array using Redim statement.

ReDim Statement:

Using Redim we can redeclare an array size. ReDim tells VBScript to "re-dimension" the array for how many elements you specify. If you use redim statement in your script it will clear all existing data which is stored in that array and declares that array as fresh array.

Redim with Preserve Keyword:

When your working redim it will clear all the existing data in an array. The preserve keyword is useful to overcome this problem. Preserve keyword will preserve the existing data and resize the array with the specified size.

Ex: Redim preserve a (20)

Using Fixed arrays:

```
**********************************************************************************

Dim x(2)

x(0)="how"
x(1)="are"
x(2)="you"

for i=lbound(x) to ubound (x)
msgbox x(i)
Next


**********************************************************************************
```

Here we cann't store more than 3 elements. Because this is a fixed length array..

Using Dynamic Arrays:

```
**********************************************************************************Dim x()
Redim preserve x(2)
x(0)="how"
x(1)="are"
x(2)="you"
Redim preserve x(3)
x(3)=123
**********************************************************************************
```

Here 'x' is a dynamic array and by redeclaring x it can able to store more values into it.

Converting a variable into an array:

We can convert a variable in to array variable using array function.

Example

```
********************************************************************************** Dim v
v=array("how","are","you")
for i=lbound(v) to ubound (v)
msgbox v(i)
Next
**********************************************************************************
```

Here 'v' is a dynamic array. We can store some more elements by redeclaring the array.

Constants

A constant is a meaningful name that takes the place of a number or string and never changes. The difference between variable and constant is we can change the variable value in run time but for constants it's not possible.

Creating constants:

const str="QTP".here str is a constant and the value will never change. We have public and private constants. By default all are public. If you want specify the type then

```
Public const str="QTP"
      or
Private const str="QTP"
```

VB Script Procedures

There are two types of procedures

1. Function Procedure
2. Sub Procedure

Function Procedure

A Function procedure is a series of VBScript statements enclosed by the Function and End Function statements. Function Procedure can able to return the value.

Example:

```
********************************************************************************* Function
demo_add (a, b)
   demo_add=a+b
End Function
oVal=demo_add (2, 3)
msgbox oVal    'Returns 5
*********************************************************************************
```

In this example demo_add function returns a value to oVal. In Function procedures we can use function name to assign a value.

Sub Procedure

A Sub procedure is a series of VBScript statements enclosed by the Sub and End Sub statements. Sub Procedure cannot return any value.

Example:

```
********************************************************************************** Sub demo_sub (a,
b, c)
   c=a+b
End sub
demo_sub 2, 3, x
msgbox x    'Returns 5

**********************************************************************************
```

This example will do the same as what function procedure is doing above. But in sub Procedure we need to use one more parameter to get values from the sub procedure.

Types of arguments in procedures

1. ByVal
2. ByRef

ByVal:
Indicates that the argument is passed by value.

ByRef:
Indicates that the argument is passed by reference.

By default all arguments are 'ByRef'.

Syntax

```
********************************************************************************** Function
demo_add(a,b)
   demo_add=a+b
End Function

**********************************************************************************
```

Here a,b are the arguments. By default these are 'ByRef'.

In simple words ByRef Means the value which is assigned to the variable with in the function is permanent and we can use that value out side of that function also.

ByVal means the value which is assigned to the variable with in the function is temporary and we can use that value only with in that function.

Example:

```
********************************************************************************** Function
demo_parameters (byref x, byval y)
   x=20
   y=50
   demo_parameters=x+y
End Function

a=10
b=20
```

```
msgbox demo_parameters (a, b)
msgbox a
msgbox b
********************************************************************************
```

In the above function x and y are the arguments, declared as byref and byval.
With in that function i assigned values to x and y.

Outside of the function i assigned values to two variables and passing those variables in to the function.' a' is passing reference to x and b is passing value to y. With in that function i am changing the value for x. This value is permanent for 'a'. Because 'a' is passed as 'ByRef'.But the value of 'b' will not be changed because it is passed as 'ByVal'.

Advanced and Most useful VBScript functions

Input box

Description

Displays a prompt in a dialog box, waits for the user to input text or click a button, and returns the contents of the text box.

Syntax

InputBox (*prompt* [**,** *title*][**,** *default*][**,** *xpos*][**,** *ypos*][**,** *helpfile***,** *context*]**)**

Examples

```
******************************************************************************** Dim Input
Input = InputBox ("Enter your name")
MsgBox ("You entered: " & Input)
********************************************************************************
```

Msgbox

Description

Displays a message in a dialog box, waits for the user to click a button, and returns a value indicating which button the user clicked.

Syntax

MsgBox (*prompt* [**,** *buttons*][**,** *title*][**,** *helpfile***,** *context*]**)**

Examples

```
********************************************************************************
Dim MyVar
MyVar = MsgBox ("Hello World!", 65, "MsgBox Example")
********************************************************************************
```

InStr

Description

Returns the position of the first occurrence of one string within another.

Syntax

InStr ([*start*,] **string1,** *string2* [**,** *compare*]**)**

Examples

```
*******************************************************************************
Dim SearchString, SearchChar, MyPos
SearchString ="XXpXXpXXPXXP"           ' String to search in.
SearchChar = "P"                          ' Search for "P".
'A textual comparison starting at position 4. Returns 6.
MyPos = Instr (4, SearchString, SearchChar, 1)

'A binary comparison starting at position 1. Returns 9.
MyPos = Instr (1, SearchString, SearchChar, 0)

'Comparison is binary by default (last argument is omitted).
MyPos = Instr (SearchString, SearchChar)    ' Returns 9.

'A binary comparison starting at position 1. Returns 0 ("W" is not found).
MyPos = Instr (1, SearchString, "W")
*******************************************************************************
```

Join

Description

Returns a string created by joining a number of substrings contained in an array.

Syntax

Join (*list* [, *delimiter*]**)**

```
*******************************************************************************
Dim MyString
Dim MyArray (2)
MyArray (0) = "Mr."
MyArray (1) = "kanakrajan"
MyString = Join (MyArray)
Msgbox MyString        'MyString contains "Mr. Kanakarajan"
*******************************************************************************
```

LTrim, RTrim, and Trim

Description

Returns a copy of a string without leading spaces (**LTrim**), trailing spaces (**RTrim**), or both leading and trailing spaces (**Trim**).

Syntax

LTrim (*string***)**

RTrim (*string***)**

Trim (*string*)

```
*******************************************************************************
Dim MyVar
MyVar = LTrim ("  vbscript ") ‘MyVar contains "vbscript ".
MyVar = RTrim ("  vbscript ") ‘MyVar contains "  vbscript".
MyVar = Trim ("  vbscript ")   'MyVar contains "vbscript".
*******************************************************************************
```

Left

```
*******************************************************************************
Dim MyString, LeftString
MyString = "VBSCript"
LeftString = Left (MyString, 3) 'LeftString contains "VBS".

*******************************************************************************
```

Right

```
*******************************************************************************
Dim AnyString, MyStr
AnyString = "Hello World"      'Define string.
MyStr = Right (AnyString, 1)    ' Returns "d".
MyStr = Right (AnyString, 6)    ' Returns “World".
MyStr = Right (AnyString, 20)   ' Returns "Hello World".

*******************************************************************************
```

Len

```
*******************************************************************************
Dim oStr
Dim oLength
oStr="kanakarajan"
oLength=len (oStr)
print oLength          ‘it display 11
*******************************************************************************
```

Mid

Description

Returns a specified number of characters from a string.

