UNIT-6
AUTOMATION TESTING

Automation testing:

Manual Testing is performed by a human sitting in front of a computer carefully executing the test steps. Automation Testing means using an automation tool to execute your test case suite.

Automation testing is a technique uses an application to implement entire life cycle of the software in less time and provides efficiency and effectiveness to the testing software.

Automation testing is an Automatic technique where the tester writes scripts by own and uses suitable software to test the software. It is basically an automation process of a manual process.

In other word, Automation testing uses automation tools to write and execute test cases, no manual involvement is required while executing an automated test suite. Usually, testers write test scripts and test cases using the automation tool and then group into test suites.

The main goal of Automation testing is to increase the test efficiency and develop software value.

MANUAL TETING	AUTOMATED TESTING
Manual Testing is not accurate at all times due to human error, hence it is less reliable.	Automated testing is more reliable, as it is performed by tools and/or scripts.
Manual Testing is time consuming, taking up human resources.	Automated testing is executed by software tools so it is significantly faster than a manual approach.
Investment is required for human resources	Investment is required for testing tools
Manual testing is only practical when the test cases are run once or twice and frequent repetition is not required.	Automated testing is practical option when the test cases are repeatedly over a long time period.
Manual testing allows for human observation, which may be more useful if the goal is user-friendliness or improved customer experience.	Automated testing does not entail human observation and cannot guarantee user-friendliness or positive customer experience.

Test cases to be automated can be selected using the following criterion to increase the automation ROI

- High Risk - Business Critical test cases
- Test cases that are repeatedly executed
- Test Cases that are very tedious or difficult to perform manually
- Test Cases which are time-consuming

The following categories of test cases are not suitable for automation:

- Test Cases that are newly designed and not executed manually at least once
- Test Cases for which the requirements are frequently changing
- Test cases which are executed on an ad-hoc basis.

Simple Steps to follow in Automation testing:

There are lots of helpful tools to write automation scripts, before using those tools it's better to identify the process which can be used to automate the testing,

- Identify areas within software to automate
- Choose the appropriate tool for test automation
- Write test scripts
- Develop test suits
- Execute test scripts

- Build result reports
- Find possible bugs or performance issue

Benefits of Automation Testing

Following are benefits of automated testing:

- 70% faster than the manual testing
- Wider test coverage of application features
- Reliable in results
- Ensure Consistency
- Saves Time and Cost
- Improves accuracy
- Human Intervention is not required while execution
- Increases Efficiency
- Better speed in executing tests
- Re-usable test scripts
- Test Frequently and thoroughly
- More cycle of execution can be achieved through automation
- Early time to market

What is a Test Framework?

A testing framework is a set of guidelines or rules used for creating and designing test cases. A framework is comprised of a combination of practices and tools that are designed to help QA professionals test more efficiently.

These guidelines could include coding standards, test-data handling methods, object repositories, processes for storing test results, or information on how to access external resources.

Benefits of a Test Automation Framework

- Improved test efficiency
- Lower maintenance costs
- Minimal manual intervention
- Maximum test coverage
- Reusability of code

Types of Automated Testing Frameworks

There are five common types of test automation frameworks, each with their own architecture and differing benefits and disadvantages. When building out a test plan, it's important to choose the framework that is right for you.

1. Linear Automation Framework
2. Modular Based Testing Framework
3. Data-Driven Framework
4. Keyword-Driven Framework
5. Hybrid Testing Framework

1 Linear Automation Framework

Linear Scripting Framework is a basic level test automation framework which is in the form of 'Record and Playback' in a linear fashion. This framework is also known as 'Record and Playback' framework. This type of framework is used to test small sized applications. In this type, creation, and execution of test script are done individually for each test case individually.

Using this framework, we could generate test scripts (Record and playback) without planning much or consume much time but it has its own drawbacks such as lack of reusability and hard coding the data does not allow to run with multiple data sets.

Advantages of a linear framework:

- There is no need to write custom code, so expertise in test automation is not necessary.
- This is one of the fastest ways to generate test scripts since they can be easily recorded in a minimal amount of time.
- The test workflow is easier to understand for any party involved in testing since the scripts are laid out in a sequential manner.
- This is also the easiest way to get up and running with automated testing, especially with a new tool. Most automated testing tools today will provide record-and-playback features, so you also won't need to plan extensively with this framework.

