UNIT-5

5.1 WIN RUNNER

Win Runner is widely used Automated Software Testing Tool for Functional testing. It was developed by Mercury Interactive. It supports C and web technologies such as (VB, VC++, D2K, Java, HTML, Power Builder, Delphe, Cibell (ERP)).

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). We can further enhance our test scripts with manual programming. WinRunner includes the Function Generator, which helps quickly and easily add functions to our recorded tests.

The important aspects of WinRunner are:

1. We can do functional/regression testing of a variety of application software written in programming languages such as PowerBuilder, Visual Basic, C/C++, and Java. We can also carry out the testing on ERP/CRM software packages.
2. Performs testing in all flavors of Windows operating systems and different browser environments such as Internet Explorer and Netscape Navigator.
3. We can record the GUI operations in the 'record' mode. WinRunner automatically creates a test script.
4. We can add checkpoints to compare actual and expected results. The checkpoints can be GUI checkpoints, bitmap checkpoints and web links.
5. It provides a facility for synchronization of test cases.
6. Data Driver Wizard provides the facility to convert a recorded test into a data driven test. So, We can replace data with variables within a test script.
7. Database checkpoints are used to verify data in a database during automated testing. The records that are inserted, deleted, modified, or updated will be highlighted so that We can ensure database integrity and transaction accuracy.
8. The Virtual Object Wizard of WinRunner is used to teach WinRunner to recognize, record, and replay custom objects.
9. The reporting tools provide the facility to generate automatically the test reports and analyze the defects.
10. WinRunner can be integrated with the testing management tool TestDirector to automate many of the activities in the testing process.

WinRunner includes two modes for recording tests:

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.

This allows you to easily reuse your Context Sensitive test scripts on future versions of your application.

Analog

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

Starting WinRunner

To start WinRunner:

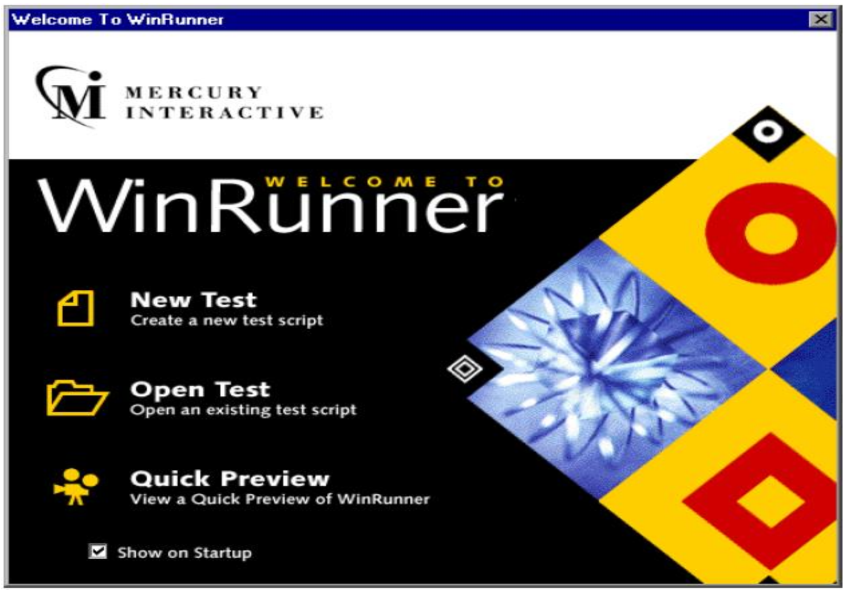
Testing an Application using WinRunner

After installing the WinRunner on your computer, invoke the WinRunner application

Start -> Programs ->WinRunner ->WinRunner

The opening screen of the WinRunner application is displayed, prompting you to select one of the three options:

- New Test: To create a new test script
- Open Test: To open an existing test script
- Quick Preview: To view the quick preview of WinRunner



Recording Test Cases

To test any application, first we can run the application and understand its operation. Then, you can invoke WinRunner, again run the application and record the GUI operations.

During the recording mode, WinRunner will capture all your actions, which button you pressed, where you clicked the mouse etc. You need to work with the application as usual and perform all the actions to be tested.

Once the recording is completed, WinRunner generates a script in TSL (Test Script Language). You can run this test script generated by WinRunner to view the results. The test results will show whether the test has passed or failed.

Each test you create or run is displayed by WinRunner in a test window. You can open many tests at one time.

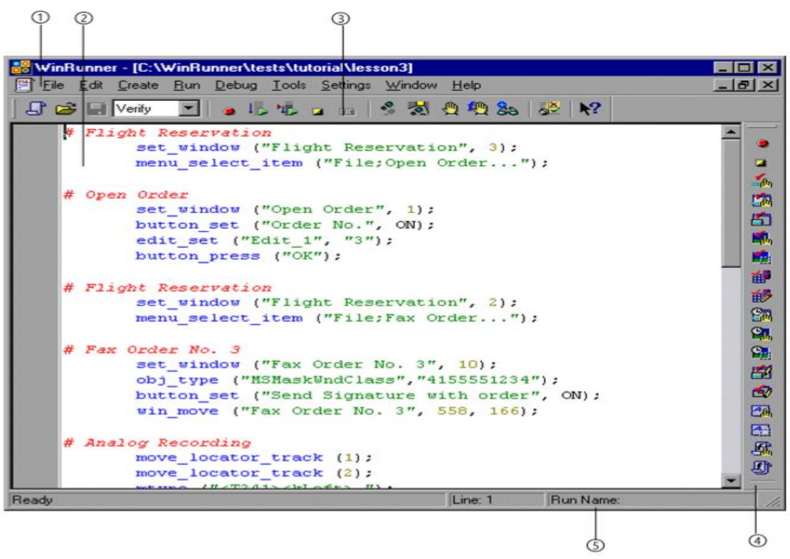
1 *The WinRunner window displays all open tests.*

2 *Each test appears in its own test window. You use this window to record, program, and edit test scripts.*

3 *Buttons on the Standard toolbar help you quickly open, run, and save tests.*

4 *The User toolbar provides easy access to test creation tools.*

5 *The status bar displays information about selected commands and the current test run.*


①
②
③
④
⑤
WinRunner - [C:\WinRunner\tests\tutorial\lesson3]
File Edit Create Run Debug Tools Settings Window Help
Verify
Ready
Line: 1
Run Name:

```
# Flight Reservation
        set_window ("Flight Reservation", 3);
        menu_select_item ("File;Open Order...");

# Open Order
        set_window ("Open Order", 1);
        button_set ("Order No.", ON);
        edit_set ("Edit_1", "3");
        button_press ("OK");

# Flight Reservation
        set_window ("Flight Reservation", 2);
        menu_select_item ("File;Fax Order...");

# Fax Order No. 3
        set_window ("Fax Order No. 3", 10);
        obj_type ("MSMaskWndClass","4155551234");
        button_set ("Send Signature with order", ON);
        win_move ("Fax Order No. 3", 558, 166);

# Analog Recording
        move_locator_track (1);
        move_locator_track (2);
```

The procedure for recording a test case is as follows:

Step 1: Open a new document: File -> New (or) Select "New Test" from the WinRunner's Welcome screen.

Step 2: Open (run) the application to be tested.

Step 3: Start recording a test case.

Create ->Record - Context Sensitive (or) click on the toolbar's "Record" button once, to record in Context Sensitive mode.

Step 4: Select the application to be tested by clicking on the application's title bar.

Step 5: Perform all the actions to be recorded.

Step 6: Once all required actions are recorded, stop the recording.

Create -> Stop (or) Click on the toolbar's "Stop" button to stop the recording WinRunner generates the script for the recoded actions.

The procedure for running a test case is as follows:

Step 1: Open the test script to be executed.

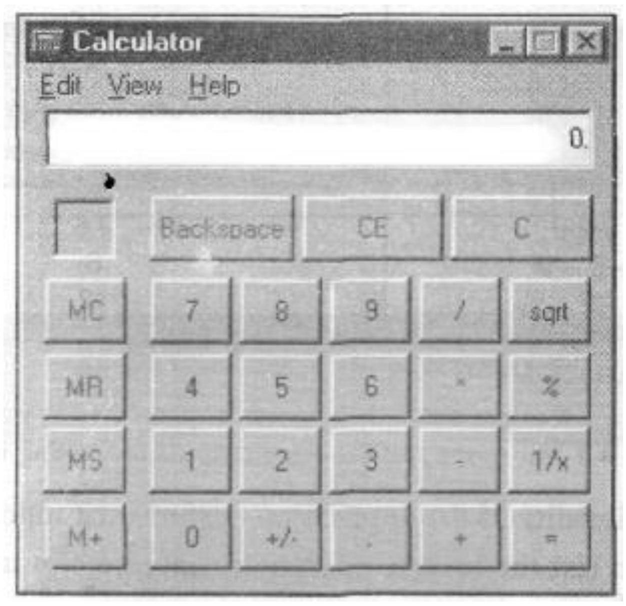
Step 2: Run the test

Run -> Run from top (or) press F5 to run the test.

WinRunner executes the generated script and displays the results in the Test Results window.

We will now illustrate using WinRunner to test the "Standard Calculator" application available on your Windows system.we can invoke the calculator application from the desktop

Start -> Programs -> Accessories -> Calculator. The GUI of the "Calculator" application is shown in Fig.



The symbols on the buttons of Calculator application represent the following functions:

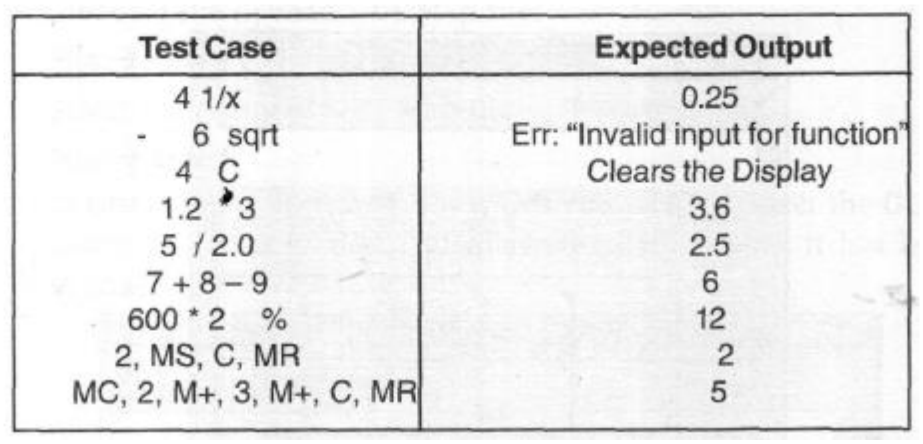
+ :	To perform addition
- :	To perform subtraction
* :	To perform multiplication
/ :	To perform division
. :	Decimal point
sqrt :	To find square root of a number
% :	To find percent
1/x :	To find inverse of a number
MC :	To clear the memory
MR :	To recall from memory
MS :	To save in the memory

M+ :	To add to the memory
C :	To clear the current calculation
CE :	To clear the displayed number
+/- :	To give sign to a number (positive or negative)

Backspace: To remove left most digit

To test the complete functionality of the application, we need to generate test cases in such a way that all the buttons are made use of. We need to generate some test cases which will give correct output and also some test cases which will give error messages. Table gives such test cases and the expected output for each test case.

Test Cases and the Expected Output for Testing the Calculater



Test Case	Expected Output
4 1/x	0.25
- 6 sqrt	Err: "Invalid input for function"
4 C	Clears the Display
1.2 * 3	3.6
5 / 2.0	2.5
7 + 8 – 9	6
600 * 2 %	12
2, MS, C, MR	2
MC, 2, M+, 3, M+, C, MR	5

To test the functionality of the application perform the following steps:

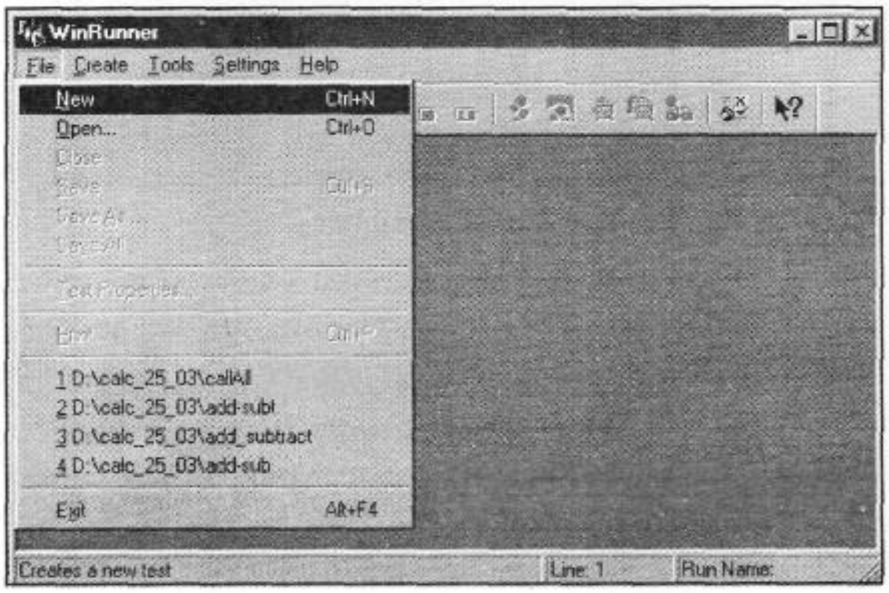
Test Case #1: To test the Inverse operation (inverse of 4 using 1/x button)

Step 1: Open WinRunner application.

Step 2: Open Calculator application.

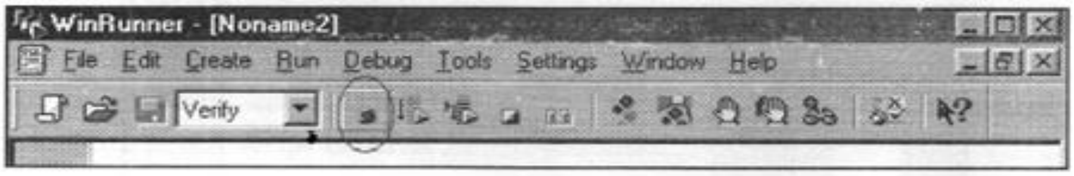
Step 3: Create a new document as shown in Figure.

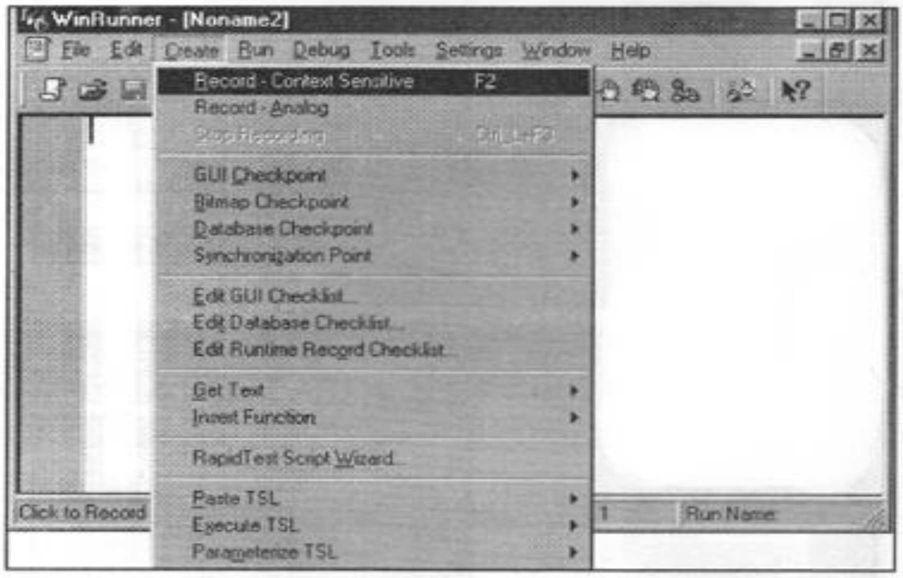
File -> New or Click New option on tool bar or press Ctrl+N



Step 4: Start recording

Create -> Record-Context Sensitive (or) press F2 (or) Click on the toolbar
Click the (Record-Context Sensitive) button on the toolbar of WinRunner as shown in Figure or Select "Record - Context sensitive" option from the "Create" menu as shown in Figure.



