JMETER - QUICK GUIDE

Before digging into the JMeter, let us first understand few jargons mostly associated with the testing of any application.

- **Performance Test**: This test sets the 'best possible' performance expectation under a given configuration of infrastructure. It also highlights early in the testing process if changes need to be made before application goes into production.
- **Load Test**: This test is basically used for exercising \discovering the system under the top load it was designed to operate under.
- **Stress Test**: This test is an attempt to break the system by overwhelming its resources.

What is JMeter?

JMeter is a software allowing to load test or performance oriented business (functional) test on different protocols or technologies. **Stefano Mazzocchi** of the Apache Software Foundation was the original developer of JMeter. He wrote it primarily to test the performance of Apache JServ (Now called as Apache Tomcat project). Apache later redesigned JMeter to enhance the GUI and to add functional-testing capabilities.

This is a Java desktop application with a graphical interface using the Swing graphical API, can therefore run on any environment / workstation accepting 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
- Directory: LDAP
- Messaging Oriented service via JMS
- Service: POP3, IMAP, SMTP
- FTP Service

JMeter Features

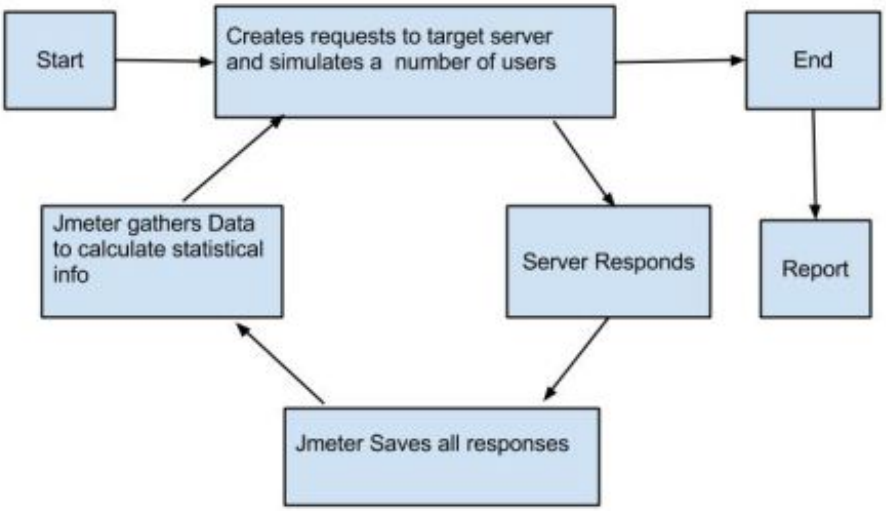
Following are some of the features of JMeter:

- Its free. Its an open source software.
- It has simple and intuitive GUI.
- JMeter can load and performance test many different server types: Web - HTTP, HTTPS, SOAP, Database via JDBC, LDAP, JMS, Mail - POP3
- It is 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 store its test plans in XML format. This means you can generate a test plan using a text editor.
- It's full multi-threading framework allows concurrent sampling by many threads and simultaneous sampling of different functions by separate thread groups.
- It is highly Extensible.

- Can also be used to perform automated and functional testing of your application.

How JMeter Works?

JMeter simulates a group of users sending requests to a target server, and return statistics that show the performance/functionality of the target server / application via tables, graphs etc. The figure below depicts this process:



ENVIRONMENT SET-UP

JMeter is a framework for Java, so the very first requirement is to have JDK installed in your machine.

System Requirement

JDK	1.6 or above.
Memory	no minimum requirement.
Disk Space	no minimum requirement.
Operating System	no minimum requirement.

Step 1 - verify Java installation in your machine

Now, open console and execute the following **java** command.

OS	Task	Command
Windows	Open Command Console	c:\> java -version
Linux	Open Command Terminal	$ java -version

Mac	Open Terminal	machine:~ joseph$ java -version

Let's verify the output for all the operating systems:

OS	Output
Windows	java version "1.7.0_25" Java(TM) SE Runtime Environment (build 1.7.0_25-b15) Java HotSpot(TM) 64-Bit Server VM (build 23.25-b01, mixed mode)
Linux	java version "1.7.0_25" Java(TM) SE Runtime Environment (build 1.7.0_25-b15) Java HotSpot(TM) 64-Bit Server VM (build 23.25-b01, mixed mode)
Mac	java version "1.7.0_25" Java(TM) SE Runtime Environment (build 1.7.0_25-b15) Java HotSpot(TM) 64-Bit Server VM (build 23.25-b01, mixed mode)

If you do not have Java installed, install the Java Software Development Kit (SDK) from http://www.oracle.com/technetwork/java/javase/downloads/index.html. We are assuming Java 1.7.0_25 as installed version for this tutorial.

Step 2: Set JAVA environment

Set the **JAVA_HOME** environment variable to point to the base directory location, where Java is installed on your machine. For example;

OS	Output
Windows	Set the environment variable JAVA_HOME to C:\Program Files\Java\jdk1.7.0_25
Linux	export JAVA_HOME=/usr/local/java-current
Mac	export JAVA_HOME=/Library/Java/Home

Append Java compiler location to System Path.

OS	Output
Windows	Append the string; C:\Program Files\Java\jdk1.7.0_25\bin to the end of the system variable, Path.
Linux	export PATH=$PATH:$JAVA_HOME/bin/
Mac	not required

Verify Java Installation using **java -version** command explained above.

Step 3: Download JMeter

Download latest version of JMeter from http://jmeter.apache.org/download_jmeter.cgi. At the time of writing this tutorial, I downloaded *apache-jmeter-2.9* and copied it into C:\>JMeter folder.

The directory structure should look like as below:

- apache-jmeter-2.9

- apache-jmeter-2.9\bin
- apache-jmeter-2.9\docs
- apache-jmeter-2.9\extras
- apache-jmeter-2.9\lib\
- apache-jmeter-2.9\lib\ext
- apache-jmeter-2.9\lib\junit
- apache-jmeter-2.9\printable_docs

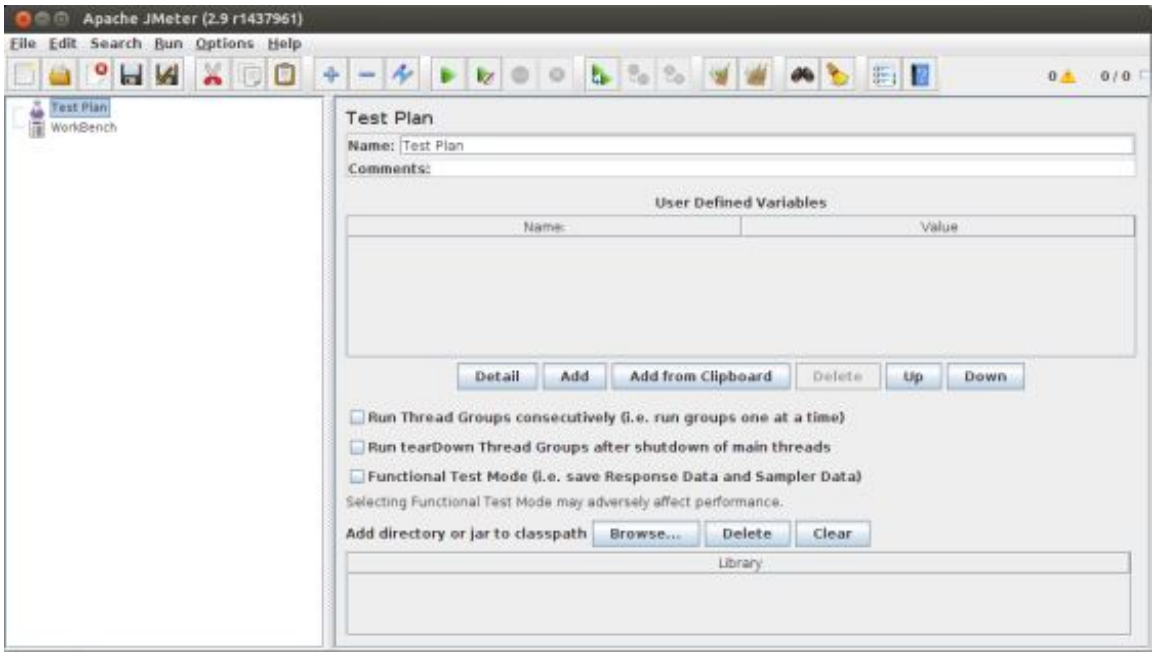
You can rename the parent directory (i.e. apache-jmeter-2.9) if you want, but do not change any of the sub-directory names.

Step 4: Run JMeter

Once the you download the JMeter, goto to the *bin* directory. In our case it would be **/home/manisha/apache-jmeter-2.9/bin**. Now click on the following:

OS	Output
Windows	jmeter.bat
Linux	jmeter.sh
Mac	jmeter.sh

Once clicked on the above file, after a short pause, the JMeter GUI should appear which is a Swing application as seen in the image below:



This is the main page and default page of the tool.

BUILD TEST PLAN

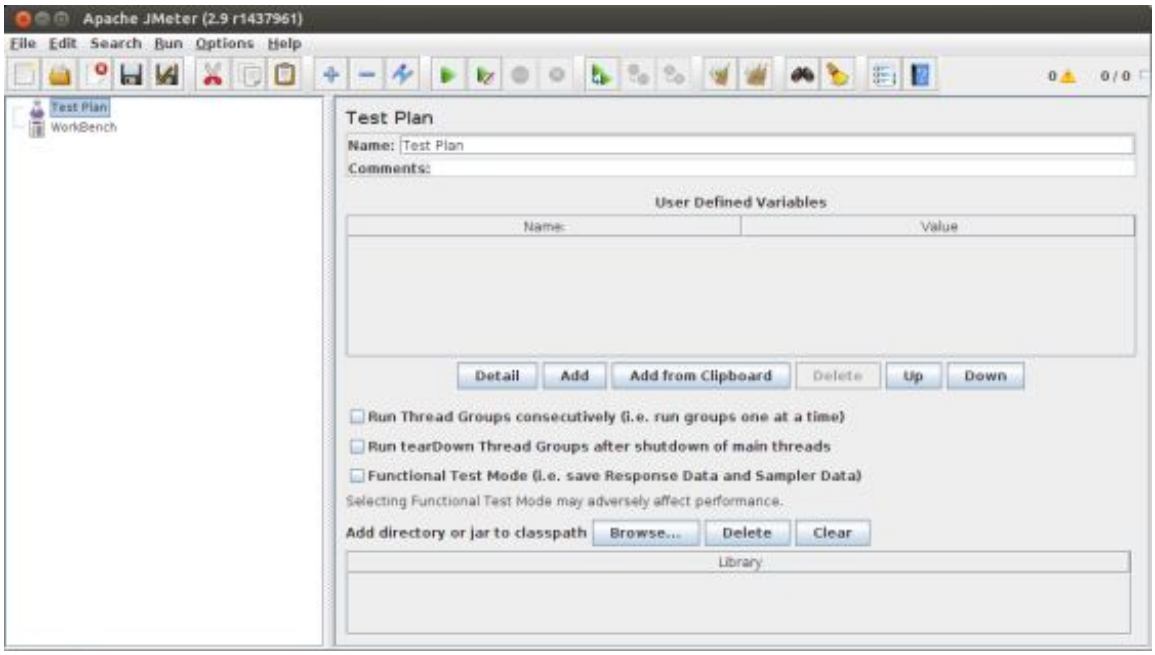
What is a Test Plan?

