QTP/UFT Material

Address:
#104,Nandini Residency,
Addagutta Society,
Pragathi Nagar Road
JNTU X-Roads,HYD .

Quick Test Professional (QTP)/Unified Functional Tester (UFT) is an automated functional Graphical User Interface (GUI) testing tool that allows the automation of user actions on a web or client based computer application.

It is primarily used for functional regression test automation. UFT uses a scripting language built on top of VBScript to specify the test procedure, and to manipulate the objects and controls of the application under test.

What is Test Automation?

Software Test Automation is the process of automating the steps of manual test cases using an automation tool or utility to shorten the testing life cycle with respect to time.

Benefits of Automated Testing

Manual testing is time-consuming and tedious, requiring a heavy investment in human resources. It is impossible to manually test every feature thoroughly before the application is released. This leaves you wondering whether serious bugs have gone undetected.

Automated testing with Quick Test addresses these problems by dramatically speeding up the testing process. You can create tests that check all aspects of your application or Web site, and then run these tests every time your site or application changes.

Benefits of Automated Testing	
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 users interface changes.

Factors to be considered in automation planning,

- Stability of AUT (Application under test)
- No of regression cycles to be performed
- Compatibility of App platform with testing tools
- Cost benefit analysis (ROI)
- Availability of skilled resources

Regression Testing & Automation

When Automation is applicable?

- Regression Testing Cycles are long and iterative.
- If the application is planned to have multiple releases / builds
- If it's a long running application where in small enhancements / Bug Fixes keeps happening
- Test Repeatability is required

Default Support

- Standard Windows applications
- Web objects / Applications
- ActiveX controls
- Visual Basic applications

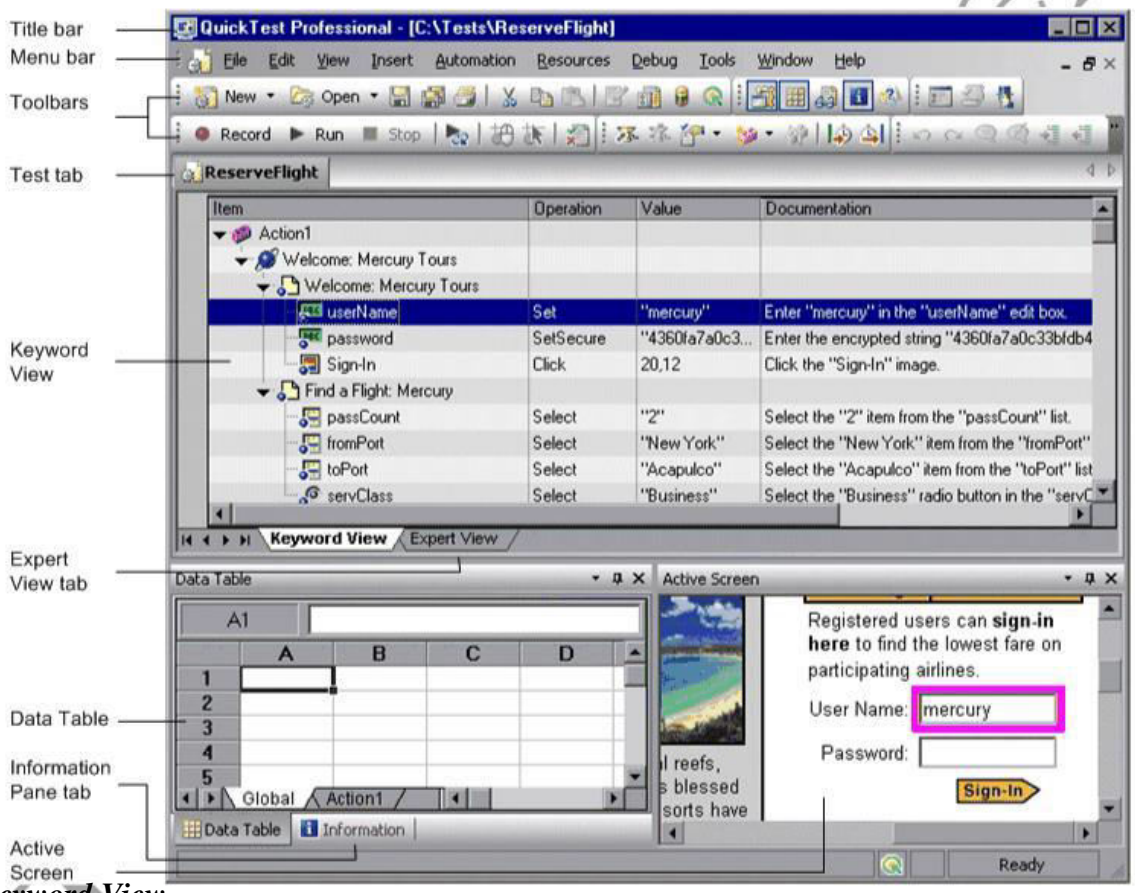
Additional UFT add-ins Support,

1. Java
2. Oracle
3. SAP Solutions,
4. .NET Windows
5. Web Forms,
6. Siebel,
7. PeopleSoft,
8. Web services, and terminal emulator applications, etc.

Quick Test Main window

Different Types of View

- In the Expert View, Quick Test displays each step as a VBScript line or statement.
- In the Keyword View, Quick Test displays information about each step and shows the object hierarchy in an icon-based table.



Keyword View

The Keyword View provides an easy way to create, view, and modify tests in a graphical easy-to-use format. As you recorded your test, Quick Test generated steps in the Keyword View representing each operation you performed in the application.

The Keyword View enables you to create and view the steps of your test in a modular, table format.

- **QuickTest title bar**—Displays the name of the currently open test.
- **Menu bar**—Displays menus of QuickTest commands.
- **File toolbar**—Contains buttons to assist you in managing your test.
- **Test toolbar**—Contains buttons to assist you in the testing process.
- **Debug toolbar**—Contains buttons to assist you in debugging your test (not displayed by default).
- **Action toolbar**—Contains buttons and a list of actions, enabling you to view the details of an individual action or the entire test flow (available only in the Tree View, not displayed by default).
- **Test pane**—Contains the Tree View and Expert View tabs.
- **Test Details pane**—Contains the Active Screen.
- **Data Table**—Assists you in parameterizing your test. The Data Table contains the **Global** tab and a tab for each action.
- **Debug Viewer pane**—Assists you in debugging your test. The Debug Viewer pane contains the **Watch Expressions**, **Variables**, and **Command** tabs (not displayed by default).
- **Status bar**—Displays the status of the QuickTest application.

❑ **Test Pane:**

- **Test Pane contains two labs to view the tests,**

  - Keyword View
  - Expert View

❑ **Keyword View:**

- Quick Test Pro displays your test in the form of a collapsible, icon based tree…

❑ **Expert View:**

- Quick Test Pro displays the source code (VB Script) of the tests in this view.

Quick Test Pro Commands

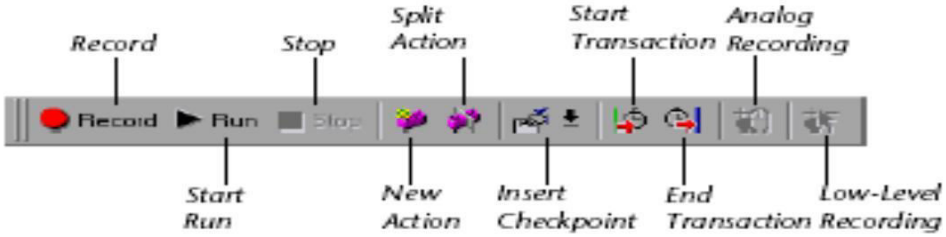
❑ Quick Test Pro Commands :

The Quick Test Pro commands can be chosen from the menu bar or from a Tool bar.

❑ **File Tool bar:** File tool bar contains buttons for managing the test.

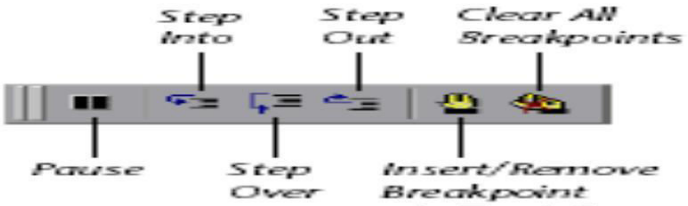
❑ **Test Tool bar :**

Test tool bar contains buttons for the commands used when creating or maintaining the tests.



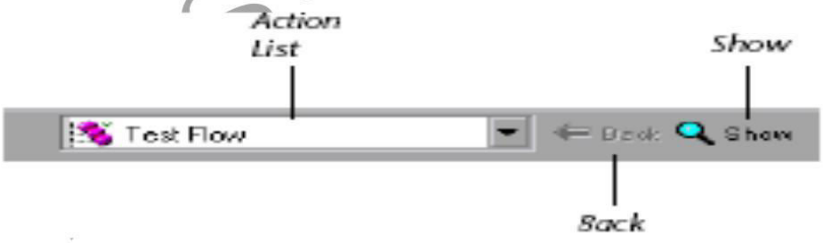
❑ **Debug Tool Bar :**

It contains buttons for commands used when debugging the steps in the tests.



❑ **Action Tool Bar :**

To view all actions in the test flow or to view the details of a selected action.



Understanding the Keyword View

The Keyword View is comprised of a table-like view, in which each step is a separate row in the table, and each column represents the different parts of the steps.

The Keyword View can contain any of the following columns: **Item**, **Operation**, **Value**, **Assignment**, **Comment**, and **Documentation**. A brief description of each column is provided below

- **Item.** The item for the step (test object, utility object, function call, or statement) in a hierarchical icon-based tree.
- **Operation.** The operation to be performed on the item.
- **Value.** The argument values for the selected operation
- **Documentation.** Auto-documentation of what the step does, in an easy-to-understand sentence.
- **Assignment.** The assignment of a value to or from a variable so you can use the value later in the test. This column is not visible by default.
- **Comment.** Any textual information you want to add regarding the step, for example, Return to page used in first step of the test. This column is not visible by default.

Item	Operation	Value	Documentation
Action1			
Welcome: Mercury Tours			
Welcome: Mercury Tours			
userName	Set	"mercury"	Enter "mercury" in the "userName" edit box.
password	SetSecure	"4082820183...	Enter the encrypted string "4082820183afe512e8bc91c1f7222db...
Sign-In	Click	2,2	Click the "Sign-In" image.
Find a Flight: Mercury			
fromPort	Select	"New York"	Select item "New York" in the "fromPort" list.
fromMonth	Select	"Dec"	Select item "Dec" in the "fromMonth" list.
fromDay	Select	"29"	Select item "29" in the "fromDay" list.
toPort	Select	"San Francisco"	Select item "San Francisco" in the "toPort" list.
toMonth	Select	"Dec"	Select item "Dec" in the "toMonth" list.
toDay	Select	"31"	Select item "31" in the "toDay" list.
servClass	Select	"Business"	Select radio button "Business" in the "servClass" radio button group.
findFlights	Click	2,2	Click the "findFlights" image.
Select a Flight: Mercury			
reserveFlights	Click	2,2	Click the "reserveFlights" image.
Book a Flight: Mercury			
passFirst0	Set	"Nicole"	Enter "Nicole" in the "passFirst0" edit box.
passLast0	Set	"Jones"	Enter "Jones" in the "passLast0" edit box.
creditnumber	Set	"12345"	Enter "12345" in the "creditnumber" edit box.
ticketLess	Set	"ON"	Set the state of the "ticketLess" check box to "ON".
buyFlights	Click	2,2	Click the "buyFlights" image.
Flight Confirmation: Mercury			
home	Click		Click the "home" image.
Welcome: Mercury Tours	Sync		Wait for the Web page to synchronize before continuing the run.

Keyword View | Expert View

For every step performed on an object, Quick Test displays a row in the Keyword View with an icon and details of the step.

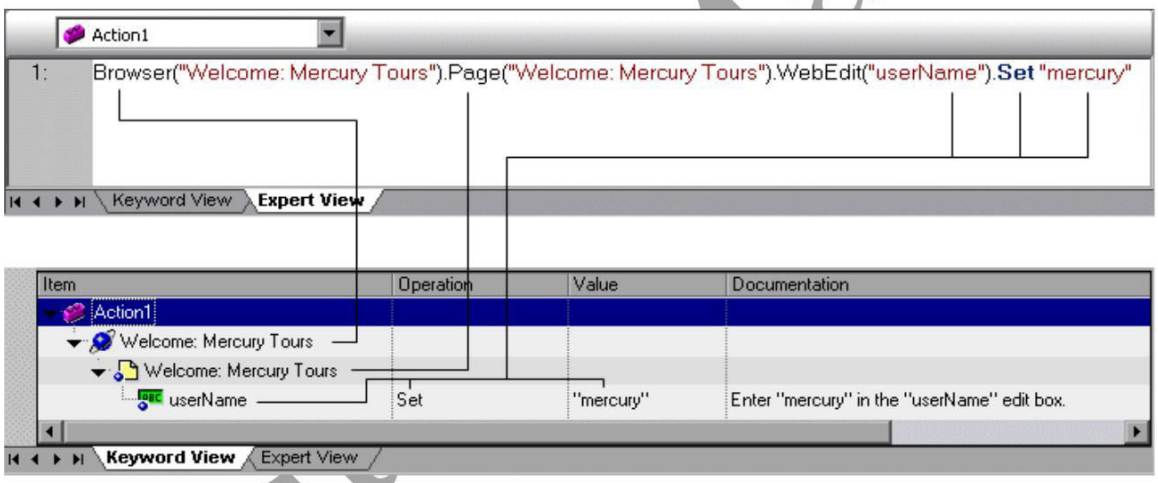
Expert View

You can use the Expert View tab to view a text-based version of your test. The test is composed of statements written in VBScript (Microsoft Visual Basic Scripting Edition) that correspond to the steps and checks displayed in the Keyword View.

The Expert View displays the same steps and objects as the Keyword View, but in a different format:

- In the Keyword View, Quick Test displays information about each step and shows the object hierarchy in an icon-based table.
- In the Expert View, Quick Test displays each step as a VBScript line or statement. In object-based steps, the VBScript statement defines the object hierarchy.

The following diagram shows how the same object hierarchy is displayed in the Expert View and in the Keyword View:



Quick Test Window Consists of the following

1. **Active Screen:** The Active Screen provides a snapshot of your application in a recording session and enables you to insert additional steps on the test objects captured in that screen after the recording session.
1. **Debug Viewer:** The Debug Viewer pane contains three tabs to assist you in debugging your function library—Watch, Variables, and Command. It assists in debugging tests with the help of Watch Expressions, Variables, and Command
2. **Data Table:** The Data Table assists you in parameterizing your test.
3. **Missing Resources Pane:** The Missing Resources pane provides a list of the resources that are specified in your test but cannot be found, such as missing calls to actions,

unmapped shared object repositories, and parameters that are connected to share object repositories.

To view the Active Screen pane, click the Active Screen button or choose View > Active Screen.

Missing Resource Pane

It provides a list of the resources that are specified in your test but cannot be found.

- Missing calls to actions
- Unmapped shared object repositories
- Parameters that are connected to shared object repositories.

If one of the resources listed in this pane is unavailable during a run session, the test may fail. You can map a missing resource, or you can remove it from the test, as required.

To show or hide the Missing Resources pane, choose View > Missing Resources or click the Missing Resource button. The Missing Resources pane may list any of the following types of missing resources:

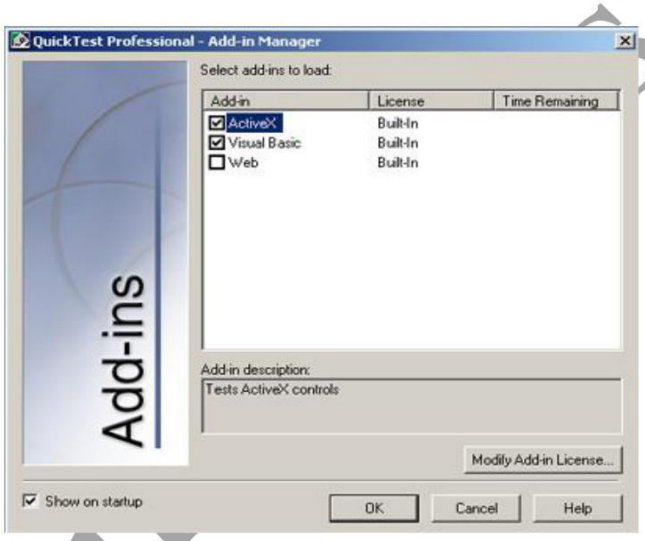
1. Missing Action Name
2. Missing Object Repository
3. Repository Parameters

ADD-IN MANAGER

Add-in: - It is a plug in program which provides compatibility between QTP and application.

There are two types of Add-in's:-

1. Build in
2. External



Build in: - By default some Add-ins are provided by Mercury interactive with QTP license these add-ins are called Built in Add-ins.

Ex: - ActiveX
Visual Basic
Web

External: - If you want to test the applications for example Java, .net etc., we have to buy the corresponding add-ins explicitly. This add-ins are called external add-ins.

Ex: - Java, .Net, Siebel, People soft, oracle apps, SAP, etc.

Add-in manager will be opened when we launch QTP. It displays the list of available add-ins with license type and time remaining. We can modify the existing add-in license with new license using "modify add in license button".

While developing/executing the script required add-ins should be selected in Add-in manager window.

QTP supported technologies and object classes

ActiveX Environment

Object	Description
ActiveX	An ActiveX control.
AcxButton	An ActiveX button.
AcxCalendar	An ActiveX calendar object.
AcxCheckBox	An ActiveX check box.
AcxComboBox	An ActiveX combo box object.
AcxEdit	An ActiveX edit box.
AcxRadioButton	An ActiveX radio button.
AcxTable	An ActiveX table.
AcxUtil	An object that enables you to work with objects returned by performing an operation (usually via the Object property) on an ActiveX test object.

Delphi Environment

Object	Description
DelphiButton	A Delphi button.
DelphiCheckBox	A Delphi check box.
DelphiComboBox	A Delphi combo box.
DelphiEdit	A Delphi edit box.
DelphiEditor	A Delphi multi-line editor.
DelphiList	A Delphi list.
DelphiListView	A Delphi list-view control.
DelphiNavigator	A Delphi navigator control.
DelphiObject	A Delphi object.
DelphiRadioButton	A Delphi radio button.
DelphiScrollBar	A Delphi scroll bar.

DelphiSpin	A Delphi spin box.
DelphiStatic	A Delphi static control.
DelphiStatusBar	A Delphi status bar.
DelphiTable	A Delphi table.
DelphiTabStrip	A Delphi tab strip.
DelphiTreeView	A Delphi tree-view control.
DelphiWindow	A Delphi window or dialog box.

Java Environment

Object	Description
JavaApplet	A Java applet.
JavaButton	A Java button.
JavaCalendar	A Java calendar.
JavaCheckBox	A Java check box.
JavaDialog	A Java dialog box.
JavaEdit	A Java edit box.
JavaExpandBar	A Java control that contains labeled bar items, which can be expanded or collapsed by the user.
JavaInternalFrame	An internal frame that can be activated from the Java applet.
JavaLink	A Java control that displays text with links.
JavaList	A Java list box with single or multiple selection.
JavaMenu	A Java menu item.
JavaObject	A generic Java object.
JavaRadioButton	A Java radio button.
JavaSlider	A Java slider.
JavaSpin	A Java spin object.
JavaStaticText	A Java static text object.
JavaTab	A Java tabstrip control containing tabbed panels.
JavaTable	A Java table.

JavaToolbar	A Java toolbar.
JavaTree	A Java tree.
JavaWindow	A Java window.

.NET Web Forms Environment

Object	Description
WbfCalendar	A .NET Web Forms calendar control.
WbfGrid	A .NET Web Forms DataGrid object.
WbfTabStrip	A .NET Web Forms tabstrip control.
WbfToolbar	A .NET Web Forms toolbar control.
WbfTreeView	A .NET Web Forms tree view object.
WbfUltraGrid	A .NET Web Forms UltraGrid object.