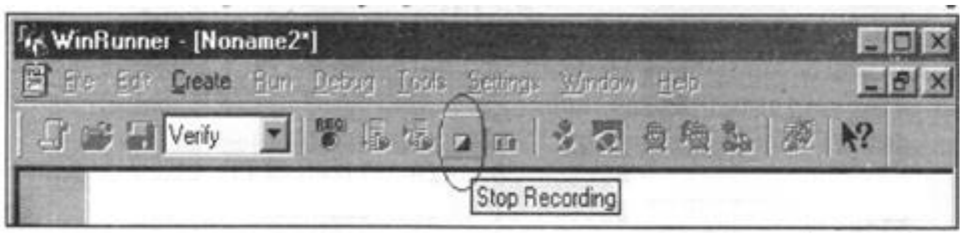


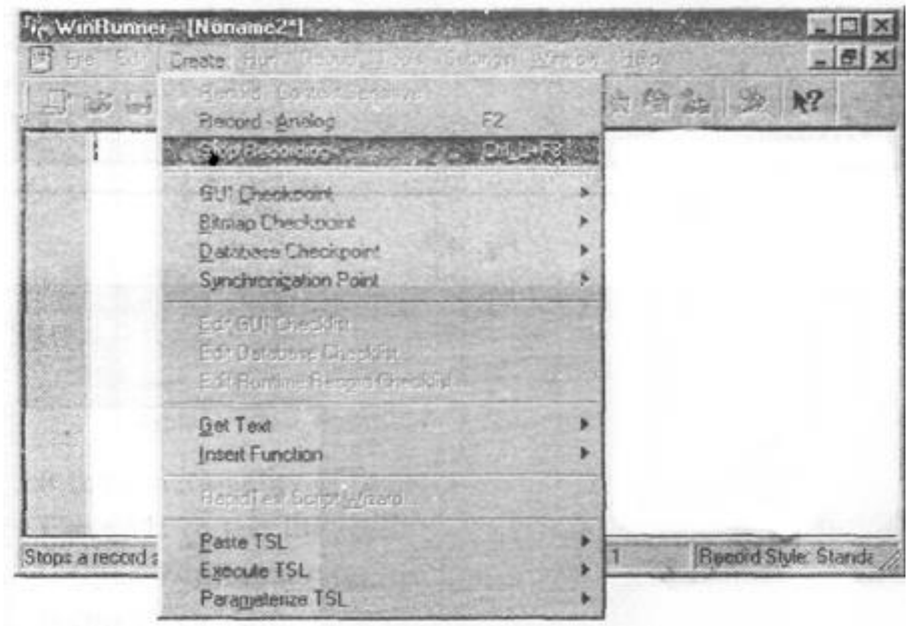
Step 5: Select the Calculator application and start recording the actions. a Click "4" on the Calculator

- Click the "1/x" button on the Calculator to find the inverse of 4.
- The result, 0.25 will be displayed on the Calculator.

Step 6: Stop the Recording process.

Create -> Stop Recording (or) Click (Stop) on toolbar
Click (Stop Recording) button on the toolbar of WinRunner as shown in Figure or Select the "Stop Recording" option from the "Create" menu as shown in Figure.

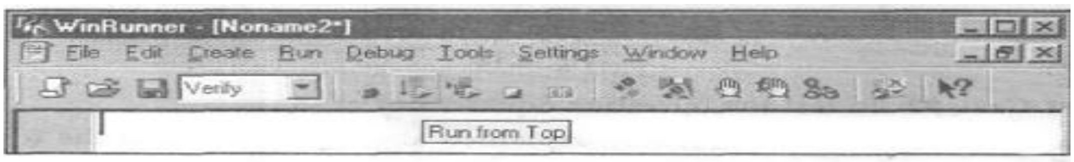


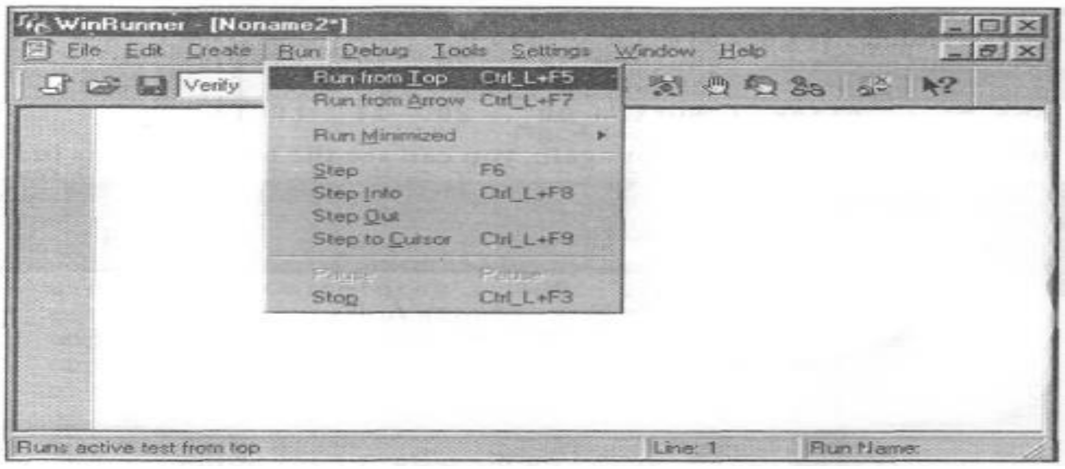


Step 8: Save the file as "inverse" in the selected folder.

File-»Save
In the "Save" dialog box that appears, save the test script with name "inverse".

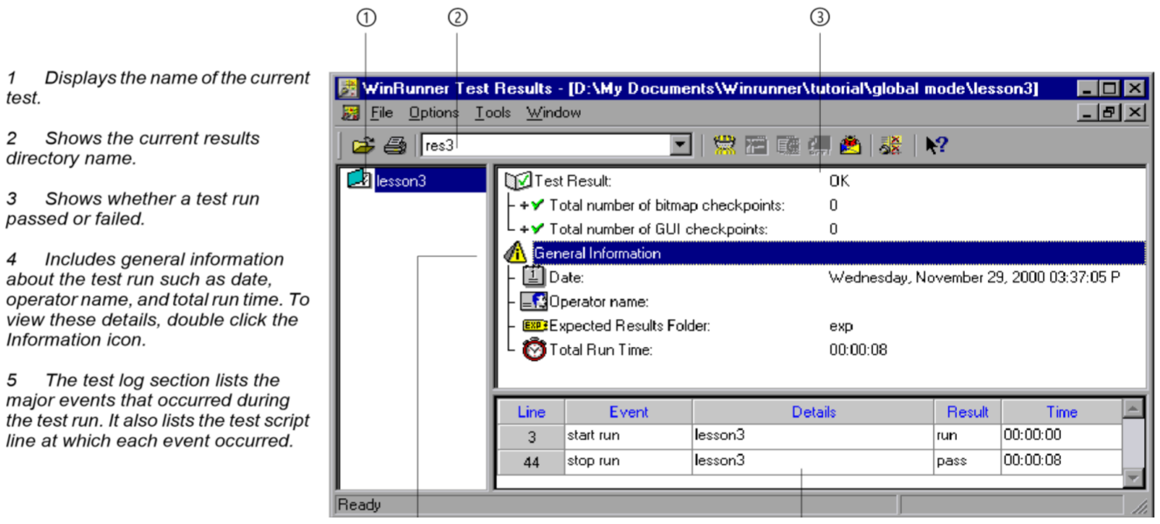
Step 9: Run the test script generated by WinRunner.
Press F5 or Click (Run from Top) on the toolbar





Step 10: After executing the TSL statements, WinRunner generates test results as shown in Figure.

The Results column indicates whether the test has "Passed" or "Failed". The test results also give useful information such as the name of the test case, the line numbers in the test script and the time taken for executing the test case.


1 Displays the name of the current test.

2 Shows the current results directory name.

3 Shows whether a test run passed or failed.

4 Includes general information about the test run such as date, operator name, and total run time. To view these details, double click the Information icon.

5 The test log section lists the major events that occurred during the test run. It also lists the test script line at which each event occurred.

①
②
③
WinRunner Test Results - [D:\My Documents\Winrunner\tutorial\global mode\lesson3]
File Options Tools Window
res3
lesson3
Ready

Test Result:	OK
Total number of bitmap checkpoints:	0
Total number of GUI checkpoints:	0
General Information	
Date:	Wednesday, November 29, 2000 03:37:05 P
Operator name:	
Expected Results Folder:	exp
Total Run Time:	00:00:08

Line	Event	Details	Result	Time
3	start run	lesson3	run	00:00:00
44	stop run	lesson3	pass	00:00:08

Calling the Test Cases using "call" Function

The "call" function can *is* used to execute a series of test cases without any user interaction. The syntax of call function is:

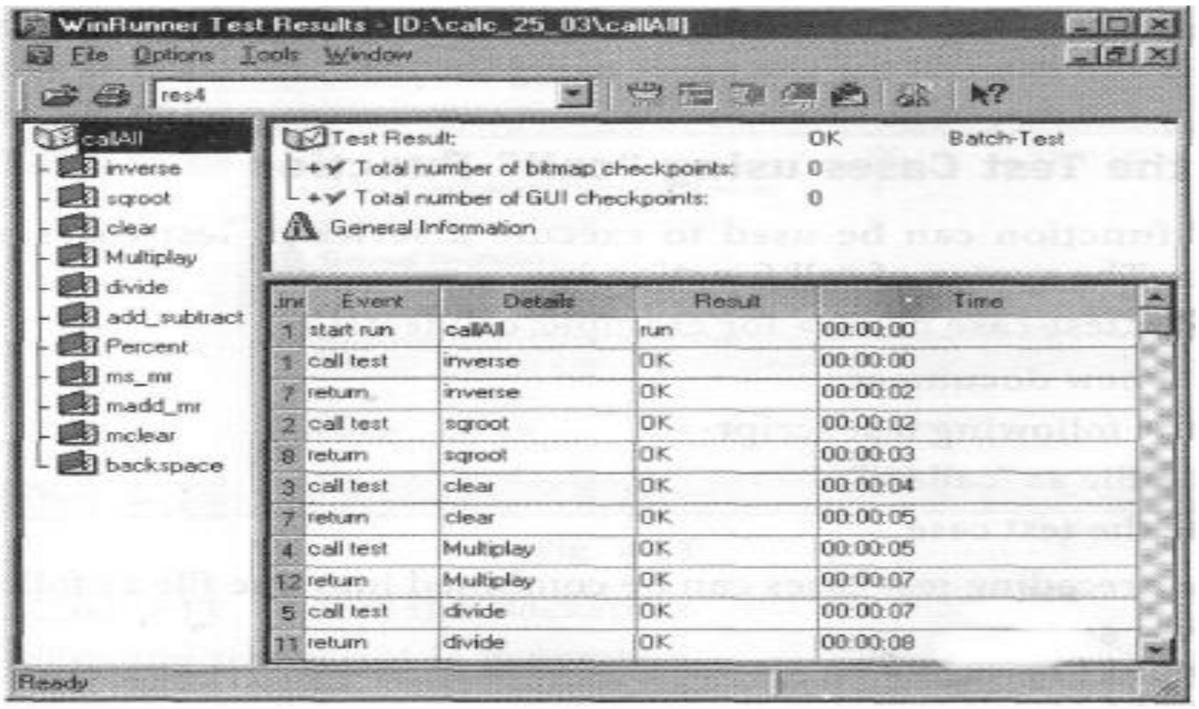
call<test-case name> for example, call testl();

- Create a new document
- Write the following test script
- Save the file as "callAll"
- Execute the test case

All the preceding test cases can be combined into one file as follows:

Call inverseO; call sqrootO; call clearO; call MultiplayO; call divideO; call add_subtract() call PercentO; call msjnrO; call maddmrO; call mclearO; call backspaceO;

When you execute this test script, all the earlier test cases are executed in one shot. The test results screen will be as shown in Figure. As you can see from the table, the "Details" column gives the various test cases executed. The "Result" column shows whether the test has passed or failed. The "Time" column gives the time taken to execute the test case.



Test Result: OK Batch-Test
Total number of bitmap checkpoints: 0
Total number of GUI checkpoints: 0
General Information

Line	Event	Details	Result	Time
1	start run	callAll	run	00:00:00
1	call test	inverse	OK	00:00:00
7	return	inverse	OK	00:00:02
2	call test	sqroot	OK	00:00:02
8	return	sqroot	OK	00:00:03
3	call test	clear	OK	00:00:04
7	return	clear	OK	00:00:05
4	call test	Multiplay	OK	00:00:05
12	return	Multiplay	OK	00:00:07
5	call test	divide	OK	00:00:07
11	return	divide	OK	00:00:08

When you have to retest the application using the same test cases, you can run the script in unattended mode. You can save the script in a file and run the script at specified time.

This feature of WinRunner is extremely useful for regression testing. When you are developing the software, you need to run the same set of test cases many times.

Test Script Language(TSL)

- Test Script Language (TSL) is a scripting language with syntax similar to C language.
- TSL is a C-like procedural programming language. It has constructs like statements, comments, constants, variables, mathematical operators, control statements and functions.
- There are 4 categories of TSL functions.

1. **Analog functions:** These functions are used when you record in Analog mode, a mode in which the exact coordinates of the GUI map are required.
2. **Context sensitive functions:** These functions are used where the exact coordinates are not required.
3. **Customization functions:** These functions allow the user to improve the testing tool by adding functions to the Function Generator.
4. **Standard functions:** These functions include all the basic elements of programming language like control flow statements, mathematical functions, string related functions etc.

The most frequently used functions in each category are as follows:

- **click:** To click a mouse button.
  Syntax: click(mouse_button,[time])
  mouse button - specify LEFT, RIGHT or MIDDLE
- **dbl_click:** To double click a mouse button.
  Syntax: dbl_click(mouse_button,[time])
  mouse_button - specify LEFT, RIGHT or MIDDLE
- **get_x:** To get the current x coordinate of the mouse pointer.
  Syntax: get_x();
  Return value: integer
- **get_y:** To get the current y coordinate of the mouse pointer.
  Syntax: get_y();
  Return value: integer
- **waitjwindow:** Waits for a window bitmap to appear for synchronizing the test procedure.
  Syntax: wait_window(time, image, window, width, height, x, y [, rebel, relyl, relx2, rely2])
- **get_text:** To read the text from the screen specified by the location.
  Syntax: get_text(location);
  The location can be xl>yl, x2, y2 or x ,y or ()(no location).When no location is specified, it considers the point closest to the mouse pointer.
- **button_check_info:** To check the value of the button property.
  Syntax: button_check_info (Button, property, property_value);
  Button - Name of the button
  Property - The property to be checked
  Property_value - The property value

The other important features of WinRunner application are as follows:

- Synchronization of test cases
- Data driven testing

- Rapid testing GUI
- Checkpoints

Synchronization of Test Cases

In WinRunner, it takes by default one second to execute the next statement. But sometimes there may be a case where the WinRunner has to wait for a few seconds to accept the data from the user or wait till the current operation is completed, before executing the next statement.

The Synchronization is required in the following cases:

- When data has to be retrieved from the database.
- When a progress bar has to reach 100%.
- To wait till some message appears.

Though, by default, WinRunner takes one second to execute the next statement, it is possible to change the default time to any desired value.

Changing the value of the "Timeout for Checkpoints and CS Statement" option in the Run tab of the General option's dialog can do this.

Settings -> General Options.

Creating the Test Case

Creating a synchronized Test

step 1: Start WinRunner and open a new test.

step 2: Start the application which you are required to test

step 3: set to synchronization test

step 4: Start recording

step 5: Create a new order

Synchronizing the Test Cases

We will now create a synchronization point into the test script..

Step 1: Open the test script if it is closed.