A Test Plan defines and provides a layout of how and what to test for example the web application as well as the client server application. It can be viewed as a container for running tests. A complete test plan will consist 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. We shall discuss these elements in detail in the next chapter Test Plan Elements.

Follow the below steps to write a test plan:

Start the JMeter window

Open the JMeter window by clicking on **/home/manisha/apache-jmeter-2.9/bin/jmeter.sh**. The JMeter window will appear as below:



This is a JMeter window having nothing added yet. Details of the above window are:

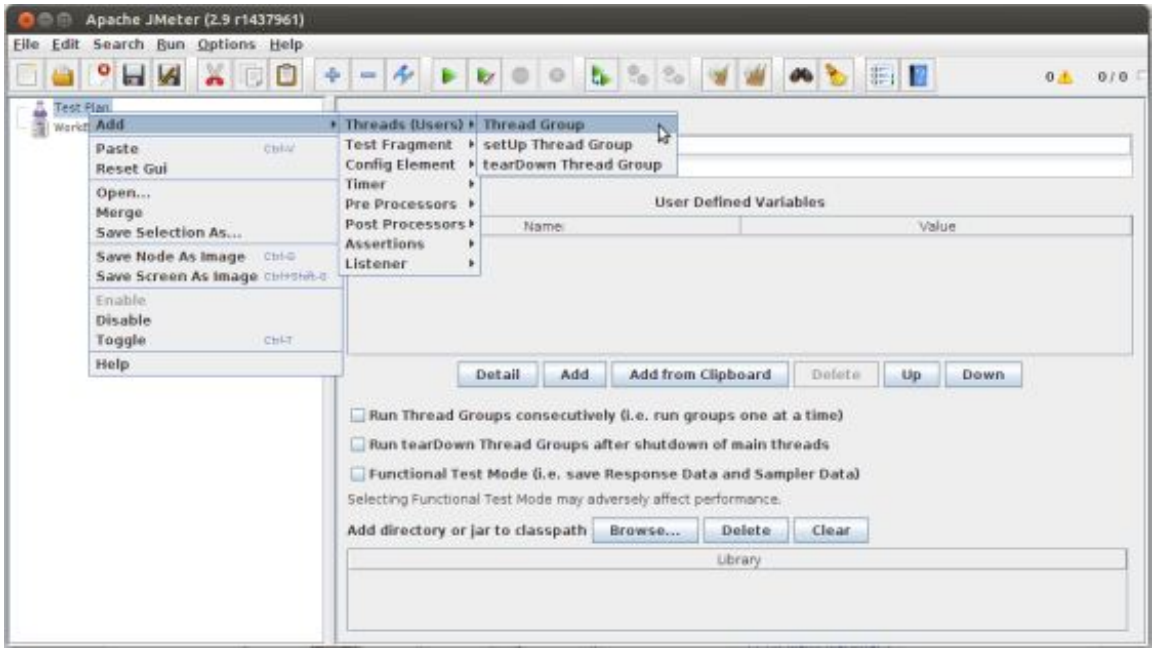
- Test Plan node is where the real test plan is kept.
- Workbench node simply provides a place to temporarily store test elements while not in use, for copy/paste purposes. When you save your test plan, WorkBench items are not saved with it.

Add/Remove Elements

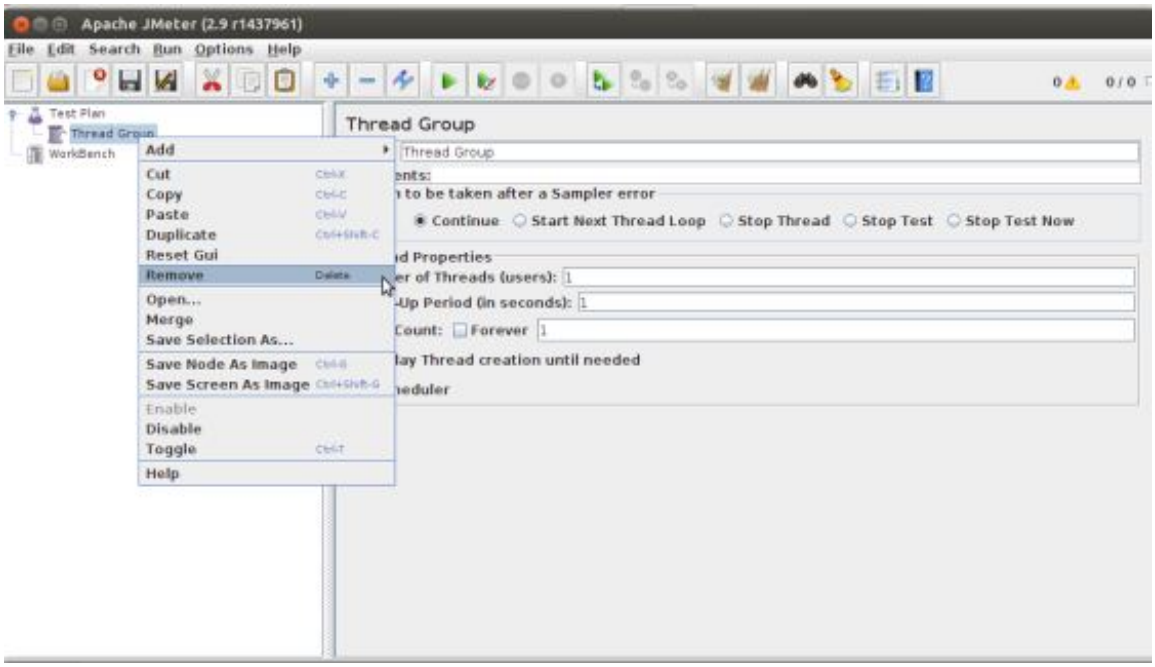
Elements (which will be discussed in the next chapter Test Plan Elements) of a test plan can be added by right clicking on the Test Plan node and choosing a new element from the "add" list.

Alternatively, elements can be loaded from file and added by choosing the "merge" or "open" option.

For example let's add a Thread Group element to a Test Plan as shown below:

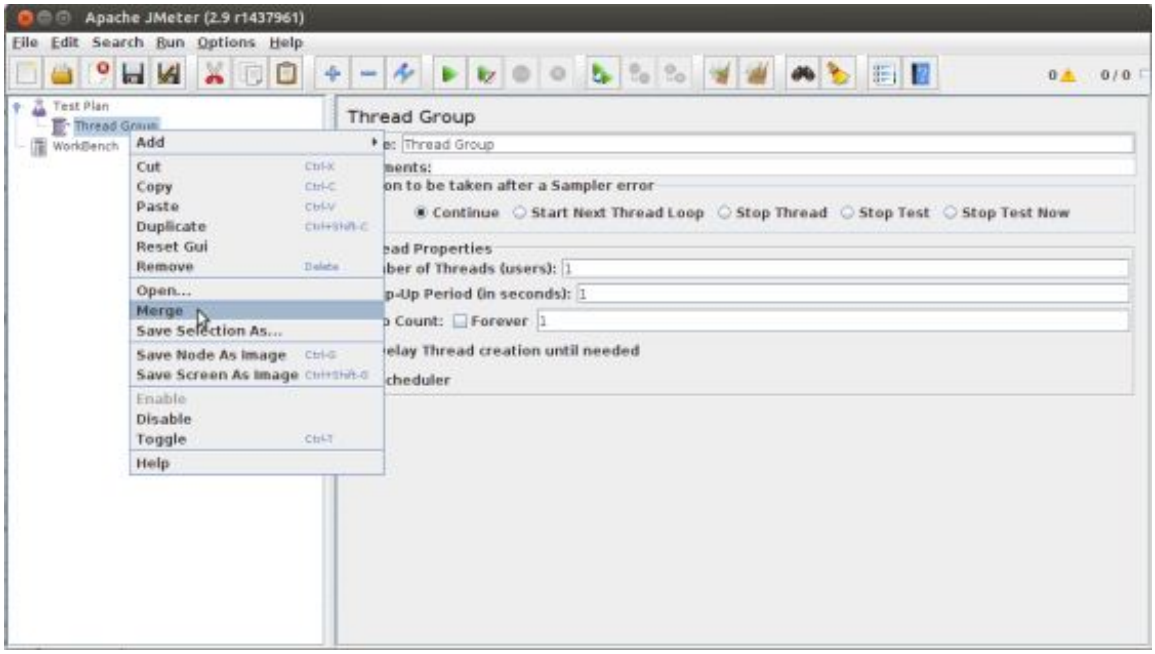


To remove an element, make sure the element is selected, right-click on the element, and choose the "remove" option.



Loading and Saving Elements

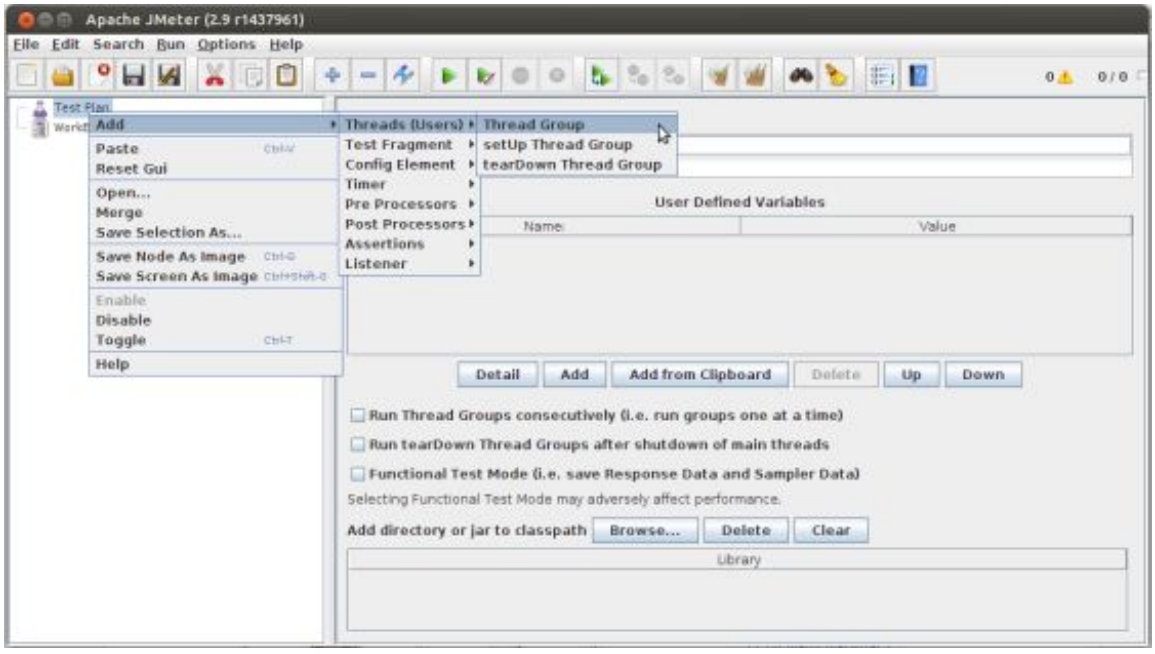
To load an element from file, right click on the existing tree element to which you want to add the loaded element, and select the "merge" option. Choose the file where your elements are saved. JMeter will merge the elements into the tree.



To save tree elements, right click on an element and choose the *Save Selection As ...* option. JMeter will save the element selected, plus all child elements beneath it. By default JMeter doesn't save the elements, you need to explicitly save it as mentioned earlier.