Disadvantages:

- The scripts developed using this framework aren't reusable. The data is hardcoded into the test script, meaning the test cases cannot be re-run with multiple sets and will need to be modified if the data is altered.
- Maintenance is considered a hassle because any changes to the application will require a lot of rework. This model is not particularly scalable as the scope of testing expands.

2 Modular Based Testing Framework

In the modular testing framework, testers create test scripts on module wise by breaking down the complete application under test into smaller, independent tests. In simple words, testers divide the application into multiple modules and create test scripts individually. These individual test scripts can be combined to make larger test scripts by using a master script to achieve the required scenarios. This master script is used to invoke the individual modules to run end to end test scenarios. In this framework, testers write function libraries to use it whenever required. This is AKA modularity framework or module-based framework.

Advantages of a Modular Framework:

- If any changes are made to the application, only the module and it's associated individual test script will needs to be fixed, meaning you won't have to tinker with the rest of the application and can leave it untouched.
- Creating test cases takes less effort because test scripts for different modules can be reused.

Disadvantages of a Modular Framework:

- Data is still hard-coded into the test script since the tests are executed separately, so you can't use multiple data sets.
- Programming knowledge is required to set up the framework.

3 Data-Driven Framework

Data-driven test automation framework is focused on separating the test scripts logic and the test data from each other. Allows us to create test automation scripts by passing different sets of test data. The test data set is kept in the external files or resources such as MS Excel Sheets, MS Access Tables, SQL Database, XML files etc., The test scripts connect to the external resources to get the test data. By using this framework we could easily make the test scripts work properly for different sets of test data. This framework significantly reduces the number of test scripts compared to module-based framework.

This framework gives more test coverage with reusable tests and flexibility in the execution of tests only when required and by changing only the input test data and reliable in terms of no impact on tests by changing the test data but it has its own drawbacks such as testers who work on this framework needs to have hands-on programming knowledge to develop test scripts.

Ex: Developing the Flight Reservation Login script using this method will involve two steps.

Step 1) Create a Test - Data file which could be Excel , CSV , or any other database source.

AgentName	**Password**
Jimmy	Mercury
Tina	MERCURY
Bill	MerCURY

Step 2) Develop Test Script and make references to your Test- Data source.

```
SystemUtil.Run "flight4a.exe","","","open"

Dialog("Login").WinEdit("Agent Name:").Set DataTable("AgentName", dtGlobalSheet)

Dialog("Login").WinEdit("Password:").Set DataTable("Password", dtGlobalSheet)

Dialog("Login").WinButton("OK").Click

'Check Flight Reservation Window has loaded

Window("Flight Reservation").Check CheckPoint("Flight Reservation")

**Note "dtGlobalSheet" is the default excel sheet provided by QTP.
```

Advantages of a Data-Driven Framework:

- Tests can be executed with multiple data sets.
- Multiple scenarios can be tested quickly by varying the data, thereby reducing the number of scripts needed.
- Hard-coding data can be avoided so any changes to the test scripts do not affect the data being used and vice versa.
- You'll save time by executing more tests faster.

Disadvantages

- You'll need a highly-experienced tester who is proficient in various programming languages to properly utilize this framework design. They will need to identify and format the external data sources and to write code (create functions) that connect the tests to those external data sources seamlessly.

- Setting up a data-driven framework takes a significant amount of time.

4 Keyword-Driven Framework

It is also known as table-driven *testing* or action word based *testing.* In Keyword-driven testing, we use a table format to define keywords or action words for each function or method that we would execute. It performs automation test scripts based on the keywords specified in the excel sheet. By using this Framework, testers can work with keywords to develop any test automation script, testers with less programming knowledge would also be able to work on the test scripts. The logic to read keywords and call the required action mentioned in the external excel sheet is placed in the main class. Keyword-driven testing is similar to data-driven testing.

Even though to work on this framework doesn't require much programming skills but the initial setup (implement the framework) requires more expertise.

In the table, keywords are stored in a step-by-step fashion with an associated object, or the part of the UI that the action is being performed on. For this approach to work properly, a shared object repository is needed to map the objects to their associated actions.