Syntax

Mid (*string***,** *start* [**,** *and length*]**)**

```
*******************************************************************************
Dim MyVar
MyVar = Mid ("VB Script is fun!", 4, 6) 'MyVar contains "Script".
*******************************************************************************
```

Replace

Description

Returns a string in which a specified substring has been replaced with another substring a specified number of times.

Syntax

Replace (*expression***,** *find***,** *replacewith* [**,** *start* [**,** *count* [**,** *compare*]]]**)**

```
**********************************************************************************
Dim MyString

'A binary comparison starting at the beginning of the string. Returns "XXYXXPXXY".
MyString = Replace ("XXpXXPXXp", "p", "Y")
Msgbox MyString

'A textual comparison starting at position 3. Returns "YXXYXXY".
MyString = Replace ("XXpXXPXXp", "p", "Y", 3, -1, 1)
Msgbox MyString

**********************************************************************************
```

Split

Description

Returns a zero-based, one-dimensional array containing a specified number of substrings.

Syntax

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

```
**********************************************************************************
Dim MyString, MyArray, msg
MyString=”Nisha Kanakarajan”
Msgbox MyString

MyArray= Split (MyString,” ”,-1, 1)
Msg=b (1) & “” &b (0)
Msgbox Msg                    ‘Kanakarajan Nisha

**********************************************************************************
```

StrComp

Description

Returns a value indicating the result of a string comparison.

Syntax

StrComp (*string1***,** *string2* [**,** *compare*]**)**

Settings

The *compare* argument can have the following values:

Constant	Value	Description
vbBinaryCompare	0	Perform a binary comparison.
vbTextCompare	1	Perform a textual comparison.

Return Values

The **StrComp** function has the following return values:

If	StrComp returns
string1 is less than *string2*	-1
string1 is equal to *string2*	0
string1 is greater than *string2*	1
string1 or *string2* is **Null**	**Null**

```
*******************************************************************************
Dim MyStr1, MyStr2, MyComp
MyStr1 = "NISHA": MyStr2 = "nisha"       ' Define variables.
MyComp = StrComp (MyStr1, MyStr2, 1)    ' Returns 0.
MyComp = StrComp (MyStr1, MyStr2, 0)    ' Returns -1.
MyComp = StrComp (MyStr2, MyStr1)       ' Returns 1.
*******************************************************************************
```

StrReverse

Description

Returns a string in which the character order of a specified string is reversed.

Syntax

StrReverse(*string1***)**

```
*******************************************************************************
Dim MyStr
MyStr = StrReverse ("kanakrajan") 'MyStr contains "najarakanak".
*******************************************************************************
```

VBScript samples

```
‘*****************************************************************************
'1    Print Hello World
Print "Hello World"
‘*****************************************************************************
‘*****************************************************************************
```

```
'2    Find whether given number is a odd number
Dim oNumber

oNumber=4
If oNumber mod 2 <>0 Then
   Print "The Number "& oNumber &" is an Odd Number"
else
   Print "The Number "& oNumber &" is not an Odd Number"
End If
‘*****************************************************************************
‘*****************************************************************************
'3    Print odd numbers between given range of numbers

Dim RangeStart
Dim RangeEnd
Dim iCounter
RangeStart=10
RangeEnd=20

For iCounter=RangeStart to RangeEnd

   If iCounter mod 2 <>0 Then
      Print  oNumber
   End If

Next

‘*****************************************************************************
‘*****************************************************************************
'4    Find the factorial of a given number
Dim oNumber
Dim iCounter
Dim fValue

oNumber=6
fValue=1

For iCounter=oNumber to 1 step-1
   fValue=fValue*iCounter
Next
print  fValue

‘*****************************************************************************
‘*****************************************************************************
'5    Find the factors of a given number
Dim oNumber
Dim iCounter

oNumber=10

For iCounter=1 to oNumber/2
```

```
   If oNumber mod iCounter=0 Then
      print iCounter
   End If
Next
print oNumber
‘*****************************************************************************
‘*****************************************************************************
```

'6 Print prime numbers between given range of numbers

```
Dim RangeStart
Dim RangeEnd
Dim iCounter
RangeStart=1
RangeEnd=30

For iCounter=RangeStart to RangeEnd

   For iCount=2 to round(iCounter/2)
         If iCounter mod iCount=0 Then
            Exit for
         End If
   Next

   If iCount=round(iCounter/2)+1 or iCounter=1 Then
         print iCounter
   End If
Next

‘*****************************************************************************
‘*****************************************************************************
```

'7 Swap 2 numbers with out a temporary variable

```
Dim oNum1
Dim oNum2

oNum1=1055
oNum2=155

oNum1=oNum1-oNum2
oNum2=oNum1+oNum2
oNum1=oNum2-oNum1
print oNum1
print oNum2
‘*****************************************************************************
‘*****************************************************************************
```

'8 Write a program to Perform specified Arithmetic Operation on two given numbers

```
Dim oNum1
Dim oNum2
Dim oValue
```

```
oNum1=10
oNum2=20

OperationtoPerform="div"

Select Case lcase(OperationtoPerform)

        Case "add"
                oValue=oNum1+oNum2
        Case "sub"
                oValue=oNum1-oNum2
        Case "mul"
                oValue=oNum1*oNum2
        Case "div"
                oValue=oNum1/ oNum2
End Select
print oValue
'*****************************************************************************
'*****************************************************************************
'9    Find the length of a given string
Dim oStr
Dim oLength
oStr="sudhakar"
oLength=len(oStr)
print oLength
'*****************************************************************************
'*****************************************************************************
'10    Reverse given string
Dim oStr
Dim oLength
Dim oChar
Dim iCounter

oStr="sudhakar"
oLength=len(oStr)

For iCounter=oLength to 1 step-1
      oChar=oChar&mid(oStr,iCounter,1)
Next
print oChar
'*****************************************************************************
'*****************************************************************************
'11    Find how many alpha characters present in a string.
Dim oStr
Dim oLength
Dim oChar
Dim iCounter

oStr="su1h2kar"
oLength=len(oStr)
```

```
oAlphacounter=0

For iCounter=1 to oLength

   If not isnumeric (mid(oStr,iCounter,1)) then
      oAlphacounter=oAlphacounter+1
   End if

Next
print oAlphacounter

‘*************************************************************************
‘*************************************************************************
```

'12 Find occurrences of a specific character in a string

```
Dim oStr
Dim oArray
Dim ochr
oStr="sudhakar"
ochr="a"

oArray=split(oStr,ochr)
print ubound(oArray)
‘*************************************************************************
‘*************************************************************************
```