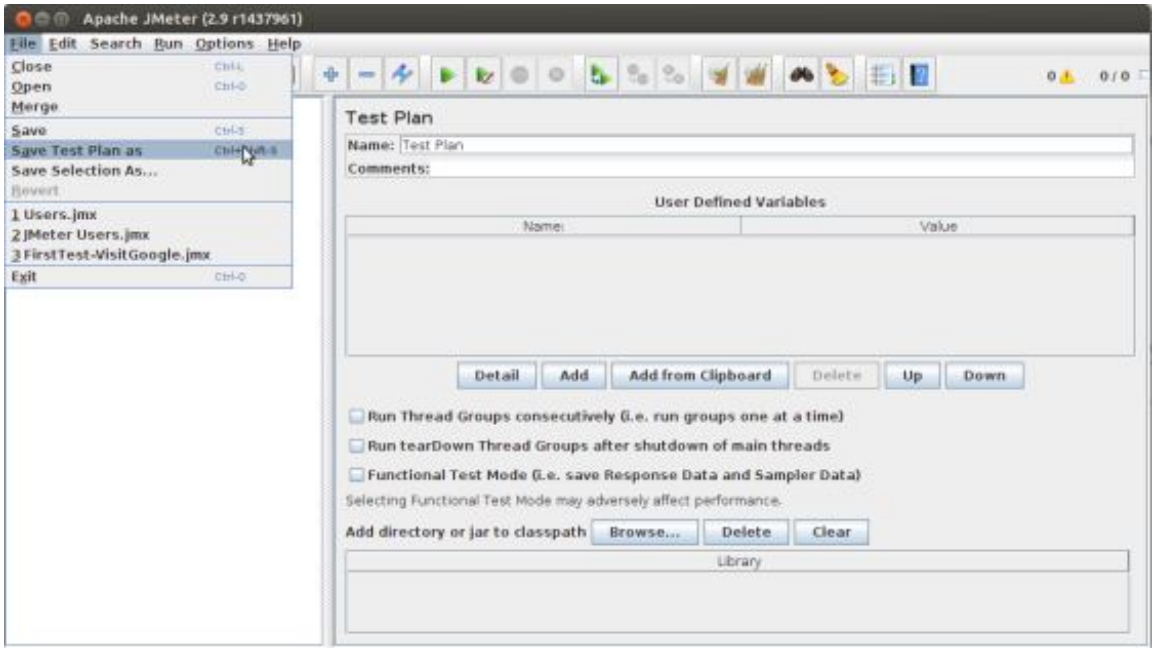
Configuring Tree Elements

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



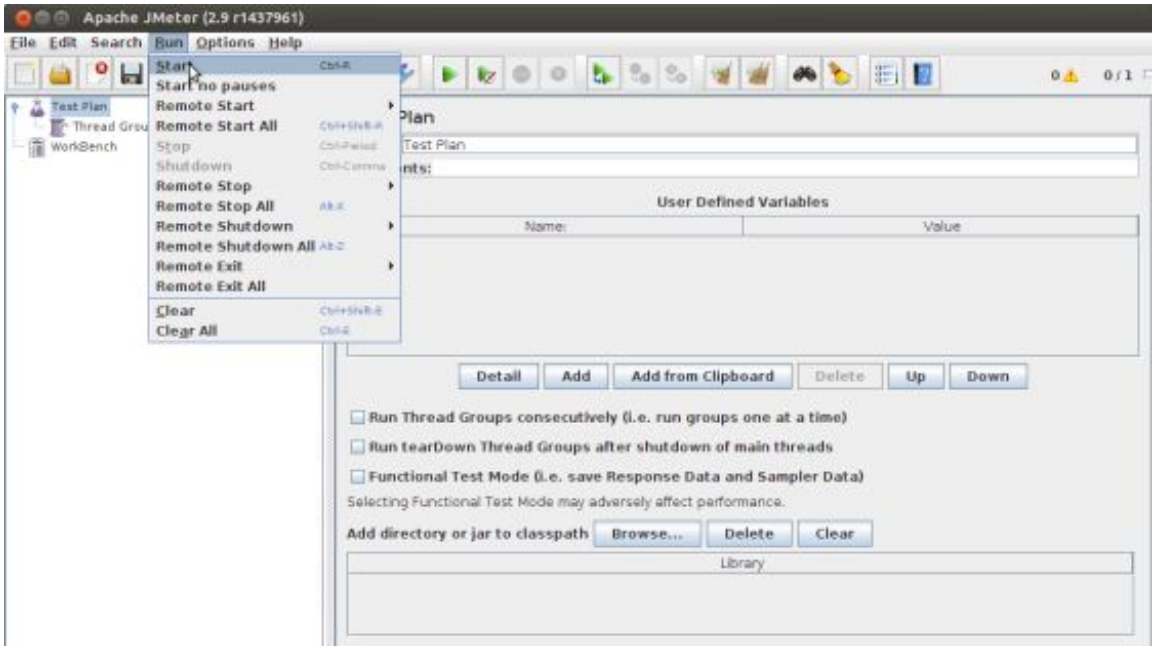
Saving the Test Plan

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



Running a Test Plan

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



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.

Stopping a Test

You can stop your test in two ways:

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

TEST PLAN ELEMENTS

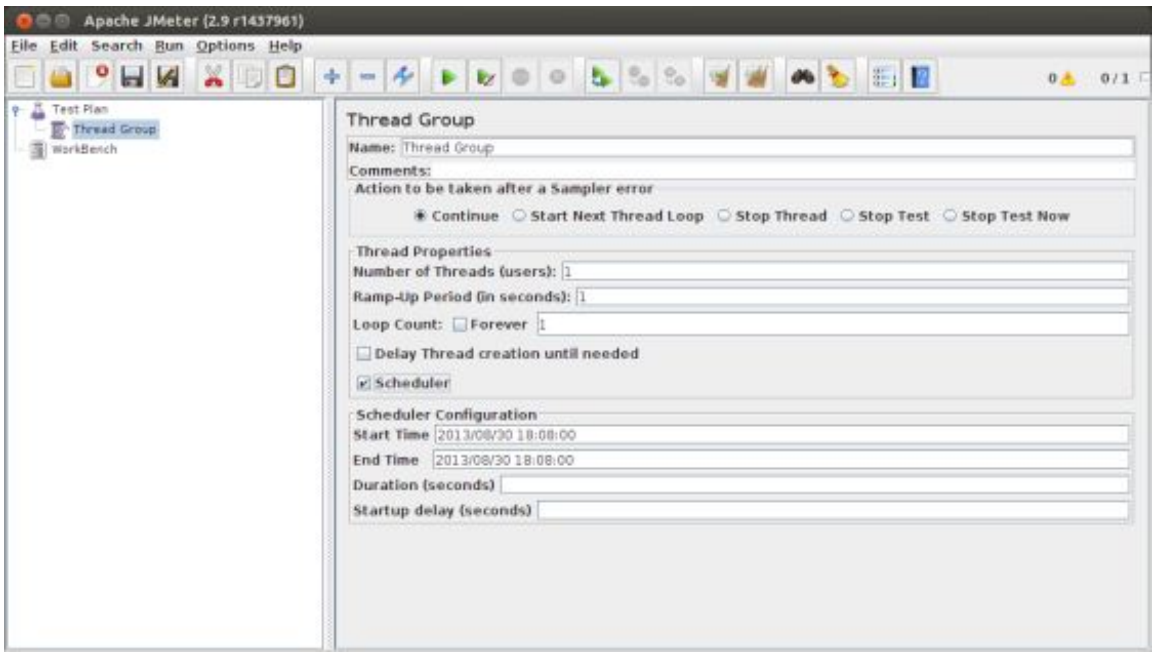
A JMeter Test Plan comprises of test elements which are discussed below. A Test Plan would comprise at least one Thread Group. Within each Thread Group we may place a combination of one or more of other elements: Sampler, Logic Controller, Configuration Element, Listener, and Timer. Each Sampler can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

ThreadGroup

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:

- By setting the number of Threads.
- By setting the Ramp Up Time
- By setting the number of test iterations.

The Thread Group Control Panel looks like this:



Details of each component on the above panel are:

- **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 continue running.
- **Number of Threads:** Simulates the number of user(s) or connection(s) 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 will appear 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*.

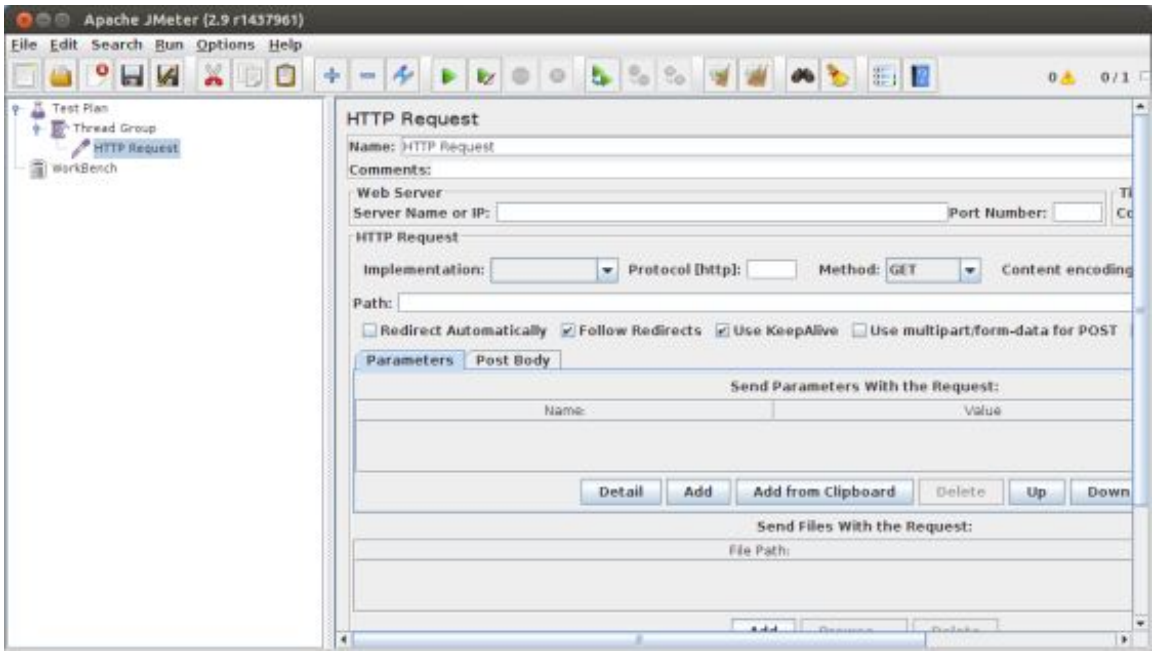
Samplers

Samplers allow JMeter to send specific types of requests to a server. They simulate a user's 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, DELETE on a HTTP service

Some useful samplers are:

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

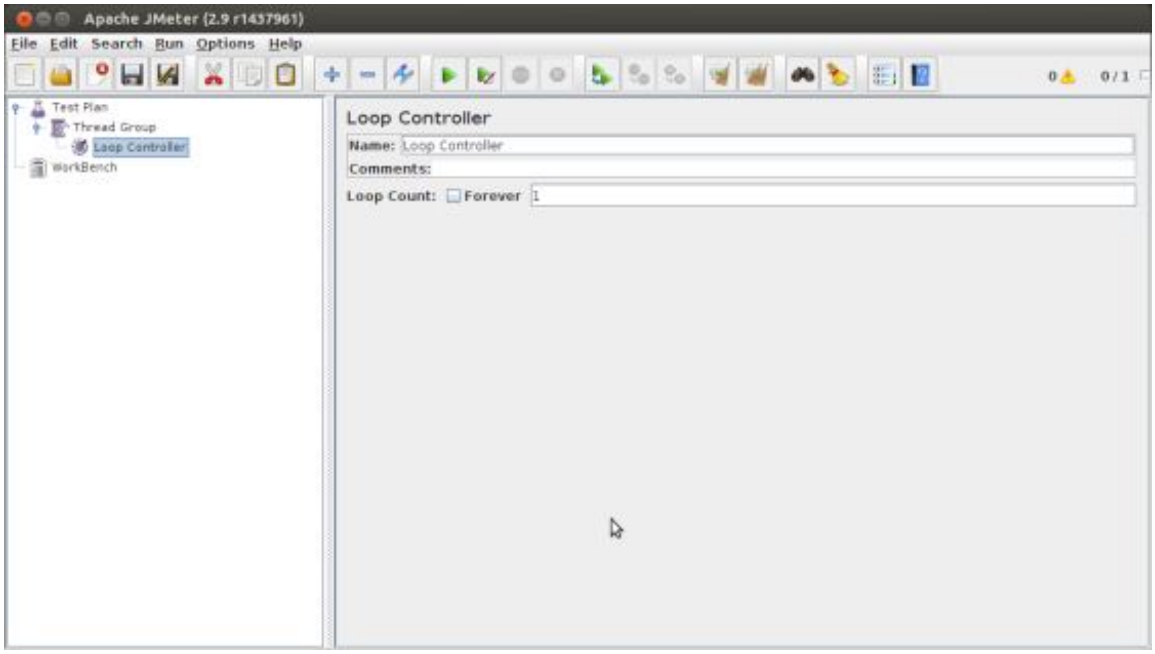
An HTTP Request Sampler Control Panel looks like the following figure:



Logic Controllers

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

A Loop Controller Control Panel looks like the following figure:



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

- Simple Controller
- Loop Controller
- Once Only Controller
- Interleave Controller
- Random Controller
- Random Order Controller
- Throughput Controller
- Runtime Controller
- If Controller
- While Controller
- Switch Controller
- ForEach Controller
- Module Controller
- Include Controller
- Transaction Controller
- Recording Controller

Test Fragments

Test Fragments is a special type of element placed at the same level as 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 let you 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 Save Configuration
- Graph Full Results
- Graph Results
- Spline Visualizer
- Assertion Results
- View Results Tree
- Aggregate Report
- View Results in Table
- Simple Data Writer
- Monitor Results
- Distribution Graph (alpha)
- Aggregate Graph
- Mailer Visualizer
- BeanShell Listener
- Summary Report

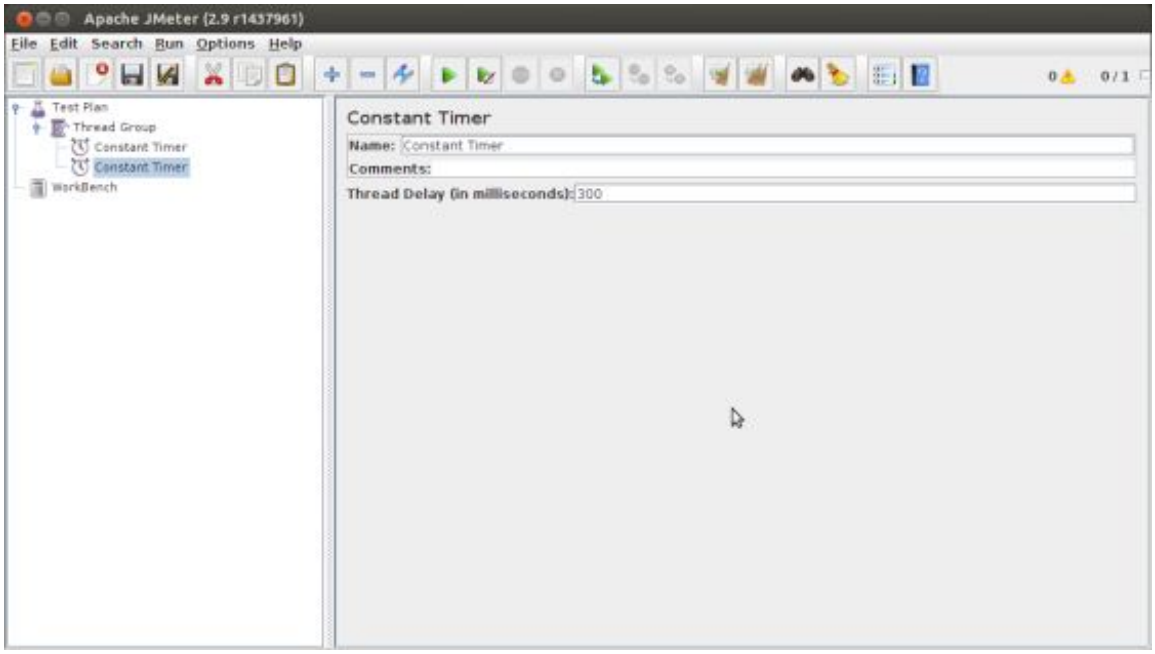
Timers

By default, a JMeter thread will send requests without pausing between each sampler. This may not be what you want. We can add a timer element which will allow us to define a period to wait between each request.

The following list consists of all the Timers JMeter provides:

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

As an example, the Constant Timer Control Panel looks like this:



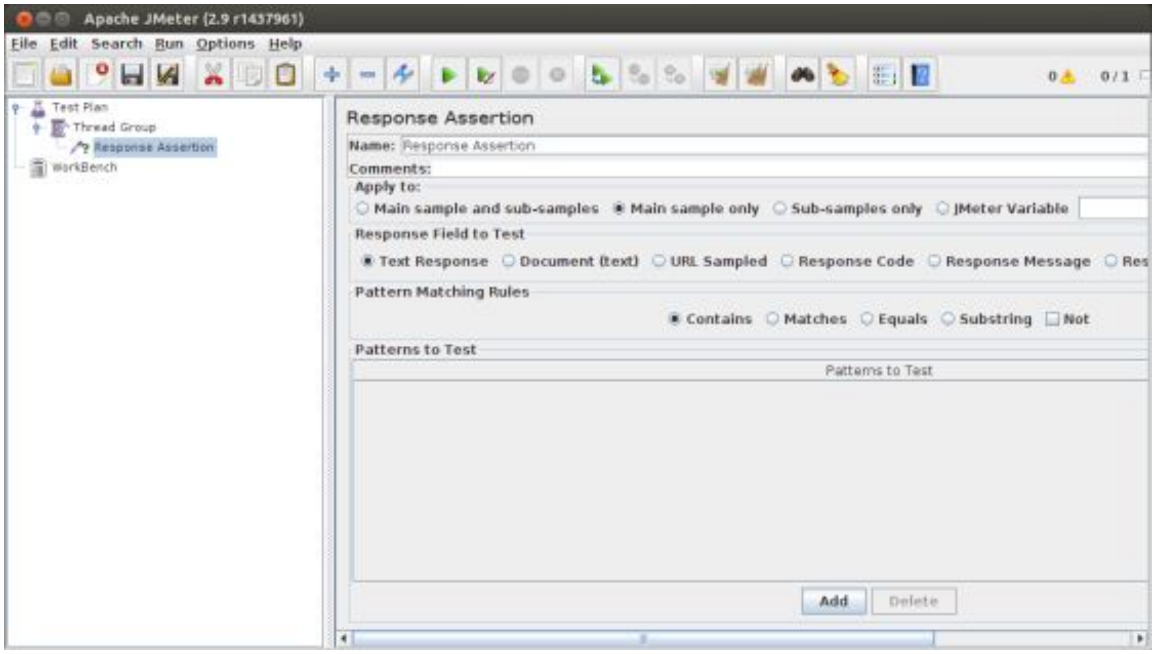
Assertions

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 will highlight 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
- BeanShell Assertion
- MD5Hex Assertion
- HTML Assertion
- XPath Assertion
- XML Schema Assertion

As an example, the Response Assertion Control Panel looks like this:



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 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
- Keystore Configuration
- JDBC Connection Configuration
- Login Config Element
- LDAP Request Defaults
- LDAP Extended Request Defaults
- TCP Sampler Config
- User Defined Variables

- Simple Config Element
- Random Variable

Pre-Processor Elements

A Pre-Procesor is something that will happen 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 JMeter provides:

- HTML Link Parser
- HTTP URL Re-writing Modifier
- HTTP User Parameter Modifier
- User Parameters
- JDBC PreProcessor
- JSR223 PreProcessor
- Reg Ex User Parameters
- BeanShell PreProcessor
- BSF PreProcessor

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 particular value for later use.

The following list consists of all the Post-Processor Elements JMeter provides:

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

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

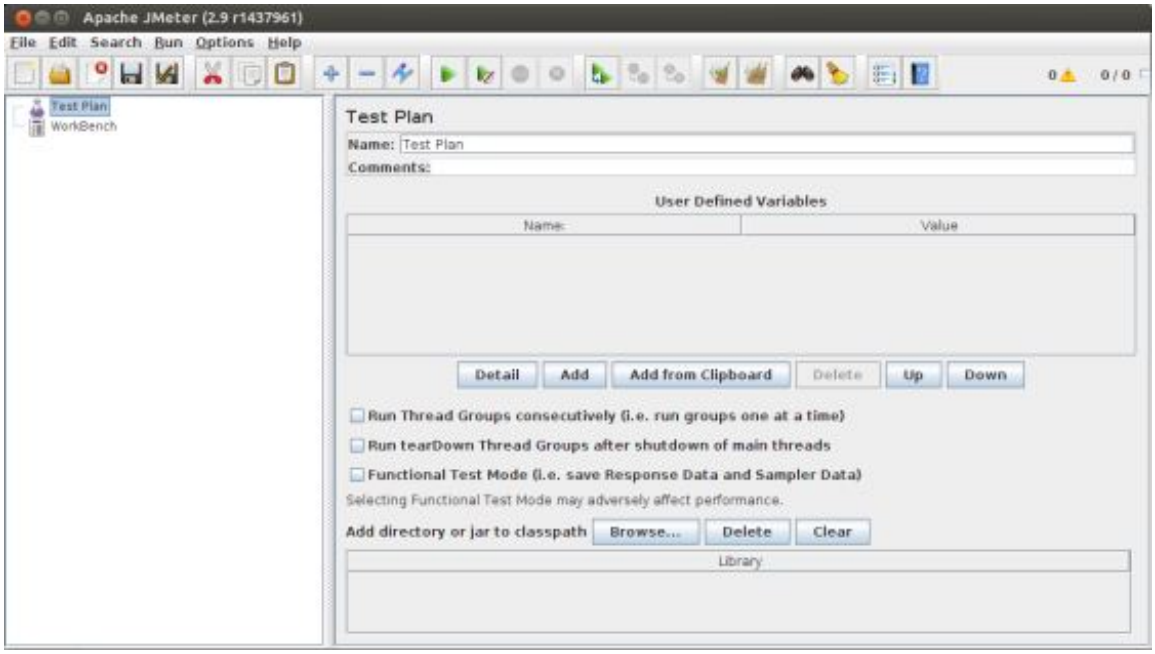
5. Post-Processors (unless SampleResult is null)
6. Assertions (unless SampleResult is null)
7. Listeners (unless SampleResult is null)

WEB TEST PLAN

Let's build a simple test plan which tests a web page. We will write a test plan in Apache JMeter so that we can test performance of one web page say page shown by the URL:*http://www.tutorialspoint.com/*.