.NET Windows Forms Environment

Object	Description
SwfButton	A .NET Windows Forms button object.
SwfCalendar	A DateTimePicker or a Month Calendar .NET Windows Forms calendar object.
SwfCheckBox	A .NET Windows Forms check box.
SwfComboBox	A .NET Windows Forms combo box.
SwfEdit	A .NET Windows Forms edit box.
SwfEditor	A .NET Windows Forms multi-line edit box.
SwfLabel	A .NET Windows Forms static text object.
SwfList	A .NET Windows Forms list.
SwfListView	A .NET Windows Forms ListView control.
SwfObject	A standard .NET Windows Forms object.
SwfPropertyGrid	A property grid control based on the .NET Windows Forms library.
SwfRadioButton	A .NET Windows Forms radio button.
SwfScrollBar	A .NET Windows Forms scroll bar.
SwfSpin	A .NET Windows Forms spin object.
SwfStatusBar	A .NET Windows Forms status bar control.

SwfTab	A .NET Windows Forms tab control.
SwfTable	A grid control based on the .NET Windows Forms library.
SwfToolBar	A .NET Windows Forms toolbar.
SwfTreeView	A .NET Windows Forms TreeView control.
SwfWindow	A .NET Windows Forms window.

Windows Presentation Foundation Environment

Object	Description
WpfButton	A button control in a Windows Presentation Foundation application.
WpfCheckBox	A check box control in a Windows Presentation Foundation application.
WpfComboBox	A combo box control in a Windows Presentation Foundation application.
WpfEdit	A document, rich text box, or text control in a Windows Presentation Foundation application.
WpfGrid	A grid control in a Windows Presentation Foundation application.
WpfImage	An image control in a Windows Presentation Foundation application.
WpfLink	A hyperlink control in a Windows Presentation Foundation application.
WpfList	A list control in a Windows Presentation Foundation application.
WpfMenu	A menu control in a Windows Presentation Foundation application.
WpfObject	An object control in a Windows Presentation Foundation application.
WpfProgressBar	A progress bar control in a Windows Presentation Foundation application.
WpfRadioButton	A radio button control in a Windows Presentation Foundation application.
WpfScrollBar	A scroll bar control in a Windows Presentation Foundation application.
WpfSlider	A slider control in a Windows Presentation Foundation application.
WpfStatusBar	A status bar control in a Windows Presentation Foundation application.
WpfTabStrip	A tab control in a Windows Presentation Foundation application.

WpfToolbar	A toolbar control in a Windows Presentation Foundation application.
WpfTreeView	A tree control in a Windows Presentation Foundation application.
WpfWindow	A window control in a Windows Presentation Foundation application.

Oracle Environment

Object	Description
OracleApplications	An Oracle Applications session window.
OracleButton	An Oracle button.
OracleCalendar	An Oracle calendar.
OracleCheckbox	A check box Oracle field.
OracleFlexWindow	An Oracle flexfield window.
OracleFormWindow	An Oracle Form window.
OracleList	An Oracle poplist (combo box) or list.
OracleListOfValues	An Oracle window containing a list of values for selection.
OracleLogon	An Oracle Applications sign-on window.
OracleNavigator	An Oracle Navigator window.
OracleNotification	An Oracle error or message window.
OracleRadioGroup	An Oracle option (radio button) group.
OracleStatusLine	The status line and message line at the bottom of an Oracle Applications window.
OracleTabbedRegion	An Oracle tabbed region.
OracleTable	An Oracle block of records.
OracleTextField	An Oracle text field.
OracleTree	An Oracle tree.

PeopleSoft Environment

Object	Description
PSFrame	A frame object within a PeopleSoft application.

PowerBuilder Environment

Object	Description

PbButton	A PowerBuilder button.
PbCheckBox	A PowerBuilder check box.
PbComboBox	A PowerBuilder combo box.
PbDataWindow	A PowerBuilder DataWindow control.
PbEdit	A PowerBuilder edit box.
PbList	A PowerBuilder list.
PbListView	A PowerBuilder listview control.
PbObject	A standard PowerBuilder object.
PbRadioButton	A PowerBuilder radio button.
PbScrollBar	A PowerBuilder scroll bar.
PbTabStrip	A PowerBuilder tab strip control
PbTreeView	A PowerBuilder tree-view control.
PbWindow	A PowerBuilder window.

SAP Web Environment

Object	Description
SAPButton	An SAP Gui for HTML application button, including icons, toolbar buttons, regular buttons, buttons with text, and buttons with text and image.
SAPCalendar	A calendar in a Web-based SAP application.
SAPCheckBox	An SAP Gui for HTML application toggle button, including check boxes and toggle images.
SAPDropDownMenu	A menu that is opened by clicking a menu icon within an SAP Gui for HTML application.
SAPEdit	An SAP Gui for HTML application edit box, including single-line edit boxes and multi-line edit boxes (text area).
SAPFrame	An SAP Gui for HTML application frame.
SAPiView	An SAP Enterprise Portal application iView frame.
SAPList	A drop-down or single/multiple selection list in an SAP Gui for HTML application.
SAPMenu	An SAP Gui for HTML application top-level menu.
SAPNavigationBar	A navigation bar in a Web-based SAP application.
SAPOKCode	An OK Code box in an SAP Gui for HTML application.
SAPPortal	An SAP Enterprise Portal desktop.
SAPRadioGroup	An SAP Gui for HTML application radio button group.
SAPStatusBar	An SAP Gui for HTML application status bar.
SAPTable	An SAP Gui for HTML application table or grid.

SAPTabStrip	An SAP Gui for HTML application tab strip object (an object that enables switching between multiple tabs).
SAPTreeView	An SAP Gui for HTML application tree object.

SAP GUI for Windows Environment

Object	Description
SAPGuiAPOGrid	An APO grid control in an SAP GUI for Windows application.
SAPGuiButton	A button in an SAP GUI for Windows application.
SAPGuiCalendar	A calendar object in an SAP GUI for Windows application.
SAPGuiCheckBox	A check box in an SAP GUI for Windows application.
SAPGuiComboBox	A combo box in an SAP GUI for Windows application.
SAPGuiEdit	An edit box in an SAP GUI for Windows application.
SAPGuiElement	Any object in an SAP GUI for Windows application.
SAPGuiGrid	A grid control in an SAP GUI for Windows application.
SAPGuiLabel	A static text label in an SAP GUI for Windows application.
SAPGuiMenubar	A menu bar in an SAP GUI for Windows application.
SAPGuiOKCode	An OK Code box in an SAP GUI for Windows application.
SAPGuiRadioButton	A radio button in an SAP GUI for Windows application.
SAPGuiSession	Represents the SAP GUI for Windows session on which an operation is performed.
SAPGuiStatusBar	A status bar in an SAP GUI for Windows application.
SAPGuiTable	A table control in an SAP GUI for Windows application.
SAPGuiTabStrip	A tab strip in an SAP GUI for Windows application.
SAPGuiTextArea	A text area in an SAP GUI for Windows application.
SAPGuiToolbar	A toolbar in an SAP GUI for Windows application.
SAPGuiTree	A column tree, list tree, or simple tree control in an SAP GUI for Windows application.
SAPGuiUtil	A utility object in an SAP GUI for Windows application.
SAPGuiWindow	A window or dialog box containing objects in an SAP GUI for Windows application.

Siebel Environment

Object	Description
SblAdvancedEdit	An edit box whose value can be set by a dynamic object that opens after clicking on a button inside the edit box

SblButton	A Siebel button.
SblCheckBox	A check box with an ON and OFF state.
SblEdit	An edit box.
SblPickList	A drop-down pick list.
SblTable	A Siebel table containing a variable number of rows and columns.
SblTabStrip	A number of tabs and four arrows that move its visible range to the left and to the right.
SblTreeView	A tree view of specific screen data.
SiebApplet	An applet in a Siebel test automation environment.
SiebApplication	An application in a Siebel test automation environment.
SiebButton	A button control in a Siebel test automation environment.
SiebCalculator	A calculator control in a Siebel test automation environment.
SiebCalendar	A calendar control in a Siebel test automation environment.
SiebCheckbox	A checkbox in a Siebel test automation environment.
SiebCommunicationsToolbar	The communications toolbar in a Siebel test automation environment.
SiebCurrency	A currency calculator in a Siebel test automation environment.
SiebList	A list object in a Siebel test automation environment.
SiebMenu	A menu or menu item in a Siebel test automation environment.
SiebPageTabs	A page tab in a Siebel test automation environment.
SiebPDQ	A predefined query in a Siebel test automation environment.
SiebPicklist	A pick list in a Siebel test automation environment.
SiebRichText	A rich text control in a Siebel test automation environment.
SiebScreen	A screen object in a Siebel test automation environment.
SiebScreenViews	A screen view in a Siebel test automation environment.
SiebTaskAssistant	The Task Assistant in a Siebel test automation environment.
SiebTaskUIPane	The task UI pane in a Siebel test automation environment.
SiebText	A text box in a Siebel test automation environment.
SiebTextArea	A text area in a Siebel test automation environment.
SiebThreadbar	A threadbar in a Siebel test automation environment.
SiebToolbar	A toolbar in a Siebel test automation environment.
SiebTree	A tree view object in a Siebel test automation environment.
SiebView	A view object in a Siebel test automation environment.
SiebViewApplets	A view applet in a Siebel test automation environment.

Standard Windows Environment

Object	Description
Desktop	An object that enables you to access top-level items on your desktop.
Dialog	A Windows dialog box.
Static	A static text object.
SystemUtil	An object used to control applications and processes during a run session.
WinButton	A Windows button.
WinCalendar	A Windows calendar.
WinCheckBox	A Windows check box.
WinComboBox	A Windows combo box.
Window	A standard window.
WinEdit	A Windows edit box.
WinEditor	A Windows multi-line editor.
WinList	A Windows list.
WinListView	A Windows list-view control.
WinMenu	A Windows menu.
WinObject	A standard (Windows) object.
WinRadioButton	A Windows radio button.
WinScrollBar	A Windows scroll bar.
WinSpin	A Windows spin box.
WinStatusBar	A Windows status bar.
WinTab	A Windows tab strip in a dialog box.
WinToolbar	A Windows toolbar.
WinTreeView	A Windows tree-view control.

Stingray Environment

Object	Description
WinTab	A Windows tab strip in a dialog box.

WinTable	A Stingray grid.
WinToolbar	A Windows toolbar.
WinTreeView	A Stingray tree control.

Terminal Emulators Environment

Object	Description
TeField	A terminal emulator field that fully supports HLLAPI.
TeScreen	A terminal emulator screen that fully supports HLLAPI.
TeTextScreen	A terminal emulator screen that uses text-only HLLAPI or does not support HLLAPI.
TeWindow	A terminal emulator window.

Visual Basic Environment

Object	Description
VbButton	A Visual Basic button.
VbCheckBox	A Visual Basic check box.
VbComboBox	A Visual Basic combo box.
VbEdit	A Visual Basic edit box.
VbEditor	A Visual Basic multi-line editor.
VbFrame	A Visual Basic frame.
VbLabel	A static text object.
VbList	A Visual Basic list.
VbListView	A Visual Basic list-view control.
VbRadioButton	A Visual Basic radio button.
VbScrollBar	A Visual Basic scroll bar.
VbToolbar	A Visual Basic toolbar.
VbTreeView	A Visual Basic tree-view control.
VbWindow	A Visual Basic window.

VisualAge Smalltalk Environment

Object	Description

WinButton	A button in the VisualAge Smalltalk application.
WinEdit	An edit box in the VisualAge Smalltalk application.
WinList	A list in the VisualAge Smalltalk application.
WinObject	An object in the VisualAge Smalltalk application.
WinTab	A tab strip in the VisualAge Smalltalk application.
WinTable	A table in the VisualAge Smalltalk application.
WinTreeView	A tree-view control in the VisualAge Smalltalk application.

Web Environment

Object	Description
Browser	A Web browser (or browser tab).
Frame	An HTML frame.
Image	An image with or without a target URL link.
Link	A hypertext link.
Page	An HTML page.
ViewLink	A Viewlink object.
WebArea	A section of an image (usually a section of a client-side image map).
WebButton	An HTML button.
WebCheckBox	A check box with an ON and OFF state.
WebEdit	An edit box, usually contained inside a form.
WebElement	A general Web object that can represent any Web object.
WebFile	An edit box with a browse button attached, used to select a file from the File dialog box.
WebList	A drop-down box or multiple selection list.
WebRadioGroup	A set of radio buttons belonging to the same group.
WebTable	A table containing a variable number of rows and columns.
WebXML	An XML document contained in a Web page.

Web Services Environment

Object	Description
Attachments	An object that supports attachment-related test object operations.
Configuration	An object that supports configuration-related test object operations.
headers	An object that supports header-related test object operations.
Security	An object that supports security-related test object operations.
WebService	A test object representing a Web service.
WSUtil	A utility object used to check WSDL files.

RECORDING MODES

QTP provides the feature called recording and playback. When test engineer performs actions on application QTP records each action and generates the corresponding VBScript statements on QTP test pane (IDE). If you execute these script statements, QTP repeats same actions on application (AUT).

There are three types of recording modes:-

1. Normal Recording Mode
2. Analog Recording Mode
3. Low Level Recording Mode

Differentiate Recordings Modes

Normal Recording	**Analog Recording**	**Low level Recording**
Records Object information based on test object model.	Records Mouse Movements respect to window or screen.	Records all objects as window or win objects.
Records Keyboard i/p's.	Records Keyboard i/p's	Records Keyboard i/p's Records mouse clicks with co-ordinates.
Records Mouse Clicks.	Records Mouse clicks.	Records object information in the form of window or win objects.
Object Information will store in object repository.	It can't record object information.	
It is possible to edit the scripts after recording with in QTP.	Not possible to edit the script after recording with in QTP.	It is possible to edit the scripts after recording with in QTP.
Recording a signature is not possible	Alternative mode for recording signatures.	Recorded steps may not run correctly on all objects.
Recorded steps will run	When we are recording with respect to window,	supports the following methods for each test object: **WinObject test objects:** Click, DblClick, Drag,

correctly on all objects. Support all methods for every object. This is Default Recording method.	the window object information will stores in object repository. Analog recording and low-level recording require more disk space than normal recording mode.	Drop, Type **Window test objects:** Click, DblClick, Drag, Drop, Type, Activate, Minimize, Restore, Maximize

1. Normal Recording Mode: - It is used to record objects and operations performed on them. Mouse movements are not recorded in this recoding mode. It is a default recording mode in QTP.

Navigation: - Automation Menu -> Record.

Ex: - Click on button
Enter values into text box
Select Radio buttons etc.

QTP Script Syntax:

Object. operation [input value]

Object -> Test object hierarchy

Operation -> Operation/action to be performed on object

Value -> input value for object

Example:

Browser ("Gmail").Page ("Gmail").WebEdit ("Mail Id").Set "info@testingmasters.com"
Browser ("Gmail").Page ("Gmail").WebEdit ("Password").Set "abc123" Browser ("Gmail").Page ("Gmail").WebButton ("Sign in").Click

In above example Browser, Page, WebEdit are Test object class names for Web application.

Below are few sample object class names for different technologies.

Technology	Object class names
Web Application	Browser, Page, Frame, WebTable, Link, WebList, WebEdit, WebButton, WebCheckbox, WebRadioGroup, Image, WebElement, WebArea, WebFile,etc..
Standard Windows	Window, Dialog, WinButton, WinEdit, WinCheckBox, WinRadioButton, WinComboBox, Static,etc..
Activex Controls	Activex, AcxButton, AcxEdit, AcxRadioButton,etc..
Visual Basic (Windows)	VbWindow, VbEdit, VbButton, VbComboBox, VbCheckBox, VbRadioButton,etc..
Java (Windows)	JavaWindow, JavaEdit, JavaButton, JavaTable, etc..
.NET	SwfWindow, swfEdit, swfButton, etc..

Exercise 1: Write a script to login Gmail Page

Steps:

step1. Open Gmail login page

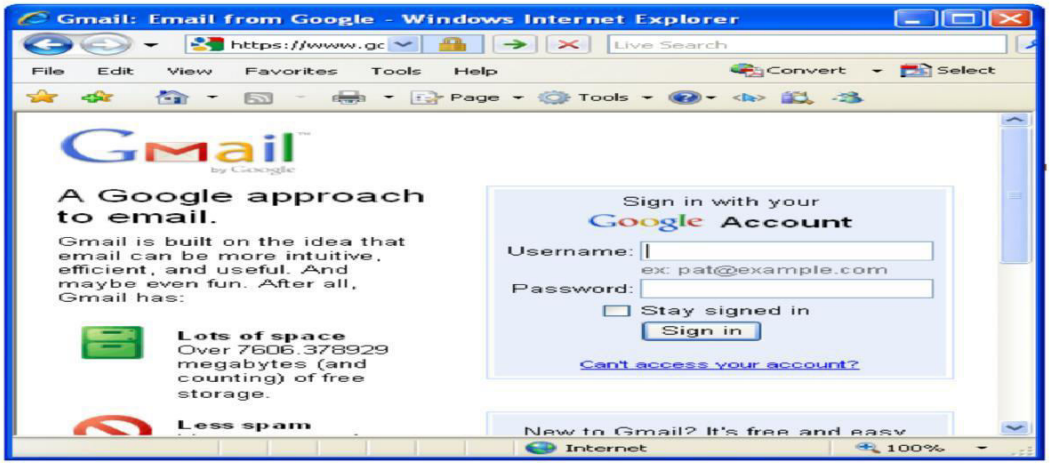
step2. Enter email id

step3. Enter password

step4. Click on sign in button

step5. Verify Gmail Inbox page is displayed

Application snapshot:



QTP Script:

```
Systemutil.Run "iexplorer","http://www.gmail.com"

Browser("Gmail_Login").Page("Gmail_Login").WebEdit("Email").Set "toveerender@gmail.com"
Browser("GmGmail_Loginail").Page("Gmail_Login").WebEdit("Passwd").SetSecure "4e26aa07c12f531ab32189bb8d0236feae3e"
Browser("Gmail_Login").Page("Gmail_Login").WebButton("Sign in").Click

If Browser("Gmail_Inbox").Page("Gmail_Inbox").Exist Then
 msgbox "Gmail - inbox page displayed,pass"
 Browser("Gmail_Inbox").Close
Else
 msgbox "Gmail - inbox page not displayed,Fail"
End If
```

Methods used in above script:

Systemutil.Run -> Used to run a specified file or application

Set -> Used to set value to edit box

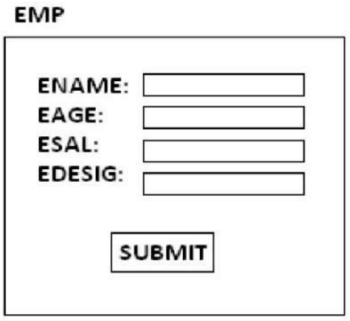
Setsecure -> Used to set encrypted text to password edit box

Click -> Used to click on button

Exist -> Used to check availability of object on AUT. If object exists it will return True, if not exist it will return False

Msgbox -> Displays user message in popup window

Script Examples for different technologies:



QTP for Visual Basic application:

```
VbWindow("emp")VbEdit("ename").set "Kumar"
VbWindow("emp")VbEdit("eage").set "26"
VbWindow("emp")VbEdit("esal").set "40,000"
VbWindow("emp")VbEdit("edesig").set "SE"
VbWindow("emp")VbButton("submit").click
```

QTP for standard Windows application:

```
Window("emp")WinEdit("ename").set "Kumar"
Window("emp")WinEdit("eage").set "26"
Window("emp")WinEdit("esal").set "40,000"
Window("emp")WinEdit("edesig").set "SE"
Window("emp")WinButton("submit").click
```