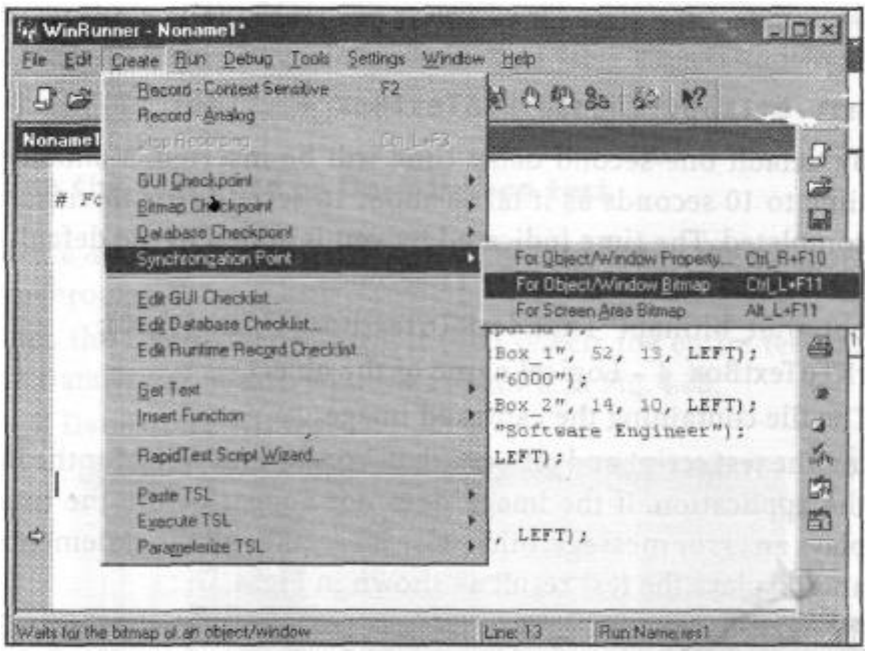
File -> Open and select the "sync_be-fore" file.

Step 2: Place the cursor at the point where the test has to be synchronized.Insert a line after the statement

obj_mouse_click ("Add", 49,12, LEFT);
as our aim is to make the WinRunner application wait till the insertion is over.

Step 3: Insert a synchronization point as shown in Figure to make WinRunner wait until the insertion is completed. Create -> Synchronization Point -> For Object/Window Bitmap

Once you select the status message, it inserts the following statement into the test script: obj_wait_bitmap("ThunderRT6TextBox_4"."Irngl",1);

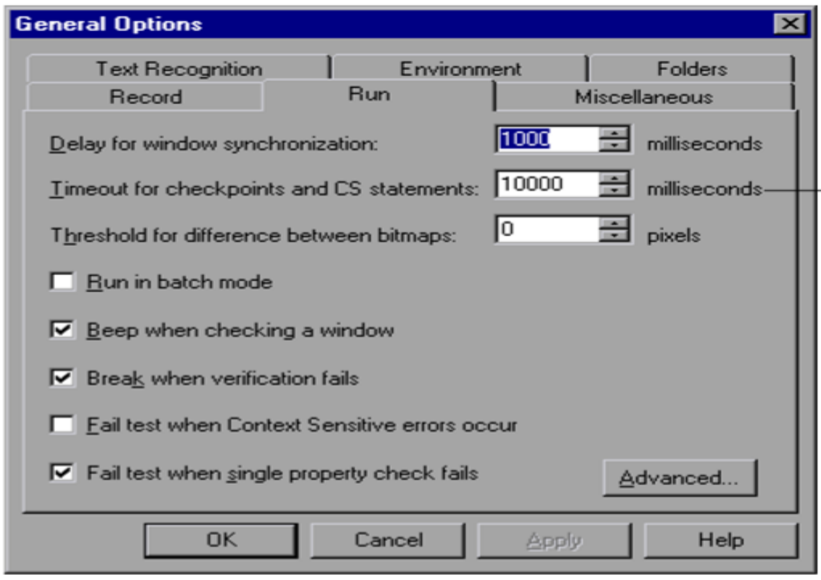

WinRunner - Noname1*
File Edit Create Run Debug Tools Settings Window Help
Record - Context Sensitive F2
Record - Analog
Stop Recording Ctrl_L+F3
GUI Checkpoint
Bitmap Checkpoint
Database Checkpoint
Synchronization Point
Edit GUI Checklist...
Edit Database Checklist...
Edit Runtime Record Checklist
Get Text
Insert Function
RapidTest Script Wizard...
Paste TSL
Execute TSL
Parameterize TSL
For Object/Window Property... Ctrl_R+F10
For Object/Window Bitmap Ctrl_L+F11
For Screen Area Bitmap Alt_L+F11
Noname1

```
Box_1", 52, 13, LEFT);
"6000");
Box_2", 14, 10, LEFT);
"Software Engineer");
LEFT);
, LEFT);
```

Waits for the bitmap of an object/window   Line: 13   Run Name: res1

Step 5: By default one-second delay time will be inserted. Manually change the wait time to 10 seconds as it takes about 10 seconds for the insertion action to be completed.

Click the run tab to change the time



General Options

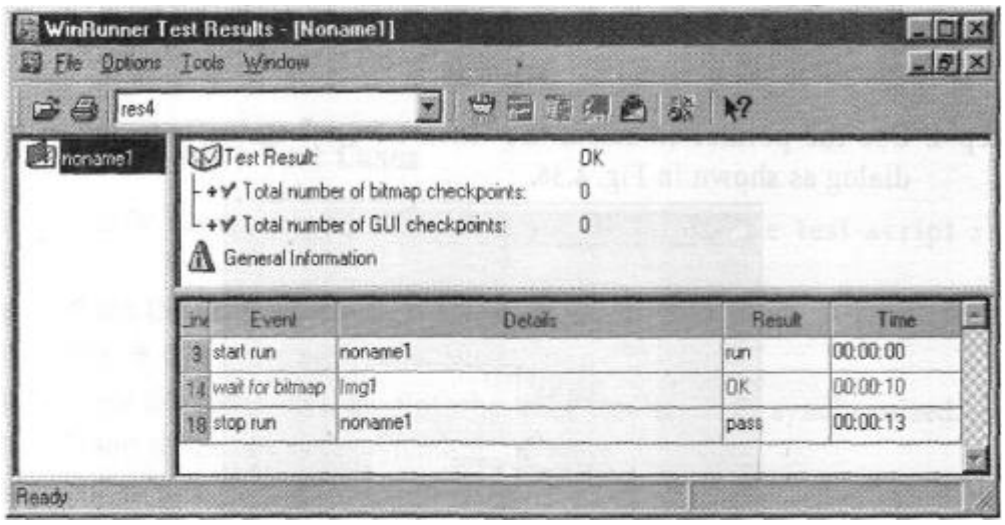
Text Recognition | Environment | Folders
Record | Run | Miscellaneous

Delay for window synchronization: 1000 milliseconds
Timeout for checkpoints and CS statements: 10000 milliseconds
Threshold for difference between bitmaps: 0 pixels

☐ Run in batch mode
☑ Beep when checking a window
☑ Break when verification fails
☐ Fail test when Context Sensitive errors occur
☑ Fail test when single property check fails

Change the value to 1000.

Step 6: Run the test script and observe that, WinRunner waits for the image to appear in the application. If the image does not appear before the timeout time, it displays an error message; otherwise it executes all the statements in the test script and displays the test result as shown in Figure



Test Result: OK
Total number of bitmap checkpoints: 0
Total number of GUI checkpoints: 0
General Information

Line	Event	Details	Result	Time
3	start run	noname1	run	00:00:00
14	wait for bitmap	Img1	OK	00:00:10
18	stop run	noname1	pass	00:00:13

Data Driven Test

Once the test script is created, we may sometimes want to check how the test script behaves for multiple data. This can be done by creating that many number of test cases and by running each test case individually, which is a very tedious process. In such cases we make use of Data-Driver wizard. This involves 3 steps.

1. Inserting the statements to open and close the data table.
2. Retrieving the data from the data table
3. Replacing the static values (for example, the employee name) with the variables containing the retrieved value from the data table. This is known as parameterizing the test.

Rapid Test Script Wizard

The Rapid test script wizard is the fastest way of performing the test process. It systematically opens up all the windows in the application, stores the learnt information in the GUI Map file and generates the test cases based on the information learnt from the application.

It is possible to apply these tests only on those applications, which open windows upon performing some task (like clicking a button, selecting the menu item etc

GUI Checkpoint used to Verify Properties Of objects which has 3 options such as

- For Single Property (Verify a single property of an object)
- For Object/Window (Verify more than 1 property of a Single Object)
- For Multiple Objects (Verify multiple objects with multiple properties)

Checking GUI Objects

- GUI checkpoints are used to check the GUI object properties
- It is possible to check the behavior of the objects in the application by creating the GUI Checkpoints.
- The GUI Checkpoints help to find the defects in the application, by examining the objects.

There are 3 modes in which the application can be executed

1. Verify mode: By default it is in verify mode. This mode is used to run the test script. It compares with the expected and actual values

2. Debug mode:This mode is used to check the values of the variables during runtime to monitor the variables. The results will be stored in debug folder

3. Update mode: This mode is used to change the expected value that you assigned during the checkpoint

5.2 LoadRunner

- Mercury Interactive's load runner is used to test the client/server applications such as database management systems and websites.
- Load runner accurately measures and analyzes the performance of the client/server application.
- Load runner creates Virtual users(Vusers). The Vusers submit the requests to the server. Vuser script is generated and this script is executed for simulating multiple users

What are the LoadRunner components?

LoadRunner contains the following components:

➢ The Virtual User Generator captures end-user business processes and creates an automated performance testing script, also known as a virtual user script.

➢ The Controller organizes, drives, manages, and monitors the load test.

➢ The Load Generators create the load by running virtual users.

➢ The Analysis helps you view, dissect, and compare the performance results.

➢ The Launcher provides a single point of access for all of the LoadRunner components.

LoadRunner addresses the drawbacks of manual performance testing:

- LoadRunner reduces personnel requirements by replacing human users with virtual users or ***Vusers***. These Vusers emulate the behavior of real users operating real applications.
- The LoadRunner Controller allows you to easily and effectively control all the Vusers from a single point of control.
- LoadRunner monitors the application performance online, enabling you to fine-tune your system during test execution.

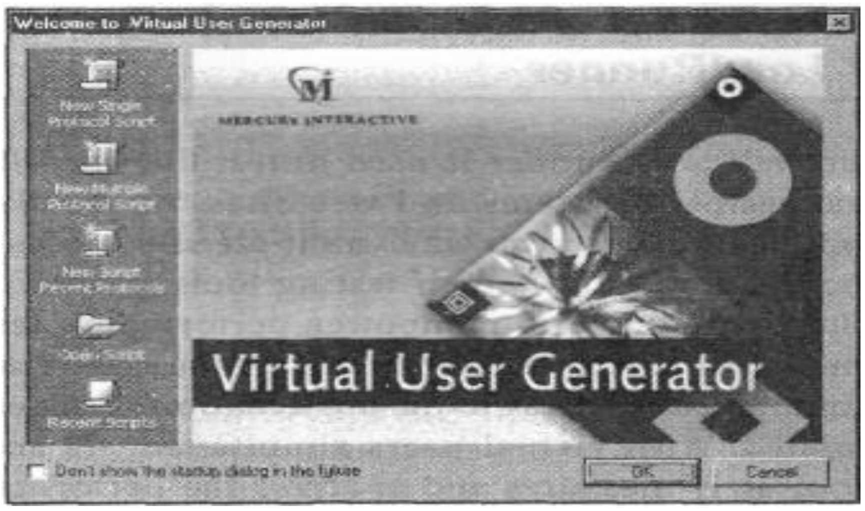
- LoadRunner automatically records the performance of the application during a test. You can choose from a wide variety of graphs and reports to view the performance data.
- LoadRunner checks where performance delays occur: network or client delays, CPU performance, I/O delays, database locking, or other issues at the database server. LoadRunner monitors the network and server resources to help you improve performance.
- Because LoadRunner tests are fully automated, you can easily repeat them as often as you need.

Creating Vuser script using Virtual User Generator

For creating the Vuser script, follow the steps given below:

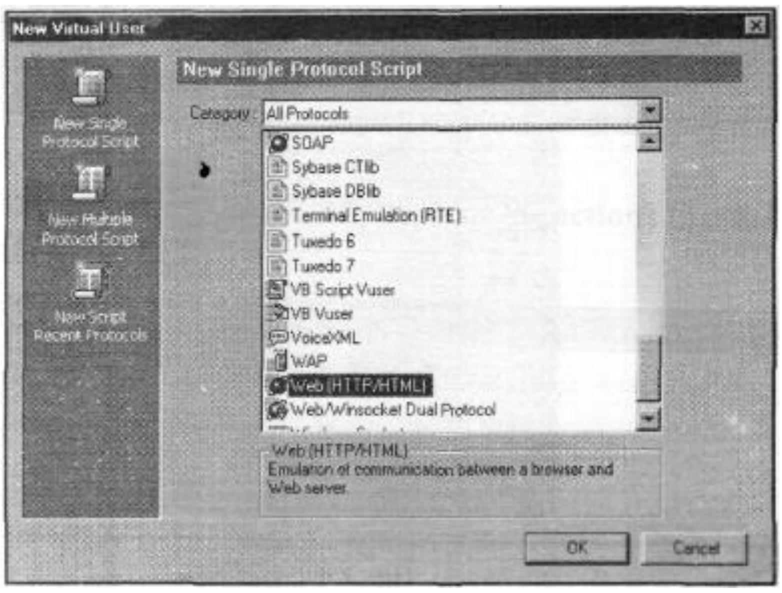
Step 1: Start the Virtual User Generator
Start -> Programs ->LoadRunner -> Virtual User Generator

Step 2: LoadRunner displays the welcome screen as shown in Figure.


Welcome to Virtual User Generator
New Single Protocol Script
New Multiple Protocol Script
New Script Recent Protocols
Open Script
Recent Scripts
MERCURY INTERACTIVE
Virtual User Generator
Don't show the startup dialog in the future
OK
Cancel

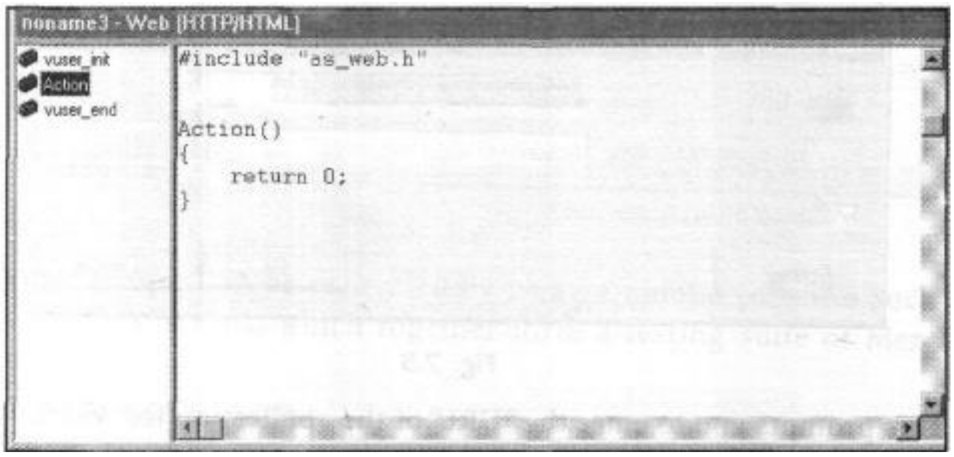
Step 3: Click the "New Single Protocol Script" button. It displays the list of protocols as shown in Figure.

Select the "Category" under which the application to be tested falls. In case of our example, select the "Web (HTTP/HTML)" option and click "OK" button.



Web (HTTP/HTML)
Emulation of communication between a browser and Web server

Step 3: When you click the "OK" button, it displays the dialog as shown in Figure.



```
#include "as_web.h"


Action()
{
    return 0;
}
```

The Virtual user script is divided into 3 sections: Vuser_init, Actions, Vuser_end.

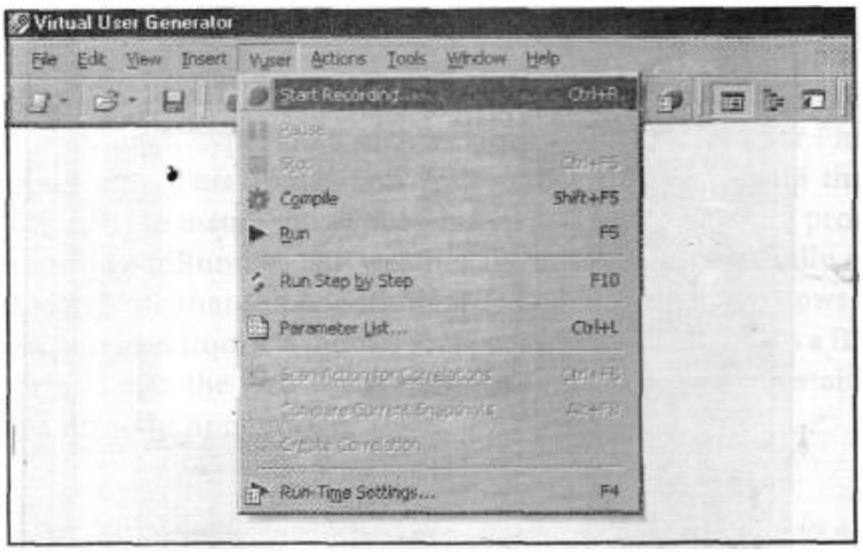
Vuser_init: These actions are performed when the Vuser is loaded or initialized.