Start JMeter

Open the JMeter window by clicking on **/home/manisha/apache-jmeter-2.9/bin/jmeter.sh**. The JMeter window will appear as below:

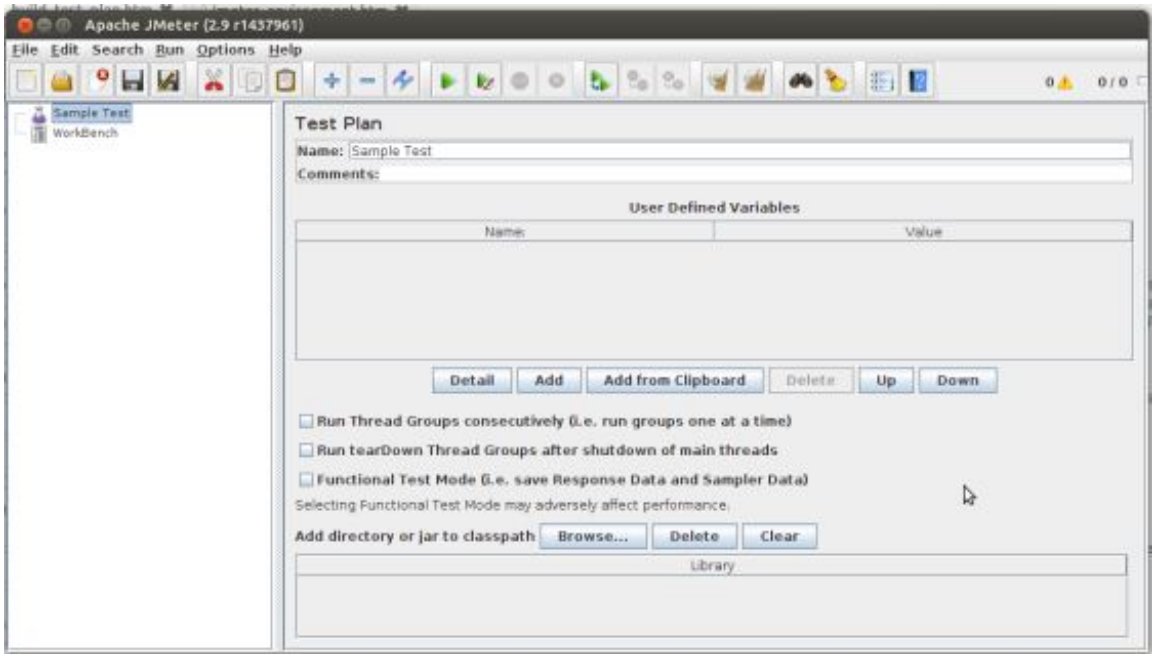


This is a JMeter window having nothing added yet. Details of the above window are:

- Test Plan node is where the real test plan is kept.
- Workbench node is where the temporary stuff is kept.

Rename Test Plan

Change the name of test plan node to Sample Test in the *Name* text box. You have to change focus to workbench node and back to Test Plan node to see the name getting reflected.

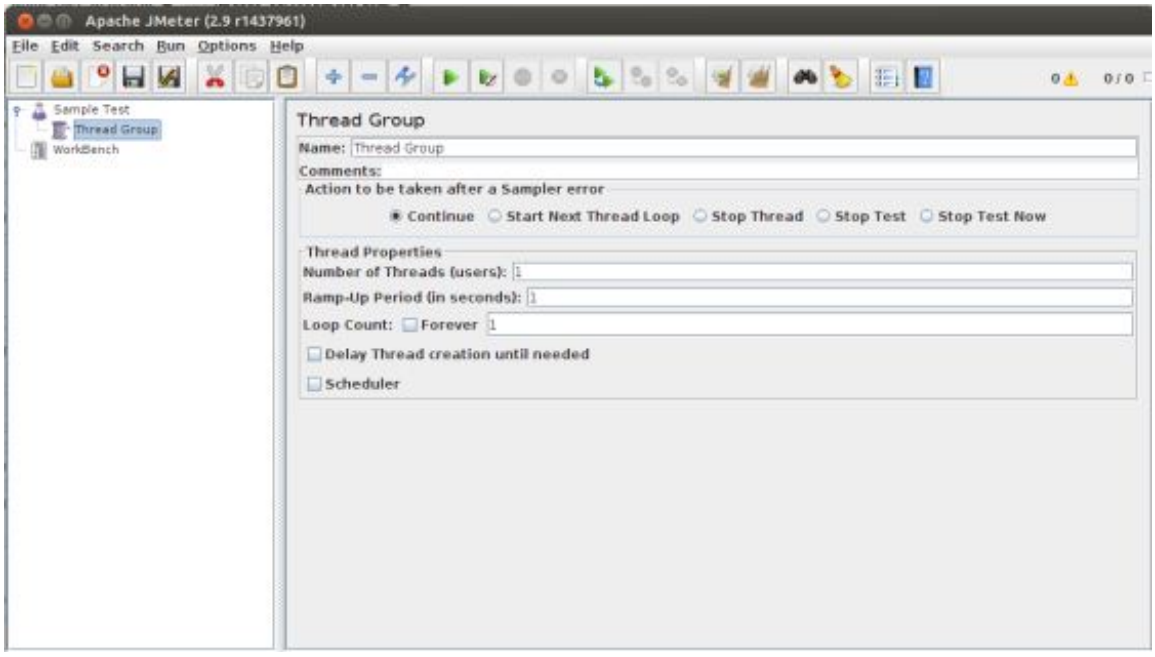


Add Thread Group

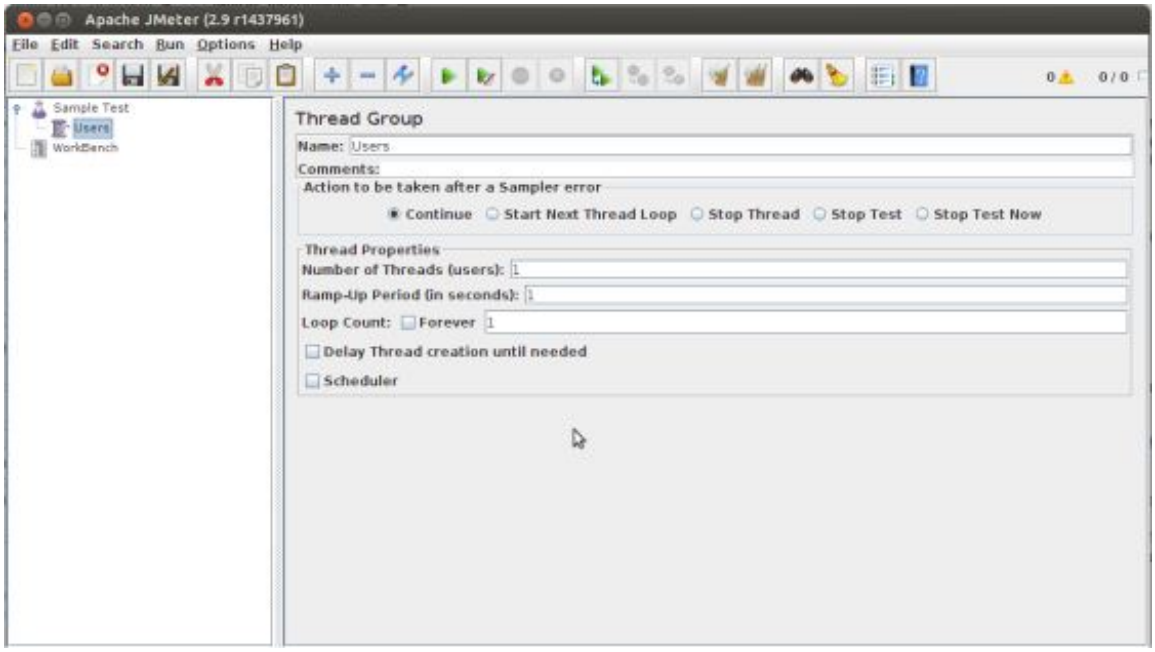
Now we will add our first element in the window. We will add one Thread Group, which is placeholder for all other elements like Samplers, Controllers, Listeners. We need one so we can configure number of users to simulate.

In JMeter all the node elements are added by using the context menu. You have to right click the element where you want to add a child element node. Then choose the appropriate option to add.

Right click on Sample Test(our Test Plan)> Add> Threads(Users)> Thread Group. Thread Group will get added under the Test Plan (Sample Test) node.

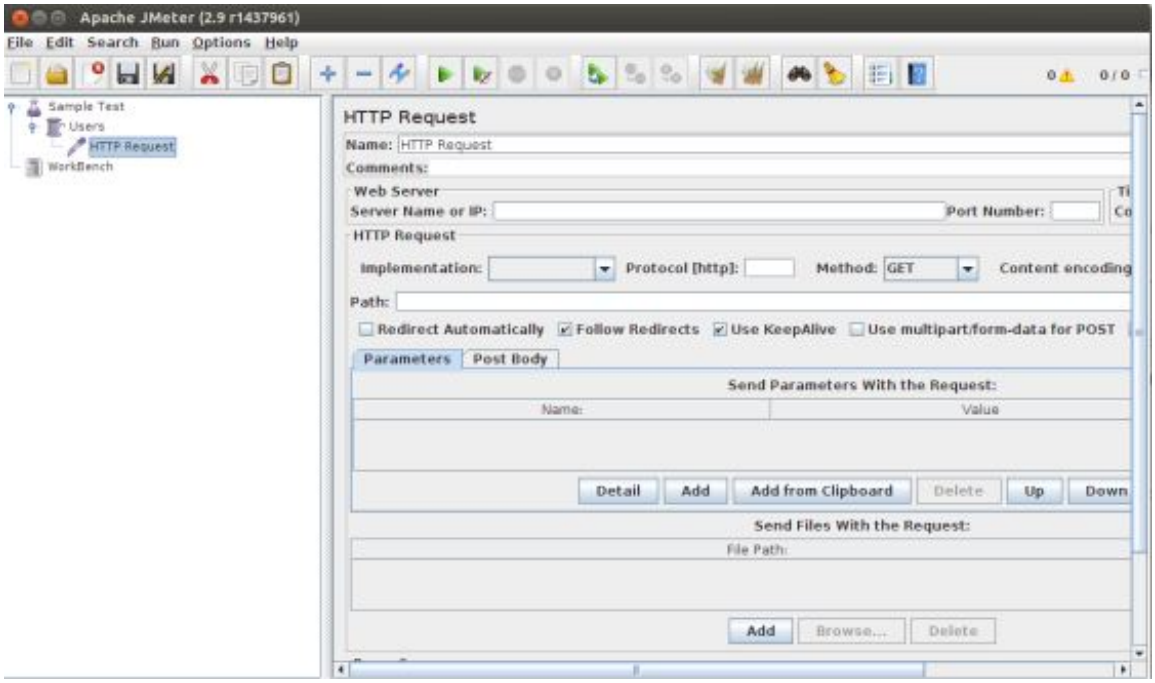


We will name Thread Group as *Users*. For us this element means Users visiting the TutorialsPoint Home Page.