Keyword Table:

Step Number	Description	Keyword	Object	Action
Step 1	Click user portal link on homepage	clicklink	Login Button	
Step 2	Enter user name	Inputdata	Login Username	
Step 3	Enter password	Inputdata	Login password	
Step 4	Verify user log in information	verifylogin		
Step 5	Log in to the application	login	Submit Button	

Once the table has been set up, all the testers have to do is write the code that will prompt the necessary action based on the keywords. When the test is run, the test data is read and pointed towards the corresponding keyword which then executes the relevant script.

Advantages of Keyword-Driven Frameworks:

- Minimal scripting knowledge is needed.
- A single keyword can be used across multiple test scripts, so the code is reusable.
- Test scripts can be built independent of the application under test.

Disadvantages:

- The initial cost of setting up the framework is high. It is time-consuming and complex. The keywords need to be defined and the object repositories / libraries need to be set up.
- You need an employee with good test automation skills.
- Keywords can be a hassle to maintain when scaling a test operation. You will need to continue building out the repositories and keyword tables.

5 Hybrid Test Automation Framework

Hybrid Test automation framework is the combination of two or more frameworks mentioned above. It attempts to leverage the strengths and benefits of other frameworks for the particular test environment it manages. Most of the teams are building this hybrid driven framework in the current market.

Every application is different, and so should the processes used to test them. As more teams move to an agile model, setting up a flexible framework for automated testing is crucial. A hybrid framework can be more easily adapted to get the best test results.

Conclusion of Test Automation Framework

One recommended approach for implementing a hybrid framework for automated testing, is to find a tool that can quickly and easily adapt to your processes. When choosing an automated testing tool, you should look for one that is flexible and can support a wide range of applications and languages. This will enable your team, regardless of background and skill set, to contribute to your testing efforts. **Test Complete**, our automated testing tool that allows QA teams to create and run UI and functional tests across mobile, desktop and web applications, provides a comprehensive environment for building and maintaining automated testing projects.

Test Complete supports a wide variety of scripting languages, such as Python, JavaScript, Java and VBScript, and comes with an extensive object library that has over fifty thousand object properties and more than five hundred control types. The tool's record-and-playback feature enables non-programmers to create complex tests without writing a single line of code. With these features, any team can easily build a hybrid framework that Test Complete supports and will allow teams to implement a myriad of testing types, including data-driven testing and keyword-driven testing.

WinRunner :

- HP **WinRunner** software was an automated functional GUI testing tool that allowed a user to record and play back user interface (UI) interactions as *test scripts.*
- As a functional test suite, it worked with HP QuickTest Professional and supported enterprise quality assurance.
- It captured, verified and replayed user interactions automatically, in order to identify defects and determine whether business processes worked as designed.
- The software implemented a proprietary Test Script Language (TSL) that allowed customization and parameterization of user input.
- HP WinRunner was originally written by Mercury Interactive.
- Mercury Interactive was subsequently acquired by Hewlett Packard (HP) in 2006.
- On February 15, 2008, HP Software Division announced the end of support for HP WinRunner versions 7.5, 7.6, 8.0, 8.2, 9.2—suggesting migration to HP Functional Testing software as a replacement.

LoadRunner

- LoadRunner is a software testing tool from Micro Focus.
- It is used to test applications, measuring system behaviour and performance under load.
- LoadRunner can simulate thousands of users concurrently using application software, recording and later analyzing the performance of key components of the application.
- LoadRunner simulates user activity by generating messages between application components or by simulating interactions with the user interface such as keypresses or mouse movements.
- The messages and interactions to be generated are stored in scripts. LoadRunner can generate the scripts by recording them, such as logging HTTP requests between a client web browser and an application's web server.

- Hewlett Packard Enterprise acquired LoadRunner as part of its acquisition of Mercury Interactive in November 2006. In Sept 2016, Hewlett Packard Enterprise announced it is selling its software business, including Mercury products, to Micro Focus. As of 01-Sept-2017, the acquisition was complete.

JMeter:

- The Apache JMeter is pure Java open source software, which was first developed by Stefano Mazzocchi of the Apache Software Foundation, designed to load test functional behavior and measure performance.
- You can use JMeter to analyze and measure the performance of web application or a variety of services.
- Performance Testing means testing a web application against heavy load, multiple and concurrent user traffic.
- JMeter originally is used for testing Web Application or FTP application. Nowadays, it is used for a functional test, database server test etc.