QTP for Java application:

```
JavaWindow("emp")JavaEdit("ename").set "Kumar"
```

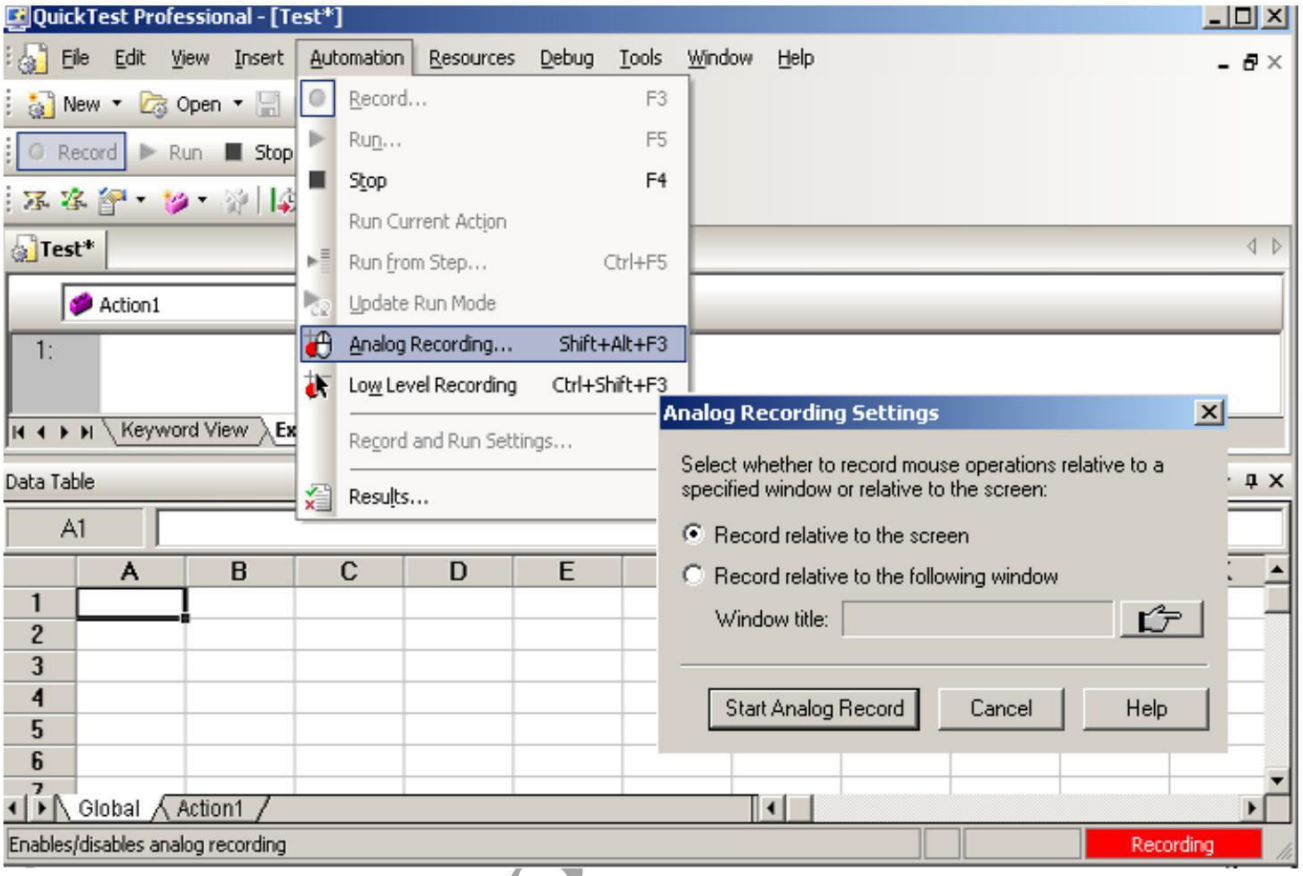
```
JavaWindow("emp")JavaEdit("eage").set "26"
JavaWindow("emp")JavaEdit("esal").set "40,000"
JavaWindow("emp")JavaEdit("edesig").set "SE"
JavaWindow("emp")JavaButton("submit").click
```

QTP for .NET application:

```
SWFWindow("emp")SWFEdit("ename").set "Kumar"
SWFWindow("emp")SWFEdit("eage").set "26"
SWFWindow("emp")SWFEdit("esal").set "40,000"
SWFWindow("emp")SWFEdit("edesig").set "SE"
SWFWindow("emp")SWFButton("submit").click
```

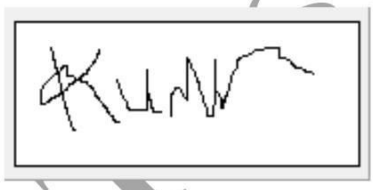
2. Analog Recording Mode: - When the tool is in this recording mode it records the continuous movements performed on the application through keyboard or mouse with respect to desktop or application window, and it records the co-ordinates of movements.

Ex: - Drawing Pictures, Signatures, Graphs, etc.



Script Example:

Desktop. Run Analog “Track1”
Window (“Flight reservation”), Window (“Fax order”). Run Analog “Track1”



Navigation:-

Automation Menu -> Start Normal Recording -> Select Analog Recording -> Select Record relative to the following window -> Click on Start Analog Record button

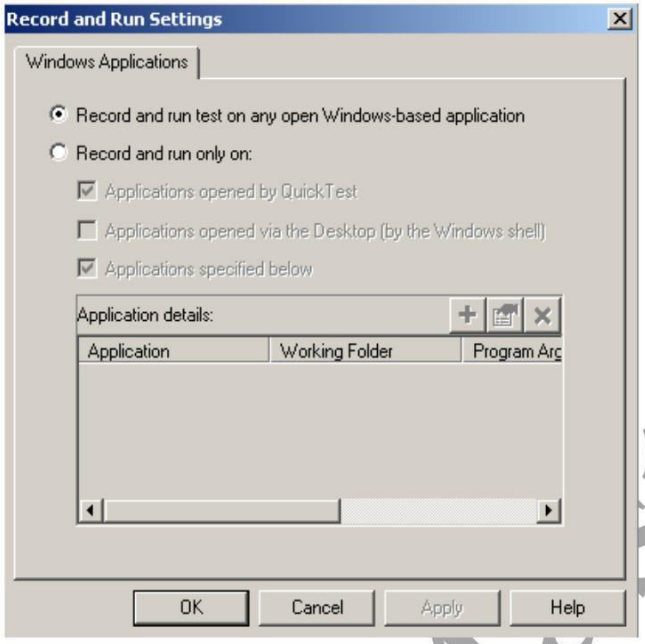
Low Level Recording: - It is used to record operations performed on the objects whose environment is not supported by QTP.

Navigation:-

Automation Menu -> Start Normal Recording -> Select Low level Recording

Record & Run Settings

Record and Run settings are used to instruct to QTP, whether it has to Record/Run the Script on already opened application or application has to be opened by QTP itself.



There are two options:-

a. Record & run test on any opened applications
b. Record & run only on specified applications

If we select 1st option Quick Test records the script on any application which is opened by test engineer. If we select 2nd option by specifying application details, Quick Test records only on the specified application and it is opened by QTP when we start Recording or script execution.

Navigation:

Automation Menu -> -Record & Run Settings -> Select Required Option (if we select 2nd option specify application details by clicking on '+' icon button.

Automation Testing Process with QTP

A) The QuickTest testing process consists of the following main phases:

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.

Preparing to record

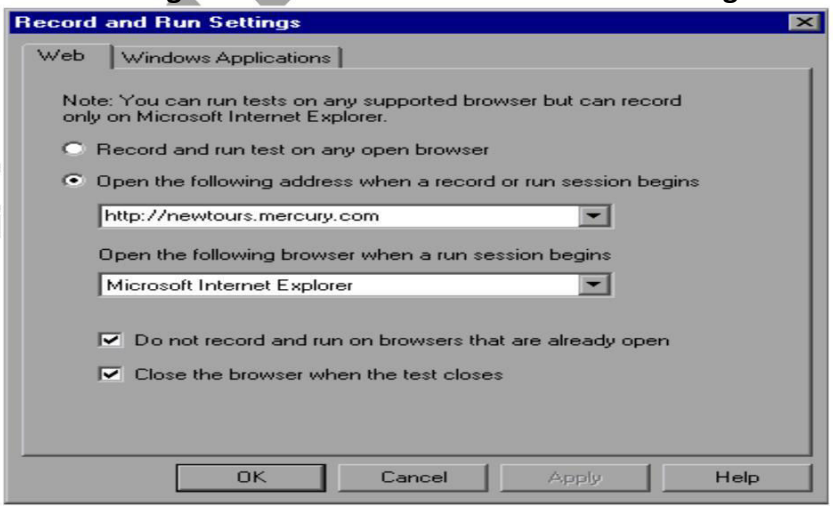
- Test Environment
- Test Conditions

Recording an application

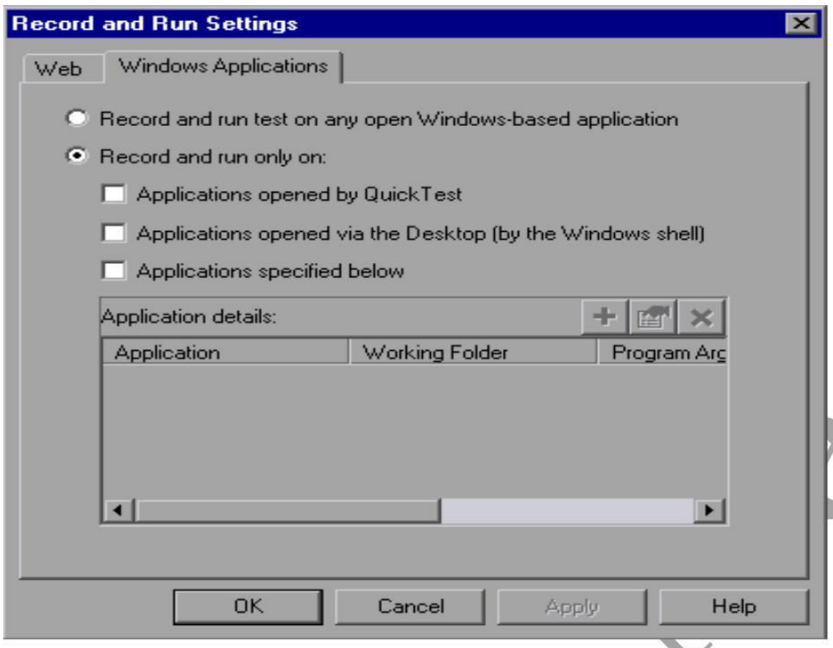
- Navigate according to Test Conditions

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.

1. Start Quick Test and open a new test
2. Choose **Automation > Record** or click the **Record** button. The Record and Run Settings dialog box opens.
   a. If you are working with web application, in the Web tab, select **Open the following address when a record or run session begins**.



b. If you are working with windows application, in Window Applications tab, confirm that Record and run only on is selected, and make sure that all three check boxes are cleared.



Enhancing your test

- Adding logic and conditional statements
- Parameterization
- Inserting checkpoints

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.

Debug Viewer : Assists you in debugging

- **The Debug Viewer pane contains three tabs to assist in debugging the test or function library — Watch, Variables, and Command.**
  - **Watch :** The Watch tab enables you to view the current value of any variable or VBScript expression that you added to the Watch tab.
  - **Variables:** During a run session, the Variables tab displays the current value of all variables that have been recognized up to the last step performed in the run session.
  - **Command:** The Command tab enables you to run a line of script to set or modify the current value of a variable or VBScript object in your test or function library. When you continue the run session, QuickTest uses the new value that was set in the command.

Debugging your test

- Check that it operates smoothly and without interruption.

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

Using Breakpoints: You can use breakpoints to instruct Quick Test to pause a run session at a predetermined place in a test or function library. Quick Test pauses the run when it reaches the breakpoint, before executing the step. You can then examine the effects of the run up to the breakpoint, make any necessary changes, and continue running the test or function library from the breakpoint.

You can use breakpoints to:

- suspend a run session and inspect the state of your site or application
- mark a point from which to begin stepping through a test or function library

Setting Breakpoints

- Click in the left margin of a step in the test or function library where you want the run to stop
- Click a step and then:
    - Click the **Insert/Remove Breakpoint** button
    - Choose **Debug > Insert/Remove Breakpoint**

To enable/disable a specific breakpoint:

1. Click in the line containing the breakpoint you want to disable/enable.
2. Choose **Debug > Enable/Disable Breakpoint** or press CTRL+F9. The breakpoint is either disabled or enabled (depending on its previous state).

Using the Single Step Commands

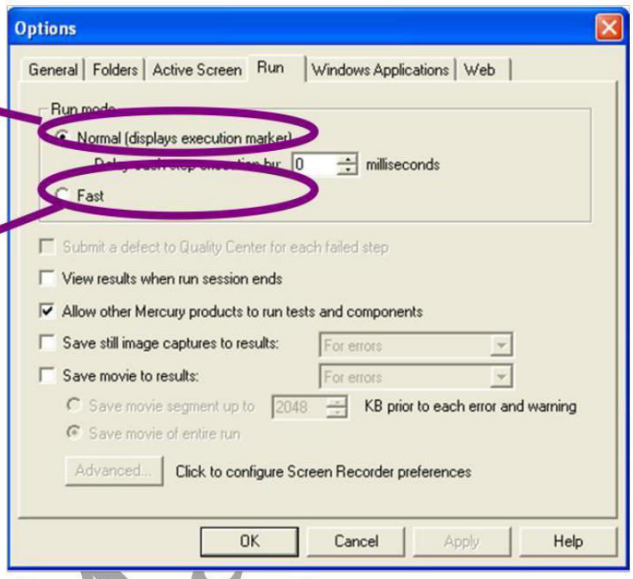
You can run a single step of a test or function library using the **Step Into**, **Step Out**, and **Step Over** commands.

1. **Step Into**: Choose Debug > Step Into, click the Step Into button, or press F11 to run only the current line of the active test or function library. If the current line of the active test or function library calls another action or a function, the called action/function is displayed in the Quick Test window, and the test or function library pauses at the first line of the called action/function.
2. **Step Out:** Choose **Debug > Step Out**, click the **Step Out** button, or press SHIFT+F11 only after using **Step Into** to enter an action or a user-defined function. **Step Out** runs to the end of the called action or user-defined function, then returns to the calling test or function library and pauses the run session.
3. **Step Over**: Choose **Debug > Step Over**, click the **Step Over** button, or press F10 to run only the current step in the active test or function library. When the current step calls another action or a user-defined function, the called action or function is executed in its entirety, but the called action or function script is not displayed in the Quick Test window.

Choose **Debug > Step Over**, click the **Step Over** button, or press F10 to run only the current step in the active test or function library. When the current step calls another action or a user-defined function, the called action or function is executed in its entirety, but the called action or function script is not displayed in the Quick Test window.

RUN MODES

- **There are two execution (Run) modes available in QTP.**
  - **Normal**
    - Shows the execution arrow during the script execution.
    - We can instruct QTP to wait between each step's execution.
    - This mode is useful for debugging the script.
  - **Fast**
    - Runs your component without the execution arrow.
    - This option requires fewer system resources comparing to the Normal execution mode.



Running your test

When you run your test, Quick Test opens the appropriate application or Web site and performs each step as it was originally recorded in the test. When Quick Test finishes running the test, it displays the results of the run.

- **Run Test** : Check the behavior of your application

Running your test on a new version of your application

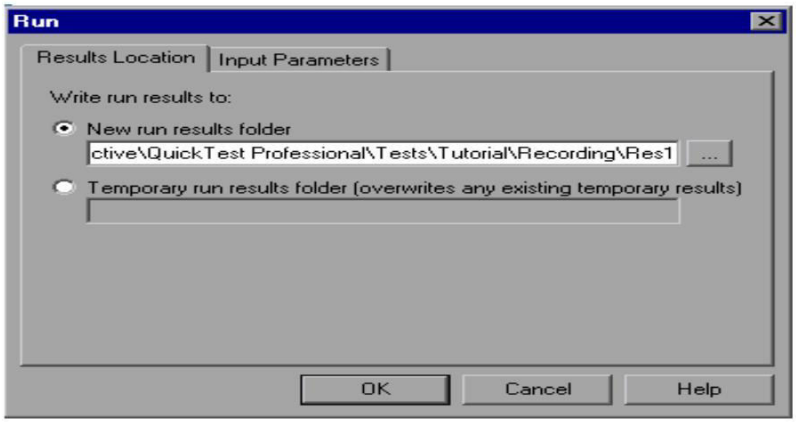
You run a test to check the behavior of your application. While running QuickTest connects to your application and performs each step in your test.

When you run your test, Quick Test opens the appropriate application or Web site and performs each step as it was originally recorded in the test. When Quick Test finishes running the test, it displays the results of the run.

Click **Run** or choose **Automation** > **Run**. The Run dialog box opens.

Select **New run results folder**. Accept the default results folder name.

Click **OK** to close the Run dialog box.



Results Analysis

Analyzing the test results

- You examine the test results to pinpoint defects in your application. After running a test, you can view a report of major events that occurred during the run session.
- When a run session ends, you can view the run session results in the Test Results window. By default, the Test Results window opens automatically at the end of a run. If you want to change this behavior, clear the **View results when run session ends** check box in the Run tab of the Options dialog box.
- The Test Results window contains a description of the steps performed during the run session. For a test that does not contain Data Table parameters, the Test Results window shows single test iteration.
- If the test contains Data Table parameters, and the test settings are configured to run multiple iterations, the Test Results window displays details for each iteration of the test run. The results are grouped by the actions in the test.

  Major events that occur during test run can be captured in images.

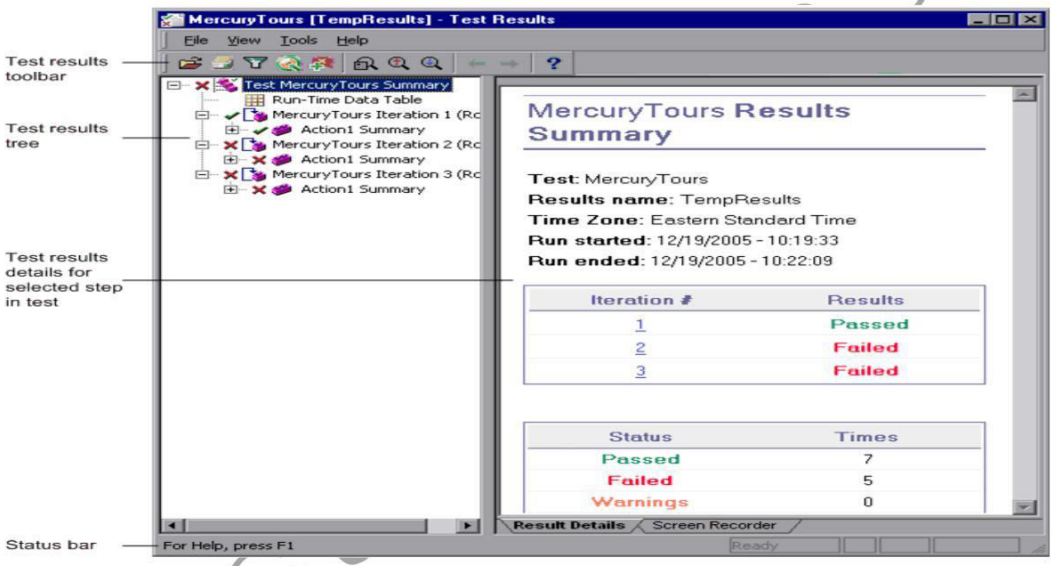
  The test execution can be captured in a movie file for further presentations or for debugging.