'13 Replace space with tab in between the words of a string.

```
Dim oStr
Dim fStr

oStr="Quick Test Professional"

fStr=replace(oStr," ",vbtab)
print fStr

‘*************************************************************************
‘*************************************************************************
```

'14 Write a program to return ASCII value of a given character

```
Dim ochr
Dim aVal

ochr="A"

aVal=asc(ochr)
print aVal

‘*************************************************************************
‘*************************************************************************
```

'15 Write a program to return character corresponding to the given ASCII value

```
Dim ochr
Dim aVal

aVal=65

oChr=chr(aVal)
print oChr

‘*****************************************************************************
‘*****************************************************************************
'16    Convert string to Upper Case
Dim oStr
Dim uStr

oStr="QuickTest Professional"

uStr=ucase(oStr)
print uStr
‘*****************************************************************************
‘*****************************************************************************
'17    Convert string to lower case
Dim oStr
Dim lStr

oStr="QuickTest Professional"
lStr=lcase(oStr)
print lStr

‘*****************************************************************************
‘*****************************************************************************
'18    Write a program to Replace a word in a string with another word

Dim oStr
Dim oWord1
Dim oWord2
Dim fStr

oStr="Mercury Quick Test Professional"
oWord1="Mercury"
oWord2="HP"

fStr=replace(oStr,oWord1,oWord2)
print fStr

‘*****************************************************************************
‘*****************************************************************************
'19    Check whether the string is a POLYNDROM

Dim oStr

oStr="bob"
```

```
fStr=StrReverse(oStr)
If oStr=fStr Then
   Print "The Given String "&oStr&" is a Palindrome"
   else
   Print "The Given String "&oStr&" is not a Palindrome"
End If
'*****************************************************************************
'*****************************************************************************
'20    Verify whether given two strings are equal
Dim oStr1
Dim ostr2

oStr1="qtp"
oStr2="qtp"
If  oStr1=oStr2 Then
      Print "The Given Strings are Equal"
      else
      Print "The Given Strings are not Equal"
End If
'*****************************************************************************
'*****************************************************************************
'21    Print all values from an Array
Dim oArray
Dim oCounter
oArray=array(1,2,3,4,"qtp","Testing")

For oCounter=lbound(oArray) to ubound(oArray)
   print oArray(oCounter)
Next

'*****************************************************************************
'*****************************************************************************
'22    Sort Array elements
Dim oArray
Dim oCounter1
Dim oCounter2
Dim tmp

oArray=array(8,3,4,2,7,1,6,9,5,0)

For oCounter1=lbound(oArray) to ubound(oArray)

      For oCounter2=lbound(oArray) to ubound(oArray)-1

            If oArray(oCounter2)>oArray(oCounter2+1) Then
               tmp=oArray(oCounter2)
               oArray(oCounter2)=oArray(oCounter2+1)
               oArray(oCounter2+1)=tmp
            End If

      Next
```

```
Next

For oCounter1=lbound(oArray) to ubound(oArray)
   print oArray(oCounter1)
Next

‘*****************************************************************************
‘*****************************************************************************
'23    Add two 2X2 matrices
Dim oArray1(1,1)
Dim oArray2(1,1)
Dim tArray(1,1)

oArray1(0,0)=8
oArray1(0,1)=9
oArray1(1,0)=5
oArray1(1,1)=-1

oArray2(0,0)=-2
oArray2(0,1)=3
oArray2(1,0)=4
oArray2(1,1)=0

tArray(0,0)=oArray1(0,0)+ oArray2(0,0)
tArray(0,1)=oArray1(0,1)+oArray2(0,1)
tArray(1,0)=oArray1(1,0)+oArray2(1,0)
tArray(1,1)=oArray1(1,1)+oArray2(1,1)
‘*****************************************************************************
‘*****************************************************************************

'24    Multiply Two Matrices of size 2X2

Dim oArray1(1,1)
Dim oArray2(1,1)
Dim tArray(1,1)

oArray1(0,0)=8
oArray1(0,1)=9
oArray1(1,0)=5
oArray1(1,1)=-1

oArray2(0,0)=-2
oArray2(0,1)=3
oArray2(1,0)=4
oArray2(1,1)=0

tArray(0,0)=oArray1(0,0)* oArray2(0,0)+ oArray1(0,1)* oArray2(1,0)
tArray(0,1)=oArray1(0,0)* oArray2(0,1)+ oArray1(0,1)* oArray2(1,1)
tArray(1,0)=oArray1(1,0)* oArray2(0,0)+ oArray1(1,1)* oArray2(1,0)
tArray(1,1)=oArray1(1,0)* oArray2(0,1)+ oArray1(1,1)* oArray2(1,1)
```

```
‘*********************************************************************************
‘*********************************************************************************
'25    Convert a String in to an array

Dim oStr
Dim iCounter
oStr="Quick Test Professional"
StrArray=split(oStr)

For iCounter=0 to ubound(StrArray)
      print StrArray(iCounter)
Next

‘*********************************************************************************
‘*********************************************************************************
'26    Convert a String in to an array using ‘i‘ as delimiter

Dim oStr
Dim iCounter
oStr="Quick Test Professional"
StrArray=split(oStr,"i")

For iCounter=0 to ubound(StrArray)
      print StrArray(iCounter)
Next

‘*********************************************************************************
‘*********************************************************************************
'27    Find number of words in string

Dim oStr
Dim iCounter
oStr="Quick Test Professional"
StrArray=split(oStr," ")
print "Theere are "&ubound(StrArray)+1&" words in the string"

‘*********************************************************************************
‘*********************************************************************************
'28    Write a program to reverse the words of a given string.

Dim oStr
Dim iCounter
oStr="Quick Test Professional"
StrArray=split(oStr," ")

For iCounter=0 to ubound(StrArray)
      print strreverse(StrArray(iCounter))
Next
```

```
‘*****************************************************************************
‘*****************************************************************************
'29    Print the data as a Pascal triangle
'The formulae for pascal triangle is nCr=n!/(n-r)!*r!

Dim PascalTriangleRows
Dim nCr
Dim NumCount
Dim RowCount

PascalTriangleRows = 10
For NumCount = 0 To PascalTriangleRows
   toPrint= Space(PascalTriangleRows - NumCount)
   For RowCount = 0 To NumCount
         If (NumCount = RowCount) Then
               nCr = 1
         Else
            nCr = Factorial(NumCount) / (Factorial(NumCount - RowCount) *
Factorial(RowCount))
         End If
    toPrint=toPrint&nCr&" "
   Next
   print toPrint
Next


Function Factorial(num)
  Dim iCounter
   Factorial = 1
   If num <> 0 Then
      For iCounter = 2 To num
         Factorial = Factorial * iCounter
      Next
   End If
End Function
‘*****************************************************************************
‘*****************************************************************************
'30    Join elements of an array as a string

Dim oStr
Dim iCounter
oStr="Quick Test Professional"
StrArray=split(oStr," ")

print join(StrArray," ")

‘*****************************************************************************
‘*****************************************************************************
'31    Trim a given string from both sides

Dim oStr
```

```
oStr="    QTP    "
print trim(oStr)

‘*****************************************************************************
‘*****************************************************************************
```