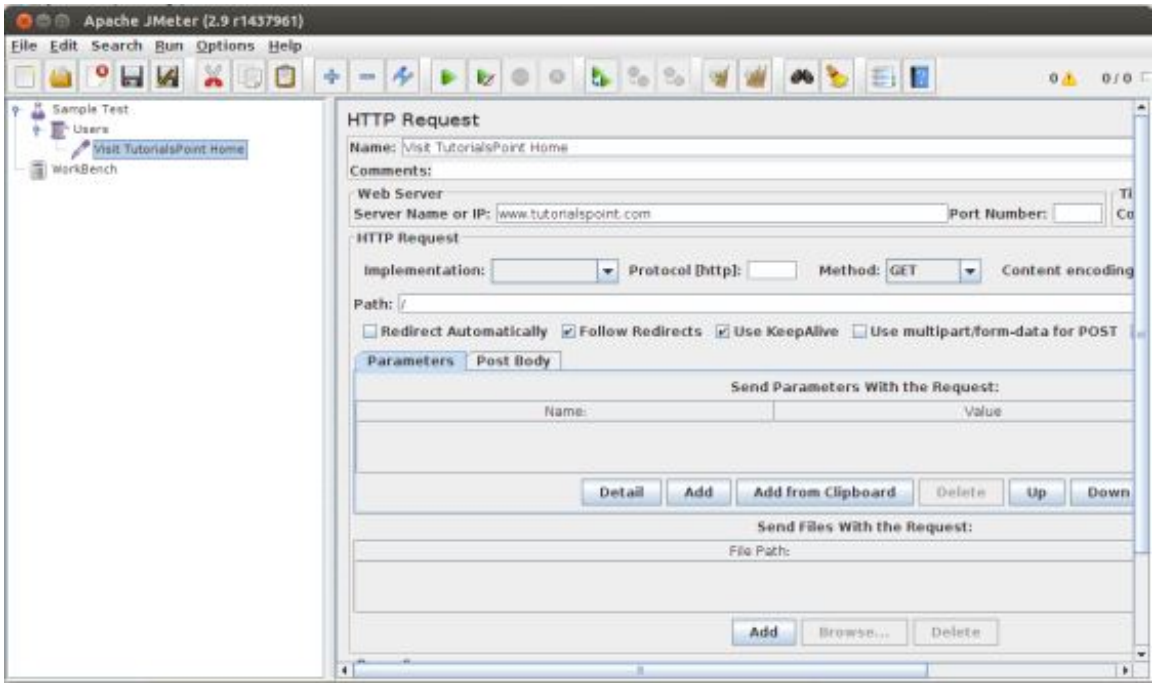


Add Sampler

Now we have to add one Sampler in our Thread Group (Users). As done earlier for adding Thread group, this time we will open the context menu of the Thread Group (Users) node by right clicking and we will add HTTP Request Sampler by choosing Add > Sampler> HTTP request option.



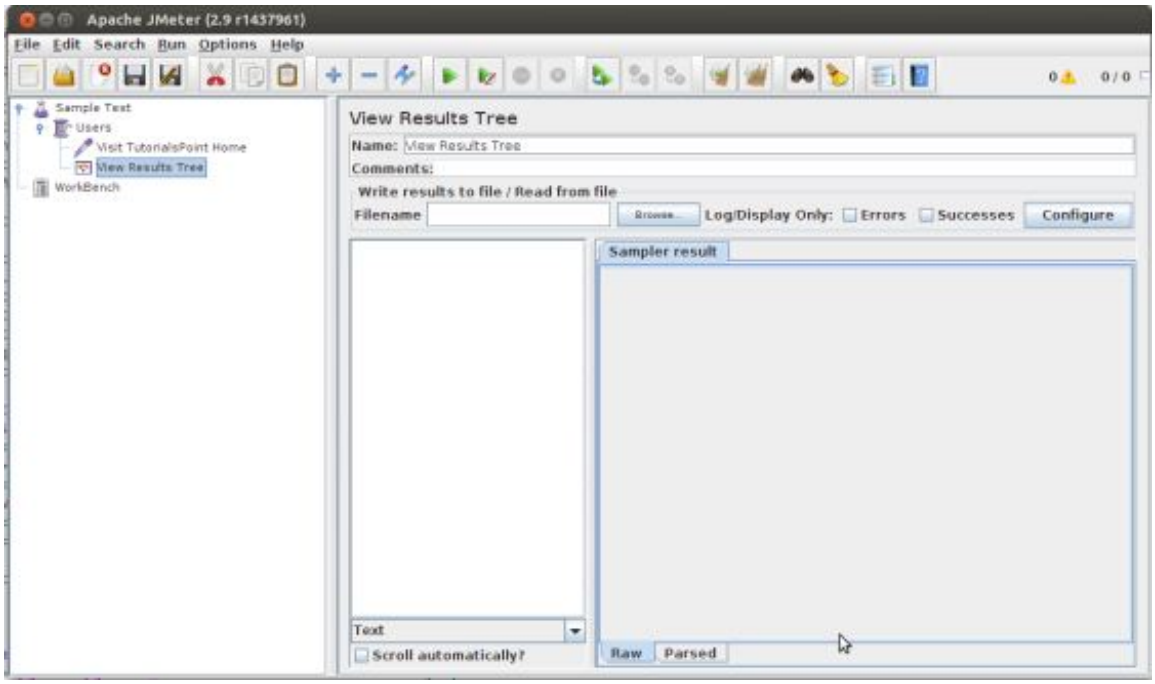
This will add one empty HTTP Request Sampler under the Thread Group (Users) node. Let us configure this node element:



- **Name:** We will change the name to reflect the action what we want to achieve. We will name it as **Visit TutorialsPoint Home Page**
- **Server Name or IP:** Here we have to type the web server name. In our case it is **www.tutorialspoint.com**. (http:// part is not written this is only the name of the server or its IP)
- **Protocol:** We will keep this blank, which means we want HTTP as the protocol.
- **Path:** We will type path as / (slash). This means we want the root page of the server.

Add Listener

We will add a listener. Let us add View Results Tree Listener under the Thread Group (User) node. This will ensure that the results of the Sampler will be available to view in this Listener node element. Open the context menu and Right click on Thread Group(Users) choose Add > Listener > View Results Tree option to add the listener.

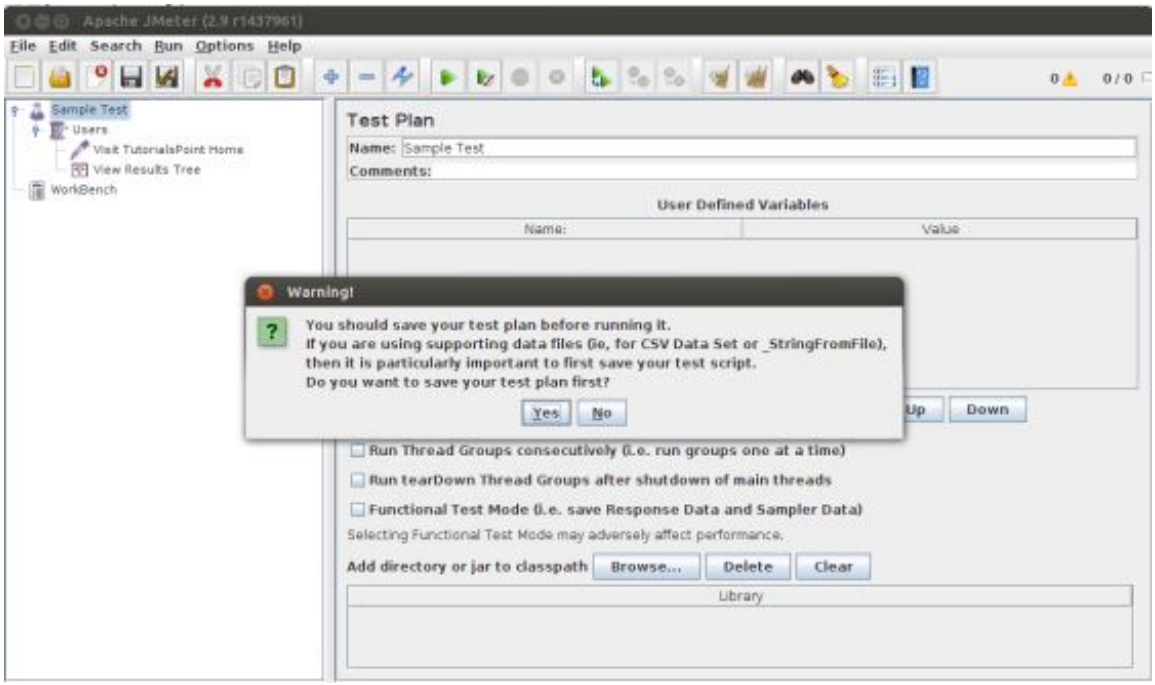


Run the Test Plan

Now with all the setup, let's execute the test plan. With the configuration of the Thread Group (Users) we have kept it all default values. This means JMeter will execute the sampler only once. It will be like a single user and only one time.

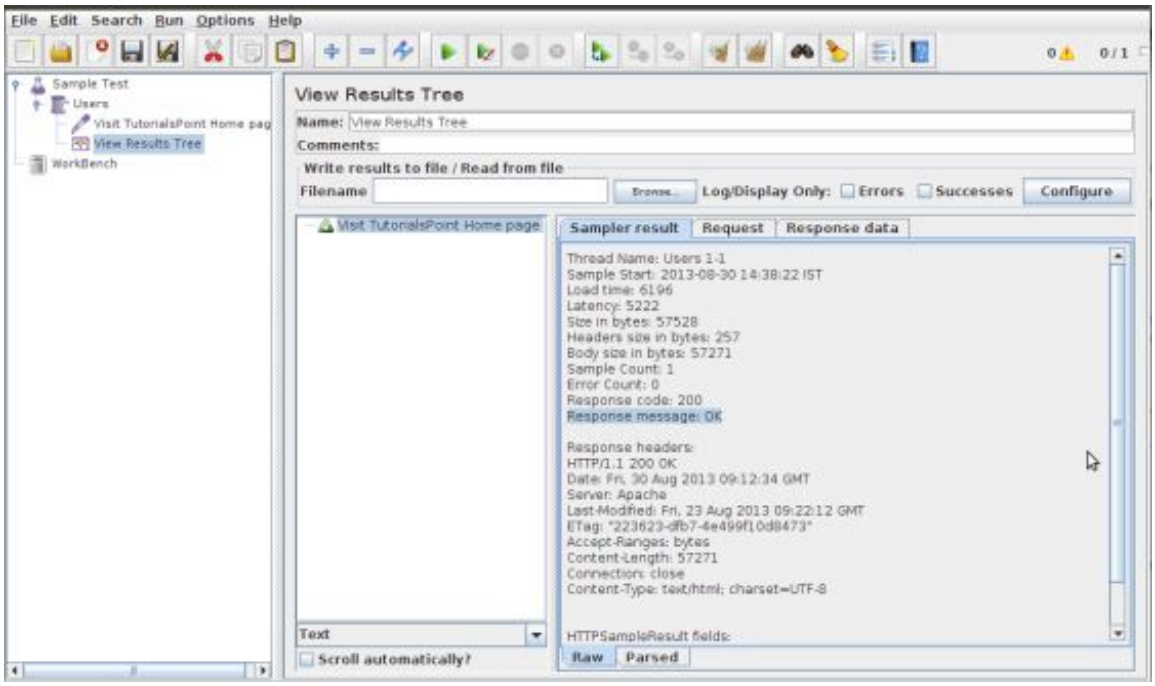
This is similar to like a user visiting a web page through browser, only here we are doing that through JMeter sampler. We will execute the test plan using Run > Start option.

Apache JMeter asks us to save the test plan in a disk file before actually starting the test. This is important if we want to run the test plan again and again. If we say not to save by clicking No option it will run without saving.