JMeter Advantages:

- **Open source license**: JMeter is totally free, allows developer use the source code for the development
- **Friendly GUI**: JMeter is extremely easy to use and doesn't take time to get familiar with it
- **Platform independent**: JMeter is 100% pure Java desktop application. So it can run on multiple platforms
- **Full multithreading framework**. JMeter allows concurrent and simultaneous sampling of different functions by a separate thread group
- **Visualize Test Result:** Test result can be displayed in a different format such as chart, table, tree and log file
- **Easy installation**: You just copy and run the *.bat file to run JMeter. No installation needed.
- **Highly Extensible**: You can write your own tests. JMeter also supports visualization plugins allow you to extend your testing
- **Multiple testing strategy**: JMeter supports many testing strategies such as Load Testing, Distributed Testing, and Functional Testing.
- **Simulation**: JMeter can simulate multiple users with concurrent threads, create a heavy load against web application under test
- **Support multi-protocol**: JMeter does not only support web application testing but also evaluate database server performance. All basic protocols such as HTTP, JDBC, LDAP, SOAP, JMS, and FTP are supported by JMeter
- **Record & Playback - Record** the user activity on the browser and simulate them in a web application using JMeter
- **Script Test**: Jmeter can be integrated with Bean Shell & Selenium for automated testing

Introduction to winrunner:

WinRunner is Mercury Interactive's enterprise functional testing tool for Microsoft Windows applications. WinRunner helps you automate the testing process, from test development to execution. You create adaptable and reusable test scripts that challenge the functionality of your application. Prior to a software release, you can run these tests in a single overnight run—enabling you to detect defects and ensure superior software quality.

Features

WinRunner is:

- Functional Regression Testing Tool
- Windows Platform Dependent
- Only for Graphical User Interface (GUI) based Application
- Based on Object Oriented Technology (OOT) concept
- Only for Static content
- Record/Playback Tool

The WinRunner Testing Process

Testing with WinRunner involves six main stages:

1. Create the GUI Map
2. Create Tests
3. Debug Tests
4. Run Tests
5. View Results
6. Report Defects

1 Create the GUI Map

- The first stage is to create the GUI map so WinRunner can recognize the GUI objects in the application being tested.
- Use the RapidTest Script wizard to review the user interface of your application and systematically add descriptions of every GUI object to the GUI map.
- Alternatively, you can add descriptions of individual objects to the GUI map by clicking objects while recording a test.
- Note that when you work in *GUI Map per Test* mode, you can skip this step.

2 Create Tests

- Next, you create test scripts by recording, programming, or a combination of both.
- While recording tests, insert checkpoints where you want to check the response of the application being tested.
- You can insert checkpoints that check GUI objects, bitmaps, and databases.
- During this process, WinRunner captures data and saves it as *expected results*—the expected response of the application being tested.

3 Debug Tests

- You run tests in Debug mode to make sure they run smoothly.
- You can set breakpoints, monitor variables, and control how tests are run to identify and isolate defects.
- Test results are saved in the debug folder, which you can discard once you've finished debugging the test.
- When WinRunner runs a test, it checks each script line for basic syntax errors, like incorrect syntax or missing elements in **If**, **While**, **Switch**, and **For** statements.

- You can use the **Syntax Check** options (**Tools >Syntax Check**) to check for these types of syntax errors before running your test.

4 Run Tests

- You run tests in Verify mode to test your application.
- Each time WinRunner encounters a checkpoint in the test script, it compares the current data of the application being tested to the expected data captured earlier.
- If any mismatches are found, WinRunner captures them as *actual results*.

5 View Results

- You determine the success or failure of the tests.
- Following each test run, WinRunner displays the results in a report.
- The report details all the major events that occurred during the run, such as checkpoints, error messages, system messages, or user messages.
- If mismatches are detected at checkpoints during the test run, you can view the expected results and the actual results from the Test Results window.
- In cases of bitmap mismatches, you can also view a bitmap that displays only the difference between the expected and actual results.
- You can view your results in the standard WinRunner report view or in the Unified report view.
- The WinRunner report view displays the test results in a Windows-style viewer.
- The Unified report view displays the results in an HTML-style viewer (identical to the style used for QuickTest Professional test results).