The Test Results Window

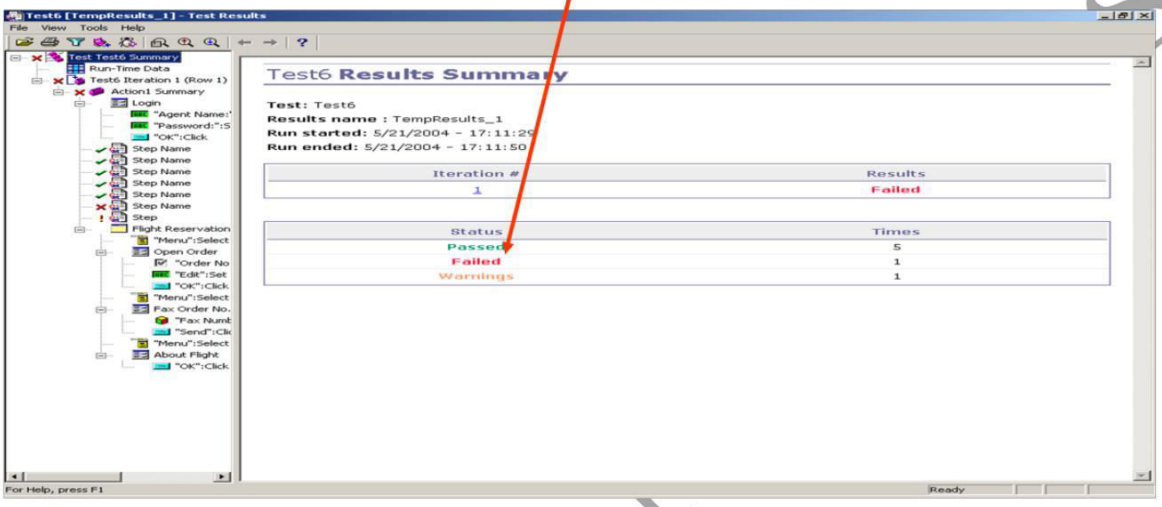
After a run session, you view the results in the Test Results window. By default, the Test Results window opens when a run session is completed.

The left pane of the Test Results window contains the test results tree. The right pane of the Test Results window contains the details for a selected step in the test results tree. The details for a selected step may include a test summary, step details, a still image of your application, or a movie of your application.

You can open the Test Results window as a standalone application from the **Start** menu. To open the Test Results window, choose **Start > Programs > Quick Test Professional > Test Results Viewer**. The following example shows the results of a test with three iterations:

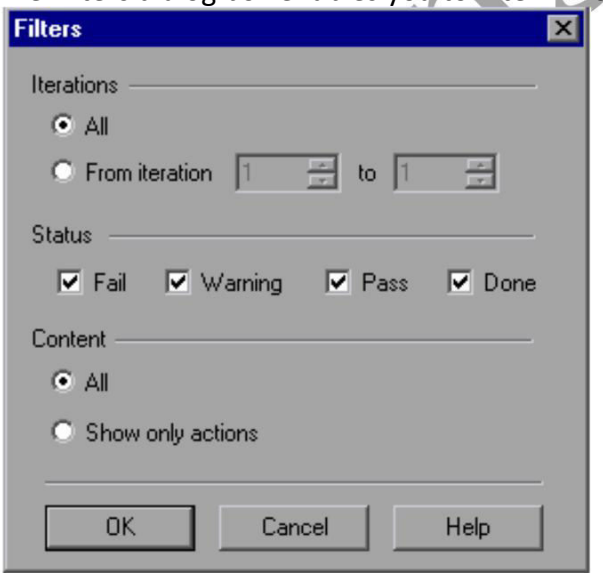


```
Reporter.ReportEvent micPass,"Step Name", "Log in successfully"
Reporter.ReportEvent micFail,"Step Name", "Log in Failed"
Reporter.ReportEvent micWarning,"Step Name", "Log in pending"
Reporter.ReportEvent micDone,"Step Name", "Log in pending"
```



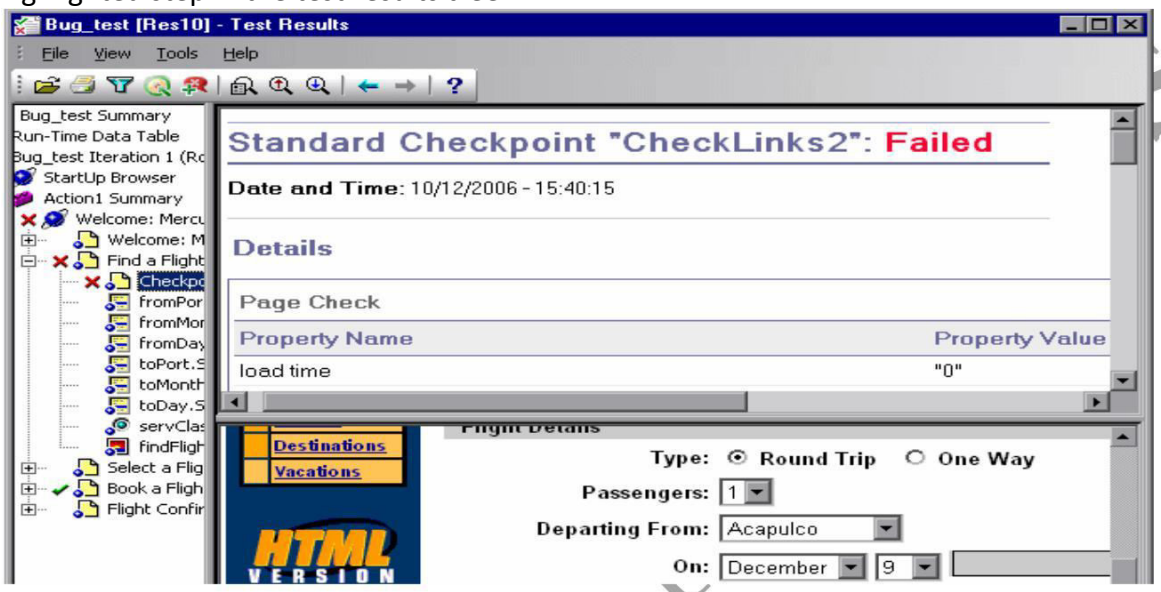
Filtering Test Results

The Filters dialog box enables you to filter which iterations are displayed in the test results tree.



Viewing Still Images of Your Application

By default, Quick Test saves a still image of your application for failed steps. When you select a failed step in the test results tree and select the **Result Details** tab, the bottom right pane of the Test Results window displays a screen capture of your application corresponding to the highlighted step in the test results tree.



Viewing Movies of Your Run Session

Quick Test can save a movie of your application during a run session. This can be useful to help you see how your application behaved under test conditions or to debug your test. You can view the entire movie or select a particular segment to view. When you select a step in the test results tree and click the **Screen Recorder** tab, the right pane of the Test Results window displays the frame in the movie corresponding to the highlighted step in the test results tree.

You can customize the criteria Quick Test uses to save movies by selecting **Always**, **For errors**, or **For errors and warnings** in the **Save movies to results** list in the Run tab of the Options dialog box.

Removing a Movie from the Test Results

You can remove a stored movie from the results of a test. This reduces the size of the test results file. To remove a movie from the test results, choose **File > Remove Movie from Results**.

Exporting Captured Movie Files

You can export a captured Screen Recorder movie to a file. The file is saved as an **.fbr** file. You can view **.fbr** files in the Mercury Micro Recorder (as described in Viewing Screen Recorder Movie Files in the Mercury Micro Player below). You can also attach **.fbr** files to defects in Quality Center. Quality Center users who have the Quick Test Add-in for Quality Center installed can view the movies from Quality Center.

To export a Screen Recorder movie:

1. Choose **File > Export Movie to File**. The Save As dialog box opens, enabling you to change the default destination folder and rename the file, if required. By default, the file is named <**test name**> **[**<**name of run results**>**]**, and is saved in the test results folder.
2. Click **Save** to save the exported (**.fbr**) file and close the dialog box.

When you capture a movie of your run session using the Screen Recorder, the movie is saved as an **.fbr** file in your test results folder. You can also view these **.fbr** files without opening the Quick Test Test Results window, using the Mercury Micro Player.

To play a Screen Recorder movie in the Mercury Micro Player:

1. Perform one of the following:
   - Double-click any **.fbr** file in Windows Explorer.
   - Choose **Start > Programs > Quick Test Professional > Tools > Mercury Micro Player** and then choose **File > Open** in the Micro Player to select any **.fbr** file.

   The movie opens in the Mercury Micro Player and begins playing.

2. Use the controls at the top of the window to access a particular location in the movie or to modify the volume settings.

Reporting defects

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

The results of each Quick Test run session are saved in a single **.xml** file (called **results.xml**). This **.xml** file stores information on each of the test result nodes in the display. The information in

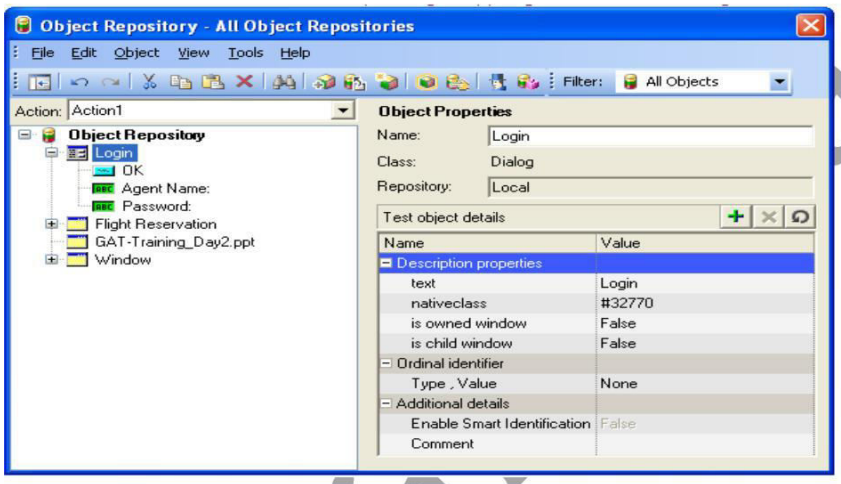
these nodes is used to dynamically create **.htm** files that are shown in the top-right pane of the Test Results window.

Each node in the test results tree is an element in the **results.xml** file. In addition, there are different elements that represent different types of information displayed in the test results. You can take test result information from the **.xml** file and use XSL to display the information you require in a customized format (either when printing from within the Quick Test Test Results window, when displaying test results in your own customized results viewer, or when exporting the test results to an HTML file).

The diagram below shows the correlation between some of the elements in the **.xml** file and the items they represent in the test results.

OBJECT REPOSITORY

- **Object Repository** is the memory space for QTP to object properties
- It is the Interface between QTP Script and the Application
- Contains Logical names and physical Descriptions of actual objects in the application
- The Object Repository window displays a tree of all objects in the current component or in the selected action (including all local objects and all objects in any shared object repositories associated with the selected action or component).



How Quick test Identifies Objects in the AUT

QTP identifies the object in the application by Object Description and Class.

For example : The Edit box is identified by using object description linked with Logical Name(PSOPTIONS_BSE_TIME20) and Class Name(WebEdit)

This concept tells about how QTP identifies objects while running and recording session.

During Runtime

Step1:- Look for class and logical name in script

Step2:- Search for the same class and logical name in object repository **Step3:-** Identify the object in application using properties of the object that stored in recording time.

During Record time

Step1:- QuickTest gets information about object from OS. Ex. Class
Step2:- Identifies what properties need to be recorded based on TOM **Step3:-** Creates an unique description based on TOM and stores in object repository as a test object.

How QTP creates unique description for an object?

- QuickTest Identifies object class with the help of OS and maps to any appropriate test object class available in object identification window.
- Records all available mandatory properties and checks for unique description
- If recorded description is unique then this description will be moved to OR
- If it is not unique then QTP will record assistive properties to create unique description
- QuickTest learns one assistive property at a time and stops as soon as it creates a unique description for the object
- 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.

QTP Object Identification Mechanism

- What is test object description?
  - QTP identifies an object using test object description which is usually stored in object repository. This description is made up of two sets of properties:
    - Mandatory Properties - Mandatory to record all the properties

- Assistive Properties - Optional. QTP will use when mandatory properties not created unique description

You use the main screen of the Object Identification dialog box to set mandatory and assistive properties, to select the ordinal identifier, and to specify whether you want to enable the Smart Identification mechanism for each test object.

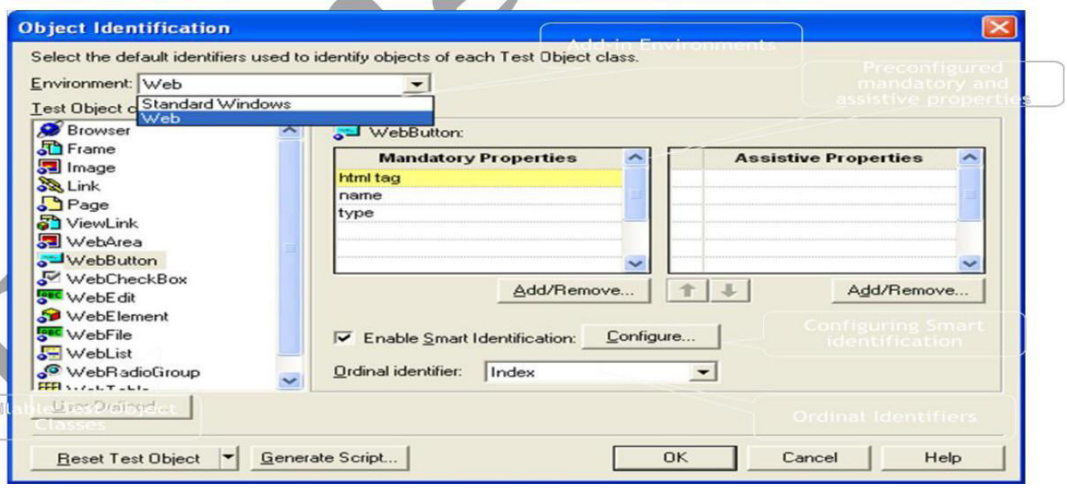
From the Object Identification dialog box, you can also define user-defined object classes and map them to Standard Windows object classes, and you can configure the Smart Identification mechanism for any object displayed in the **Test Object classes** list for a selected environment.

Configuring Mandatory and Assistive Recording Properties

If you find that the description Quick Test uses for a certain object class is not the most logical one for the objects in your application, or if you expect that the values of the properties currently used in the object description may change, you can modify the mandatory and assistive properties that Quick Test learns when you learn an object of a given class.

During the run session, Quick Test looks for objects that match all properties in the test object description—it does not distinguish between properties that were learned as mandatory properties and those that were learned as assistive properties.

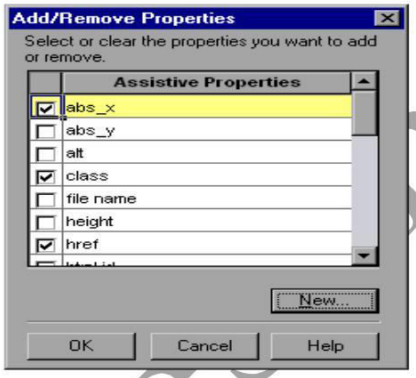
1. Choose **Tools > Object Identification**. The Object Identification dialog box opens.



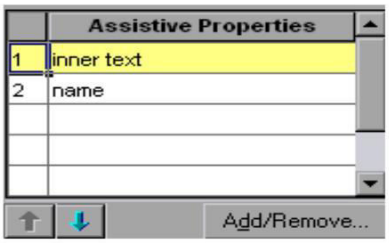
2. Select the appropriate environment in the **Environment** list. The test object classes associated with the selected environment are displayed alphabetically in the **Test Object classes** list.

3. In the **Test Object classes** list, select the test object class you want to configure.
4. In the **Mandatory Properties** list, click **Add/Remove**. The Add/Remove Properties dialog box for mandatory properties opens.

5. Select the properties you want to include in the Mandatory Properties list and/or clear the properties you want to remove from the list.
6. In the **Assistive Properties** list, click **Add/Remove**. The Add/Remove Properties dialog box for assistive properties opens.



7. Select the properties you want to include in the assistive properties list and/or clear the properties you want to remove from the list.
8. Click **OK** to close the Add/Remove Properties dialog box. The properties are displayed in the Assistive Properties list



9. Use the up and down arrows to set your preferred order for the assistive properties.

Smart Identification

When the learned definition for an object does not enable Quick Test to identify an option, Quick Test uses the Smart Identification definition (if defined and enabled) to identify the object.

The Smart Identification dialog box enables you to create and modify the Smart Identification definition that Quick Test uses for a selected test object class.

The Smart Identification mechanism uses two types of properties:

Base Filter Properties. - Very basic properties to create an object list for smart identification

The most fundamental properties of a particular test object class; those whose values cannot be changed without changing the essence of the original object. For example, if a Web link's tag was changed from <A> to any other value, you could no longer call it the same object.

Optional Filter Properties. - Useful to filter the object list created by base filter properties to identify unique object. Other properties that can help identify objects of a particular class. These properties are unlikely to change on a regular basis, but can be ignored if they are no longer applicable.

Normal Identification

- Uses only recorded description to identify an object
- Should satisfy all recorded property values
- Fails even one property is not matched with original property value

Smart Identification

- Uses recorded configuration settings in Object identification window

- No need to satisfy all configured property values
- If one property value fails SI will use another property to identify object

Difference between RO & TO Properties

- Test Object properties are QTP identified properties
- Runtime Object properties are the properties assigned by application provider

Ordinal Identifiers: - There are 3 types of ordinal identifiers.

1. Location
2. Index
3. Creation Time

1. Location: If at all the Location is selected as an ordinal identifier then the QTP will generate the sequence of numbers from 0,1,2,… based on the sequence of the objects located in the application.

2. Index: If at all the index is selected as an ordinal identifier then the QTP will generate the sequence of numbers from 0,1,2,… based on the sequence of the programs of the corresponding objects.

3. Creation time: If at all the creation time is selected as an ordinal identifier then the QTP will generate the sequence of numbers from 0,1,2,... based on the loading time of a web page.

Test object Model

The *test object model* is the large set of object types or *classes* that QuickTest uses to represent the objects in application under test. Each test object class has a list of properties that can uniquely identify objects of that class, and a set of relevant methods that QuickTest can perform during a run session.

QTP tool while recordin captures the objects and its current properites in appliction.If U run the script it will identifie the run time object in appliction by captured test object properties.

A ***test object*** is an object that QuickTest creates in the test or component to represent the actual object in your application. QuickTest stores information about the object that will help it identify and check the object during the run session.

A ***run-time object*** is the actual object in your Web site or application on which methods are performed during the run session. When you perform an operation on your application while recording

QuickTest:

➢ identifies the QuickTest test object class that represents the object on which you performed the operation and creates the appropriate test object

➢ reads the current value of the object s properties in your application and stores the list of properties and values with the test object

➢chooses a unique name for the object generally using the value of one of its prominent properties

➢records the operation that you performed on the object using the appropriate QuickTest test object method

For example suppose you click on a **Find** button with the following HTML source code:

<INPUT TYPE submit NAME Find VALUE Find > QuickTest identifies the object that you clicked as a **WebButton** test object.

It creates a WebButton object with the name Find and records the following properties and values for the Find WebButton:

It also records that you performed a **Click** method on the WebButton.

QuickTest displays your step in the Keyword View like this:

QuickTest displays your step in the Expert View like this:

Browser(Mercury Interactive).Page(Mercury Interactive).WebButton(Find).Click

Object Repository types

There are two object repository types in QTP.

1. Local/Per Action
2. Shared

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.
A shared object repository stores test objects in a file that can be accessed by multiple tests (in read-only mode).

Local Object repository

- An object repository is created for each action.(per action means you have individual OR for every 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.

- 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.

- The extension of the per action repository is .mtr

Shared Object repository

- A single objected repository is used for multiple tests (shared means you have single OR for multiple actions.)

- Shared object repositories can be created in two ways
  - From Object repository Manager
  - By exporting objects from local object repository

- QTP can use multiple shared object repositories for one action/test

- These OR's are only readable in any action / test

- Editing of a shared object repository only by using object repository manager
- The extension for shared object repository is .tsr

When to use shared / local?