'32   Write a program to insert 100values and to delete 50 values from an array

```
Dim oArray()
Dim iCounter

ReDim oArray(100)

For iCounter=0 to ubound(oArray)
        oArray(iCounter)=iCounter
        'Print total 100 Values
        print(oArray(iCounter))
Next
print "******************************"
print "******************************"
ReDim preserve oArray(50)

For iCounter=0 to ubound(oArray)
   'Print Values after deleting 50 values
        print(oArray(iCounter))
Next
‘*****************************************************************************
‘*****************************************************************************
```

'33   Write a program to force the declaration of variables

```
Option explicit    ' this keyword will enforce us to declare variables

Dim x
x=10
'Here we get an error because i have not declared y,z
y=20
z=x+y
print z
‘*****************************************************************************
‘*****************************************************************************
```

'34   Write a program to raise an error and print the error number.

```
On Error Resume Next
Err.Raise 6   ' Raise an overflow error.
print  ("Error # " & CStr(Err.Number) & " " & Err.Description)

‘*****************************************************************************
‘*****************************************************************************
```

'35   Finding whether a variable is an Array

```
Dim oArray()
```

```
if  isarray(oArray) then
   print "the given variable is an array"
 else
   print "the given variable is not an array"
End if
‘*************************************************************************
‘*************************************************************************
```

'36 Write a program to list the Timezone offset from GMT

```
Dim objWMIService
Dim colTimeZone
Dim objTimeZone

Set objWMIService = GetObject("winmgmts:" &
"{impersonationLevel=impersonate}!\\.\root\cimv2")
Set colTimeZone = objWMIService.ExecQuery("Select * from Win32_TimeZone")

For Each objTimeZone in colTimeZone
   print "Offset: "& objTimeZone.Bias
Next
‘*************************************************************************
‘*************************************************************************
```

'37 Retrieving Time Zone Information for a Computer

```
Dim objWMIService
Dim colTimeZone
Dim objTimeZone

Set objWMIService = GetObject("winmgmts:" &
"{impersonationLevel=impersonate}!\\.\root\cimv2")
Set colTimeZone = objWMIService.ExecQuery("Select * from Win32_TimeZone")

For Each objItem in colTimeZone

   print "Bias: " & objItem.Bias
   print "Caption: " & objItem.Caption
   print "Daylight Bias: " & objItem.DaylightBias
   print "Daylight Day: " & objItem.DaylightDay
   print "Daylight Day Of Week: " & objItem.DaylightDayOfWeek
   print "Daylight Hour: " & objItem.DaylightHour
   print "Daylight Millisecond: " & objItem.DaylightMillisecond
   print "Daylight Minute: " & objItem.DaylightMinute
   print "Daylight Month: " & objItem.DaylightMonth
   print "Daylight Name: " & objItem.DaylightName
   print "Daylight Second: " & objItem.DaylightSecond
   print "Daylight Year: " & objItem.DaylightYear
   print "Description: " & objItem.Description
   print "Setting ID: " & objItem.SettingID
   print "Standard Bias: " & objItem.StandardBias
   print "Standard Day: " & objItem.StandardDay
   print "Standard Day Of Week: " & objItem.StandardDayOfWeek
   print "Standard Hour: " & objItem.StandardHour
```

```
    print "Standard Millisecond: " & objItem.StandardMillisecond
    print "Standard Minute: " & objItem.StandardMinute
    print "Standard Month: " & objItem.StandardMonth
    print "Standard Name: " & objItem.StandardName
    print "Standard Second: " & objItem.StandardSecond
    print "Standard Year: " & objItem.StandardYear

Next

‘*************************************************************************
‘*************************************************************************
'38    Write a program to Convert an expression to a date
Dim StrDate
Dim actualDate
Dim StrTime
Dim actualTime

StrDate = "October 19, 1962"   ' Define date.
actualDate = CDate(StrDate)   ' Convert to Date data type.
print actualDate
StrTime = "4:35:47 PM"         ' Define time.
actualTime = CDate(StrTime)   ' Convert to Date data type.
print actualTime

‘*************************************************************************
‘*************************************************************************
'39 Display current date and Time

print now
‘*************************************************************************
‘*************************************************************************
'40    Find difference between two dates.

'Date difference in Years
print DateDiff("yyyy","12/31/2002",Date)

'Date difference in Months
print DateDiff("m","12/31/2002",Date)

'Date difference in Days
print DateDiff("d","12/31/2002",Date)

‘*************************************************************************
‘*************************************************************************
'41    Add time interval to a date

print DateAdd("m", 1, "31-Jan-95")

‘*************************************************************************
‘*************************************************************************
'42    Print current day of the week
```

```
Print day(date)
‘*************************************************************************
‘*************************************************************************
'43    Find whether current month is a long month
Dim oCurrentMonth
Dim ocurrentYear
Dim oDaysinMonths

oCurrentMonth = Month(date)
ocurrentYear = Year(date)
oDaysinMonths=Day(DateSerial(ocurrentYear, oCurrentMonth + 1, 0))
print oDaysinMonths&" Days in Current Month"
If oDaysinMonths=31 Then
   print "Current Month is a long month"
else
   print "Current Month is not a long month"
End If
‘*************************************************************************
‘*************************************************************************
'44    Find whether given year is a leap year

'1st Method

'The rules for leap year:
'1. Leap Year is divisible by 4    (This is mandotory Rule)
'2. Leap Year is not divisible by 100 (Optional)
'3. Leap Year divisble by 400 (Optional)

Dim oYear

oYear=1996

If ((oYear Mod 4 = 0) And (oYear Mod 100 <> 0) Or (oYear Mod 400 = 0)) then
   print "Year "&oYear&" is a Leap Year"
else
   print "Year "&oYear&" is not a Leap Year"
End If


'45.    2nd Method
' Checking 29 days for February month in specified year
Dim oYear
Dim tmpDate

oYear=1996
tmpDate = "1/31/" & oYear
DaysinFebMonth = DateAdd("m", 1, tmpDate)

If  day(DaysinFebMonth )=29 then
   print "Year "&oYear&" is a Leap Year"
```

```
else
   print "Year "&oYear&" is not a Leap Year"
End If

‘*************************************************************************
‘*************************************************************************
```

'46 Format Number to specified decimal places

```
Dim oNum
Dim DecimaPlacestobeFormat
oNum = 3.14159
DecimaPlacestobeFormat=2
print Round(oNum , DecimaPlacestobeFormat)

‘*************************************************************************
‘*************************************************************************
```

'47 Write a program to Generate a Random Numbers

```
'This script will generate random numbers between 10 and 20
Dim rStartRange
Dim rEndRange

rStartRange=10
rEndRange=20

For iCounter=1 to 10
   print Int((rEndRange - rStartRange + 1) * Rnd + rStartRange)
Next
‘*************************************************************************
‘*************************************************************************
```