6 Report Defects

- If a test run fails due to a defect in the application being tested, you can report information about the defect directly from the Test Results window.
- This information is sent via e-mail to the quality assurance manager, who tracks the defect until it is fixed.
- You can also insert **tddb_add_defect** statements to your test script that instruct WinRunner to add a defect to a TestDirector project based on conditions you define in your test script.

WinRunner Testing Record Modes

- WinRunner facilitates easy test creation by recording how you work on your application.
- As you point and click GUI (Graphical User Interface) objects in your application, WinRunner generates a test script in the C-like Test Script Language (TSL).
- You can further enhance your test scripts with manual programming.
- WinRunner includes the Function Generator, which helps you quickly and easily adds functions to your recorded tests.

WinRunner includes two modes for recording tests:

1 Context Sensitive

- *Context Sensitive* mode records your actions on the application being tested in terms of the GUI objects you select (such as windows, lists, and buttons), while ignoring the physical location of the object on the screen.
- Every time you perform an operation on the application being tested, a TSL statement describing the object selected and the action performed is generated in the test script.
- As you record, WinRunner writes a unique description of each selected object to a GUI map. The GUI map consists of files maintained separately from your test scripts.
- If the user-interface of your application changes, you have to update only the GUI map, instead of hundreds of tests.
- This allows you to easily reuse your Context Sensitive test scripts on future versions of your application.
- To run a test, you simply play back the test script.

- WinRunner emulates a user by moving the mouse pointer over your application, selecting objects, and entering keyboard input.
- WinRunner reads the object descriptions in the GUI map and then searches in the application being tested for objects matching these descriptions.
- It can locate objects in a window even if their placement has changed.

2 Analog

- *Analog* mode records mouse clicks, keyboard input, and the exact x- and y-coordinates traveled by the mouse.
- When the test is run, WinRunner retraces the mouse tracks.
- Use Analog mode when exact mouse coordinates are important to your test, such as when testing a drawing application.

GUI MAP

- When WinRunner runs tests, it simulates a human user by moving the mouse cursor over the application, clicking GUI objects and entering keyboard input.
- Like a human user, WinRunner must learn the GUI of an application in order to work with it.
- WinRunner does this by learning the GUI objects of an application and their properties and storing these object descriptions in the GUI map.
- You can use the GUI Spy to view the properties of any GUI object on your desktop, to see how WinRunner identifies it.
- WinRunner can learn the GUI of your application in the following ways:
  1. by using the RapidTest Script wizard to learn the properties of all GUI objects in every window in your application
  2. by recording in your application to learn the properties of all GUI objects on which you record
  3. by using the GUI Map Editor to learn the properties of an individual GUI object, window, or all GUI objects in a window
- If the GUI of your application changes during the software development process, you can use the GUI Map Editor to learn individual windows and objects in order to update the GUI map.

Identifying GUI Objects

- When you work in Context Sensitive mode, you can test your application as the user sees it—in terms of GUI objects—such as windows, menus, buttons, and lists.
- Each object has a defined set of properties that determines its behavior and appearance.
- WinRunner learns these properties and uses them to identify and locate GUI objects during a test run.
- Note that in Context Sensitive mode, WinRunner does not need to know the physical location of a GUI object to identify it.
- You can use the GUI Spy to view the properties of any GUI object on your desktop, to see how WinRunner identifies it.
- WinRunner stores the information it learns in a *GUI map*.
- When WinRunner runs a test, it uses the GUI map to locate objects: It reads an object's description in the GUI map and then looks for an object with the same properties in the application being tested. You can view the GUI map in order to gain a comprehensive picture of the objects in your application.

The GUI map is actually the sum of one or more *GUI map files*. There are two modes for organizing GUI map files:

1 *Global GUI Map File* mode

2 *GUI Map File per Test* mode

1 *Global GUI Map File* mode

- You can create a GUI map file for your entire application, or for eachwindow in your application.
- Multiple tests can reference a common GUI map file.
- This is the default mode in WinRunner.
- For experienced WinRunner users, this is the most efficient way to work.