View Output

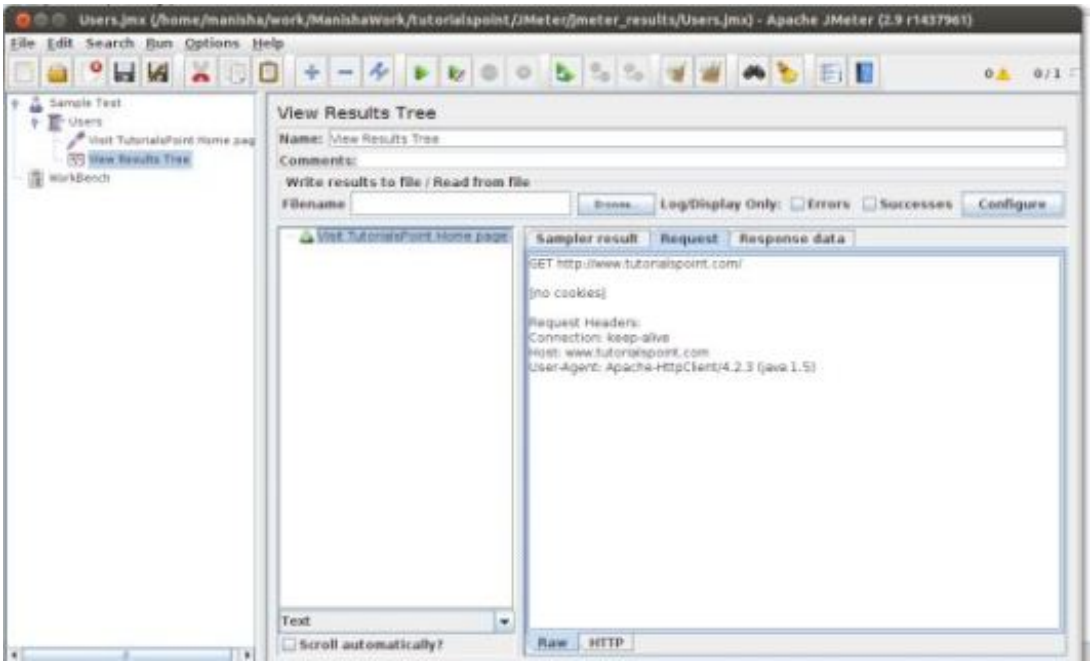
We have kept the setting of the thread group as single thread (that means one user only) and loop for 1 time (that means run only one time), hence we will get result of one single transaction in the View Result Tree Listener.



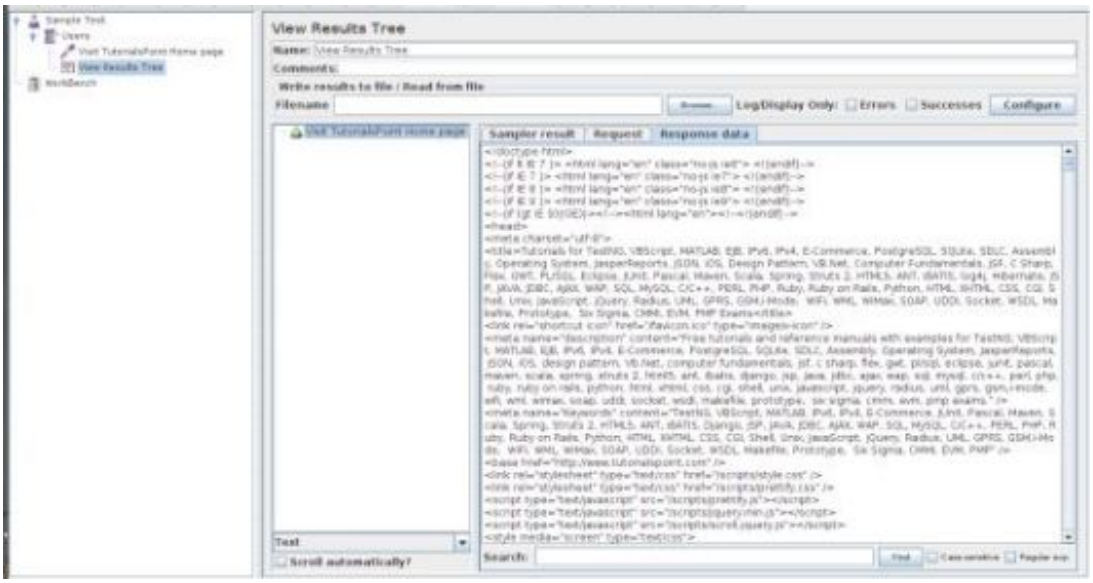
Details of the above result are:

- Green color against the name *Visit TutorialsPoint Home Page* indicates success.

- JMeter has stored all the headers and the response sent by the web server and ready to show us the result in many ways.
- The first tab is Sampler Results. It shows JMeter data as well as data returned by the web server.
- The second tab is Request, where all the data which was sent to the web server as part of the request is shown.



- The last tab is Response data. In this tab the listener shows the data received from server as it is in text format.



This is just a simple test plan which executes only one request. But JMeter's real strength is in sending the same request like many users are sending it. To test the web servers with multiple users we will have to change the Thread Group (Users) settings.

DATABASE TEST PLAN

In this chapter we will see how to create a simple test plan to test the database server. For our test purpose we have used the MYSQL database server. You can use any other database for testing. For installation and table creation in MYSQ please refer MYSQL Tutorial.

Once MYSQL is installed, follow the steps below to setup the database:

- Create a database with name "tutorial".
- Create a table *tutorials_tbl*.
- Insert records into *tutorials_tbl* :

```
mysql> use TUTORIALS;
Database changed
mysql> INSERT INTO tutorials_tbl
    ->(tutorial_title, tutorial_author, submission_date)
    ->VALUES
    ->("Learn PHP", "John Poul", NOW());
Query OK, 1 row affected (0.01 sec)
mysql> INSERT INTO tutorials_tbl
    ->(tutorial_title, tutorial_author, submission_date)
    ->VALUES
    ->("Learn MySQL", "Abdul S", NOW());
Query OK, 1 row affected (0.01 sec)
mysql> INSERT INTO tutorials_tbl
    ->(tutorial_title, tutorial_author, submission_date)
    ->VALUES
    ->("JAVA Tutorial", "Sanjay", '2007-05-06');
Query OK, 1 row affected (0.01 sec)
mysql>
```

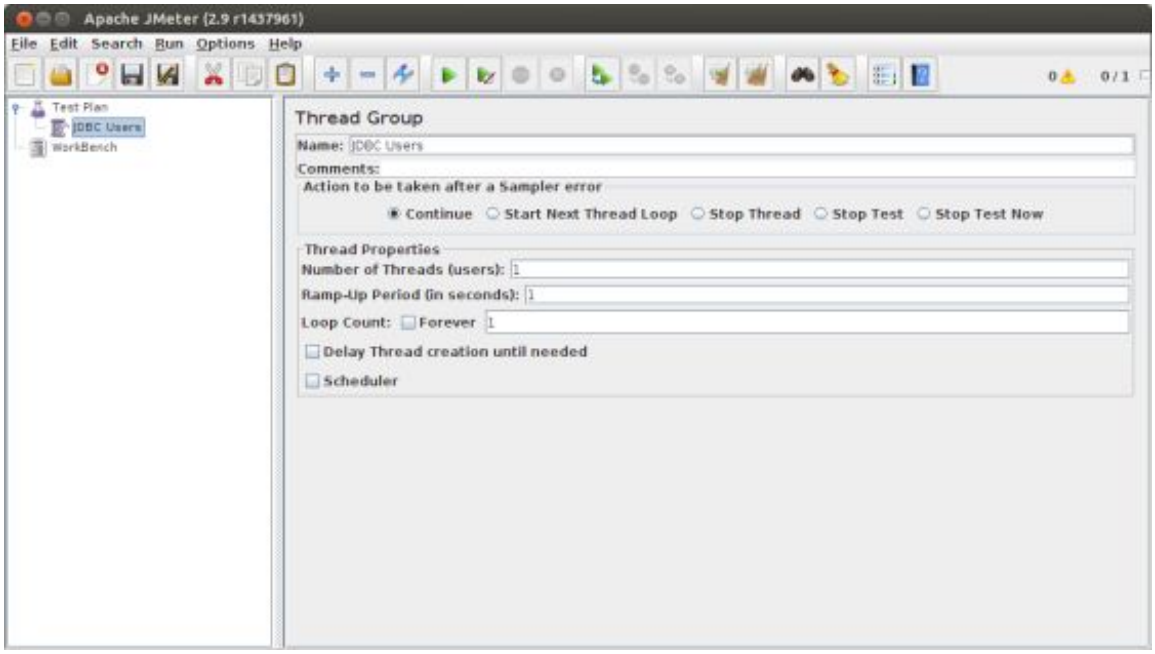
- Copy the appropriate JDBC driver to **/home/manisha/apache-jmeter-2.9/lib**.

Create JMeter Test Plan

First let's start the JMeter from **/home/manisha/apache-jmeter-2.9/bin/jmeter.sh**.

Add Users

Now create a Thread group, right click on **Test Plan > Add> Threads(Users)> Thread Group**. Thread Group will get added under the Test Plan node. Rename this Thread Group as *JDBC Users*.



We will not change the default properties of the Thread Group.

Adding JDBC Requests

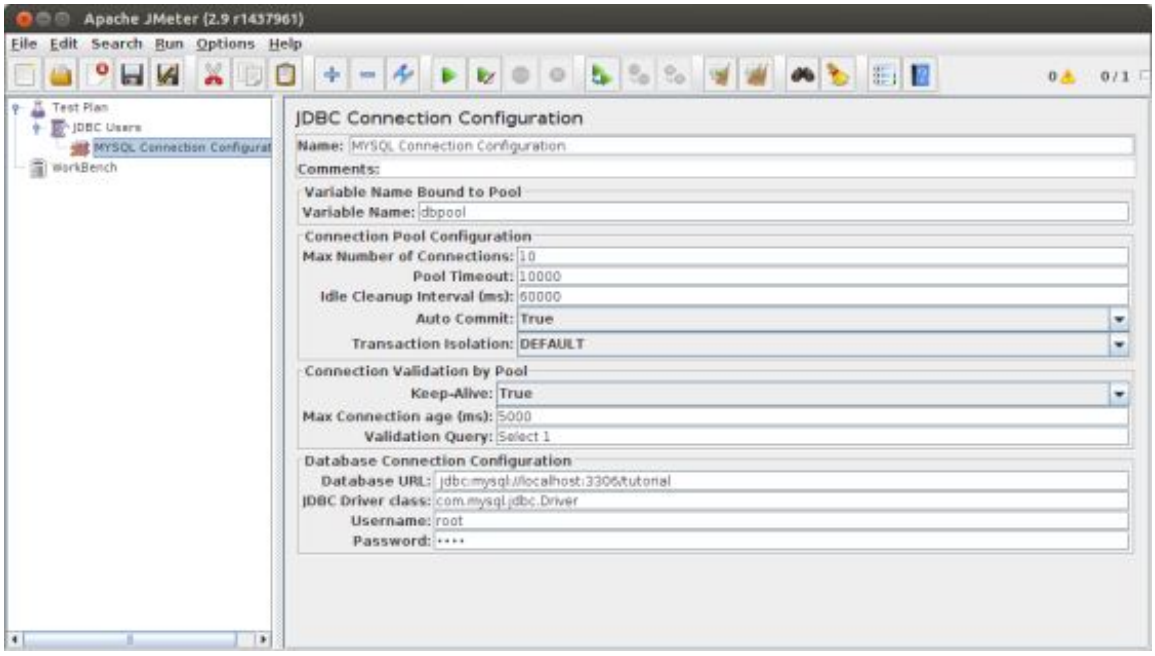
Now that we have defined our users, it is time to define the tasks that they will be performing. In this section, you will specify the JDBC requests to perform. Right click on the JDBC Users element, select **Add > Config**

Element > JDBC Connection Configuration.