'48 Write a program to show difference between Fix and Int

```
'Both Int and Fix remove the fractional part of number and return the resulting integer
value.
'The difference between Int and Fix is that if number is negative, Int returns the first
negative integer less than or equal to number,
'whereas Fix returns the first negative integer greater than or equal to number.
'For example, Int converts -8.4 to -9, and Fix converts -8.4 to -8.

print Int(99.8)    ' Returns 99.
print Fix(99.2)    ' Returns 99.
print Int(-99.8)   ' Returns -100.
print Fix(-99.8)   ' Returns -99.
print Int(-99.2)   ' Returns -100.
print Fix(-99.2)   ' Returns -99.

‘*************************************************************************
‘*************************************************************************
```

'49 Write a program to find subtype of a variable

```
Dim oVar
Dim oDatatypes
```

```
oVar="QTP"
oVartype=Typename(oVar)
print oVartype

‘*************************************************************************
‘*************************************************************************
'50    Write a program to print the decimal part of a given number
Dim oNum
oNum=3.123
oDecNum=oNum- int(oNum)
print oDecNum
‘*************************************************************************
‘*************************************************************************
```

How to take screenshot using VBScript

```
********************************************************************************* 'Taking Screenshot
using word object
Set oWordBasic = CreateObject ("Word. Basic")
oWordBasic.SendKeys "{prtsc}"
oWordBasic.AppClose "Microsoft Word"
Set oWordBasic = Nothing
WScript.Sleep 2000

'Opening Paint Application
set WshShell = CreateObject("WScript.Shell")
WshShell.Run "mspaint"
WScript.Sleep 2000

'Some times Paint Application is not activating properly
'To activate MS Paint properly i have minimized and restored the opened windows
set shl=createobject("shell. application")
shl.MinimizeAll
WScript.Sleep 1000
shl.UndoMinimizeAll
Set shl=Nothing
WScript.Sleep 1000

'Activating Paint Application
WshShell.AppActivate "untitled - Paint"
WScript.Sleep 1000

'Paste the captured Screenshot
WshShell.SendKeys "^v"
WScript.Sleep 500

'Save Screenshot
WshShell.SendKeys "^s"
WScript.Sleep 500
```

```
WshShell.SendKeys "c:\test.bmp"
WScript.Sleep 500
WshShell.SendKeys "{ENTER}"

'Release Objects
Set WshShell=Nothing
WScript.Quit
'*****************************************************************************
```

Advanced VBScript examples

```
'*****************************************************************************
'Write a program to read data from a text file
'*****************************************************************************

'Read Text File
Set fso=CreateObject("scripting.filesystemobject")
Set fl=fso.OpenTextFile(FilePath,1)
 'Reading Complete Data from File
 MsgBox fl.ReadAll

'*****************************************************************************
'Write a program to write data into a text file
'*****************************************************************************

Set fso=CreateObject("scripting.filesystemobject")
Set fl=fso.OpenTextFile(FilePath,2)

 'Write characters
 fl.Write("hello")

 'Write blank lines
 fl.WriteBlankLines(2)

 'Write data line by line
 fl.WriteLine("A New Line")

'*****************************************************************************
'Write a program to print all lines that contains a word either “testing” or “qtp”
'*****************************************************************************

Set fso=CreateObject("scripting.filesystemobject")
Set fl=fso.OpenTextFile(FilePath,1)
 'Read Data Line by Line
 While Not fl.AtEndOfStream
   If instr(1,fl.ReadLine,"testing")<>0 or instr(1,fl.ReadLine,"qtp")<>0 then
      print fl.ReadLine
   End if
 Wend
```

```
'*******************************************************************************
'Write a program to print the current foldername
'*******************************************************************************

Set fso=CreateObject("scripting.filesystemobject")
msgbox fso.GetAbsolutePathName("")

'*******************************************************************************
'Write a program to print files in a given folder
'*******************************************************************************

Dim fso,fld,sFiles,sFile
Set fso = CreateObject("Scripting.FileSystemObject")
Set fld = fso.GetFolder(FolderPath)
Set sFiles = fld.Files
For Each sFile in sFiles
 print sFile.name
Next

'*******************************************************************************
'Write a program to print subfolders in a given folder
'*******************************************************************************

Dim fso,fld,sfolders,sFld
Set fso = CreateObject("Scripting.FileSystemObject")
Set fld = fso.GetFolder(FolderPath)
Set sfolders = fld.SubFolders
For Each sFld in sfolders
 msgbox sFld.name
Next

'*******************************************************************************
'Write a program to print all drives in the file system
'*******************************************************************************
Set fso = CreateObject("Scripting.FileSystemObject")
Set drvs = fso.Drives

For Each drv in drvs
  print drv.DriveLetter
Next

'*******************************************************************************
'Write a program to print current drive name
'*******************************************************************************

Set fso = CreateObject("Scripting.FileSystemObject")
msgbox fso.GetDriveName(fso.GetAbsolutePathName(""))

'*******************************************************************************
'Print the last modified and creation date of a given file
```

```
'*******************************************************************************

Set fso = CreateObject("Scripting.FileSystemObject")
Set fl = fso.GetFile(FilePath)
print fl.DateLastModified
print fl.DateCreated

'*******************************************************************************
'Print the size of the file
'*******************************************************************************

Set fso = CreateObject("Scripting.FileSystemObject")
Set fl = fso.GetFile(FilePath)
msgbox  fl.Size

'*******************************************************************************
'Write a program to display files of a specific type
'*******************************************************************************

Function DisplaySpecificFileTypes(FolderPath,FileType)

   Set fso = CreateObject("Scripting.FileSystemObject")
   Set fld = fso.GetFolder(FolderPath)
   Set sFiles = fld.files

   For Each sFl in sFiles
      If lcase(sFl.type)=lcase(FileType) then
         print sFl.name
      End if
   Next

End Function

'Calling the Function
DisplaySpecificFileTypes "C:\","Text Document"

'*******************************************************************************
'Write a program to print the free space in a given drive
'*******************************************************************************

Set fso = CreateObject("Scripting.FileSystemObject")
Set d = fso.GetDrive(drvPath)
print "Free Space: " & FormatNumber(d.FreeSpace/1024, 0)

'*******************************************************************************
'Write a program to find whether a given folder is a special folder
'*******************************************************************************

fldPath="C:\WINDOWS"
Const WindowsFolder =0
Const SystemFolder = 1
```

```
Const TemporaryFolder = 2

Set fso = CreateObject("Scripting.FileSystemObject")
If lcase(fso.GetSpecialFolder(WindowsFolder))=lcase(fldPath) or
lcase(fso.GetSpecialFolder(SystemFolder))=lcase(fldPath) or
lcase(fso.GetSpecialFolder(TemporaryFolder))=lcase(fldPath) then
   print "Given Folder is a special Folder"
Else
   print "Given Folder is not a special Folder"
End if

'*************************************************************************
'Write a program to remove all empty files in the folder
'*************************************************************************

FolderPath="C:\Documents and Settings\sudhakar kakunuri\Desktop\"
Set fso = CreateObject("Scripting.FileSystemObject")
Set fld = fso.GetFolder(FolderPath)
Set sFiles = fld.Files
For Each sFile in sFiles

   If sFile.size=0 Then
      print sFile.name
   End If

Next

'*************************************************************************
'Write a program to Copy contents of one folder to other folder
'*************************************************************************

Function CopyContentstoOtherFolder(SourceFolder,TargetFolder)

Set fso = CreateObject("Scripting.FileSystemObject")
Set fld = fso.GetFolder(SourceFolder)

Set sFiles = fld.Files

For Each sFile in sFiles
  sFile.copy TargetFolder&"\"&sFile.name
Next

Set sFlds = fld.SubFolders

For Each sFld in sFlds
  sFld.copy TargetFolder&"\"&sFld.name
Next

End Function

'Calling the Function
```

```
CopyContentstoOtherFolder "C:\Documents and Settings\sudhakar
kakunuri\Desktop\Test1","C:\Documents and Settings\sudhakar kakunuri\Desktop\Test2"

'*******************************************************************************
'Write a program to check whether a given path represents a file or a folder
'*******************************************************************************

Set fso = CreateObject("Scripting.FileSystemObject")
If fso.FileExists(fPath) then
   Print "Path Representing a File"
Elseif fso.FolderExists(fPath) then
   Print "Path Representing a Folder"
End if

'*******************************************************************************
'Write a program to compress a folder
'*******************************************************************************

strComputer = "."
strFolder   = "Folder Path"

set objWMI = GetObject("winmgmts:\\" & strComputer & "\root\cimv2")
set objFolder = objWMI.Get("Win32_Directory='" & strFolder & "'")

fCompress = objFolder.Compress  ' To uncompress change this to objFolder.Uncompress

if fCompress <> 0 then
  WScript.Echo "There was an error compressing the folder: " & fCompress
else
  WScript.Echo "Folder compression successful"
end if

'*******************************************************************************
'Write a program to rename a folder
'*******************************************************************************

CreateObject("Scripting.FileSystemObject").GetFolder("C:\Documents and
Settings\sudhakar\Desktop\Training Session For MF Testers").Name="New Name"

'*******************************************************************************
'Write a program to print all lines in a file that ends with “world”
'*******************************************************************************

Set fso=CreateObject("Scripting.FileSystemObject")
Set fl=fso.OpenTextFile(FilePath)

Set regEx = New RegExp
regEx.Pattern = "world$"
regEx.Global = True

 While Not fl.AtEndOfStream
```

```
   Set Matches = regEx.Execute(strng)

   If Matches.count<>0 then
      print fl.ReadLine
   End if

 Wend

'*****************************************************************************
'Write a program to check whether string starts with “Error”
'*****************************************************************************
str="Errorhello"
Set regEx = New RegExp
regEx.Pattern = "^Error"
regEx.Global = True
Set Matches = regEx.Execute(str)

If Matches.count<>0 then
   msgbox "String Started with 'Error'"
   Else
   msgbox "String Not Started with 'Error'"
End if

'*****************************************************************************
'Write a program to Replace all words that contains “demo” in a file with the word “QTP”
'*****************************************************************************
FilePath="C:\Documents and Settings\sudhakar\Desktop\demo.txt"

Set fso=CreateObject("Scripting.FileSystemObject")
Set fl=fso.OpenTextFile(FilePath)
txtToReplace=replace(fl.ReadAll,"demo","QTP")
fl.Close

Set fl=fso.OpenTextFile(FilePath,2)
fl.Write(txtToReplace)
fl.Close
'*****************************************************************************
'Write a program to check whether a string contains only alpha numerics
'*****************************************************************************

str="xyz123!!"
Set regEx = New RegExp
regEx.Pattern = "[^A-Za-z0-9]"
regEx.Global = True
Set Matches = regEx.Execute(str)

If Matches.count=0 then
   msgbox "String Contains only alpha numerics"
   Else
   msgbox "String Contains other characters"
End if
```

```
'*****************************************************************************
'Write a program to check whether given string is in date format
'*****************************************************************************
MyDate = "October 19, 1962"

If isdate(MyDate) Then
   msgbox "Given String is in Date Format"
   else
   msgbox "Given String is not in Date Format"
End If

'*****************************************************************************
'Write a program to Invoke command prompt and execute dir command from the shell
'*****************************************************************************
Dim oShell
Set oShell = WScript.CreateObject ("WSCript.shell")
oShell.run "cmd /K CD C:\ & Dir"
Set oShell = Nothing
'*****************************************************************************
```