2 *GUI Map File per Test* mode

- WinRunner can automatically create a GUI map file for each test you create.
- You do not need to worry about creating, saving, and loading GUI map files.
- If you are new to WinRunner, this is the simplest way to work.
- At any stage in the testing process, you can switch from the GUI Map File per Test mode to the Global GUI Map File mode.

How a Test Identifies GUI Objects

- You create tests by recording or programming *test scripts*.
- A test script consists of statements in Mercury Interactive's test script language (TSL).
- Each TSL statement represents mouse and keyboard input to the application being tested.
- WinRunner uses a *logical name* to identify each object: for example "Print" for a Print dialog box, or "OK" for an OK button.
- The logical name is actually a nickname for the object's *physical description*.
- The physical description contains a list of the object's physical properties: the Print dialog box, for example, is identified as a window with the label "Print".
- The logical name and the physical description together ensure that each GUI object has its own unique identification.

The GUI MapEditor

- You can view the contents of the GUI map at any time by choosing **Tools > GUI Map Editor**.
- In the GUI Map Editor, you can view either the contents of the entire GUI map or the contents of individual GUI map files.
- GUI objects are grouped according to the window in which they appear in the application.

RapidTest Script Wizard

You can use the RapidTest Script wizard before you start to test in order to teach WinRunner all the GUI objects in your application at once. This ensures that WinRunner has a complete, well-structured basis for all your Context Sensitive tests. The descriptions of GUI objects are saved in GUI map files. Since all test users can share these files, there is no need for each user to individually relearn the GUI.

Note: You can use the RapidTest Script wizard only when you work in the *Global GUI Map File* mode (the default mode, which is described in this chapter). All tests created in WinRunner version 6.02 or earlier use this mode.

When you work in the *GUI Map File per Test* mode, the RapidTest Script wizard is not available. The RapidTest Script wizard is also not available if the WebTest or certain other add-ins are loaded. To find out whether the RapidTest wizard is available with the add-in(s) you are using, refer to your add-in documentation.

The simplest and most thorough way for WinRunner to learn your application is by using the RapidTest Script wizard. The RapidTest Script wizard enables WinRunner to learn all windows and objects in your application being tested at once. The wizard systematically opens each window in your application and learns the properties of the GUI objects it contains. WinRunner provides additional methods for learning the properties of individual objects.

WinRunner then saves the information in a GUI map file. WinRunner also creates a startup script which includes a **GUI_load** command that loads this GUI map file.

Creating Tests

You can create tests using both recording and programming.

Usually, you start by recording a basic *test script*. As you record, each operation you perform generates a statement in Mercury Interactive's Test Script Language (TSL).

These statements appear as a test script in a test window. You can then enhance your recorded test script, either by typing in additional TSL functions and programming elements or by using WinRunner's visual programming tool, the Function Generator.

Two modes are available for recording tests:

1 ***Context Sensitive*** records the operations you perform on your application by identifying Graphical User Interface (GUI) objects.

2 ***Analog*** records keyboard input, mouse clicks, and the precise x- and y-coordinates traveled by the mouse pointer across the screen.

You can add GUI, bitmap, text, and database checkpoints, as well as synchronization points to your test script. Checkpoints enable you to check your application by comparing its current behavior to its behavior in a previous version. Synchronization points solve timing and window location problems that may occur during a test run.

You can create data-driven tests, which are tests driven by data stored in an internal table.

To create a test script, you perform the following main steps:

1 Decide on the functionality you want to test. Determine the checkpoints and synchronization points you need in the test script.

2 Document general information about the test in the Test Properties dialog box.

3 Choose a Record mode (*Context Sensitive* or *Analog*) and record the test on your application.

4 Assign a test name and save the test in the file system or in your TestDirector project.

Test Scripts with Programming

- When you record a test, a test script is generated in Mercury Interactive's Test Script Language (TSL). Each line WinRunner generates in the test script is a *statement*.
- A statement is any expression that is followed by a semicolon.
- Each TSL statement in the test script represents keyboard and/or mouse input to the application being tested.
- A single statement may be longer than one line in the test script.
  For example:
  if (button_check_state("Underline", OFF) == E_OK)
  report_msg("Underline check box is unavailable.");

- TSL is a C-like programming language designed for creating test scripts.
- It combines functions developed specifically for testing with general purpose programming language features such as variables, control-flow statements, arrays, and user-defined functions.
- You enhance a recorded test script simply by typing programming elements into the test window, If you program a test script by typing directly into the test window, remember to include a semicolon at the end of each statement.
- TSL is easy to use because you do not have to compile. You simply record or type in the test script and immediately execute the test.