Actions: These actions are performed when the Vuser is in "Running" state.

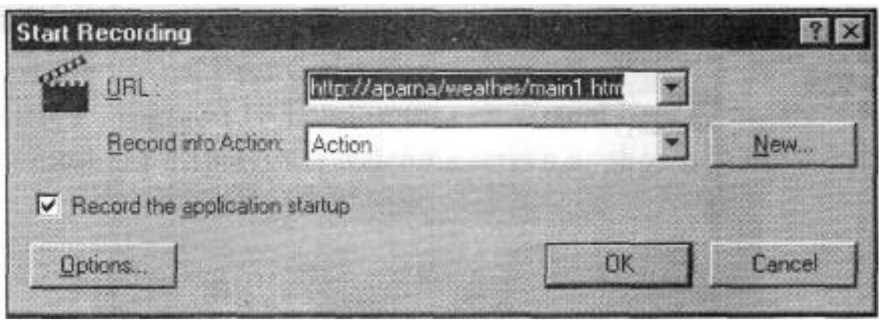
Vuser_end: These actions are performed when the Vuser finishes or stops.

You can select the section before or while the recording is under progress.

Step 4: Select the "Actions" section. Click the »»«<*,« button or select the "Start Recording" option from the "Vuser" menu as shown in Figure.


Virtual User Generator
File Edit View Insert Vuser Actions Tools Window Help
Start Recording... Ctrl+R
Pause
Stop Ctrl+F5
Compile Shift+F5
Run F5
Run Step by Step F10
Parameter List... Ctrl+L
Scan Action for Correlations Ctrl+F8
Compare Current Snapshots Alt+F8
Create Correlation
Run-Time Settings... F4

Step 5: On selecting "Start Recording" option, it displays the "Start Recording" dialog as shown in Figure.



URL: http://aparna/weather/main1.htm

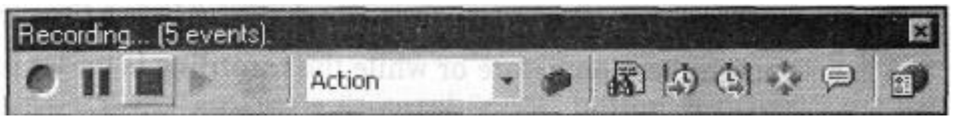
Record into Action: Action

☑ Record the application startup

This prompts you to enter the "URL" of the application for which the test has to be performed. Enter the URL and select the "Action" section from the "Record into Action" combo box as shown in Figure and click the "OK" button.

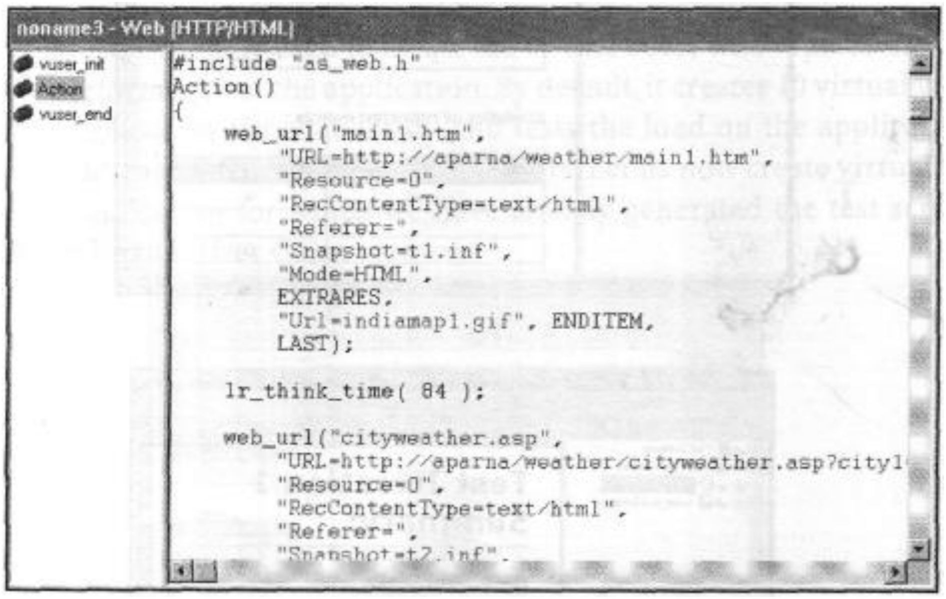
Note that you need to give the URL based on where you installed the application.

Step 6: LoadRunner now opens the specified URL and a "Recording Toolbar..." appears as shown in Figure.



LoadRunner is now in the "Recording" mode. Perform the actions that are to be recorded such as clicking on a link to obtain the weather information for a particular city. Once all the actions are recorded, stop the recording by clicking the "Stop" button in the Recording toolbajr.

Step 7: LoadRunner generates the script for the actions that are recorded as shown in Figure.



```
#include "as_web.h"
Action()
{
    web_url("main1.htm",
        "URL=http://aparna/weather/main1.htm",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t1.inf",
        "Mode=HTML",
        EXTRARES,
        "Url=indiamap1.gif", ENDITEM,
        LAST);

    lr_think_time( 84 );

    web_url("cityweather.asp",
        "URL=http://aparna/weather/cityweather.asp?city1
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t2.inf",
```

Vuser Scripts can be generated by using a number of tools such as WinRunner, VuGen, QuickTest etc. which together form a testing suite of Mercury Interactive Corporation.

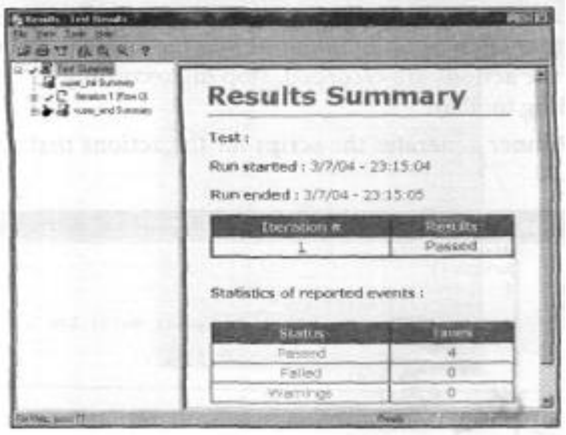
Step 8: Save the Test Script.
File -> Save

Step 9: Run the script.
Click the ► button to run the test script or click the "Run" from "Vuser" menu or press F5.
Vuser -> Run

Step 10: Once the execution is completed, examine the "Execution log" to see whether the script ran without errors or not.

Step 11: The test results are displayed in the "Test Results" window, which contains various sections:"Vuser_init_summary", "Iteration","Vuser_end_summary".

The Results Summary Report is shown in Figure, the Iteration Report is shown in Figure and Vuser_end Summary Report is shown in Figure



Results Summary

Test :

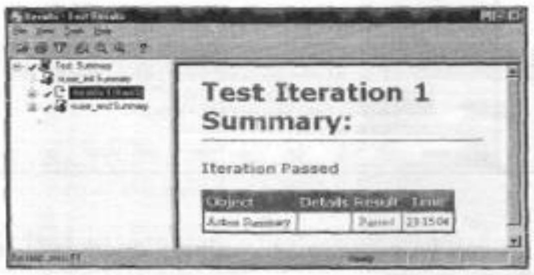
Run started : 3/7/04 - 23:15:04

Run ended : 3/7/04 - 23:15:05

Iteration #	Results
1	Passed

Statistics of reported events :

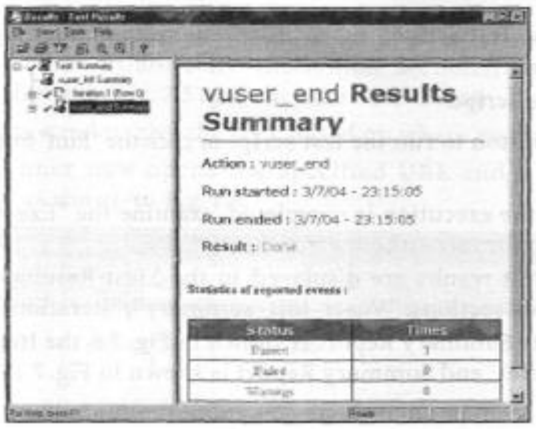
Status	Times
Passed	4
Failed	0
Warnings	0



Test Iteration 1 Summary:

Iteration Passed

Object	Details	Result	Time
Action Summary		Passed	23:15:04



vuser_end **Results Summary**

Action : vuser_end

Run started : 3/7/04 - 23:15:05

Run ended : 3/7/04 - 23:15:05

Result : Done

Statistics of reported events :

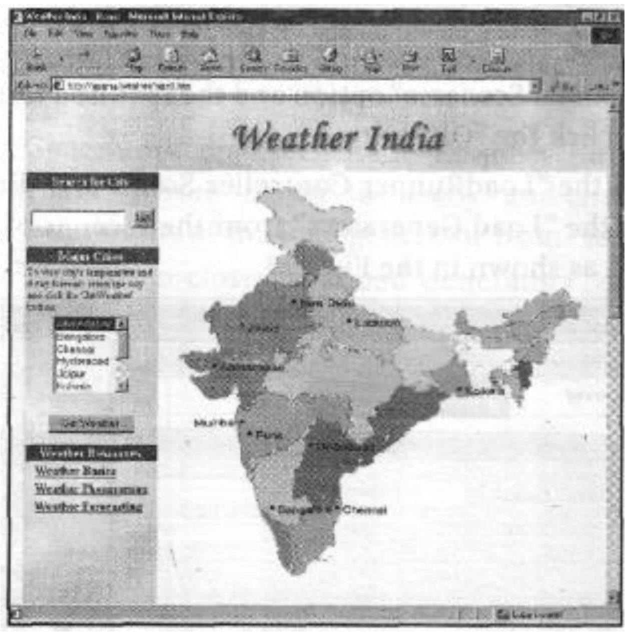
Status	Times
Passed	1
Failed	0
Warnings	0

Creating Virtual Users Using Loadrunner Controller

LoadRunner Controller is used to create the virtual users who replace the human users to test the performance of the application.

By default, it creates 10 virtual users who will access the application simultaneously and tests the load on the application. It is also possible to increase the number of virtual users.

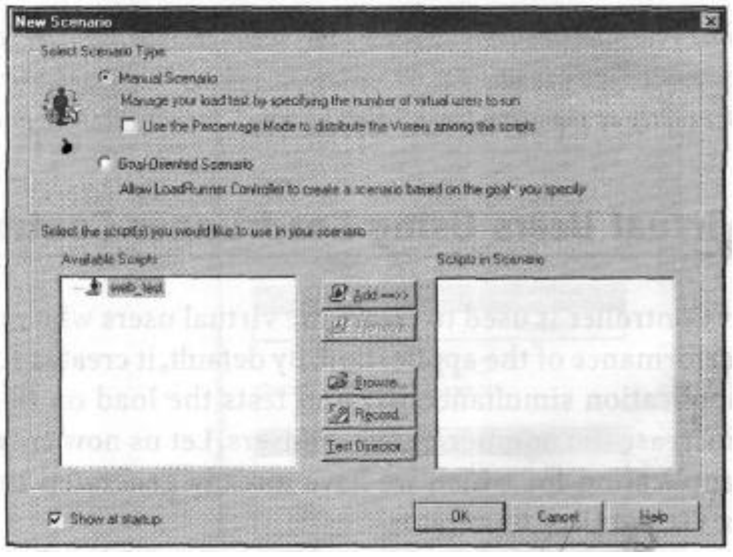
Let us now create virtual users for the web-based application for which we have already generated the test script using the LoadRunner Virtual User Generator.



To create the Virtual users and test the performance of the "Weather India" application whose home page is shown in Figure, follow the steps given below.

Step 1: Start the Load Runner - Controller
Start -> Programs ->LoadRunner -> Controller

Step 2: On starting the Controller, it displays the screen as shown in Figure.



New Scenario

Select Scenario Type

[✓ Manual Scenario]
Manage your load test by specifying the number of virtual users to run
☐ Use the Percentage Mode to distribute the Vusers among the scripts

Goal-Oriented Scenario
Allow LoadRunner Controller to create a scenario based on the goal you specify

Select the script(s) you would like to use in your scenario

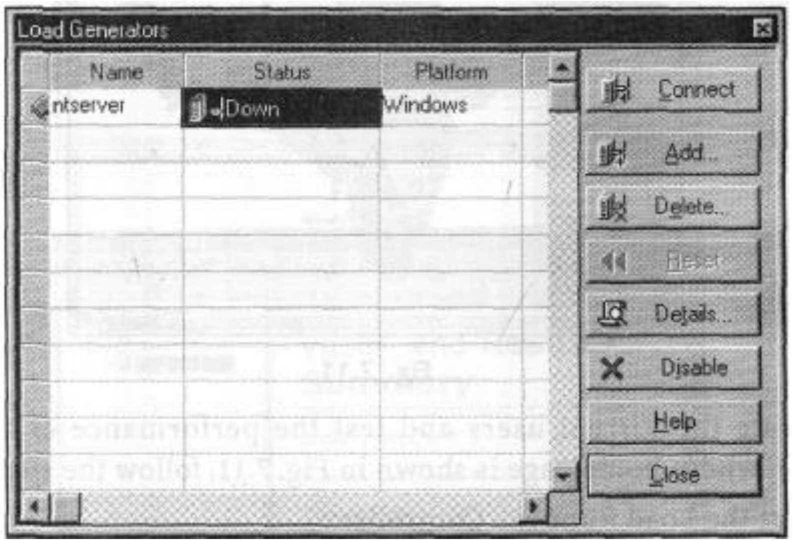
Available Scripts: web_test

Scripts in Scenario

☑ Show at startup

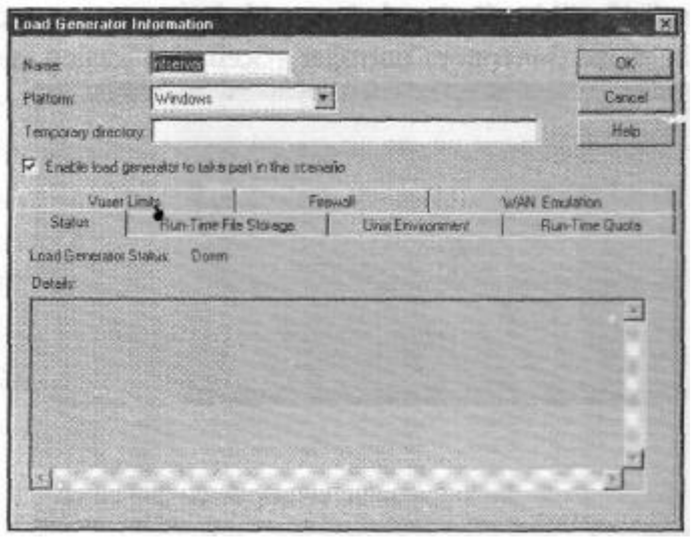
Select the "Manual Scenario" option and the required test script from the "Available Scripts" list and click the "OK" button.

Step 3: From the "LoadRunner Controller Scenario 1" dialog, click the "Generator" button or select the "Load Generators" from the "Scenario" menu. "Load Generators" dialog is opened as shown in the Figure.



Name	Status	Platform
ntserver	Down	Windows

Click the "Add" button to add a generator, or double-click the "Status" column o the default host details, which displays the "Load Generator Information" as shown in Figure.



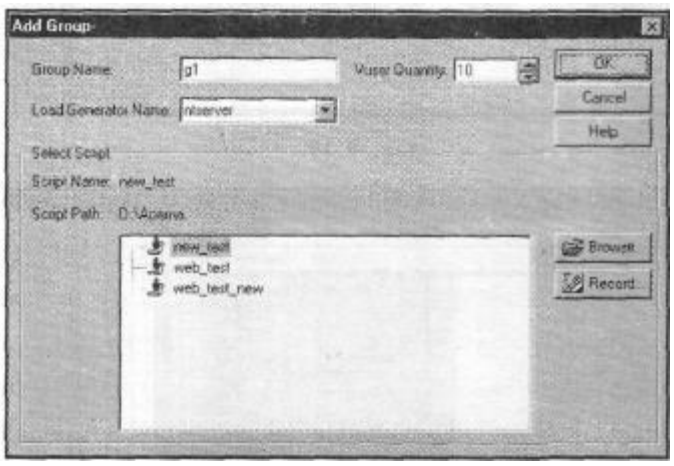
Name: ntserver
Platform: Windows
Temporary directory:
☑ Enable load generator to take part in the scenario
Load Generator Status: Down
Details:

Host Name refers to the identity of the system. Enter the name of the system in this field. Select the platform (in case of our example select "Windows") and click the "OK" button.