Sample script for Web Based Application

1. Script to enter data in login screen

```
*******************************************************************************
Browser ("Book Store").Page ("Book Store").WebEdit ("Login").Set "guest"
Browser ("Book Store").Page ("Book Store").WebEdit ("Password").Set "guest"
Browser ("Book Store").Page ("Book Store").WebButton ("Login").Click
'*******************************************************************************
```

2. Find X and Y coordinates of button

```
*******************************************************************************
x=browser ("Book Store").Page ("Book Store").WebButton ("Search").GetROProperty ("x")
y=browser ("Book Store").Page ("Book Store").WebButton ("Search").GetROProperty ("y")
Msgbox x
Msgbox y
'*******************************************************************************
```

3. Read items in a list box –YP->Admin->Entries ->Oracle->Category

```
*******************************************************************************
a=browser ("Book Store").Page ("Book Store").WebList ("category_id").GetROProperty
("items count")
Msgbox a
For i=1 to a
   Set c= browser ("Book Store").Page ("Book Store").WebList ("category_id").GetItem (i)
Msgbox c
Next
'*******************************************************************************
```

4. Check whether edit box is focused –YP->Home->Name

```
'*****************************************************************************
a=browser ("Yellow Pages").Page ("Yellow Pages").WebEdit ("name").GetROProperty
("visible")
If a=True Then
        Msgbox "edit box is focused"
        Else
        Msgbox "edit box is disabled"
End If
'*****************************************************************************
```

5. Check default selection in list box

```
'*****************************************************************************
a=browser ("Yellow Pages").Page ("Yellow Pages").WebList
("category_id").GetROProperty ("default value")
If a="Software" Then
        Msgbox "default value is present"
        Else
        Msgbox "default value is not showing"
End If

'*****************************************************************************
```

6. List all links in the web page

```
'*****************************************************************************
set a=description. Create
a ("html tag").value="A"
Set cnt=browser ("Yellow Pages").Page ("Yellow Pages").ChildObjects (a)
Msgbox cnt.count
For i=0 to cnt.count-1
        c=cnt (i).getroproperty ("name")
Msgbox c
Next
'*****************************************************************************
```

7. Print URL name

```
'*****************************************************************************
a=browser ("Yellow Pages").WinEdit ("Edit").GetROProperty ("text")
Msgbox a
                                        OR
a=browser ("Yellow Pages").page ("Yellow Pages").GetROProperty ("url")
Msgbox a

'*****************************************************************************
```

8. Display entire data in the web table

```
'*******************************************************************************
a=browser ("Yellow Pages").Page ("Yellow Pages").WebTable
("Categories").GetROProperty ("text")
 Msgbox a

'*******************************************************************************
```

9. Find the background color of the object

```
'*******************************************************************************
a=browser ("Yellow Pages").page ("yellow pages").WebElement ("name").GetROProperty
("outerhtml")
Msgbox a

'*******************************************************************************
```