TSL includes four types of functions:

1 Context Sensitive functions perform specific tasks on GUI objects, such as clicking a button or selecting an item from a list. Function names, such as **button_press** and **list_select_item**, reflect the function's purpose.

2 Analog functions depict mouse clicks, keyboard input, and the exact coordinates traveled by the mouse.

3 *Standard* functions perform general purpose programming tasks, such as sending messages to a report or performing calculations.

4 *Customization* functions allow you to adapt WinRunner to your testing environment.

Adding Checkpoints to Your Test

Checkpoints allow you to compare the current behavior of the application being tested to its behavior in an earlier version.

You can add four types of checkpoints to your test scripts:

1) GUI checkpoints verify information about GUI objects. For example, you can check that a button is enabled or see which item is selected in a list.

2) Bitmap checkpoints take a "snapshot" of a window or area of your application and compare this to an image captured in an earlier version.

3) Text checkpoints read text in GUI objects and in bitmaps and enable you to verify their contents.

4) Database checkpoints check the contents and the number of rows and columns of a result set, which is based on a query you create on your database.

RUNNING TESTS

Once you have developed a test script, you run the test to check the behavior of your application. When you run a test, WinRunner interprets your test script, line by line. The execution arrow in the left margin of the test script marks each TSL statement as it is interpreted. As the test runs, WinRunner operates your application as though a person were at the controls.

You can run your tests in three modes:

1. Verify run mode, to check your application
2. Debug run mode, to debug your test script
3. Update run mode, to update the expected results

WinRunner Test Run Modes

WinRunner provides three modes in which to run tests—Verify, Debug, and Update. You use each mode during a different phase of the testing process. You can set the default run mode in the General Options dialog box.

Verify

Use the Verify mode to check your application. WinRunner compares the *current* response of your application to its *expected* response. Any discrepancies between the current and expected responses are captured and saved as *verification results*. When you finish running a test, by default the Test-Results window opens for you to view the verification results. You can save as many sets of verification results as you need. To do so, save the results in a new folder each time you run the test. You specify the folder name for the results using the Run Test dialog box. This dialog box opens each time you run a test in Verify mode.

Note: Before your run a test in Verify mode, you must have expected results for the checkpoints you created. If you need to update the expected results of your test, you must run the test in Update mode.

Debug

Use the Debug mode to help you identify bugs in a test script. Running a test in Debug mode is the same as running a test in Verify mode, except that debug results are always saved in the *debug* folder. Because only one set of debug results is stored, the Run Test dialog box does not open automatically when you run a test in Debug mode. When you finish running a test in Debug mode, the Test Results window does not open automatically. To open the Test Results window and view the debug results,

you can click the **Test Results** button on the main toolbar

or

choose **Tools > Test Results**.

Use WinRunner's debugging facilities when you debug a test script:

- Use the Step commands to control how your tests run.
- Set breakpoints at specified points in the test script to pause tests while they run.
- Use the Watch List to monitor variables in a test script while the test runs.
- Use the Call Chain to follow and navigate the test flow.

Update

Use the Update mode to update the *expected* results of a test or to create a new expected results folder. For example, you could *update* the expected results for a GUI checkpoint that checks a push button, in the event that the push button default status changes from enabled to disabled. You may want to *create* an additional set of expected results if, for example, you have one set of expected results when you run your application in Windows 98 and another set of expected results when your run your application in Windows NT.

By default, WinRunner saves expected results in the *exp* folder, overwriting any existing expected results.

You can update the expected results for a test in one of two ways:

- by globally overwriting the full existing set of expected results by running the entire test using a Run command
- by updating the expected results for individual checkpoints and synchronization points using the Run from Arrow command or a Step command

Debugging Tests/ Controlling Your Test Run

After you create a test script you should check that it runs smoothly, without errors in syntax or logic. In order to detect and isolate defects in a script, you can use the Step and Pause commands to control test execution.