Step 4: In the "Load Generators" dialog, click the "Connect" button to change the status of the load generator from "Down" to "Ready" and click the "Disconnect" button to change the status of the load generator from "Ready" to "Down".

Step 5: Click "Close" button to close the "Load Generators" dialog.

Step 6: In the "Load Runner Controller - Scenario 1" dialog, click the "Add Group" to create the group for the virtual users. It then displays the "Add Dialog" as shown in Figure.



Group Name: g1

Vuser Quantity: 10

Load Generator Name: ntserver

Select Script

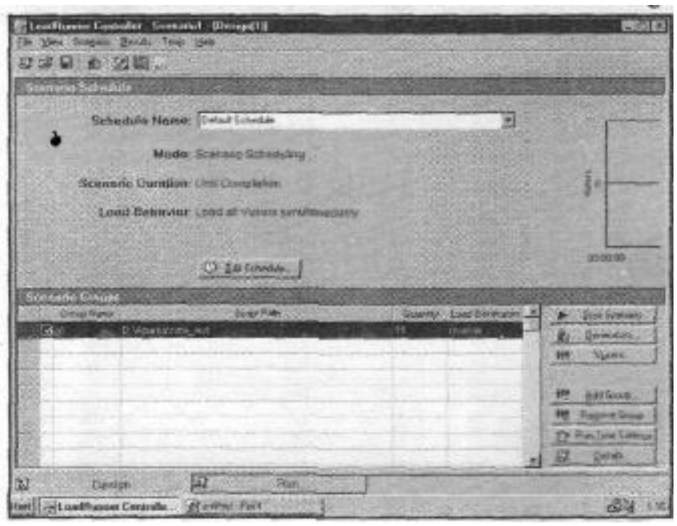
Script Name: new_test

Script Path: D:\Aparna

new_test
web_test
web_test_new

Enter the group name as "gl", select the host name from the "Load Generator Name" and select the test script from the "Select Script" list box that has to be tested. Enter the number of virtual users to be created in the "Vuser Quantity" and click the "OK" button.

Step 7: The Group "gl" will be displayed along with its host name and the number of virtual users in the "Load Generator Controller - Scenariol" dialog as shown in Figure.



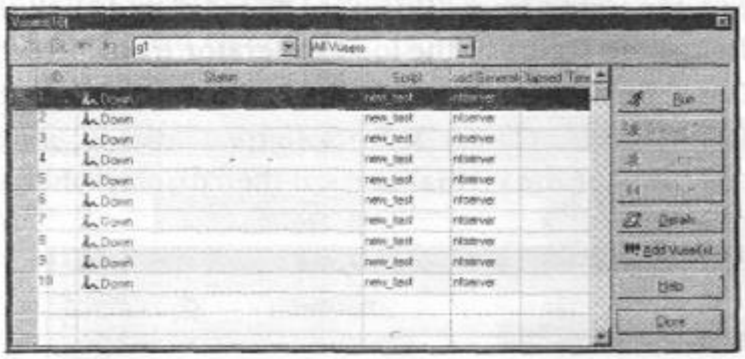
Schedule Name: Default Schedule

Mode: Scenario Scheduling

Scenario Duration: Until Completion

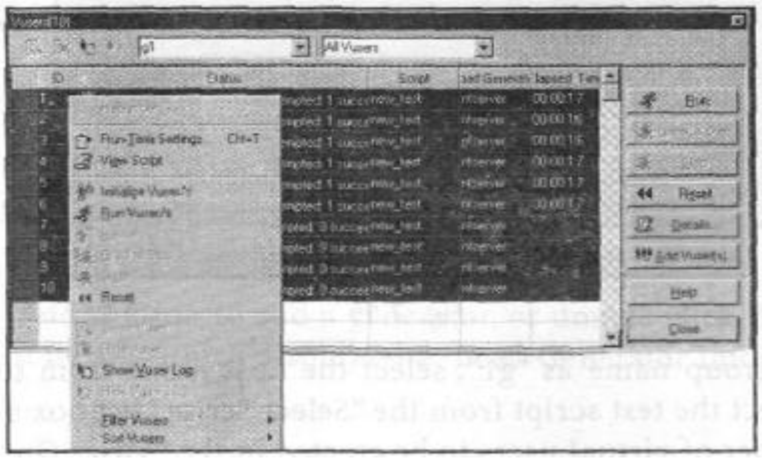
Load Behavior: Load all Vusers simultaneously

Step 8: Click "Vuser" button to view the virtual users. Initially all the users are in "Down" State as shown in Figure. Select all the users and right click to select the "Initialize Vuser/s" option as shown in Figure.



Vusers(10)

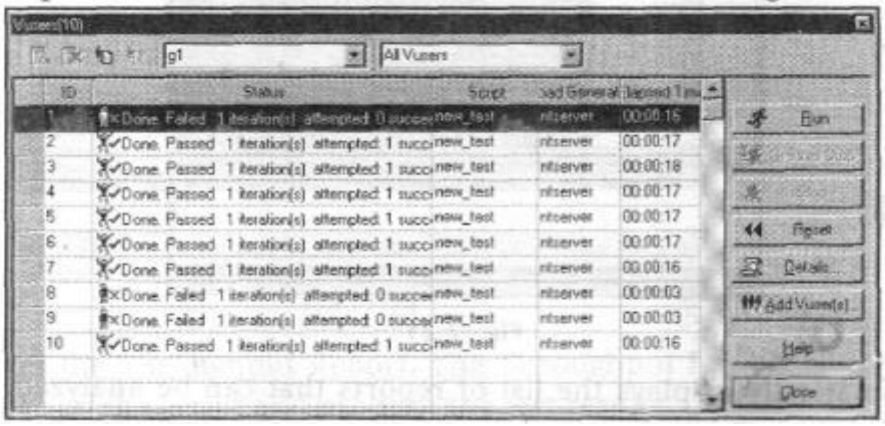
ID	Status	Script	oad Generat	lapsed Tim
1	Down	new_test	ntserver	
2	Down	new_test	ntserver	
3	Down	new_test	ntserver	
4	Down	new_test	ntserver	
5	Down	new_test	ntserver	
6	Down	new_test	ntserver	
7	Down	new_test	ntserver	
8	Down	new_test	ntserver	
9	Down	new_test	ntserver	
10	Down	new_test	ntserver	



Vusers(10)

All the Vusers will change their state from "Down" - "Initializing" - "Ready" mode. Now to run the Vusers, again select all the Vusers, right click and select "Run Vuser/s" or click the "Run" button. The status of the Vusers will change from Ready - Running - Done. Passed or Done. Failed.

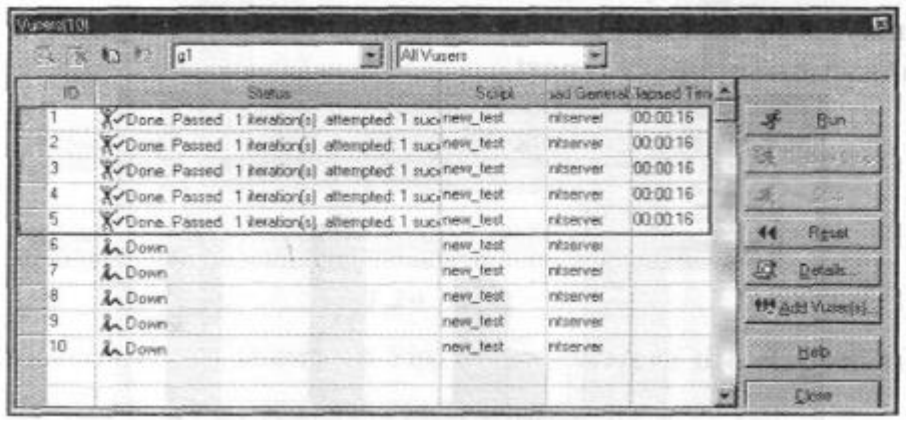
Step 9: Once all the Vusers complete the execution of the script, it displays which user has passed the test and which has not, as shown in Figure



Vusers(10)

ID	Status	Script	oad Generat	lapsed Tim
1	Done. Failed 1 iteration(s) attempted: 0 succe	new_test	ntserver	00:00:16
2	Done. Passed 1 iteration(s) attempted: 1 succ	new_test	ntserver	00:00:17
3	Done. Passed 1 iteration(s) attempted: 1 succ	new_test	ntserver	00:00:18
4	Done. Passed 1 iteration(s) attempted: 1 succ	new_test	ntserver	00:00:17
5	Done. Passed 1 iteration(s) attempted: 1 succ	new_test	ntserver	00:00:17
6	Done. Passed 1 iteration(s) attempted: 1 succ	new_test	ntserver	00:00:17
7	Done. Passed 1 iteration(s) attempted: 1 succ	new_test	ntserver	00:00:16
8	Done. Failed 1 iteration(s) attempted: 0 succe	new_test	ntserver	00:00:03
9	Done. Failed 1 iteration(s) attempted: 0 succe	new_test	ntserver	00:00:03
10	Done. Passed 1 iteration(s) attempted: 1 succ	new_test	ntserver	00:00:16

If only 5 Vusers are allowed to test the "Weather India" application simultaneously, then all the Vusers successfully complete the test as shown in Figure.This means that the "Server" that is processing the requests is not able to take the load of all the 10 users at a time.

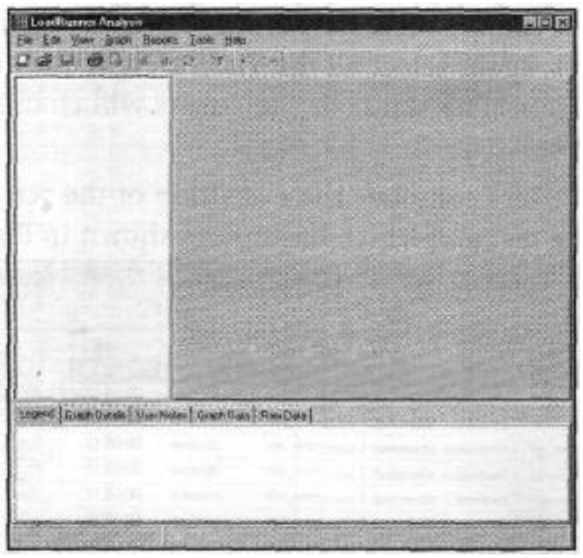


ID	Status	Script	ad General	lapsed Tim
1	Done. Passed 1 iteration(s) attempted: 1 suc	new_test	ntserver	00:00:16
2	Done. Passed 1 iteration(s) attempted: 1 suc	new_test	ntserver	00:00:16
3	Done. Passed 1 iteration(s) attempted: 1 suc	new_test	ntserver	00:00:16
4	Done. Passed 1 iteration(s) attempted: 1 suc	new_test	ntserver	00:00:16
5	Done. Passed 1 iteration(s) attempted: 1 suc	new_test	ntserver	00:00:16
6	Down	new_test	ntserver	
7	Down	new_test	ntserver	
8	Down	new_test	ntserver	
9	Down	new_test	ntserver	
10	Down	new_test	ntserver	

Step 10:Click the "Close" button to close the "Vusers" window.

Step 11: Analyze the test results.

Tools -> Analysis Load Runner internally opens the "Load Runner Analysis". It displays "LoadRunner Analysis" dialog as shown in Figure.

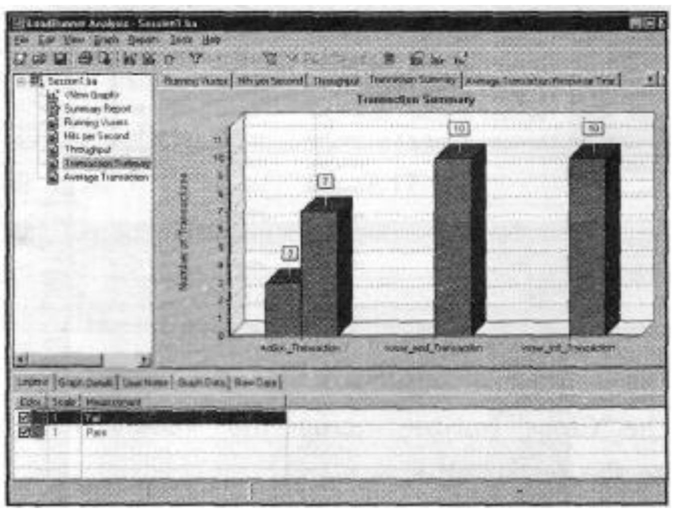


It automatically displays the list of reports that can be analyzed. You can obtain the following reports:

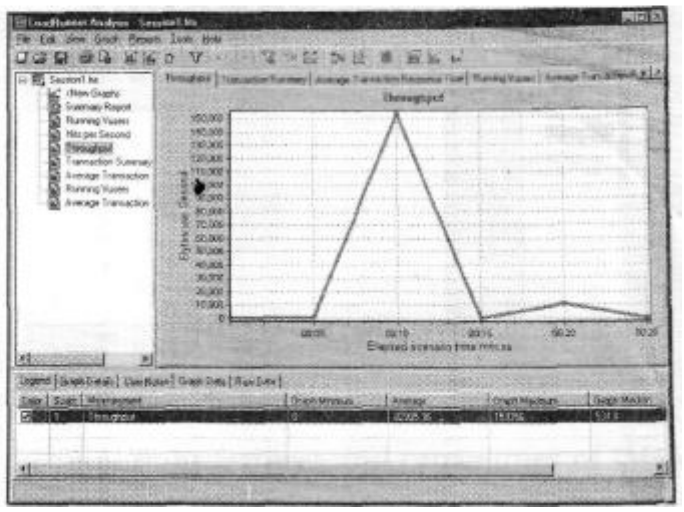
1. Running Vusers: Whole Scenario graph lets you monitor the number of Vusers that are running at a given time.
2. Hits per second: Whole Scenario graph lets you monitor the number of hits (HTTP requests) on the Web server made by Vusers during each second of the scenario run. This enables you to follow the amount of load that is generated on the server.
3. Transaction Response Time - Whole Scenario graph lets you monitor the amount of time it takes for each transaction to be completed. You can see how long it takes for your customers to log on, search flights, purchase flights, check itineraries, and log off the system
4. Windows Resources graph lets you monitor the Windows resource usage measured during a scenario (such as CPU, disk, or memory utilization).

5. Throughput

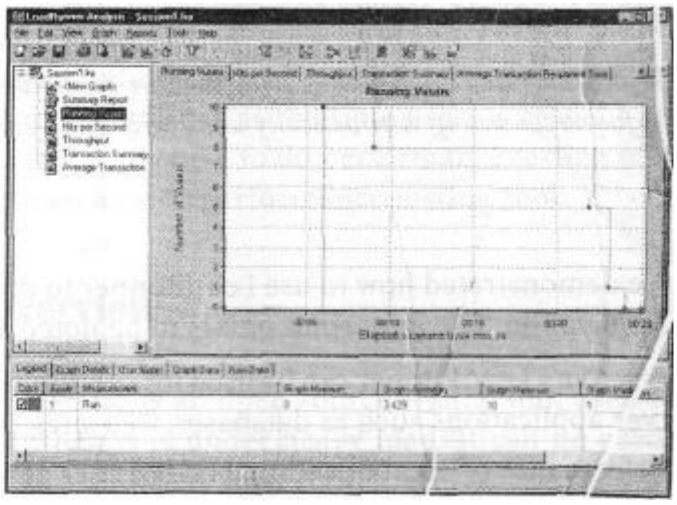
Step 12: Analyze the results by studying graphs shown in Figure which shows the transaction summary report.