Shared Object Repository

- 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.
- You are creating tests using keyword-driven methodologies (not using record).

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.

Repository Manager

- The Object Repository Manager enables you to create and maintain shared object repositories.
  - The following operations are facilitated by Object Repository Manager
    - Creating /Editing Object Repositories
    - Opening Object Repositories
    - Saving Object Repositories

- Comparing object repositories
- Manipulating Objects in Shared Object Repositories
- Managing Repository Parameters
- Modifying Test Object Details
- Locating Objects
- Merge Object Repositories
- Performing Import and Export Operations

- **Object Repository Features :**
  - New Object Repository Format - Object repository files can be imported and export from and to XML format.
  - Multiple repositories per action – Multiple repositories can be associated with a single action.
  - New Object Repository Manager (ORM) - Enables to manage all the Object Repositories from a single window, 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.
  - **Object Repository Merge Tool -** Enables you to merge two shared Object Repositories into a single shared Object Repository or to merge objects from the local Object Repository of one or more actions into a shared Object Repository.

    Merging local to shared repository

    You would need to associate the shared repository with the action containing the local repository. Go to "object repository manager - update from local repository" option.

    Follow these steps to merge -

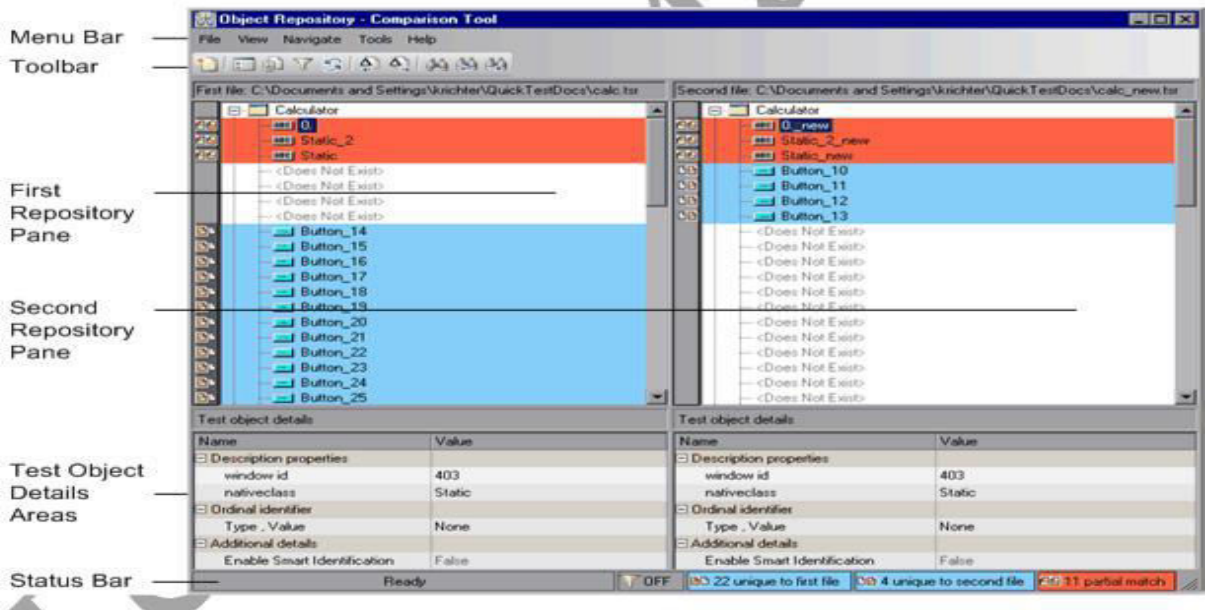
    1. Go to resources -> object repository manager.

2. In the object repository manager window, go to file -> open, and select the shared object repository file. Clear the "open in read-only mode" checkbox.
3. If the repository file opened in read-only mode, go to file -> enable editing.
4. Go to tools -> update from local repository.

Merging shared repositories

1. Open the object repository manager.
2. Go to tools -> object repository merge tool.
3. In the new merge window, browse to the primary repository file. Select the secondary repository file.
4. Click ok
5. Review the merge statistics, as described in viewing merge statistics, and resolve any merge conflicts that were reported.

- **Object Repository Comparison Tool**



Mapping custom class to standard class

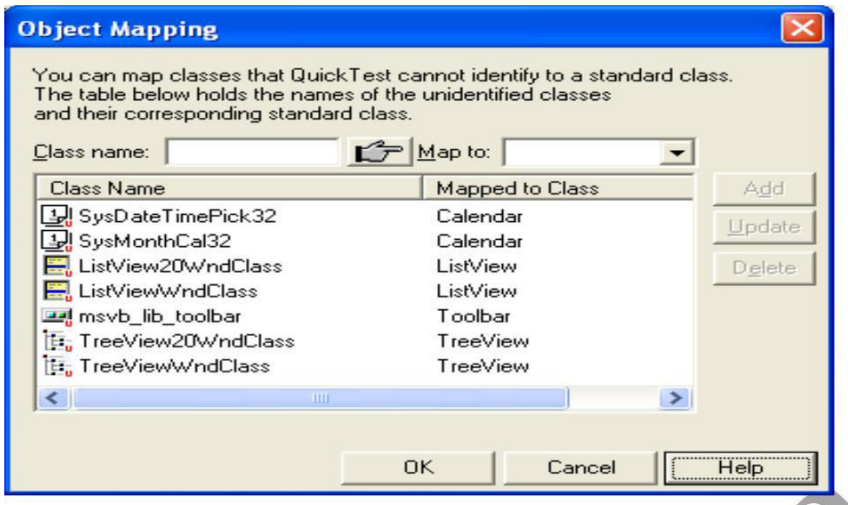
→ QuickTest gets information about object from OS. Ex. Class → What If that class is not available in test object classes list?

The solution is mapping custom class to standard class.

→ The Object Mapping dialog box enables you to map an object of an unidentified or custom class to a Standard Windows class.

→ By default an unidentified or custom class will be recorded by QTP as winobject.

→ This mapping facility is available for only standard windows environment



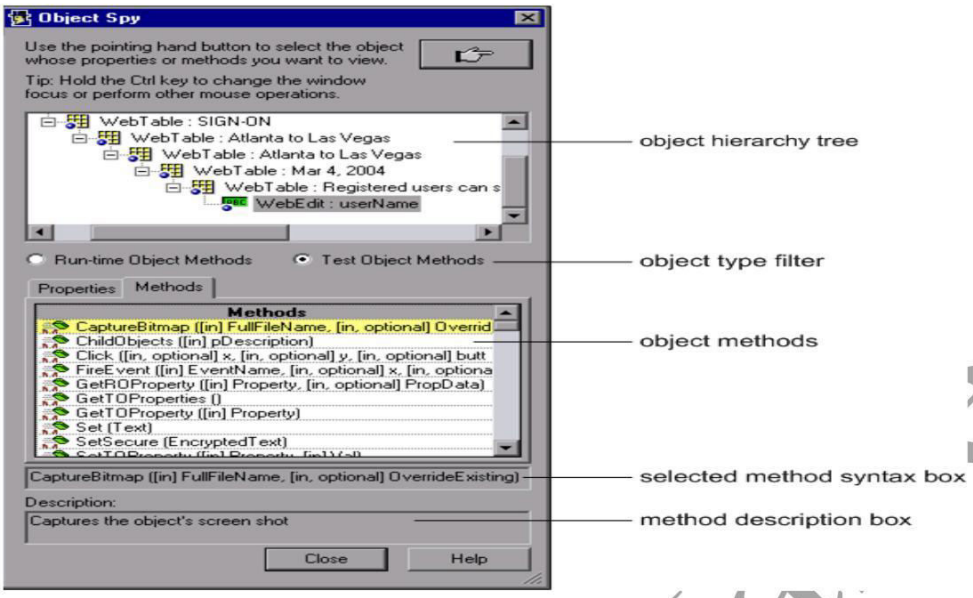
Object Spy

Object Spy enables you to view object properties with current values and also it enaled you to view both the run-time object methods and the test object methods associated with an object.

How to use object spy?

1. Choose **Tools > Object Spy** or click the **Object Spy** toolbar button to open the Object Spy dialog box.
2. Click the **Methods** tab.
3. Click the pointing hand. Both Quick Test and the Object Spy are minimized so that you can point to any object on the open application.
4. If the object on which you want to spy can be displayed only by performing an event (such as right-clicking or moving the pointer over an object to display a context menu), hold the left CTRL key. The pointing hand temporarily turns into a standard pointer and you can perform the event. When the object on which you want to spy is displayed, release the left CTRL key. The pointer becomes a pointing hand again.
5. Click the object whose associated methods you want to view. The Object Spy returns to focus and displays the object hierarchy tree and the run-time object or test object methods associated with the object that is selected in the tree.

6. To view the methods of the test object, click the **Test Object Methods** radio button. To view the methods of the run-time object, click the **Run-Time Object Methods** radio button.



Synchronization Points

Synchronizing Your Test
Options to Synchronize Your Test
Inserting Synchronization Point
Adding Exist Statement
Adding Wait Statement
Global synchronization Setting

Transactions

Inserting transactions
Ending Transactions

Synchronization

If you do not want Quick Test to perform an action or verification until an object in your application achieves a certain status, you should insert a synchronization point to instruct Quick Test to pause the test until the object property achieves the value you specify (or until a specified timeout is exceeded).

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:

Adding Exist and Wait Statements

You can enter **Exist** and/or **Wait** statements to instruct Quick Test to wait for a window to open or an object to appear. Exist statements return a boolean value indicating whether or not an object currently exists. Wait statements instruct Quick Test to wait a specified amount of time before proceeding to the next step. You can combine these statements within a loop to instruct Quick Test to wait until the object exists before continuing with the test.

Inserting synchronization point

First find where you want to insert synchronization point and click on record > insert > synchronization point > select the object > select the property and value > 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) 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("FT").WinButton("DeleteOrder").GetROProperty("enabled")

    If oPropval=true Then

        Exit for

    else

        wait(1)

    End If

Next '*************************************************************************
```

Transactions

You can measure how long it takes to run a section of your test by defining transactions. A transaction represents the process in your application that you are interested in measuring. You define transactions within your test by enclosing the appropriate sections of the test with **start** and **end** transaction statements.

```
Services.StartTransaction "ReserveSeat"

Browser("Welcome:*").Page("Find a Flight").WebList("fromPort").Select "London"

Browser("Welcome:*").Page("Find a Flight").WebList("toPort").Select "Frankfurt"

Browser("Welcome:*").Page("Find a Flight: Mercury").WebList("toDay").Select "12"

Services.EndTransaction "ReserveSeat"
```

Inserting Transactions

1. Click the step where you want the transaction timing to begin. The page is displayed in the Active Screen tab.
2. Click the **Start Transaction** button or choose **Insert > Start Transaction**. The **Start Transaction** dialog box opens.

3. Enter a meaningful name in the **Name** box.

4. Decide where you want the transaction timing to begin:
   - To insert a transaction before the current step, select **Before current step**.
   - To insert a transaction after the current step, select **After current step**.
5. Click **OK**. A **Start Transaction** step is added to the Keyword View.

Ending Transactions

1. Click the step where you want the transaction timing to end. The page opens in the Active Screen.
2. Click the **End Transaction** button or choose **Insert > End Transaction**. The **End Transaction** dialog box opens.

3. The Name box contains a list of the transaction names you defined in the current test. Select the name of the transaction you want to end.

4. Decide where to insert the end of the transaction:
   - To insert a transaction before the current step, select **Before current step**.
   - To insert a transaction after the current step, select **After current step**.

Click **OK**. An **End Transaction** step is added to the Keyword View.

Commonly used methods in QTP:

1. Click
2. Set
3. SetSecure
4. Exist
5. CaptureBitmap
6. Select
7. GetToProperty
8. GetRoProperty
9. ChildObjects
10. ChildItem
11. GetCellData
12. RowCount
13. ColumnCount
14. GetRowWithCellText

Click: It is used to click on button, Image, Link, etc.

Syntax: object.click

Set: It is used to set value to EditBox, and select CheckBox

Syntax: For EditBox

Object.set inputvalue

Syntax: For CheckBox

Object.Set "ON"

SetSecure: It is used to set value to Password EditBox

Syntax: object.Setsecure "Encrypted password"

Exist: It is used to verify the existence of an object(Browser, Page, link, Image, etc.). It will return True if object exist, if object not exist it will return False.

Syntax: object.Exist

Script Example:

```
SystemUtil.Run "iExplore.exe", "http://sys1:2014/Multi_Cloud/Login.aspx"

Set oPage = Browser("Cloud Management").Page("Cloud Management")

oPage.WebEdit("UserId"). Set "admin"
oPage.WebEdit("Password"). Set "admin"
oPage.WebButton("Login").click

If Browser("Admin Home").Page("Admin Home").Exist Then
        Reporter.ReportEvent micPass, "Verify Page Exist","Admin Home page displayed"
Else
        Reporter.ReportEvent micFail,"VerifyPageExist","Admin Home page not displayed"
End If
```

SystemUtil.Run: It is used launch application. It can be used to open any time of application.

Syntax: SystemUtil.Run "URL or application path"

InvokeApplication: It is used to launch application based on given path. Mainly it is used to open desktop based application.

Syntax: InvokeApplication "applicationpath"

SetSecure: It is used to enter encypted password into password EditBox.

Syntax: Object.Setsecure "Encrypted Password"

Select: It is used to select RadioButton and Select item in dropdown list

Syntax: Object.Select "item"

CaptureBitmap: It is used to capture screenshot of window or page or any other object and save it in the specified location.

Syntax: Object.CaptureBitmap "Filepath"

GetToProperty: It is used to get test object property Value.

Test Object: Object stored in Object Repository

Syntax: object.GetToProperty("propname")

GetRoPropery: It is used to get Run-Time object property value.

Run Time object: Object available on AUT

Syntax: object.GetRoproperty("propname")

ChildObjects: It is used to get child objects available on Page or window or any parent object.

Syntax: object.ChildObject(object description)

ChildItem: It is used get child item from WebTable

Syntax: object.ChildItem(Row, Column, Object Class, index)

GetCellData: It is used to get WebTable Cell data

Syntax: object.GetCellData(Row, Column)

RowCount: It is used to WebTable row count

Syntax: object.RowCount

ColumnCount: It is used to get WebTable column count

Syntax: object.ColumnCount(Row)

GetRowWithCellText: It is used to get row number in a WebTable based on given input text

Syntax: object.GetRowWithCellText(celltext, [column])

Parameterization

You can replace the constant/fixed input data with parameters and send data to the application during execution, it is known as parameterization. It will increase the power and flexibility of a test. A parameter is a variable that is assigned a value from an external data source or generator. Values in object input steps, checkpoints and action input values can be parameterized.

Types of parameters:

There are four types of parameters:

1. Test/action parameters

2. DataTable parameters.

3. Environment variable parameters.

4. Random number parameters.

Test/action parameters

Test parameters make possible for us to use values passed from the test. Action parameters enable us to pass values from other actions in your test.

Data Table parameters

Data Table parameters allow us to create a data-driven test (or action) that runs several times using the data that we supply. In each repetition, or iteration, Quick Test uses a different value from the Data Table.

Data Tables are 2 types:

i) Design time DataTable.

ii) Run time DataTable. We

can parameterize in two ways.

1. Using Built-in Data Table (Global or Local sheets).

2. Using External Excel Sheet.

i) Design time DataTable:

It is a default sheet for every new test which is used to maintain application input & expected data.

ii) Run time DataTable:

It is temporary table created by QTP for every Run session. Once execution is completed run time table will be exported to result window. When we start execution of a test, QTP will copy Design Time Table data to runtime data table.

There are 2 types of Sheets

i) Global Sheet ii) Action or Local sheet

i) Global Sheet:

It is a test sheet, data available in this sheet can be used by all the "Actions".

Navigation:

File → Settings-- → Run-- → DataTable Iterations.

ii) Action or Local sheet:

For each Action in a test separate sheet will be maintained by QTP. Data available in Local sheet can be used for corresponding Action only.

Navigation:

Action or Local sheet Iteration Settings: (In Key word View)

Test Flow-→ Action (Right Click)-→ Select "Action Call Properties"-→ Run Tab

We can parameterize in two ways.

1. Using Built-in Data Table (Global or Local sheets).

2. Using External Excel Sheet.

Below is the Sample Code for how to parameterize with Built-in Data Table.

DataTable "c:\Kumar\Testdata.xls","Sheet1","Action 1"

n=DataTable.Localsheet.GetRowCount Msgbox n

For i=1 to n

Set opage=Browser("name:=GMail.*").Page("Title:= Gmail.*") Opage.webedit ("name:=reqname").set DataTable ("reqname",dtlocalsheet) Opage.webedit ("name:=reqemail).set set DataTable("reqemail",dtlocalsheet) Opage.webedit ("name:=reqphone").set set DataTable("reqphone",dtlocalsheet) Opage.webbutton ("name:=reqcontact).click Browser ("name:= reqcontact").Back

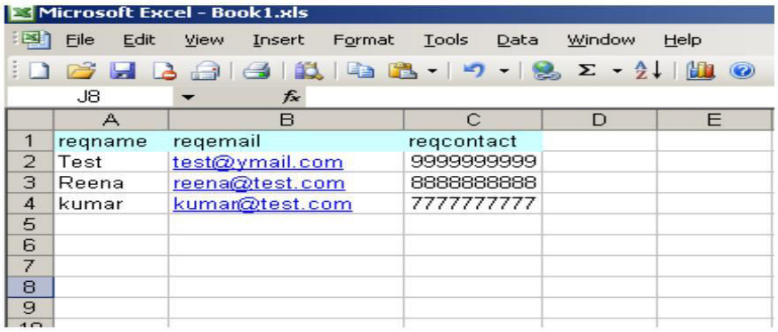
DataTable.SetNextRow

Next

Below is the Sample Code for how parameterize from External Excel Sheet.