10. Check whether selected link is having different appearance

```
'*******************************************************************************
x=Browser ("Yellow Pages").Page ("Yellow Pages").Image ("CC-YelloPages-
logo").getroproperty ("outer html")
Msgbox x

'*******************************************************************************
```

11. Check whether a link contains image to its left side.

```
'*******************************************************************************
Browser ("Yellow Pages").Page ("Yellow Pages").Image ("CC-YelloPages-
logo").getroproperty ("x")
        If x<30 Then
                Msgbox "logo is in left side"
        Else
                Msgbox "logo is not in left side"
        End If

'*******************************************************************************
```

12. Clear the cache in the web page

```
'*******************************************************************************
browser ("browser").wintoolbar ("toolbarwindow32").Press"&tools"
Browser ("browser").winmenu ("contextmenu").Select"internet options."
Browser ("browser").dialog ("internet options").winbutton ("delete cookies").Click
Browser ("browser").dialog ("internet options").dialog ("delete cookies").winbutton
("ok").Click
Browser ("browser").dialog ("internet options").winbutton ("ok").Click

'*******************************************************************************
```

13. How to test Slideshow in a page

```
'*******************************************************************************
ExpectedImagePath=Environment("ProductDir")&"\Tests\SlideImageExpected.png"
ActualImagePath=Environment("ProductDir")&"\Tests\SlideImageActual.png"

Browser("Hindustan Times: Latest").Page("Hindustan Times:
```

```
Latest").Image("SlideImage").CaptureBitmap ExpectedImagePath,True
wait(1)
Browser("Hindustan Times: Latest").Page("Hindustan Times: Latest").Image("Next").Click
Browser("Hindustan Times: Latest").Page("Hindustan Times:
Latest").Image("SlideImage").CaptureBitmap ActualImagePath,True

If CompareBitmap( ActualImagePath, ExpectedImagePath) then
   Reporter.ReportEvent micFail, "SlideShow Test", "SlideShow is Not Running"
Else
   Reporter.ReportEvent micPass, "SlideShow Test", "SlideShow is Running"
End if
************************************************************
Function CompareBitmap (ActualBmp, ExpectedBmp)
 Set fCompare = CreateObject ("Mercury.FileCompare")

 If fCompare.IsEqualBin (ExpectedBmp, ActualBmp, 0, 1) Then
   CompareBitmap= True
  else
  CompareBitmap= False
  End If

End Function

'*****************************************************************************
```

14. Parameterize links in a web page

```
*****************************************************************************
Dim lnk_array
lnk_array=array ("link1","link2","link3","Link4")

For i= 0 to ubound(lnk_array)
Browser ("Google").Page ("Google").Frame ("body").Link ("Link_1").SetTOProperty "text",
lnk_array (i)
Browser ("Google").Page ("Google").Frame ("body").Link ("Link_1").Click
Next

'*****************************************************************************
```

15. Show all properties without using Object Spy

```
*****************************************************************************
Set oDesc=Description. Create
Set qtApp = CreateObject ("QuickTest.Application")
Set qtIdent = qtApp.Options.ObjectIdentification
qtIdent.ResetAll
Set objList=browser ("micclass: =Browser").page ("micclass: =Page").ChildObjects
(oDesc)
For iCounter=0 to objList.count-1
 objList (iCounter).highlight
 oClassName=objList (iCounter).getroproperty ("micclass")
 Set qtObject = qtIdent.Item (oClassName)
 set PropColl=qtObject.AvailableProperties
```

```
 print "*****************************************************"
 For oPropCount=1 to PropColl.count
  print PropColl.item (oPropCount) & ":="& objList (iCounter).getroproperty (PropColl.item
(oPropCount))
 Next
 print "*****************************************************"
Next
'*******************************************************************************
```

16. Deleting Browsing history, Temporary Files and Cookies

```
*******************************************************************************
'To clear temporary Internet files
Set WshShell = CreateObject("WScript.Shell")
WshShell.run "RunDll32.exe InetCpl.cpl,ClearMyTracksByProcess 8"

'To clear browsing cookies
WshShell.run "RunDll32.exe InetCpl.cpl,ClearMyTracksByProcess 2"

'To Clear Browsing History
WshShell.run "RunDll32.exe InetCpl.cpl,ClearMyTracksByProcess 1"

'*******************************************************************************
```

17. How to click on all Google Search Links in all Pages

```
*******************************************************************************
browser("micclass:=Browser").page("micclass:=Page").webedit("name:=q").Set "amit"
browser("micclass:=Browser").page("micclass:=Page").webbutton("name:=Google
Search").Click

NextExist=true

do until  NextExist=false

browser("micclass:=Browser").Sync

Set oDesc=Description. Create
oDesc ("micclass").value="Link"
set lnkList=browser ("micclass: =Browser").page ("micclass: =Page").childobjects (oDesc)

Set oDictionary=createobject("scripting. dictionary")
For i=0 to lnkList.count-1
   oDictionary.add i,lnkList(i).getROproperty("name")
Next
```

```
  LnkProperties = oDictionary.Items

For i = 0 To oDictionary.Count -1

   If i=oDictionary.count-1 then
      Exit for
   End if

   If LnkProperties(i+1)="Cached" Then

browser("micclass:=Browser").page("micclass:=Page").Link("name:="&LnkProperties(i)).cli
ck
      browser("micclass:=Browser").Back
      browser("micclass:=Browser").Sync
   End If
Next
NextExist=browser("micclass:=Browser").page("micclass:=Page").Link("name:=Next").Exis
t(1)
If NextExist Then
   browser("micclass:=Browser").page("micclass:=Page").Link("name:=Next").Click
End If
Loop

'*********************************************************************************
```

18. How to click on all Google Search Links in a Page

```
*********************************************************************************
browser("micclass:=Browser").page("micclass:=Page").webedit("name:=q").Set "amit"
browser("micclass:=Browser").page("micclass:=Page").webbutton("name:=Google
Search").Click
browser("micclass:=Browser").Sync

Set oDesc=Description. Create
oDesc ("micclass").value="Link"
set lnkList=browser ("micclass: =Browser").page ("micclass: =Page").childobjects (oDesc)

Set oDictionary=createobject("scripting. dictionary")
For i=0 to lnkList.count-1
   oDictionary.add i,lnkList(i).getROproperty("name")
Next

  LnkProperties = oDictionary.Items

For i = 0 To oDictionary.Count -1

   If i=oDictionary.count-1 then
      Exit for
   End if

   If LnkProperties(i+1)="Cached" Then
```

```
browser("micclass:=Browser").page("micclass:=Page").Link("name:="&LnkProperties(i)).cli
ck
        browser("micclass:=Browser").Back
        browser("micclass:=Browser").Sync
    End If
Next

'*****************************************************************************
```