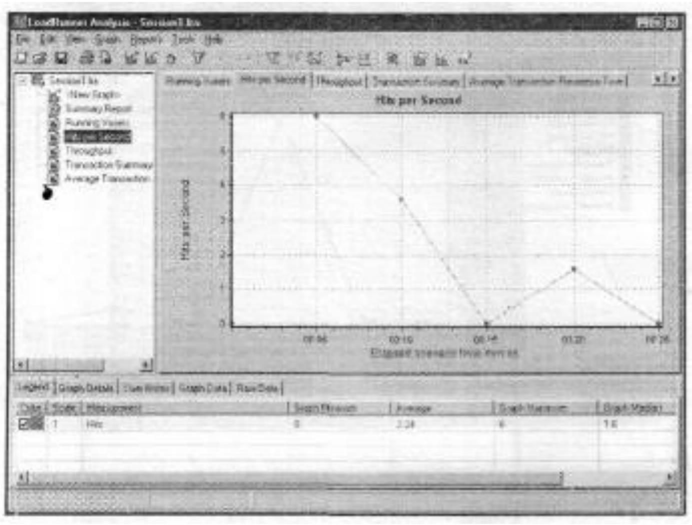
The Transaction Summary report specifies the number of Vusers that passed and failed the various sections of the script i.e. Action, Vuser_init,Vuser_end in the form of Bar graph.


LoadRunner Analysis - Session1.lra
Throughput
Bytes per Second
Elapsed scenario time mm:ss
Throughput

gives the Throughput Report. The Throughput Report shows the rate which the Vusers run the script and produce the test results.



"Running Vusers" report, shown in Figure, describes the elapsed time for each user in mnv.ss format shows the graph which indicates the number of requests made by user per second1



5.3 JMeter

JMeter is a software that can perform load test, performance-oriented business (functional) test, regression test, etc., on different protocols or technologies.

Stefano Mazzocchi of the Apache Software Foundation was the original developer of JMeter.

Apache later redesigned JMeter to enhance the GUI and to add functional testing capabilities.

JMeter is a Java desktop application with a graphical interface that uses the Swing graphical API. It can therefore run on any environment / workstation that accepts a Java virtual machine, for example: Windows, Linux, Mac, etc.

The protocols supported by JMeter are:

- Web: HTTP, HTTPS sites 'web 1.0' web 2.0 (ajax, flex and flex-ws-amf)
- Web Services: SOAP / XML-RPC
- Database via JDBC drivers
- FTP Service

JMeter Features

Following are some of the features of JMeter:

- Being an open source software, it is freely available.
- It has a simple and intuitive GUI.
- JMeter can conduct load and performance test for many different server types: Web HTTP, HTTPS, SOAP, Database via JDBC, LDAP, JMS, Mail - POP3, etc.
- It is a platform-independent tool. On Linux/Unix, JMeter can be invoked by clicking on JMeter shell script. On Windows, it can be invoked by starting the jmeter.bat file.
- It has full Swing and lightweight component support (precompiled JAR uses packages javax.swing.*).
- JMeter stores its test plans in XML format. It means you can generate a test plan using a text editor.
- Its full multi-threading framework allows concurrent sampling by many threads and simultaneous sampling of different functions by separate thread groups.
- It is highly extensible.
- It can also be used to perform automated and functional testing of the applications

How JMeter Works?

- JMeter simulates a group of users sending requests to a target server, and returns statistics that show the performance/functionality of the target server/application via tables, graphs.
- Take a look at the following figure that depicts how JMeter works:

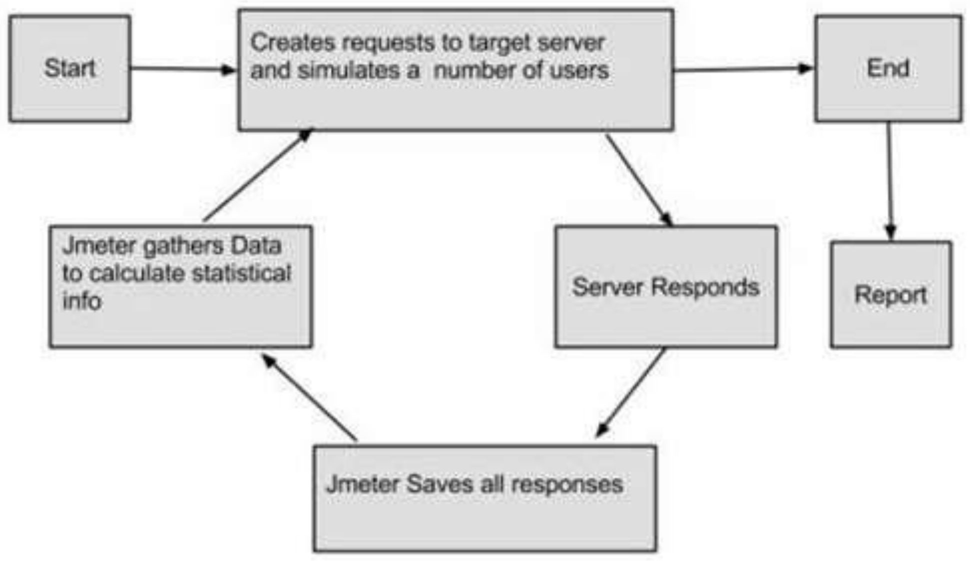

Start
Creates requests to target server and simulates a number of users
End
Jmeter gathers Data to calculate statistical info
Server Responds
Report
Jmeter Saves all responses

Fig: Working process of JMeter

JMeter test plan

Test Plan

- A Test Plan can be viewed as a container for running tests. It defines what to test and how to go about it.
- A complete test plan consists of one or more elements such as thread groups, logic controllers, sample-generating controllers, listeners, timers, assertions, and

- configuration elements. A test plan must have at least one thread group.

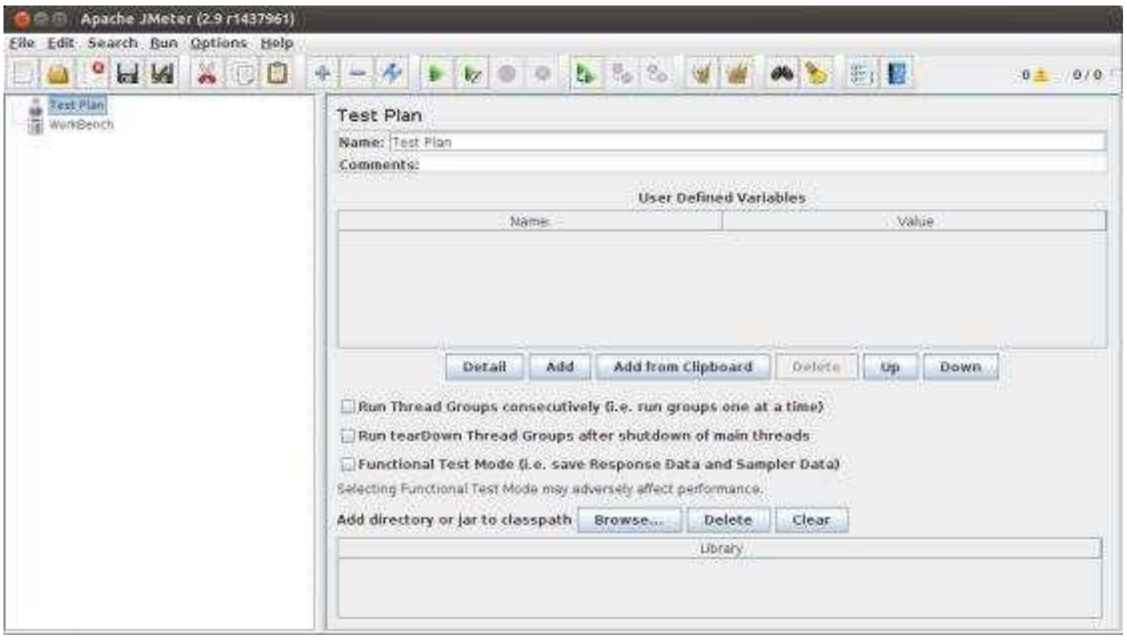
Writing a Test Plan

Follow the steps given below to write a test plan:

Step 1: Start the JMeter Window

Open the JMeter window by clicking **/home/manisha/apache-jmeter-2.9/bin/jmeter.sh**.

The JMeter window will appear as shown below:



Name: Test Plan
Comments:

☐ Run Thread Groups consecutively (i.e. run groups one at a time)
☐ Run tearDown Thread Groups after shutdown of main threads
☐ Functional Test Mode (i.e. save Response Data and Sampler Data)
Selecting Functional Test Mode may adversely affect performance.
Add directory or jar to classpath

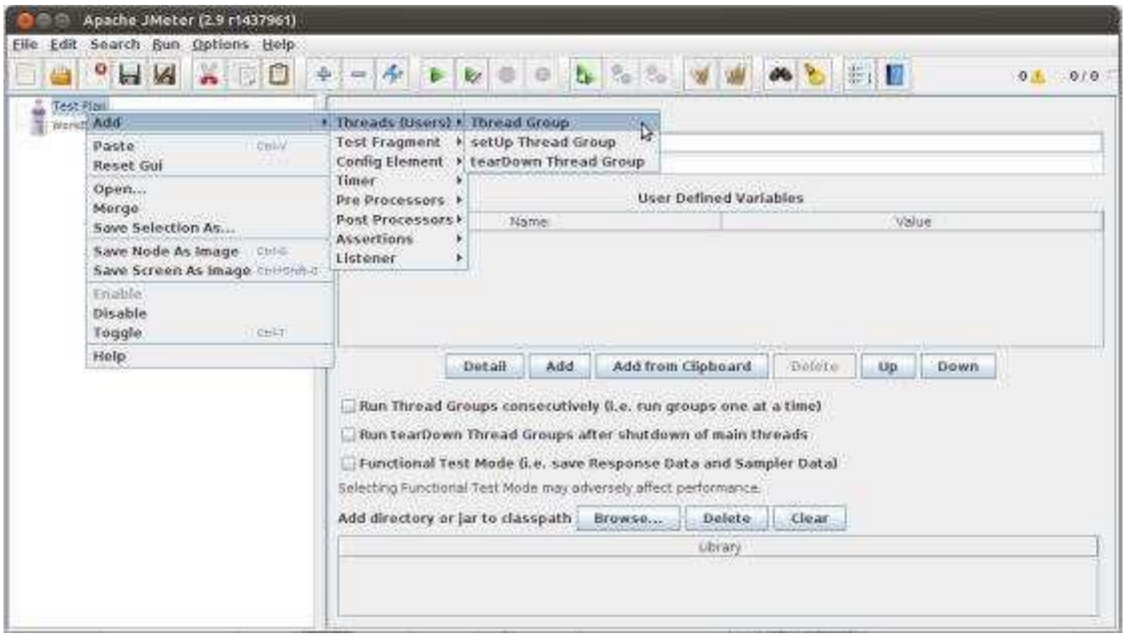
This is a plain and blank JMeter window without any additional elements added to it. It contains two nodes:

- **Test Plan node**: It is where the real test plan is kept.
- **Workbench node**: It simply provides a place to temporarily store test elements while not in use, for copy/paste purposes. When you save your test plan, Workbench items are not saved with it.

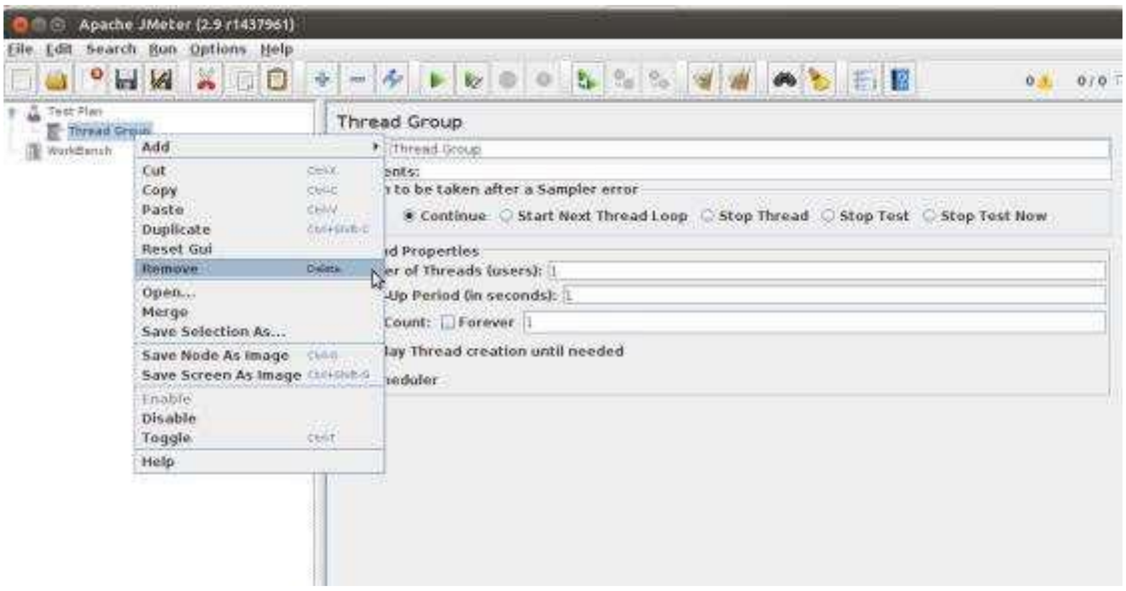
Step 2: Add/Remove Elements

- Elements can be added to a test plan by right-clicking on the Test Plan node and choosing a new element from the "add" list.
- Alternatively, you can load an element from a file and add it by choosing the "merge" or "open" option.

For example, let us add a Thread Group element to a Test Plan as shown below:



To remove an element, make sure the element is selected, right-click on the element, and choose the "remove" option.

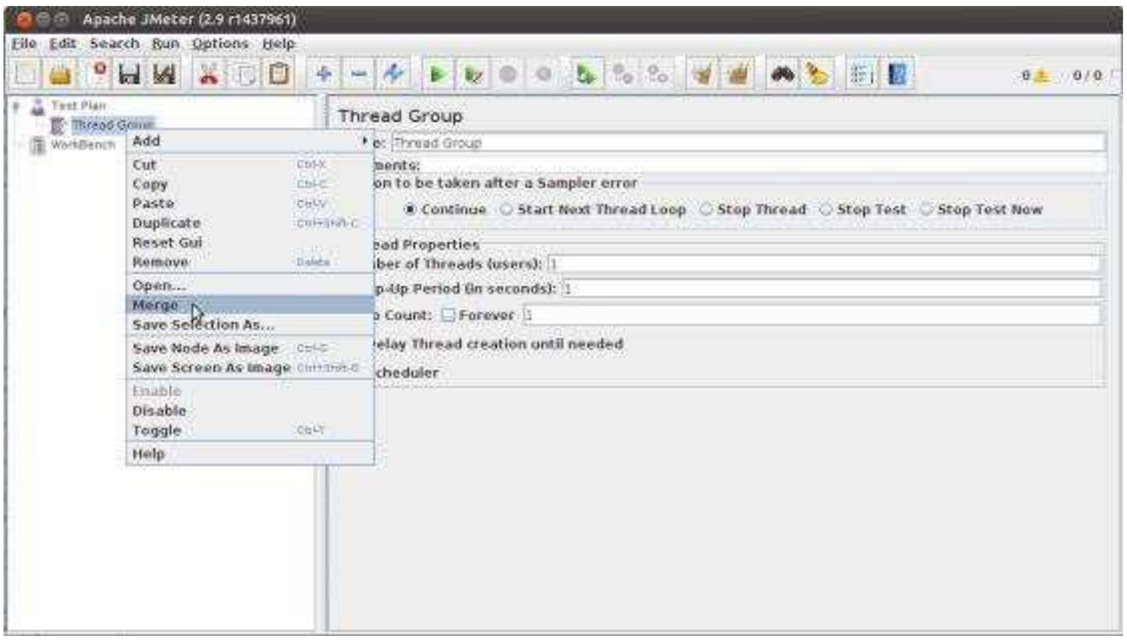


Step 3: Load and Save the Elements

To load an element from file:

- Right-click on the existing tree element to which you want to add the loaded element.
- Select Merge.
- Choose the file where you saved the elements.
- JMeter will merge the elements into the tree.

By default, JMeter does not save the element, you need to explicitly save it.