```
Call GetdataFromexcel (c:\\Kumar\Testdata.xls",”sheet 1”,ref_data)

Col=Ubound (ref_data,2)

Row= Ubound (ref_data,1)

For i=0 to row

        For j=0 to col

                Set opage=Browser (“name:=Gmail.*”).Page(“Title:= Gmail.*”)

                Opage.webedit (“name:=reqname”).set ref_date(i,j)

                Opage.webedit (“name:=reqemail).set ref_date(i,j)

                Opage.webedit (“name:=reqphone”).set ref_date(i,j)

                Opage.webbutton (“name=reqcontact).click Browser

                (“name:= reqcontact”).Back

        Next

Next
```

Below is the Test Data in excel file”Testdata.xls”.



Environment variable parameters

Environment variable parameters allow us to use variable values in many Actions or Functions.

In other words these are global variables. Environment variables are 2 types.

1.Build-in

2.User defined

Built-in: These parameters will store system generated values.

Example:

strMachineOS = Environment.value("OS")

strSystemLoginUser = Environment("UserName")

User defined Environment variables:

We can define environment variables in ".xml" or ".ini" file.

. XML file:

<Environment>

<Variable>

<Name>URL</Name>

<Value>www.gmail.com</Value>

<Variable/>

</Environment>

.ini fine:

[Environment]

URL=www.gmail.com

Note: Using “Environment.LoadFromFile” statement we can load external file environment variables to QTP test.

Example: QTP main script

Environment .LoadFromFile ”c:\\Kumar\test.xls”

SystemUtil.Run Environment .Value (”URL”)

Random number parameters:

Random number parameters can be used to send the random numbers as input to the application.

Ex: Dim a

a = Randomnumber(10, 20)

Msgbox a

Regular expressions

Regular expressions enable QuickTest to identify objects and text strings with varying values.

You can use regular expressions when:

- Defining the property values of an object
- Parameterize a step
- Creating checkpoints with varying values

A regular expression is a string that specifies a complex search phrase. By using special characters such as a period (.), asterisk (*), caret (^), and brackets ([ ]), you can define the conditions of a search. When one of these special characters is preceded by a backslash (\), Quick Test searches for the literal character.

To define a constant property value as a regular expression:

- Open the Object Properties dialog box for the object either form test tree or from Active Screen or from Object Repository
- In the Property column, select the property you want to set as a regular expression.
- In the Edit value section, click Constant.

- Click the Edit Constant Value Options button. The Constant Value Options dialog box pens.
- Select the Regular Expression check box.
- In the Value box, enter the regular expression syntax for the string.
- Click OK to close the Constant Value Options dialog box.
- Click OK to save and close the Object Properties

To define a regular expression in an object checkpoint:

- Display the page, window, or screen containing the text you want to check.
- Choose Insert > Checkpoint > Text Checkpoint, or click the arrow next to the Insert Checkpoint button and choose Text Checkpoint.
- Click the text string for which you want to create the checkpoint.
- Select the object for which you are creating the checkpoint. The Text Checkpoint Properties dialog box opens.
- In the Edit value section, click Constant.
- Click the Edit Constant Value Options button. The Constant Value Options dialog box opens.
- Select the Regular Expression check box.
- In the Value box, enter the regular expression syntax for the string.
- Click OK to close the Constant Value Options dialog box.
- Click OK to save and close Text Checkpoint Properties dialog box.

Common Special characters to create regular expressions.

.	Matching Any Single Character
[xy]	Matching Any Single Character in a List
[^xy]	Matching Any Single Character Not in a List
[x-y]	Matching Any Single Character within a Range
*	Matching Zero or More Specific Characters
+	Matching One or More Specific Characters
?Matching Zero or One Specific Character	
( )	Grouping Regular Expressions

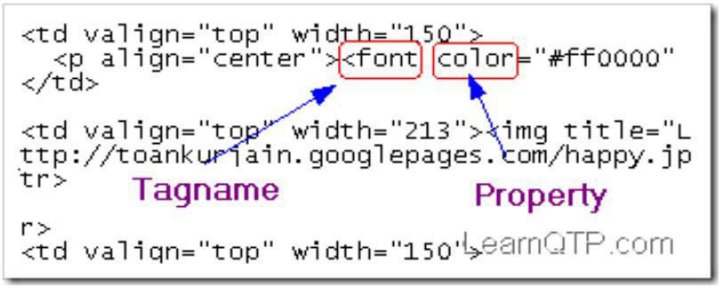
|	Matching One of Several Regular Expressions
^	Matching the Beginning of a Line
$	Matching the End of a Line
\w	Matching Any AlphaNumeric Character Including the Underscore
\W	Matching Any Non-AlphaNumeric Character

You can combine regular expression operators in a single expression to achieve the exact search criteria you need.

Document Object Model (DOM)

DOM is a method for QTP engineers to access the source (IE –> View –> Source) of any webpage direct through VB Scripting.

1. Access the DOM of webtable using **.object** notation.
2. Access the tagname corresponding to the property you wish to find out. In our case tagname is “font and property to be explored is “color”.



3. Find the count of total number of tags and loop it find out the font-color.

The corresponding VB script would be:

```
set obj = Browser("Google").Page("Google").WebTable("Table").object.all.tags("font")

msgbox obj.length

For i = 0 to obj.length - 1
        msgbox obj(i).color
Next
```

Automation Object Model (AOM)

It's a way to write scripts so as to automate your QuickTest operations. In other words user interaction with QTP can be automated using AOM.

When can we use AOM?

- 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, Load repositories, etc.
- You can use AOM to call QTP from other application: For ex: You can write a macro for calling QTP from excel
- You can use AOM to run multiple scripts without test management tools.

Example:

```
Dim qtAppn
Dim qtObjRes

Set qtAppn = CreateObject("QuickTest.Application")
qtAppn.Launch
qtAppn.Visible = True

qtApp.Open "E:\Test\Test2", False, False
Set qtObjRes = qtApp.Test.Actions("Login").ObjectRepositories

qtObjRes.Add "E:\OR\ObjRes.tsr", 1
```


Working with Library Files

Set of functions, procedures, classes, variables, etc.. defined .vbs/.txt/.qfl is called a library file.

.vbs -> VBScript file
.txt -> text file
.qfl -> Quicktest function library

Library file can be created using Notepad or QTP.

Steps to create library file with Notepad:

Open Notepad -> define functions -> save the file with .vbs file

Steps to create library file with QTP:

File menu -> New -> Function library -> define function in library tab -> save the file with .vbs/.txt/.qfl

Sample Example for library file:

```
Function add(x, y)
 dim z
 z= x + y
 add = z
End Function

Function sub(x, y)
 dim z
 z= x - y
 sub = z
End Function

Function mul(x, y)
 dim z
 z= x * y
 mul= z
End Function

Function div(x, y)
 dim z
 z= x / y
 div= z
End Function
```

Once the library file is defined we need to associate the library file(s) with QTP, then only we can call/use library functions in QTP script.

Steps to associate library file in QTP:

File -> Settings -> Resources -> Click on add (+) button -> select the library file

Or

We can load library file using QTP utility statements at runtime.

1. Execute File
2. LoadFunctionLibrary

Example:

ExecuteFile “D:/WebControls.vbs”,

LoadFunctionLibrary “D:/WebControls.vbs”, “D:/UtilityFunctions.vbs”

ExecuteFile can be used to load one library file at a time

LoadFunctionLibrary can be used to load one or more Library files at a time

Note: 1) When ExecuteFile is used, we can’t debug the script(Steo Into, Step Over, etc..)
2) When LoadFunctionLibrary is used, we can debug the functions easily with all debug options provided in QTP.

Sample QTP Main Script:

```
LoadFunctionLibrary “D:/ArithmeticFunctions.vbs

a = add(10,20)
msgbox "Addition of two values is : "& a

b = mul(10,20)
msgbox Multiplicationof two values is : "& b
```

Working with Actions

Action is a part of a test. It helps to divide a test into logical units, such as the main sections of a Web site, or specific activities that you perform in your application.

An action consists of its own test script, including all of the steps recorded in that action, and any objects in its local object repository.

When you open a test, you can choose to view the test flow (calls to actions) or you can view and edit the individual actions stored with your test.

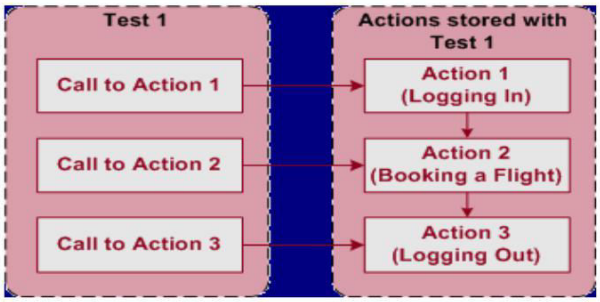
If you work with tests that include many steps or lines of script, it is recommended that you use actions to divide your test steps.

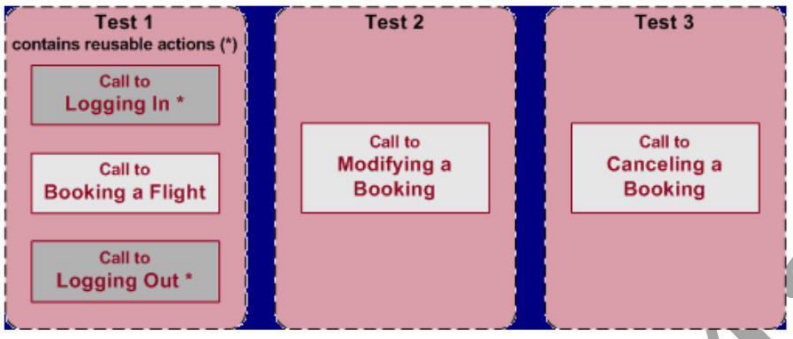
If a test is handling more functionalities / modules it's better to use multiple actions. Splitting functionality into actions avails reusability of actions in other tests.

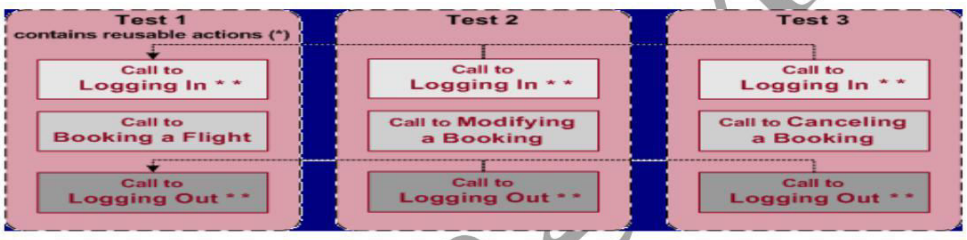
For example, suppose you want to test several features of a flight reservation system. You plan several tests to test various business processes, but each one requires the same login and logout steps. You can create one action that contains the steps required for the login process, another for the logout steps, and other actions for the main steps in your test. After you create the login and logout actions, you can insert those actions into other tests.

If you create a test in which you log into the system, book one flight, and then log out of the system, your test might be structured as shown—one test calling three separate actions:

Action Reusability







Actions enable you to parameterize and iterate over specific elements of a test. They can also make it easier to modify steps in one action when part of your application changes.

For every action called in your test, QuickTest creates a corresponding action sheet in the Data Table so that you can enter Data Table parameters that are specific to that action only.

Types of Actions/ Calling Actions

- **Non Reusable Actions / Normal Actions**

An action that can be called only in the test with which it is stored, and can be called only once.

– Insert → Call to New Action

- **Reusable Actions**

An action that can be called multiple times by the test with which it is stored (the local test), as well as by other tests.

  - Insert → Call to Existing Action

- **External Actions**

  - a reusable action stored with another test.

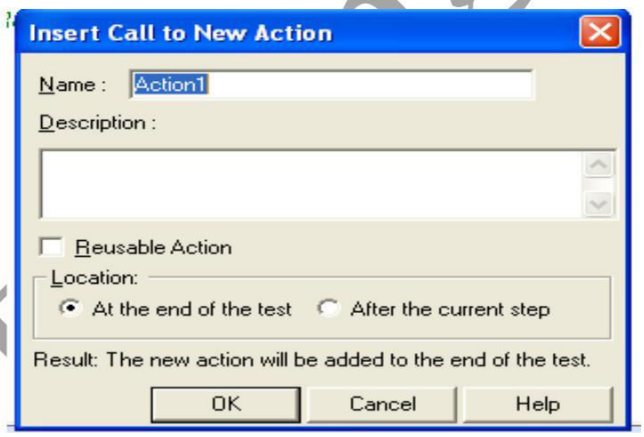
  - External actions are read-only in the calling test, but you can choose to use a local, editable copy of the Data Table information for the external action.

  - Insert → Call to Existing Action

Dividing the Tests /Working with Multiple Actions

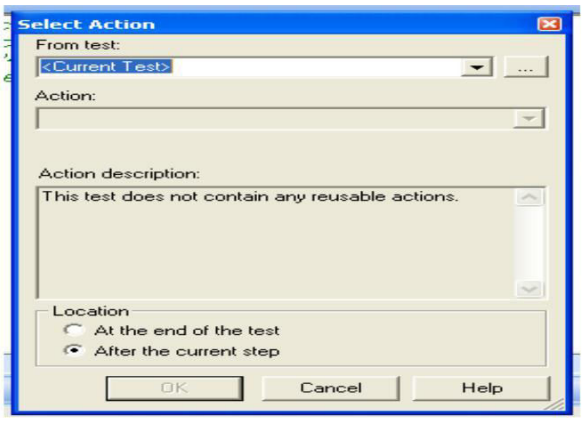
You can divide your test into multiple actions by creating new actions and inserting calls to them, or by inserting calls to existing actions.

➲ **Insert a call to a new action: Insert > Call to New Action** or use the **Insert Call to New Action** button. (or) Right click a step in the Expert View & and choose **Call to Copy of Action** or **Call to Existing Action**.



➲ **Insert a call to a copy of an existing action: Insert > Call to Copy of Action** or right-click an action and choose **Insert Call to Copy of Action**.

- **Insert a call to an existing action: Insert > Call to Existing Action** or right-click an action and choose **Insert Call to Existing Action**



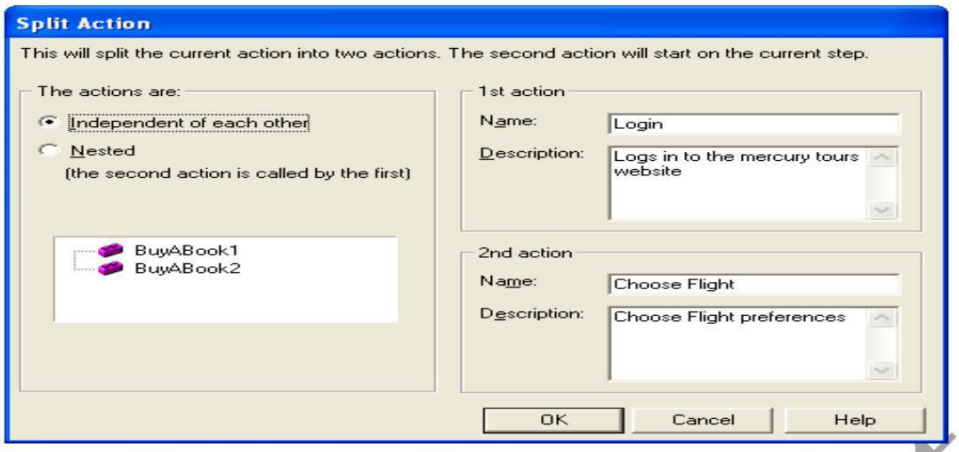
Operations done on actions

- **Split Action:** Edit > Action > Split Action or use the Split Action button
- **Rename Action:** Edit > Action > Rename Action
- **Delete Action:** Edit > Action > Delete Action
- **Edit Action Properties**: Edit > Action > Action

Splitting Actions

You can split an action that is stored with your test into two sibling actions or into parent-child nested actions. When you split an action, the second action starts with the step that is selected when you perform the split action operation.

- Select the step
- Choose **Edit > Action > Split Action**, click the **Split Action** button, or right-click the step and choose **Action > Split**.
- Choose one of the following options:
  - **Independent of each other.** Splits the action into two sibling actions.
  - **Nested (the second action is called by the first).** Splits the selected action into a parent action whose last step calls the second, child action.



Setting Action Properties

The Action Properties dialog box enables you to define options for the stored action. These settings apply each time the action is called. You can modify an action name, add or modify an action description, and set an action as reusable. For an external action, you can set the Data Table definitions.

You can open the Action Properties dialog box while recording or editing your test by:

File > Settings > Run Tab (or) Edit > Action > Action Call Properties (Keyword View)

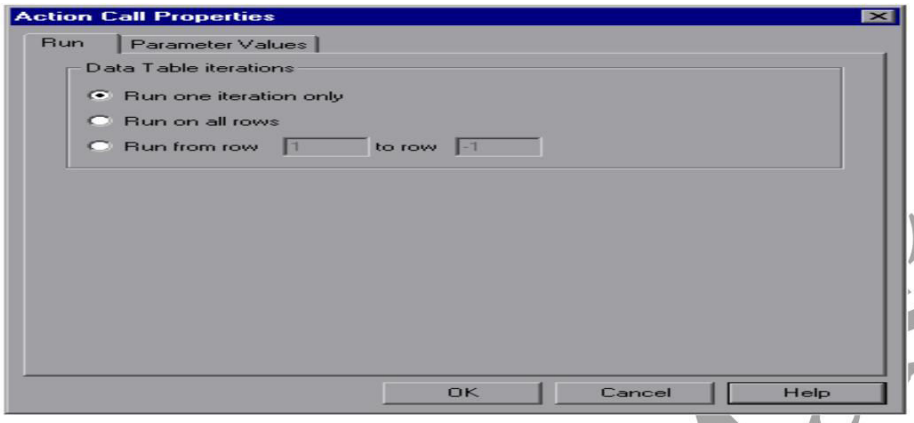
- Choosing **Edit** > **Action > Action Properties** from the Keyword View when an action node is highlighted or from the Expert View.
- Right-clicking an action node in the Keyword View and selecting **Action Properties**.

Setting the Run Properties for an Action

You can use the Run tab of the Action Call Properties dialog box to instruct Quick Test to run only one iteration on the called action, to run iterations on all rows in the Data Table, or to run iterations only for a certain row range in the Data Table.

Instructs Quick Test

- Run only one iteration
- Run iterations on all rows
- Run iterations only for a certain row range

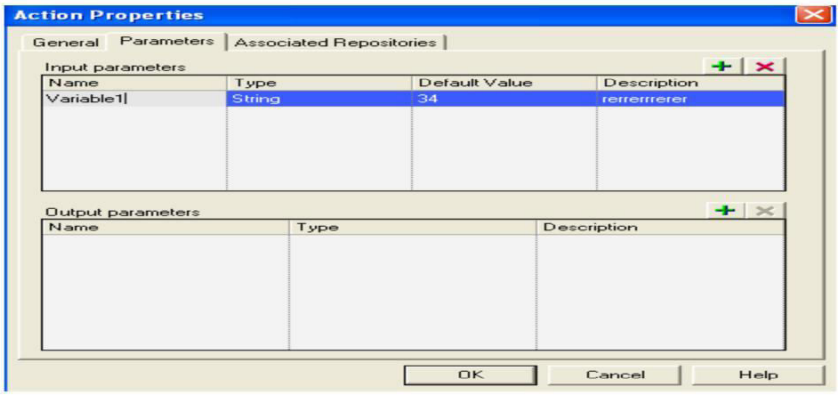


The Run tab includes the following options:

Option	Description
Run one iteration only	Runs the called action only once, using the first row in the action's data sheet.
Run on all rows	Runs the called action with the number of iterations according to the number of rows in the action's Data Table.
Run from row __ to row __	Runs the called action with the number of iterations according to the specified row range.

Parameterizing an Action

In Expert View > Action > Action Call Properties (or) Edit >Action >Action Call Properties



Iterations

- A single execution of a set of instructions that are to be repeated is called as an iteration
- It's the one which tells how many times a test or an action should be executed.
- Test Iteration
    - A single execution of all actions in a test.
- Action Iteration
    - A single execution of an action.

Differentiate Action and Function

Action	Function

1) Action is Specific to QTP 2) Action will have own local repository, Datatable and any other resources 3) Action run only in QTP. 4) Action will not return any value 5) Action doesn't supports arrays and objects in action parameters. 6) Can call a function from an action 7) Action have input and output parameters 8) If you call an action from another action you'll find script, Datatable, Object repository in that called action 9) QTP is mandatory to modify an action	1) Function is specific to VBScript 2) Functional will not have anything to it. 3) Functions can run out side of QTP 4) Functions return values 5) Function support arrays, objects as inputs. 6) Not possible to call an action in the function 7) Function have byval and byref parameters 8) If you call a function from another function or an action you don't find anything for that function 9) No need to have QTP to modify function

Creating an Action Template

If you want to include one or more statements in every new action in your test, you can create an action template. For example, if you always enter your name as the author of an action, you can add this comment line to your action template. An action template applies only to actions created on your computer.

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. The text file must be in the structure and format used in the Expert View.
Save the text file as **ActionTemplate.mst** in your **<QuickTest Installation Folder>\dat** folder. All new actions you create contain the script lines from the action template.

Note: Only the file name ActionTemplate.mst is recognized as an action template.

'**

Sample Action Template

'##

' Objective : The objective of the testcase

```
' Test Case          :  Test Case Name
' Author             :  Author Name
' Date Created       :   XX/XX/XXXX (mm/dd/yyyy)
' Date Updated       :   XX/XX/XXXX (mm/dd/yyyy)
'Updated by          :
'############################################################