The following Step commands are available:

 The Step command runs a single line of a test script.

 The Step Into command calls and displays another test or user-defined function.

 The Step Out command—used in conjunction with Step Into—completes the execution of a called test or user-defined function.

 The Step to Cursor command runs a selected section of a test script.

In addition, you can use the Pause command or the **pause** function to temporarily suspend the test run.

You can also control the test run by setting **breakpoints**. A breakpoint pauses a test run at a pre-determined point, enabling you to examine the effects of the test on your application.

To help you debug your tests, WinRunner enables you to **monitor variables** in a test script. You define the variables you want to monitor in a Watch List. As the test runs, you can view the values that are assigned to the variables. You can view the current values of monitored variables in the Watch List pane of the Debug Viewer.

You can use the **call chain to follow and navigate** the test flow. At each break during a test run—such as after a Step command, at a breakpoint, or at the end of a test, you can view the current chain of called tests and functions in the Call Chain pane of the Debug Viewer.

When you debug a test script, you run the test in the Debug mode. The results of the test are saved in a *debug* folder. Each time you run the test, the previous debug results are overwritten. Continue to run the test in the Debug mode until you are ready to run it in Verify mode.

Analyzing Test Results

After you run a test, you can view the results in the Test Results window. The appearance of this window depends on the Report View option you select in the **Run** category of the General Options dialog box.

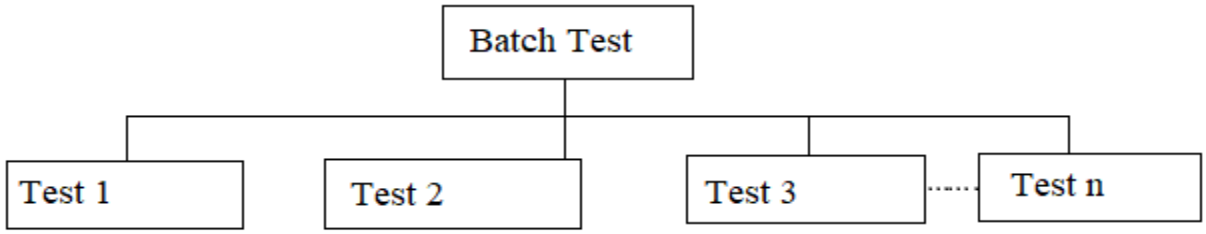
There are two types of Test Results Views:

1. **WinRunner report view**—Displays the test results in a Windows-style viewer. If you run a test that includes a call to a QuickTest test, the WinRunner report view displays only basic information about the results of the QuickTest test. When running tests that call QuickTest tests, it is recommended to use the Unified report view.
2. **Unified report view**—Displays the results in an HTML-style viewer. The unified TestFusion report viewer is identical to the style used for QuickTest Professional test results. If you run a test that includes a call to a QuickTest test (version 6.5 or later), the unified report view enables you to view detailed results of each step in the called QuickTest test. Regardless of the selected report

view, the test results window always contains a description of the major events that occurred during the test run, such as GUI, bitmap, or database checkpoints, file comparisons, and error messages. It also includes tables and pictures to help you analyze the results.

Batch Tests

You can run a group of tests unattended by creating and executing a single batch test. A batch test is a test script that contains call statements to other tests. It opens and executes each test and saves the test results.



- A batch test looks like a regular test that includes call statements.
- A test becomes a "batch test" when you select the **Run in batch mode** option in the **Run** category of the General Options dialog box before you execute the test.
- When you run a test in Batch mode, WinRunner suppresses all messages that would ordinarily be displayed during the test run, such as a message reporting a bitmap mismatch.
- WinRunner also suppresses all **pause** statements and any halts in the test run resulting from run time errors. By suppressing all messages, WinRunner can run a batch test unattended.
- This differs from a regular, interactive test run in which messages appear on the screen and prompt you to click a button in order to resume test execution.
- A batch test enables you to run tests overnight or during off-peak hours, so that you can save time while testing your application.
- At each break during a test run—such as after a Step command, at a breakpoint, or at the end of a test, you can view the current chain of called tests in the Call Chain pane of the Debug Viewer window. When a batch test run is completed, you can view the results in the Test Results window.
- The window displays the results of all the major events that occurred during the run.