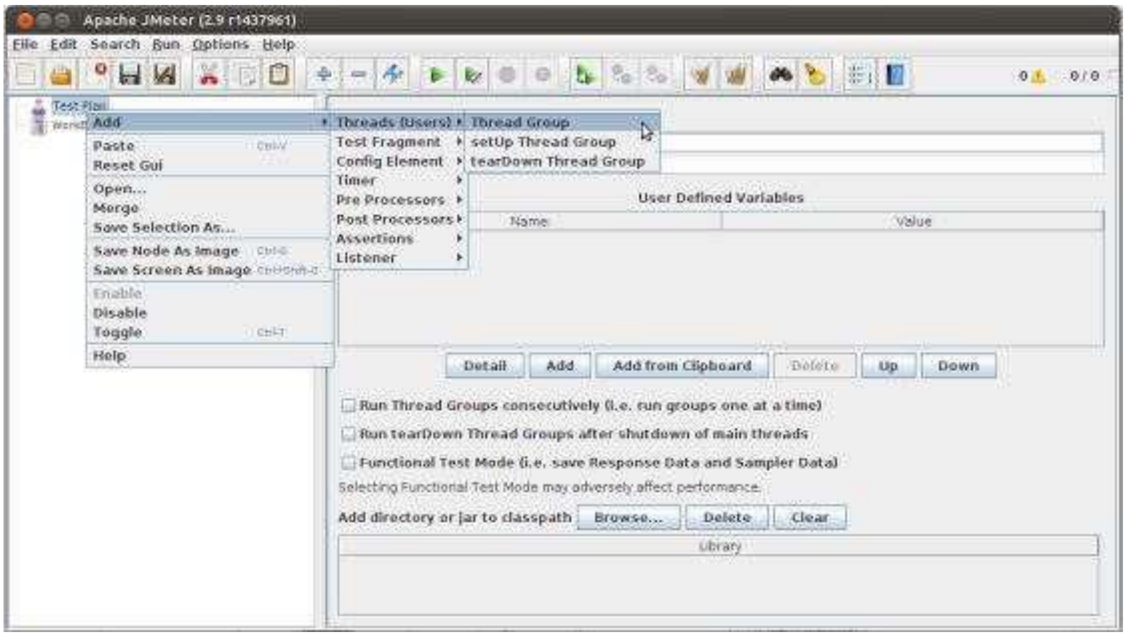


To save a tree element:

- Right-click on the element.
- Choose the *Save Selection As* ... option.
- JMeter will save the element selected, plus all the child elements beneath it. By default,
- JMeter doesn't save the elements, you need to explicitly save it as mentioned earlier.

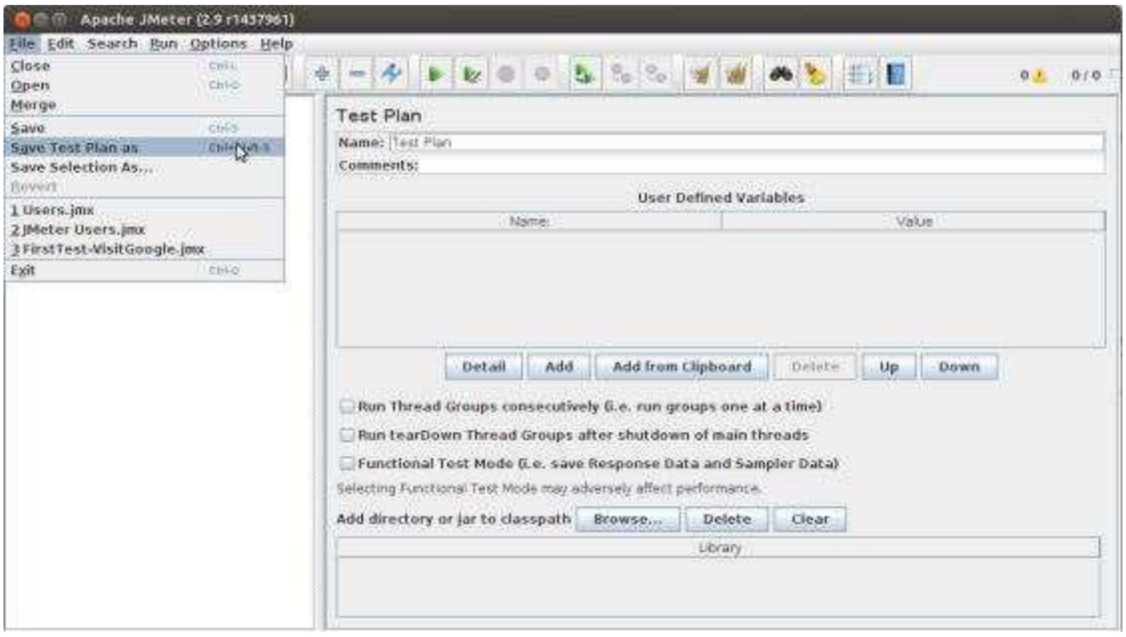
Step 4: Configure the Tree Elements

- Any element in the Test Plan can be configured using the controls present in JMeter's righthand side frame. These controls allow you to configure the behavior of that particular test element.
- For example, the Thread Group can be configured for a number of users, ramp up periods, etc., as shown below:


Apache JMeter (2.9 r1437961)
File Edit Search Run Options Help
Test Plan
Add
Paste Ctrl-V
Reset Gui
Open...
Merge
Save Selection As...
Save Node As Image Ctrl-G
Save Screen As Image Ctrl+Shift-G
Enable
Disable
Toggle Ctrl-T
Help
Threads (Users)
Test Fragment
Config Element
Timer
Pre Processors
Post Processors
Assertions
Listener
Thread Group
setUp Thread Group
tearDown Thread Group
User Defined Variables
Name
Value
Detail
Add
Add from Clipboard
Delete
Up
Down
Run Thread Groups consecutively (i.e. run groups one at a time)
Run tearDown Thread Groups after shutdown of main threads
Functional Test Mode (i.e. save Response Data and Sampler Data)
Selecting Functional Test Mode may adversely affect performance.
Add directory or jar to classpath
Browse...
Delete
Clear
Library

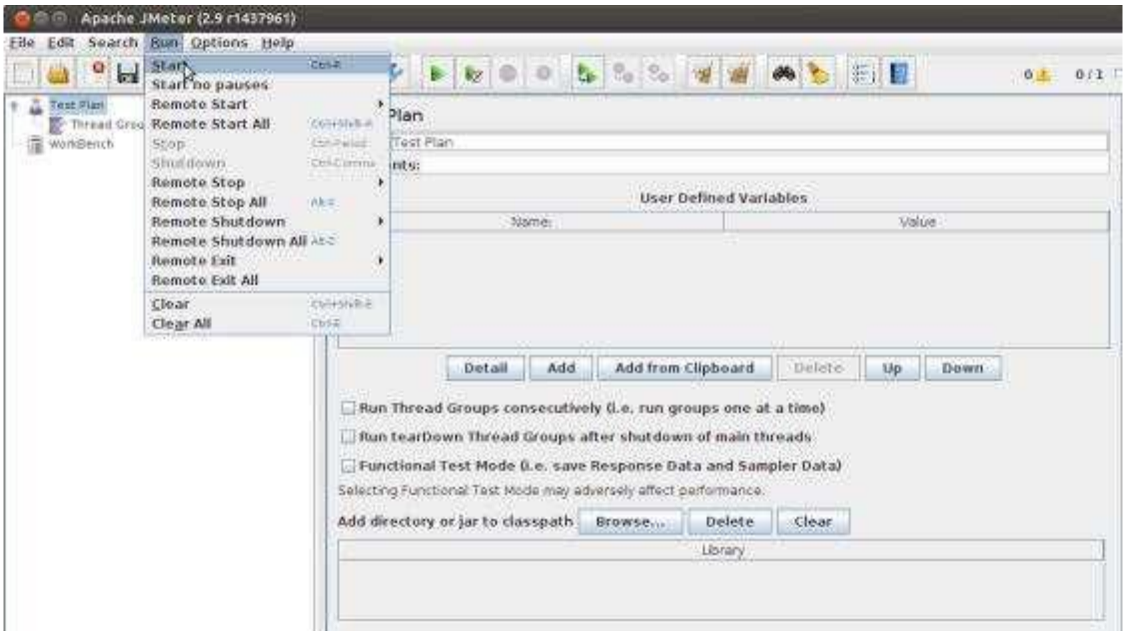
Step 5: Save the Test Plan

You can save an entire Test Plan by using either **Save** or **"Save Test Plan As ..."** from the File menu.


Apache JMeter (2.9 r1437961)
File Edit Search Run Options Help
Close
Open
Merge
Save
Save Test Plan as
Save Selection As...
Revert
1 Users.jmx
2 JMeter Users.jmx
3 FirstTest-VisitGoogle.jmx
Exit
Test Plan
Name: Test Plan
Comments:
User Defined Variables
Name:
Value
Detail
Add
Add from Clipboard
Delete
Up
Down
Run Thread Groups consecutively (i.e. run groups one at a time)
Run tearDown Thread Groups after shutdown of main threads
Functional Test Mode (i.e. save Response Data and Sampler Data)
Selecting Functional Test Mode may adversely affect performance.
Add directory or jar to classpath
Browse...
Delete
Clear
Library

Step 6: Run the Test Plan

You can run the Test Plan by clicking **Start** (Control + r) from the **Run** menu item. When JMeter starts running, it shows a small green box at the right-hand end of the section just under the menubar.


Apache JMeter (2.9 r1437961)
File Edit Search Run Options Help
Start
Start no pauses
Remote Start
Remote Start All
Stop
Shutdown
Remote Stop
Remote Stop All
Remote Shutdown
Remote Shutdown All
Remote Exit
Remote Exit All
Clear
Clear All
Test Plan
Thread Group
WorkBench
User Defined Variables
Name:
Value
Detail
Add
Add from Clipboard
Delete
Up
Down
Run Thread Groups consecutively (i.e. run groups one at a time)
Run tearDown Thread Groups after shutdown of main threads
Functional Test Mode (i.e. save Response Data and Sampler Data)
Selecting Functional Test Mode may adversely affect performance.
Add directory or jar to classpath
Browse...
Delete
Clear
Library

The numbers to the left of the green box are the number of active threads / total number of threads. These only apply to a locally run test; they do not include any threads started on remote systems when using client-server mode.

Step 7: Stop the Test Plan
You can stop your test in two ways:

- Using **Stop** (Control + '.'). It stops the threads immediately if possible.
- Using **Shutdown** (Control + ','). It requests the threads to stop at the end of any
- current work.

Jmeter Test Plan Elements

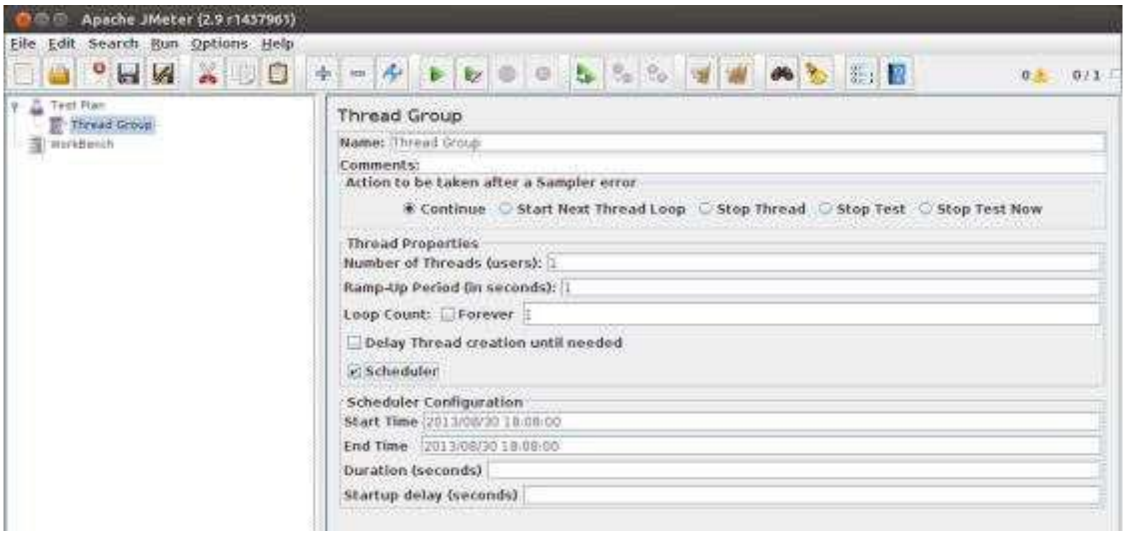
- A JMeter Test Plan comprises of test elements.
- A Test Plan comprises of at least one Thread Group. Within each Thread Group, we may place a combination of one or more of other elements: Sampler, Logic Controller, Configuration Element, Listener, and Timer.
- Each Sampler can be preceded by one or more Pre-processor element, followed by Post-processor element, and/or Assertion element. Let us see each of these elements in detail

Thread Group

Thread Group elements are the beginning points of your test plan. As the name suggests, the thread group elements control the number of threads JMeter will use during the test. We can also control the following via the Thread Group:

- Setting the number of threads
- Setting the ramp-up time
- Setting the number of test iterations

The Thread Group Control Panel looks like this:


Apache JMeter (2.9 r1437961)
File Edit Search Run Options Help
Test Plan
Thread Group
WorkBench

Thread Group
Name: Thread Group
Comments:
Action to be taken after a Sampler error
[✓ Continue] Start Next Thread Loop Stop Thread Stop Test Stop Test Now
Thread Properties
Number of Threads (users): 1
Ramp-Up Period (in seconds): 1
Loop Count: Forever 1
Delay Thread creation until needed
[✓ Scheduler]
Scheduler Configuration
Start Time 2013/08/30 18:08:00
End Time 2013/08/30 18:08:00
Duration (seconds)
Startup delay (seconds)

The Thread Group Panel holds the following components:

- **Action to be taken after a Sampler error:** In case any error occurs during test execution, you may let the test either:
- **Continue** to the next element in the test
- **Stop Thread** to stop the current Thread.
- **Stop Test** completely, in case you want to inspect the error before it continues running.

- **Number of Threads:** Simulates the number of users or connections to your server application.
- **Ramp-Up Period:** Defines how long it will take JMeter to get all threads running.
- **Loop Count:** Defines the number of times to execute the test.
- **Scheduler checkbox:** Once selected, the Scheduler Configuration section appears at the bottom of the control panel.
- **Scheduler Configuration:** You can configure the start and end time of running the test.
- **Controllers**

JMeter has two types of Controllers: *Samplers* and *Logic Controllers*.

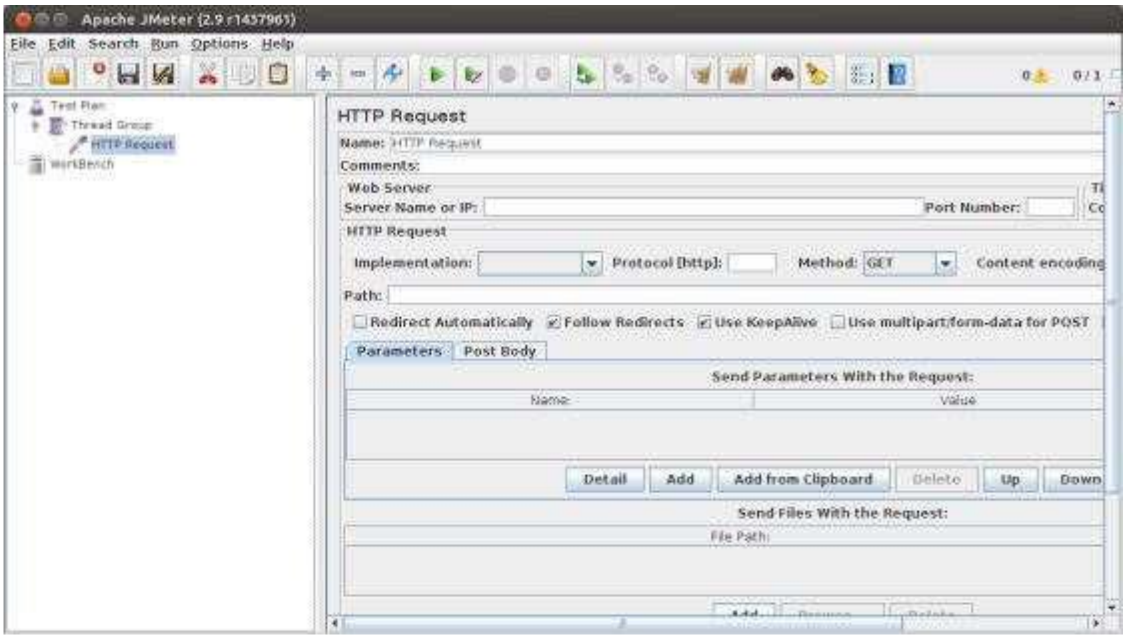
Samplers

Samplers allow JMeter to send specific types of requests to a server. They simulate a user request for a page from the target server. For example, you can add a HTTP Request sampler if you need to perform a POST, GET, or DELETE on a HTTP service.