'###############Initialize Variables and Load Libraries###################

'-----------Write Statements to initialize Variable and to execute libraries

'######################Driver Script ############################

'-----------Write the main Script here

'############### Report Generation & Memory Cleanup #################

'-----------Write code to generate report and to cleanup memory of variables

'############################################################
```

Removing Actions from a Test

The procedures and effects of removing calls to non-reusable actions, external actions, or reusable actions are different.

To remove a call to a reusable or external action from the test flow:

1. Select the **Test Flow** view from the Action List in the Keyword View.
2. Highlight the action you want to remove and either choose **Edit > Delete** or press the **Delete** key on your keyboard. Alternatively, right-click the action and choose **Delete**. The Delete Action dialog box opens.
3. Choose **Delete the selected call to the action** and click OK. The action call is deleted. The action remains in the test's Action List.

Creating New Actions

You can create new actions and add calls to them, as needed.

To create a new action in your test:

1. If you want to insert a call to the new action from an existing action in your test, click the step after which you want to insert the new action. To insert a call to the new action from the test flow as a top-level action, click any step.
2. Choose **Insert > Call to New Action** or click the **Insert Call to New Action** button

Recovery Scenarios

Recovery Scenarios are used to instruct QuickTest to recover from unexpected events and errors that occur in your testing environment during a run session.

This section describes:

- About Defining and Using Recovery Scenarios
- Deciding When to Use Recovery Scenarios
- Defining Recovery Scenarios
- Understanding the Recovery Scenario Wizard
- Managing Recovery Scenarios
- Setting the Recovery Scenarios List for Your Tests or Components
- Programmatically Controlling the Recovery Mechanism

About Defining and Using 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.

The Recovery Scenario Manager provides a wizard that guides you through the process of defining a *recovery scenario*—a definition of an unexpected event and the operation(s) necessary to recover the run session. For example, you can instruct QuickTest to detect a Printer out of paper message and recover the run session by clicking the OK button to close the message an continue the test or component.

A recovery scenario consists of the following:

- Trigger Event—the event that interrupts your run session. For example, a window that may pop up on screen, or a QuickTest run error.
- Recovery Operation(s)—the operation(s) that need to be performed in order to continue running the test or component. For example, clicking an OK button in a pop-up window, or restarting Microsoft Windows.

- Post-Recovery Test Run Option—the instructions on how QuickTest should proceed once the recovery operations have been performed, and from which point in the test or component QuickTest should continue, if at all. For example, you may want to restart a test or component from the beginning, or skip a step entirely and continue with the next step in the test or component.

Recovery scenarios are saved in recovery scenario files. A recovery scenario file is a logical collection of recovery scenarios, grouped according to your own specific requirements.

Defining Recovery Scenarios

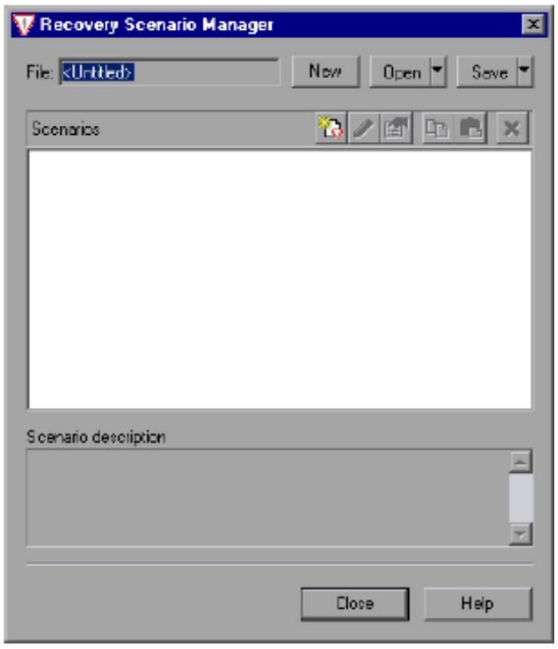
The Recovery Scenario Manager Dialog box enables you to create recovery scenarios and save them in recovery files. You create recovery scenarios using the Recovery Scenario Wizard, which leads you through the process of defining each of the stages of the recovery scenario. You then save the recovery scenarios in a recovery file, and associate them with specific tests or components.

Creating a Recovery File

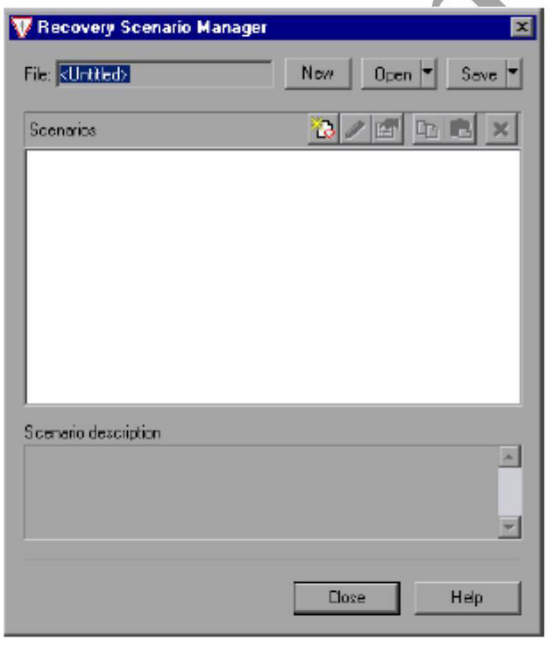
You save your recovery scenarios in a recovery file. A recovery file is a convenient way to organize and store multiple recovery scenarios together. You can create a new recovery file or edit an existing one.

To create a recovery file:

1 Choose Tools > Recovery Scenario Manager. The Recovery Scenario Manager Dialog box opens.



2 By default, the Recovery Scenario Manager Dialog box opens with a new recovery file. You can either use this new file, or click the Open button to choose an existing recovery file. Alternatively, you can click the arrow next to the Open button to select a recently-used recovery file from the list. You can now create recovery scenarios using the Recovery Scenario Wizard and save them in your recovery file, as described in the following sections.



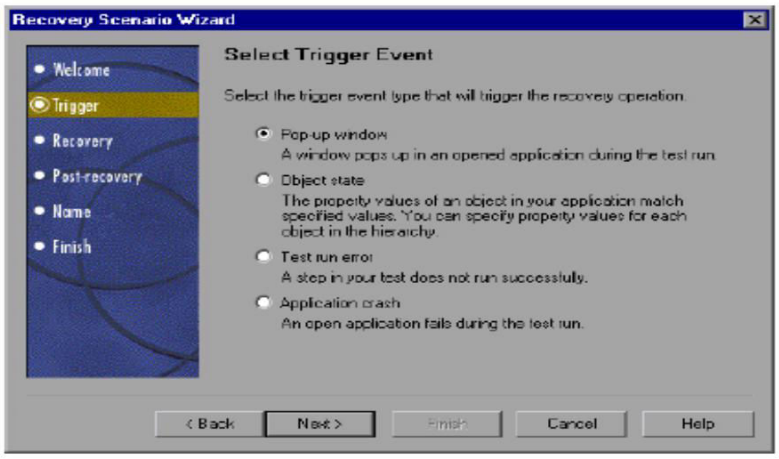
- defining the trigger event that interrupts the run session
- specifying the recovery operation(s) required to continue
- choosing a post-recovery test run operation
- specifying a name and description for the recovery scenario
- specifying whether to associate the recovery scenario to the current test and/or to all new tests

You open the Recovery Scenario Wizard by clicking the **New Scenario** button in the Recovery Scenario Manager Dialog box. **Welcome to the Recovery Scenario Wizard Screen** The Welcome to the Recovery Scenario Wizard screen provides general information about the different options in the Recovery Scenario Wizard, and provides an overview of the stages involved in defining a recovery scenario.

Click **next** to continue to the Select Trigger Event screen.

Select Trigger Event Screen

The Select Trigger Event screen enables you to define the event type that triggers the recovery scenario, and the way in which QuickTest recognizes the event.



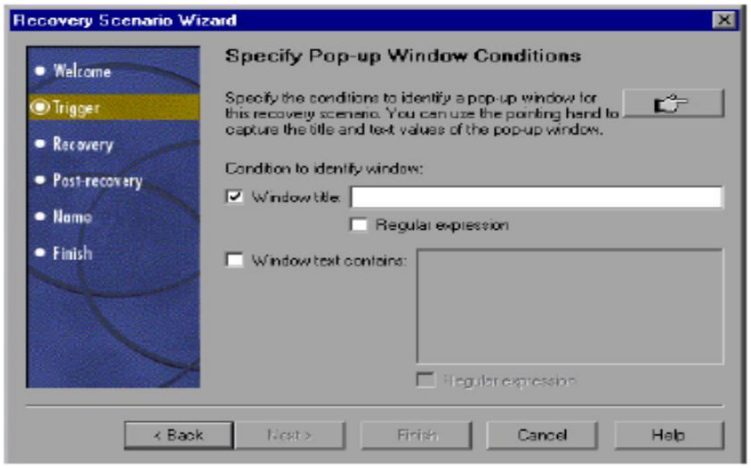
Select a type of trigger and click **next**. The next screen displayed in the wizard depends on which of the following trigger types you select:

- **Pop-up window**—QuickTest detects a pop-up window and identifies it according to the window title and textual content.
- **Object state**—QuickTest detects a specific test object state and identifies it according to its property values and the property values.
- **Test run error**—QuickTest detects a run error and identifies it by a failed return value from a method.
- **Application crash**—QuickTest detects an application crash and identifies it according to a predefined list of applications.

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.

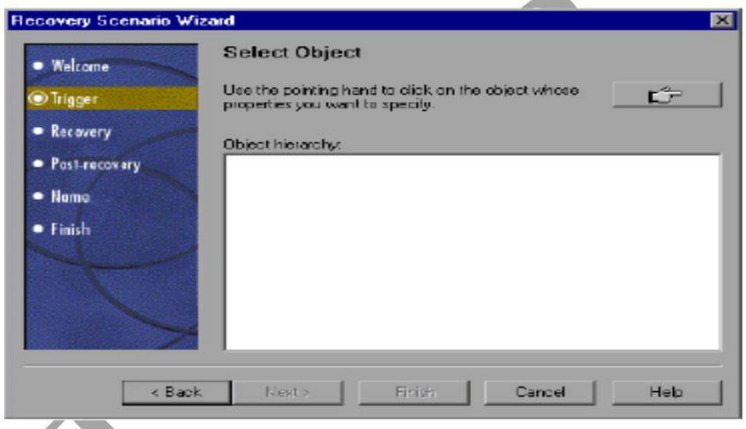
Specify Pop-up Window Conditions Screen

If you chose a **Pop-up window** trigger in the Select Trigger Event screen, the Specify Pop-up Window Conditions screen opens.



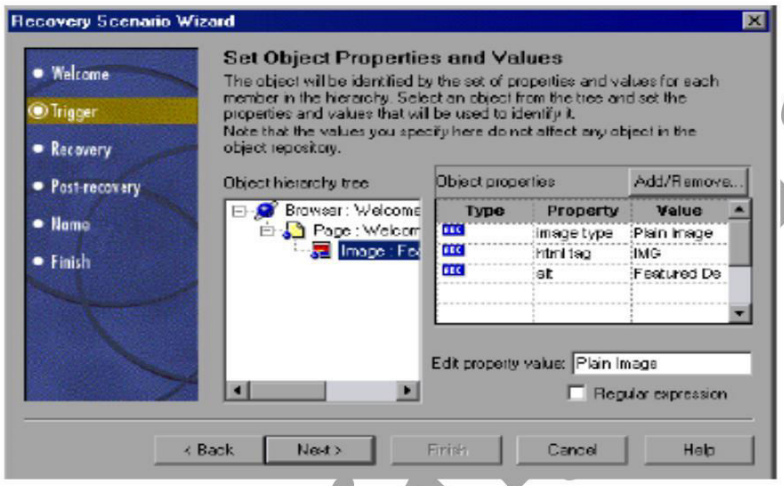
Select Object Screen

.



Click on “Hand Icon”

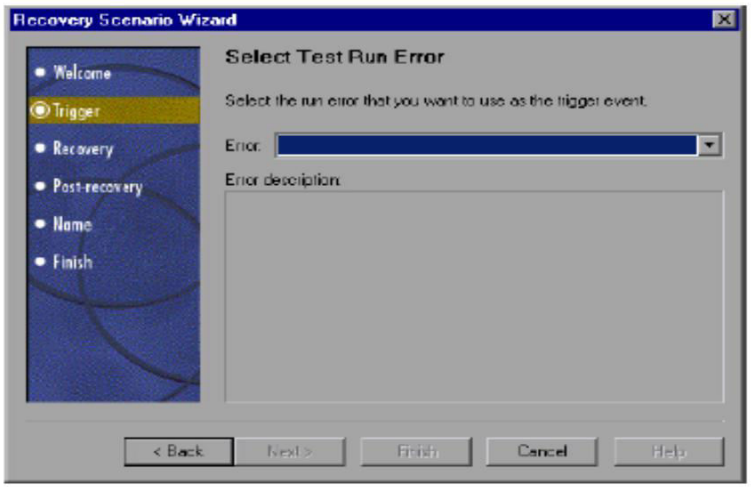
Set Object Properties and Values



Click on “Next” Buttton

Select Test Run Error Screen

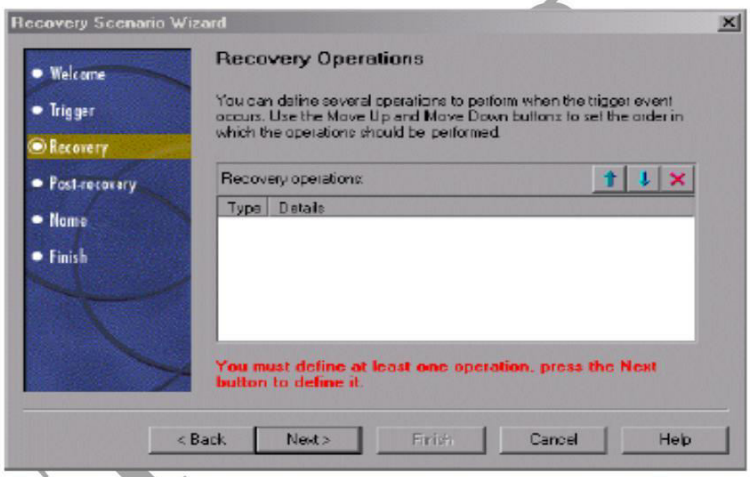
If you chose a **Test run error** trigger in the Select Trigger Event screen, the Select Test Run Error screen opens.



In the **Error** list, choose the run error that you want to use as the trigger event:

Recovery Operations Screen

The Recovery Operations screen enables you to manage the collection of recovery operations in the recovery scenario. Recovery operations are operations that QuickTest performs sequentially when it recognizes the trigger event.

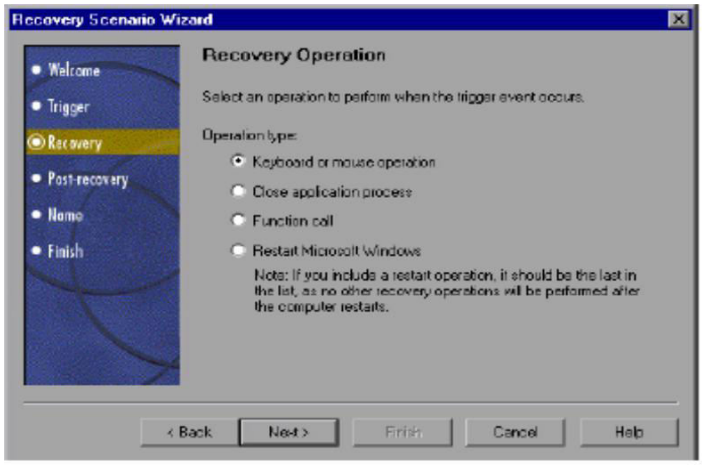


Click **next** to continue to the Recovery Operation screen.

Note: If you define a **Restart Microsoft Windows** recovery operation, it is always inserted as the last recovery operation, and you cannot change its position in the list.

Recovery Operation Screen

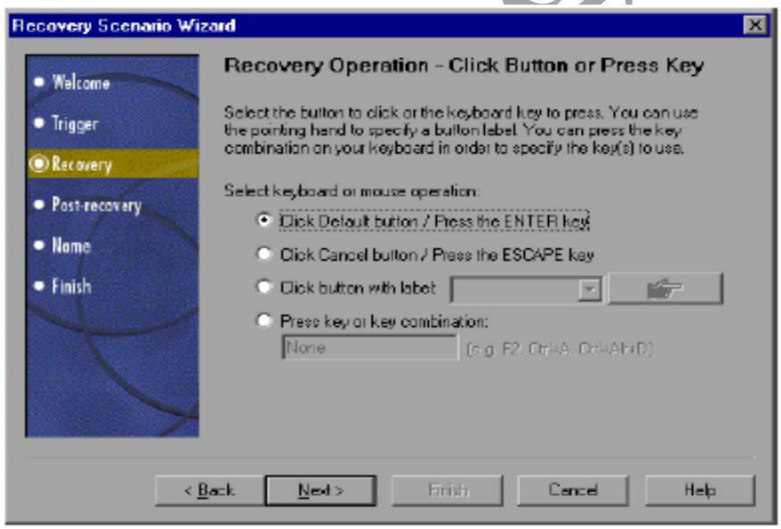
The Recovery Operation screen enables you to specify the operation(s) QuickTest performs after it detects the trigger event.



Select a type of recovery operation and click "**Next**"

Recovery Operation – Click Button or Press Key Screen

If you chose a **Keyboard or mouse operation** recovery operation in the Recovery Operation screen, the Recovery Operation – Click Button or Press Key screen opens.



Click "**Next**"

Recovery Operation – Function Call Screen

If you chose a **Function call** recovery operation in the Recovery Operation screen, the Recovery Operation – Function Call screen opens. Select a recently specified library file in the **Library file** box. Alternatively, click the browse button to navigate to an existing library file.

Note: QuickTest automatically associates the library file you select with your test or component. Therefore, you do not need to associate the library file with your test or component in the Resources tab of the Test Settings dialog box or Business Component Settings dialog box, respectively.

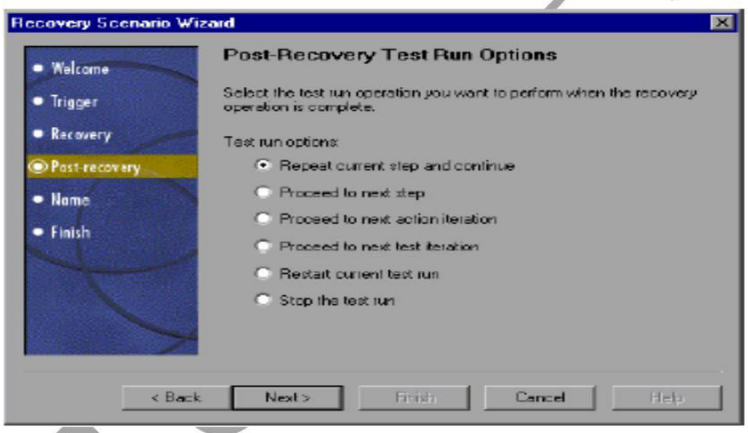
After you select a library file, choose one of the following options:

Select function—choose an existing function from the library file you selected.

Post-Recovery Test Run Options Screen

When you clear the **Add another recovery operation** check box in the Recovery Operations screen and click **Next**, the Post-Recovery Test Run Options screen opens.

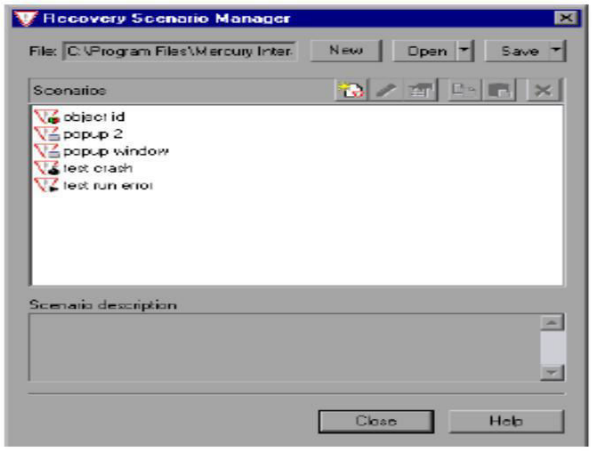
Post-recovery test run options specify how to continue the run session after QuickTest has identified the event and performed all of the specified recovery operations.



Quickest perform one of the following run session options after it performs the recovery operations **WHAT you defined**.

Managing Recovery Scenarios

Once you have created recovery scenarios, you can use the Recovery Scenario Manager to manage them.



The Recovery Scenario Manager contains the following recovery scenario icons.

Navigation to associate Recovery Scenario with QTP Test

File menu-> Settings -> Recovery tab -> click on + button -> Select file -> Select Scenario -> click Add button -> Click Finish

Creating Checkpoints

A checkpoint verifies that expected information is displayed in your application while the test is running.

Understanding Checkpoint Types

Quick Test Professional offers the following types of checkpoints:

Checkpoint Type	Description	Example of Use
Standard Checkpoint	Checks values of an object's properties.	Check that a radio button is selected.

Image Checkpoint	Checks the property values of an image.	Check that the image source file is correct.
Table Checkpoint	Checks information in a table.	Check that the value in a table cell is correct.
Page Checkpoint	Checks the characteristics of a Web page.	Check how long a Web page takes to load or if a Web page contains broken links.