19. How to get number of controls (Links, Edits, Images, etc)

```
*****************************************************************************
Function GetAllSpecificControls (Page, MicClass)
    Set Desc = Description. Create()
    Desc("micclass").Value = MicClass
    Set GetAllSpecificControls = Page.ChildObjects(Desc)
End Function

Function GetAllEdits(Page)
    Set GetAllEdits = GetAllSpecificControls(Page, "WebEdit")
End Function

Function GetAllButtons(Page)
    Set GetAllButtons = GetAllSpecificControls(Page, "WebButton")
End Function

Function GetAllLinks(Page)
    Set GetAllLinks = GetAllSpecificControls(Page, "Link")
End Function

Function GetAllImages(Page)
    Set GetAllImages = GetAllSpecificControls(Page, "Image")
End Function

Set oPage = Browser("Google Sets").Page("Google Sets")

MsgBox "Number of Edits: " & GetAllEdits(oPage).Count
MsgBox "Number of Buttons: " & GetAllButtons(oPage).Count
MsgBox "Number of Links: " & GetAllLinks(oPage).Count
MsgBox "Number of Images: " & GetAllImages(oPage).Count

'*****************************************************************************
```

19. Capturing the tool tip of the image

```
'*****************************************************************************
Set a = Description. Create
a (“html tag”).value=”IMG”
Set b = browser (“name: =Y.*”).page (“Y.*”).childobject (a)
Msgbox “No of images:”&b.count
        For I = 0 to b.count-1
        C = b (I).GetROProperty (“alt”)
        D= b (I).GetROProperty (“SRC”)
```

```
                If C <> “” then
                Msgbox “image SRC” &C
                Msgbox “Tool Tip” &D
                End if
        Loop
'*****************************************************************************
```

20. Launching Browser

```
'*****************************************************************************

'*****************************************************************************
```

21. Launching Browser

```
'*****************************************************************************
Set Browser =Createobject (“InternetExplorer.Application”)
Browser.visible = true
Browser. Navigate “www.google.com”

'*****************************************************************************
```

22. Close all browsers one by one

```
'*****************************************************************************
While Browser (“CreationTime: =0”).exist
        Browser (“CreationTime: =0”).Close
Wend
*****************************************************************************
```

23. Close all browsers one by one

```
'*****************************************************************************
While Browser (“CreationTime: =0”).exist
        Browser (“CreationTime: =0”).Close
Wend
*****************************************************************************
```

24. To close particular Browser

```
'*****************************************************************************
Set mask = “google.com”
CreationTime =”0”
        While Browser (CreationTime: =”&CreationTime).Exist
        Set URL = Browser (CreationTime: =”&CreationTime).GetROProperty (“url”)
                If Instr (URL.mask)>0 then
                Browser (CreationTime: =”&CreationTime).close
                Else
```

```
                CreationTime = CreationTime + 1
                End if
        Wend
*******************************************************************************
```

25. Get name of all open Browser

```
'*******************************************************************************
Set Browserdesc = Description. Create ()
Browserdesc (“version”).Value=”internet explorer 6”
Set browsercoll = Desktop.Childobject (Browserdesc)
Browsercnt = browsercoll.count
Msgbox Browsercnt
        For I = 0 to (Browsercnt-1)
        Msgbox “Browser” &I& “has title”=& browsercoll (I)
        Next
*******************************************************************************
```

26. To check extension of the image

```
'*******************************************************************************
Set filename = Browser (“Google”).Page (“Google”).Image (“India”).GetROProperty (“title
name”)
Set ext = Split (filename,”.”)
Msgbox ext (ubound (ext))
*******************************************************************************
```

26. Find out cookies in web page

```
'*******************************************************************************
Set cke = Browser (“Google”).Page (“Google”).Object. Cookie
Msgbox cke
*******************************************************************************
```

An overview of the *With* Statement

The VBScript ***with*** statement enables you to perform a series of operation on an object. You use the ***with*** statement to group related steps together and make a script easier to read. For example, using with grouping enables you to quickly see all the activities occurring on the same web page

The below code snippets that perform the same operations. The second code snippet uses the ***with*** statement to group steps

```
'*******************************************************************************

Browser (“Welcome Mercury Tours”).Page (“Mercury Tours”).WebEdit (“username”).Set
“kanak”
```

```
Browser (“Welcome Mercury Tours”).Page (“Mercury Tours”).WebEdit
(“username”).SetSecure “3980232390181hh121321”
Browser (“Welcome Mercury Tours”).Page (“Mercury Tours”).Image (“Sign-In”).Click
Browser (“Welcome Mercury Tours”).Page (“Mercury Tours”).Sync
Reporter.ReportEvent micDone,”Login”,”Login Completed”

'*******************************************************************************


'*******************************************************************************
With Browser (“Welcome Mercury Tours”)
        With Page (“Welcome Mercury Tours”)
                WebEdit (“username”).Set “kanak”
                WebEdit (“username”).SetSecure
                Image (“Sign-In”).Click
        End With
        Page (“Mercury Tours”).Sync
End With
Reporter.ReportEvent micDone,”Login”,”Login Completed”

'*******************************************************************************
```

About HP QTP Certification

Presently HP offers following two types of certification credentials to the individuals.

1) AIS - full form as “Accredited Integration Specialist”.

2) ASE - full form as “Accredited Systems Engineer”.

QTP Certification gets covered under following two parent streams by the broader name HP Quality Center v9.

1) **AIS - HP Quality Center v9**: Accredited Integration Specialist in HP Quality Center v9.

2) **ASE - HP Quality Center v9**: Accredited Systems Engineer in HP Quality Center v9.

We need to clear to get AIS - HP Quality Center v9 Certification?

1) Mandatory or Core Exam: Covers – Implementing HP Quality Center Software (Exam Code: HP0-M81)

2) Elective Exam: One can choose either of the following exams

a) HP QuickTest Professional 9.2 Software (Exam Code: HP0-M16)

Or

B) HP WinRunner 9.2 Software (Exam Code: HP0-M12)

We need to clear to get ASE - HP Quality Center v9 Certifications?

1) Mandatory or Core Exam: Covers – HP Quality Center 9.2 Software (Exam Code: HP0-M15)

2) Elective Exam: One can choose either of the following exams

a) Test Scripting using HP QuickTest Professional Software (Exam Code: HP0-M80)

Or

b) Test Scripting using HP WinRunner Software (Exam Code: HP0-M82)

HP does not provide any certification for individual exams may be the Mandatory / Core or the Elective exam. Hence to get the HP Certification, successful clearing of both Core exam as well as Elective exam is essential. After clearing the elective exams, a Score Card is provided by Parametric, which also commands great respect in the IT sector.

Exam preparation guides are available for all the certification exams. These guides provide following basic information like:

1) Objective: Purpose of the concerned exam preparation guide.

2) Audience: Who should appear in this exam.

3) Requirements: Basic requirements for the concerned certification.

4) Prerequisites: If any for appearing in the concerned exam.

5) Exam details: Covering information like

a) Number of questions in the exam - 58

b) Type of Questions - Multiple choices

c) Time commitment in hours - 2 hours

d) Percentage Required Passing the Exam -70%

6) Exam content: specific broad areas covered in the exam.

7) Sample Questions: 2 - 3 sample questions just to present an idea.

Reader's Note