Some useful samplers are:

- HTTP Request
- FTP Request
- JDBC Request
- Java Request
- SOAP/XML Request
- RPC Requests

The following screenshot shows an HTTP Request Sampler Control Panel:

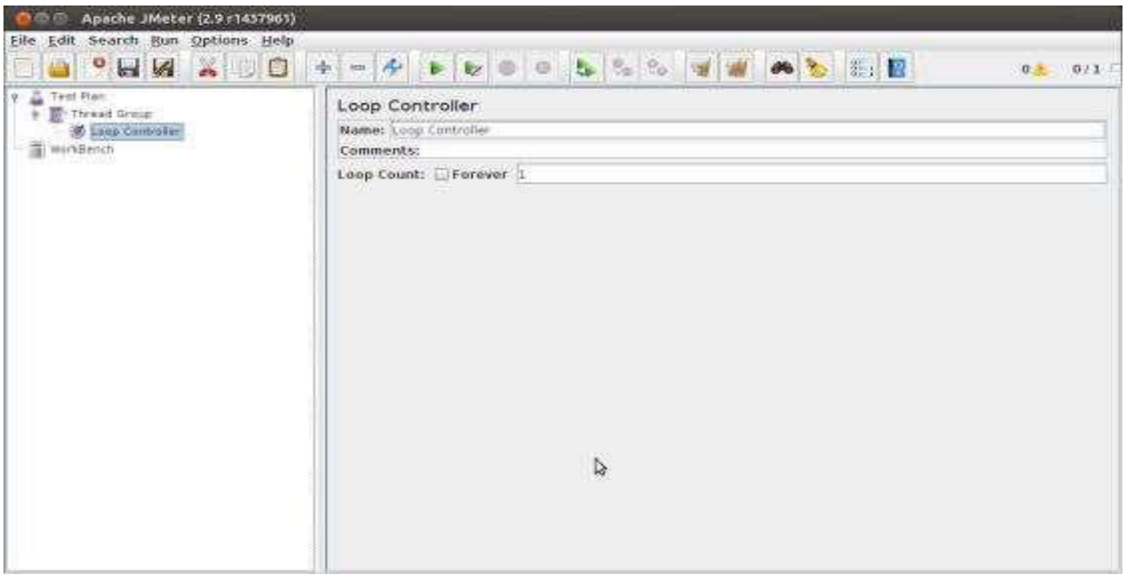


Name: HTTP Request
Comments:
Server Name or IP:
Port Number:
Implementation:
Protocol [http]:
Method: GET
Content encoding
Path:
☐ Redirect Automatically ☑ Follow Redirects ☑ Use KeepAlive ☐ Use multipart/form-data for POST

Logic Controllers

Logic Controllers control the order of processing of Samplers in a Thread. Logic controllers can change the order of a request coming from any of their child elements. Some examples are: For Each Controller, While Controller, Loop Controller, IF Controller, Run Time Controller, Interleave Controller, Throughput Controller, and Run Once Controller.

The following screenshot shows a Loop Controller Control Panel:



The following list consists of all the Logic Controllers JMeter provides:

- Simple Controller
- Loop Controller
- Random Controller
- Runtime Controller
- If Controller
- While Controller
- Switch Controller
- ForEach Controller
- Module Controller
- Include Controller
- Transaction Controller
- Recording Controller

Test Fragments

A Test Fragment is a special type of element placed at the same level as the Thread Group element. It is distinguished from a Thread Group in that it is not executed unless it is referenced by either a Module Controller or an Include_Controller. This element is purely for code re-use within Test Plans.

Listeners

Listeners are used to view the results of Samplers in the form of tables, graphs, trees, or simple text in some log files. They provide visual access to the data gathered by JMeter about the test cases as a Sampler component of JMeter is executed. Listeners can be added anywhere in the test, including directly under the test plan. They will collect data only from elements at or below their level. The following list consists of all the

Listeners JMeter provides:

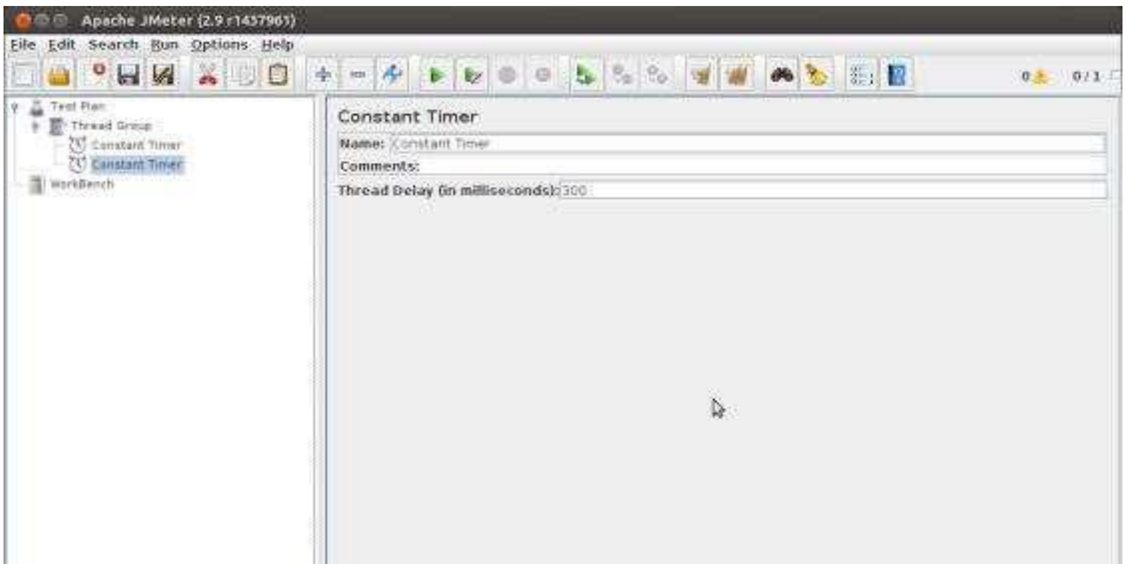
- Sample Result
- Graph Results
- Spline Visualizer
- Assertion Results
- View Results Tree
- Aggregate Report
- View Results in Table
- Simple Data Writer
- Monitor Results
- Distribution Graph (alpha)
- Summary Report

Timers

By default, a JMeter thread sends requests without pausing between each sampler. This may not be what you want. You can add a timer element which allows you to define a period to wait between each request.
The following list shows all the timers that JMeter provides:

- Constant Timer
- Gaussian Random Timer
- Uniform Random Timer
- Constant Throughput Timer
- Synchronizing Timer
- JSR223 Time
- BeanShell Time
- BSF Time
- Poisson Random Time

The following screenshot shows a Constant Timer Control Panel:


Apache JMeter (2.9 r1437961)
File Edit Search Run Options Help
0/1
Test Plan
Thread Group
Constant Timer
Constant Timer
WorkBench

Constant Timer

Name: Constant Timer

Comments:

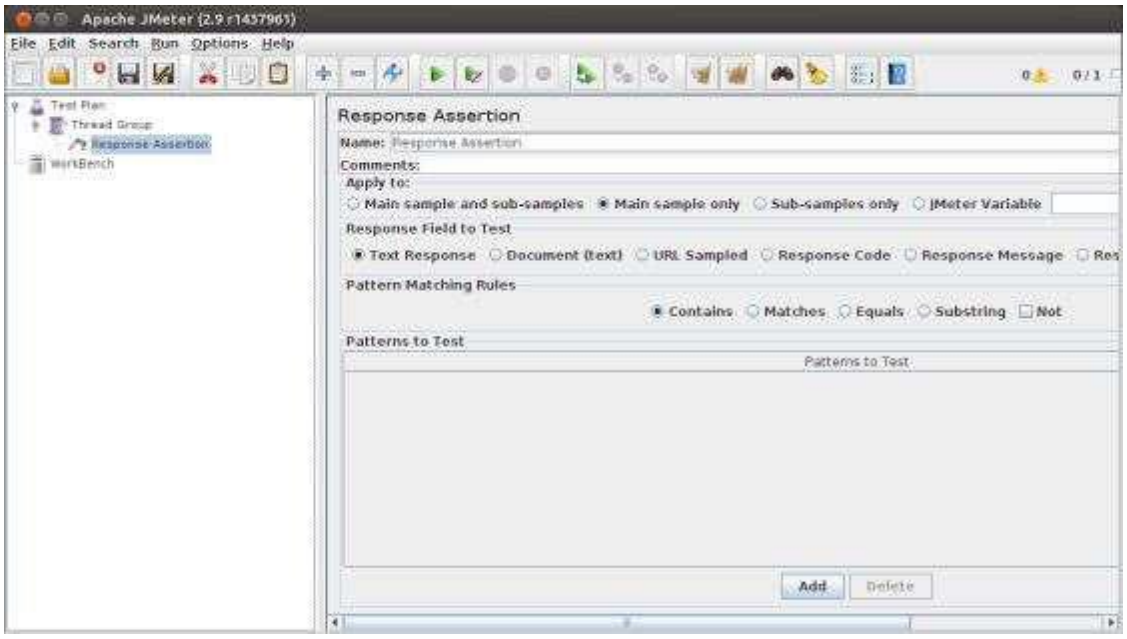
Thread Delay (in milliseconds): 300

Assertions

Assertions allow you to include some validation test on the response of your request made using a Sampler. Using assertions you can prove that your application is returning the correct data. JMeter highlights when an assertion fails.
The following list consists of all the assertions JMeter provides:

- Beanshell Assertion
- BSF Assertion
- Compare Assertion
- JSR223 Assertion
- Response Assertion
- Duration Assertion
- Size Assertion
- XML Assertion
- HTML Assertion
- XPath Assertion
- XML Schema Assertion

The following screenshot shows a Response Assertion Control Panel:


Apache JMeter (2.9 r1437961)
File Edit Search Run Options Help
Test Plan
Thread Group
Response Assertion
WorkBench
Add
Delete

Response Assertion

Name: Response Assertion

Comments:

Apply to: Main sample and sub-samples [✓ Main sample only] Sub-samples only JMeter Variable

Response Field to Test: [✓ Text Response] Document (text) URL Sampled Response Code Response Message Res

Pattern Matching Rules: [✓ Contains] Matches Equals Substring Not

Patterns to Test

Patterns to Test

Configuration Elements

Configuration Elements allow you to create defaults and variables to be used by Samplers. They are used to add or modify requests made by Samplers. They are executed at the start of the scope of which they are part, before any Samplers that are located in the same scope. Therefore, a Configuration Element is accessed only from inside the branch where it is placed.
The following list consists of all the Configuration Elements that JMeter provides:

- Counter
- CSV Data Set Config
- FTP Request Defaults
- HTTP Authorization Manager
- HTTP Cache Manager
- HTTP Cookie Manager
- HTTP Proxy Server
- HTTP Request Defaults
- HTTP Header Manager
- Java Request Defaults

Pre-processor Elements

A pre-processor element is something that runs just before a sampler executes. They are often used to modify the settings of a Sample Request just before it runs, or to update variables that are not extracted from response text. The following list consists of all the pre-processor elements that JMeter provides:

- HTML Link Parser
- HTTP URL Re-writing Modifier
- HTTP User Parameter Modifier
- User Parameters
- JDBC Pre-processor

Post-processor Elements

A post-processor executes after a sampler finishes its execution. This element is most often used to process the response data, for example, to retrieve a particular value for later use.

The following list consists of all the post-processor elements that JMeter provides:

- Regular Expression Extractor
- XPath Extractor
- Result Status Action Handler
- JSR223 Post-processor
- JDBC Post-processor
- BSF Post-processor
- CSS/JQuery Extractor
- BeanShell Post-processor
- Debug Post-processor

Execution Order of Test Elements

Following is the execution order of the test plan elements:

1. Configuration elements
2. Pre-Processors
3. Timers
4. Sampler
5. Post-Processors (unless SampleResult is null)
6. Assertions (unless SampleResult is null)
7. Listeners (unless SampleResult is null)

JMeter has some limitations especially when it is run in a distributed environment. Following these guidelines will assist in creating a real and continuous load:

- Use multiple instances of JMeter in case, the number of threads are more.
- Check the Scoping Rules and design accordingly.
- Use naming conventions always for all elements.
- Check the default browser Connectivity settings, before executing scripts.
- Add Listeners appropriately.

5.4 TEST DIRECTOR

Mercury Interactive TestDirector is an excellent tool for managing the testing process effectively. To deliver quality software, the testing process has to be very well defined and managed.

The important features of TestDirector are listed below:

- It is a web-based tool (the earlier versions being Client/Server based) and hence it facilitates distributed testing.
- As testing the software is linked to the requirements of the software, it provides the feature of linking the software requirements to the testing plan.
- It provides the features to document the testing procedures.

- It provides the feature of scheduling the manual and automated tests—the testing can be done during nighttimes or when the system load is less
- It provides the feature of setting groups of machines to carry out testing. For example, if you want to test the software on both Windows and Linux machines, you can group the machines based on their OS.
- It keeps a history of all the test runs.
- The audit trail feature allows keeping track of changes in the tests and test runs.
- It keeps a log of all defects (or bugs) found and the status of each bug can be changed by authorized persons only.
- It provides the feature of creating different users with different privileges (e.g., developer, tester, QA manager, beta tester etc.).
- It generates test reports and analysis for the QA manager to decide when the software can be released into the market.

TEST MANAGEMENT PROCESS

While using the TestDirector, the testing management process can be defined using the following four steps:

- Testing Requirements management
- Test planning
- Design and develop tests
- Run the tests in manual mode or automatic mode
- Test execution
- Test results analysis
- Analyze the defects Accordingly,

Managing the Testing Process Using Testdirector

The TestDirector testing process includes four phases:

Test Requirements Management

Requirements Manager is used to link the requirements with the tests to be carried out. Each requirement in the Software Requirement Specification(SRS) has to be tested at least once.

In the SRS, the functional requirements and performance requirements are specified. Functional requirements are generated from use-case scenarios. Performance requirements are dependent on the application.

Specifying Requirements

- Requirements are linked to tests and defects to provide complete traceability and aid the decision-making process.
- See what percent of requirements are covered by tests

- Each requirement in the tree is described in detail, and can include any relevant attachments. The QA tester assigns the requirement a priority level which is taken into consideration when the test team creates the test plan

Test Planning

In test planning, the QA manager does a detailed planning and addresses the following issues:

- Hardware and software platforms on which the testing has to be carried out.
- The various tests to be performed (functional/regression testing, performance testing, source code testing etc).
- Time schedule for conducting the tests.
- Roles and responsibilities of the persons associated with the project.
- Procedure for running the tests (manual or automatic).
- Various test cases to be generated.
- Procedure for tracking the progress of the testing.
- Documents to be generated during the testing process.
- The test engineers also identify the common test scripts that can be reused to test different modules and map the workflow between tests.
- The test plan is communicated to all the test engineers and also the development team.

Note: TestDirector can be integrated with other tools such as WinRunner and LoadRunner. Hence, the test scripts generated using these tools can be incorporated in the test plan.

Test Execution

- As the application constantly changes, using test lab, run manual and automated tests in the project in order to locate defects and assess quality.
- By creating test sets and choosing which tests to include in each set, test suite can be created. A test set is a group of tests in a Test Director project database designed to achieve specific testing goals.
- Tests can be run manually or scheduled to run automatically based on application dependencies.
- In the case of automated testing, the test scheduling is done as per the plan.
- A history of all test runs is maintained and audit trail, to trace the history of tests and test runs, is also maintained. During this phase, Test Sets are created. A test set is a set of test cases

Test Results Analysis

In this phase, the test results are analyzed—which tests passed and which tests failed. For the tests that failed, an analysis has to be carried out as to why they failed. Also, each bug is classified based on its severity. A simple way of classification is

- Critical
- Major
- Minor

When a bug is reported to the developer, it is not enough if you inform that there is a bug. we need to give additional information such as what is the problem, what is the system configuration on which the test was run, what is the version of the software ?The bug report is stored in a database.

Based on the bug tracking and analysis tools, the QA manager and the project manager can take the decision whether the software can be released to the customer or still more testing is required.

Tracking Defects:

Locating and repairing application defects efficiently is essential to the testing

process. Defects can be detected and added during all stages of the testing process.

In this phase you perform the following tasks:

- This tool features a sophisticated mechanism for tracking software defects, enabling Testing Team and the project Team to monitor defects closely from initial detection until resolution
- By linking TestDirector to e-mail system, defect tracking information can be shared by all Development and Management Teams
- Software Quality Assurance personnel