Text Checkpoint	Checks that a text string is displayed in the appropriate place in a Web page.	Check whether the expected text string is displayed in the expected location on a Web page.
Bitmap Checkpoint	Checks an area of a Web page or application after capturing it as a bitmap.	Check that a Web page (or any portion of it) is displayed as expected.
Database Checkpoint	Checks the contents of databases accessed by an application or Web site.	Check that the value in a database query is correct.
Accessibility Checkpoint	Identifies areas of a Web site to check for Section 508 compliancy.	Check if the images on a Web page include ALT properties, required by the W3C Web Content Accessibility Guidelines.
XML Checkpoint	Checks the data content of XML documents.	**Note:** XML file checkpoints are used to check a specified XML file; XML application checkpoints are used to check an XML document within a Web page.

Creating Standard Checkpoint

You can check that a specified object in your application or on your Web page has the property values you expect, by adding a standard checkpoint to your test while recording or editing the test. To set the options for a standard checkpoint, you use the Checkpoint Properties dialog box.

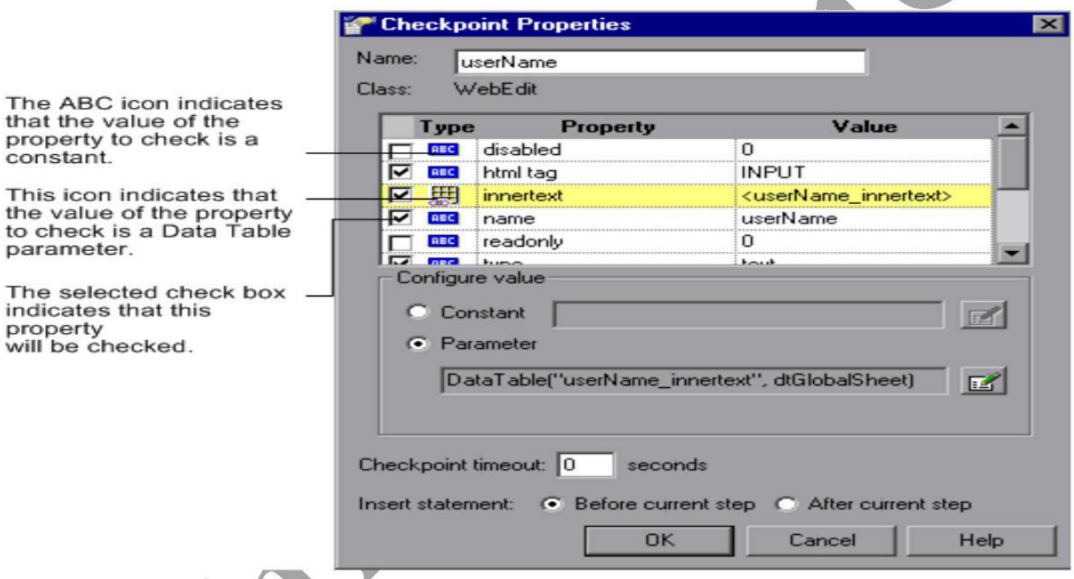
To add a standard checkpoint while recording:

1. Choose **Insert > Checkpoint > Standard Checkpoint**, or click the **Insert Checkpoint or Output Value** toolbar button
2. Click the object you want to check. The Object Selection - Checkpoint Properties dialog box opens.

3. Select the item you want to check from the displayed object tree.
4. Specify the settings for the checkpoint in the checkpoint properties dialog box.
5. Click **OK** to close the dialog box.

Checkpoint Properties Dialog Box

In the Checkpoint Properties dialog box, you can specify which properties of the object to check and edit the values of these properties. While the specific elements vary slightly depending on the type of object you are checking, the Checkpoint Properties dialog box generally includes the following basic elements:

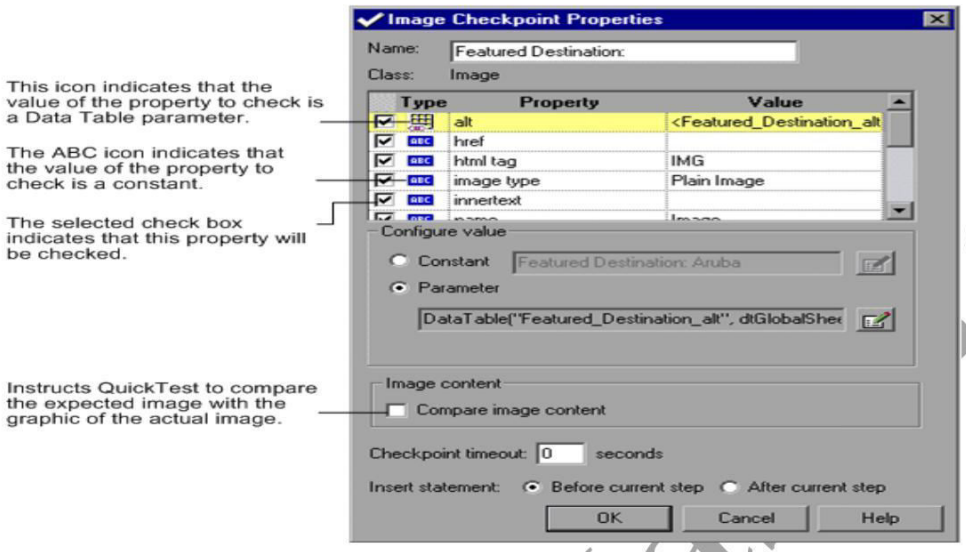


The dialog box described above is used to configure most standard checkpoints. Certain standard checkpoint types, however, employ different dialog boxes, as follows:

- Image checkpoint properties
- Page checkpoint properties
- Table checkpoint properties

Image Checkpoint

Image checkpoints enable you to check the properties of a Web image. In the Image Checkpoint Properties dialog box, you can specify which properties of the image to check and edit the values of those properties. This dialog box is similar to the standard Checkpoint Properties dialog box, except that it contains the **Compare image content** option. This option enables you to compare the expected image source file with the actual image source file.



Page Checkpoint

The Page Checkpoint Properties dialog box enables you to choose which properties to check.

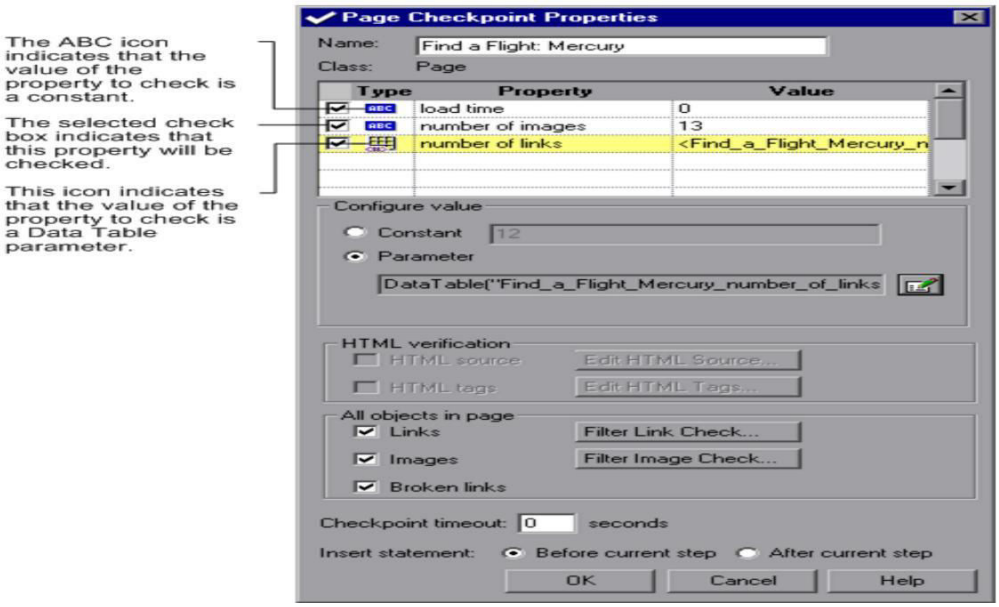
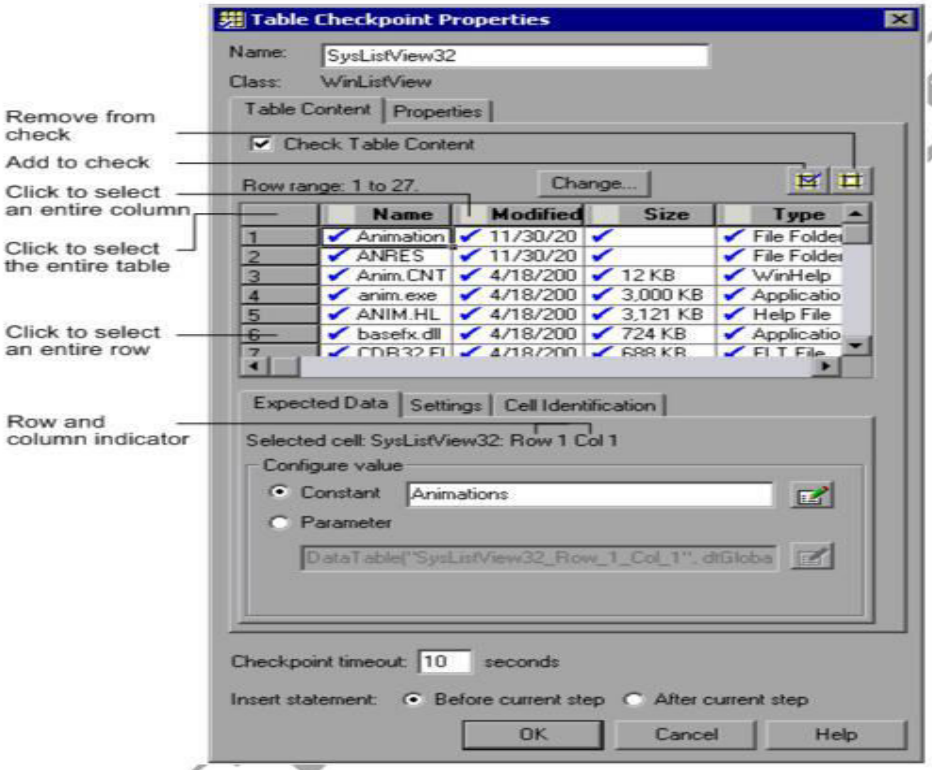


Table Checkpoint

The table check point dialog box enables you to select the table cells which needs to be verified.

Checking Table Content



Output Values

An *output value* is a value captured during the test run and entered in the run-time Data Table for use at another point in the test run. When you create an output value, the test retrieves a value at a specified point during the test run and stores it in a column in the current row of the run-time Data Table.

Eg: The Order Number created during the Insert order function

- **Standard OutPut Value**
- **Text OutPut Value**
- **Page OutPut Value**
- **Bitmap OutPut Value**
- **Data Base OutPut Value**
- **Accessibility OutPut Value**
- **XML OutPut Value**

Navigation to work with standard output value:

Take sample application

→ Record
→ Insert a new record
→ Insert
→ Output value
→ Standard output value
→ Place the hand pointer on target object
→ Order number
→ Click ok
→ Select the appropriate property
→ Text
→ Ok
→ Go to application and open an order number
→ Stop recording.

Script :

Window ("Flight Reservation").winEdit("order No").output checkpoint ("Order no")

Test out put value: Take sample application
→ Click on open order number
→ Insert
→ Output value
→ Text
→ Take the hand pointer and place it on the target object (name)
→ Click ok
→ Select the appropriate output property (text)
→ Click ok.

Again go to application
→ Insert a new record
→ Stop recording

Database:

Make sure that the application data available in Database
→ Insert
→ Output value
→ Database
→ Select 2nd radio button (specify sql statement manually)
→ Next
→ Click create
→ Click on messing Data source
→ Qt_flight_32
→ Select * from orders
→ Finish
→ Select the appropriate database output property from table → Insert before the current statement → Click ok.

Script: Dbtable ("Dbtable").output checkpoint ("Dbtable")

Xml: Application data is available in the xml file

Script:

<xml>

<ordno>10</ordno>
<ordname>cd</ordname>
<date>10_10_2007</date>
<xml>
Save the file with extention .xml;
→ Insert
→ Output value
→ Xml output value
→ Provide the path of the xml file
→ Select the appropriate attribute (ordno)
→ Insert before current statement
→ Click ok.

Script:
Xmlfile ("output value.xml").output checkpoint ("outputvalue.xml")

Working with Virtual Objects

Virtual objects enable you to record and run tests or components on objects that are not normally recognized by QuickTest.

Your application may contain objects that behave like standard objects but are not recognized by QuickTest. You can define these objects as virtual objects and map them to standard classes, such as a button or a check box.

QuickTest emulates the user's action on the virtual object during the run session. In the test results, the virtual object is displayed as though it is a standard class object.

For example, suppose you want to record a test on a Web page containing a bitmap that the user clicks. The bitmap contains several different hyperlink areas, and each area opens a different destination page. When you record a test, the Web site matches the coordinates of the click on the bitmap and opens the destination page.

It is not a real object, if multiple objects having similarities, then they will be treated as virtual objects.

Any application contains 2 types of objects **Standard** object and **Non-Standard objects.**

QTP cannot identify non standard object, it is available in application, for this case we need to map the non standard object to standard object.

Virtual objects are a non standard objects. By using virtual object wizard method, we can map virtual objects to standard class. Once it is connected to standard class, QTP will not be confused to identify that particular object.

Following tasks can be performed by using virtual object wizard method:

1. we can map virtual objects to standard class
2. we can drag the boundary area of the virtual objects
3. we can assign a unique logical name to the virtual object
4. we can Assing virtual objects to a particular collection.

Navigation steps to convert virtual objects to standard class:

→ Make sure your application have virtual objects
→ Click tools
→ Virtual objects
→ Virtual object manager
→ Click new
→ Next
→ Select Class names as Buttons
→ Next
→ Click on mark object
→ Take this "+" pointer (crosshair mark)
→ Drag over the object area
→ Next
→ Specify a name to virtual objects
→ Specify the collection
→ Finish

- Repeat the same steps again in case of multiple objects.
- Object → can be used for multipurpose uses, we can select is if we get confused in selecting the class of any object
- More of virtual objects can be scene in multimedia projects.

Script:

Window ("paint").Winobject ("colors").virtualButton ("Red").Click
Window ("paint").winobject("colors").Click
Window ("paint").winobject ("colors").virtualbutton ("green").Click

Defining a Virtual Object:

Virtual objects can be defined using the Virtual Object Wizard.

Virtual object collection is a group of virtual objects that is stored in the Virtual Object Manager under a descriptive name.

Note:

QuickTest does not support virtual objects for analog or low-level recording. For additional information about low-level recording

Automation Framework

What is Automation Framework?

Framework is nothing but the approach that we consistently follow during the automation process – a set of guidelines.

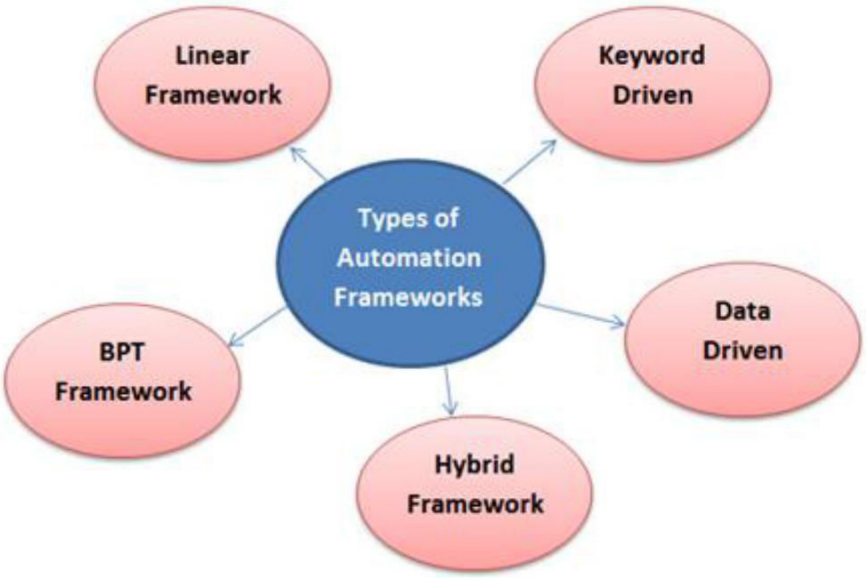
Any testing strategy that tries to incorporate some or all of these below points is your Test Automation Framework.

1. Reusability
2. Script's easy maintenance
3. Readability of scripts
4. Good workable folder structure for all the test assets
5. No hard coding values

6. No cascade of failures. (i.e. if one test fails, it should not cause the failure or stopping of the others)

Types of Frameworks:

Below are common frameworks used in QTP.

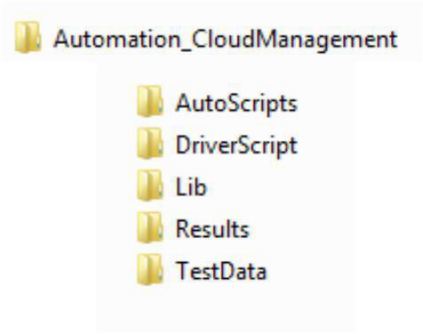


1. **Linear** – Simplest form of creating a test. Just write a one single program without modularity in sequential steps
2. **Keyword driven** – Create different keywords for different set of operations and in the main script we can just refer to these keywords.
3. **Data driven** – To run same set of operations on multiple sets of data that are kept in separate files, mostly excel sheets.
4. **Hybrid** – A combination framework that can be partly data driven and partly keyword driven/Functional Architecture.
5. **BPT** – This just means that programs are broken down into business components and are used with one or the other of the above types of frameworks

Hybrid Framework:

A combination of Data-driven and Functional decomposition (or functional architectural) framework is our Hybrid framework.

Framework folder structure:



Automation_CloudManagement: It is a automation folder for current project, where in all other automation assets are maintained.

AutoScripts Folder: It will have all the automation test cases/ test scripts

DriverScript Folder: It will have the script which is used to run one more automation test scripts based on the tester selection of automation test scripts in a spreadsheet.

Lib Folder: It will have all the function library files

1. Generic library files
2. Business library files

Generic Library files are technology specific functions (Web, Java, VB, SAP), Excel functions, Flat file function, etc. These functions can be used for any applications without any modifications.

Business library Functions are Application specific. For example Login to application, log out from application, application common navigations, application validation functions, etc.

TestData Folder: It will have application test data excel files

Results Folder: It will have execution results (HTML/XML/PDF)