Set up the following fields (we are using MySQL database called tutorial):

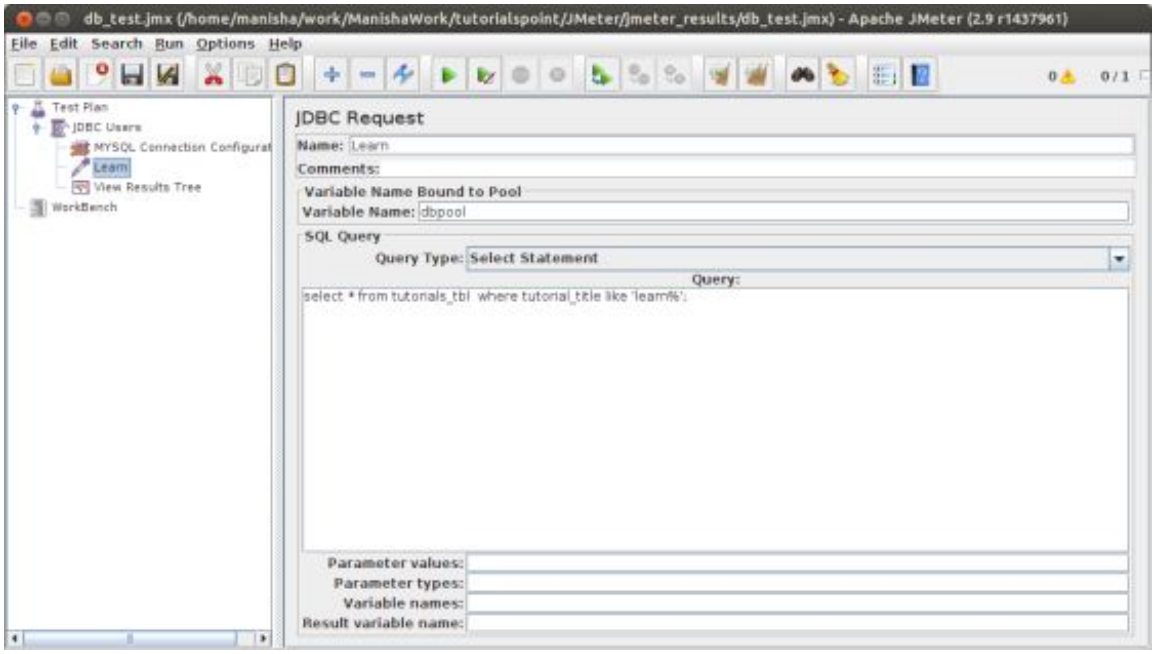
- Variable name bound to pool. This needs to uniquely identify the configuration. It is used by the JDBC Sampler to identify the configuration to be used. We have named it as *test*
- Database URL: jdbc:mysql://localhost:3306/tutorial
- JDBC Driver class: com.mysql.jdbc.Driver
- Username: root
- Password: password for root

The other fields on the screen can be left as the defaults as shown below:



Now add a JDBC Request which refers to the JDBC Configuration pool defined above. Select JDBC Users element, click your right mouse button to get the Add menu, and then select **Add > Sampler > JDBC Request**. Then, select this new element to view its Control Panel. Edit the properties as below:

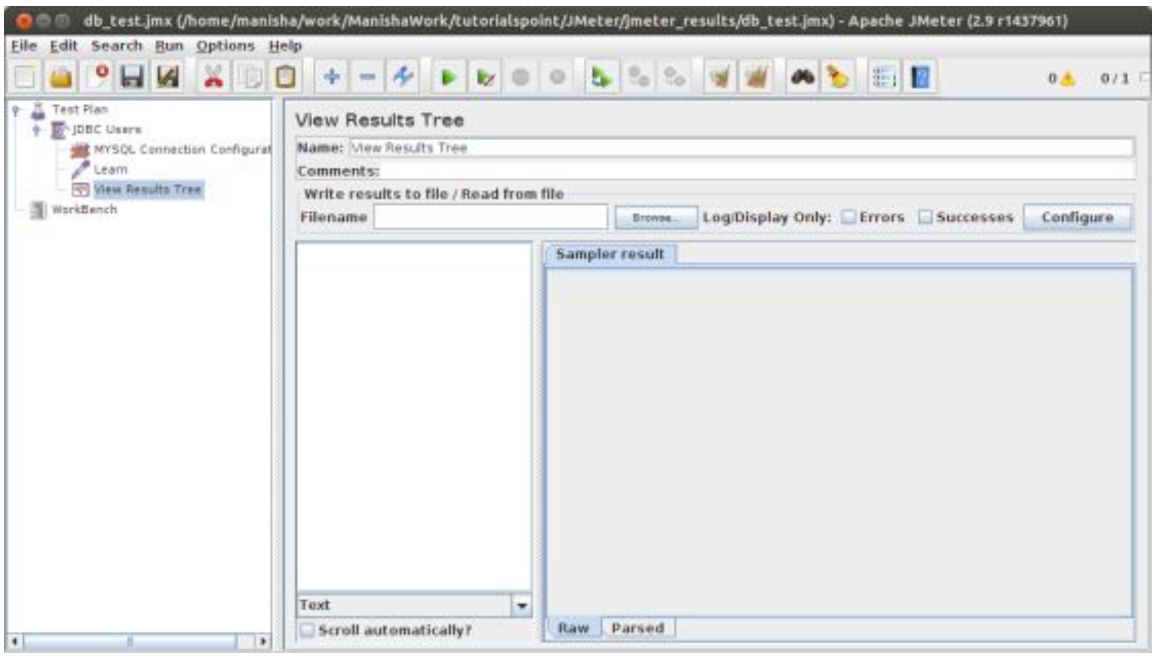
- Variable name bound to pool. This needs to uniquely identify the configuration. It is used by the JDBC Sampler to identify the configuration to be used. We have named it as *test*
- Name: Learn
- Enter the Pool Name: test (same as in the configuration element)
- Query Type: Select statement
- Enter the SQL Query String field.



Create Listener

Now add the Listener element. This element is responsible for storing all of the results of your JDBC requests in a file and presenting a visual model of the data.

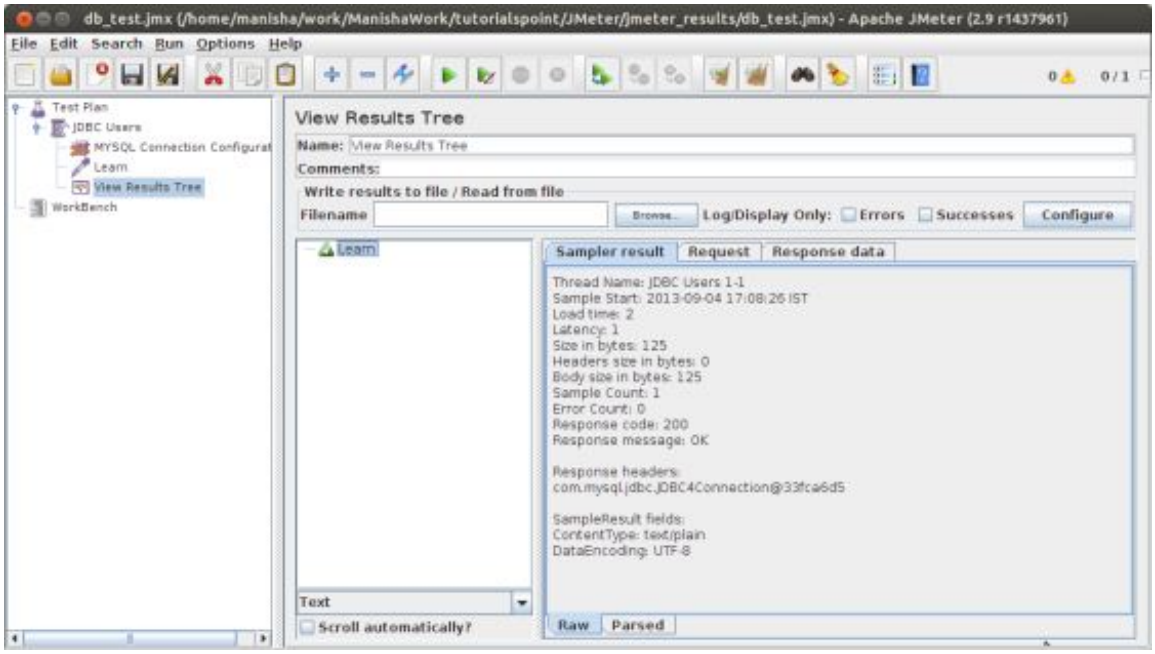
Select the JDBC Users element and add a View Results Tree listener (**Add > Listener > View Results Tree**).

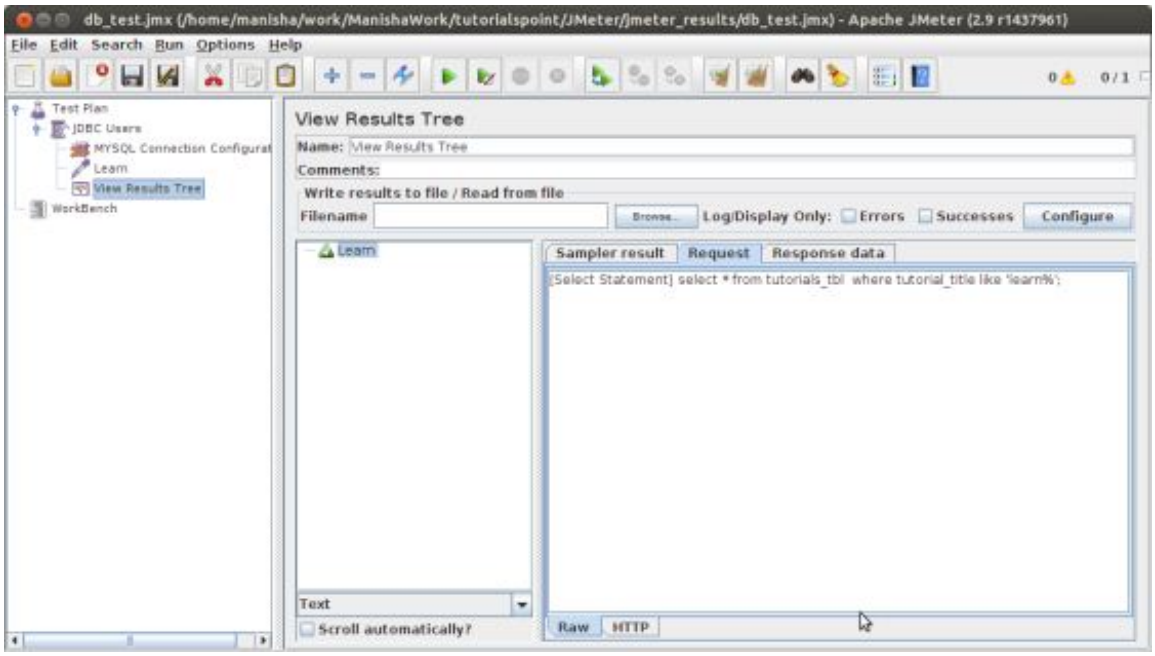


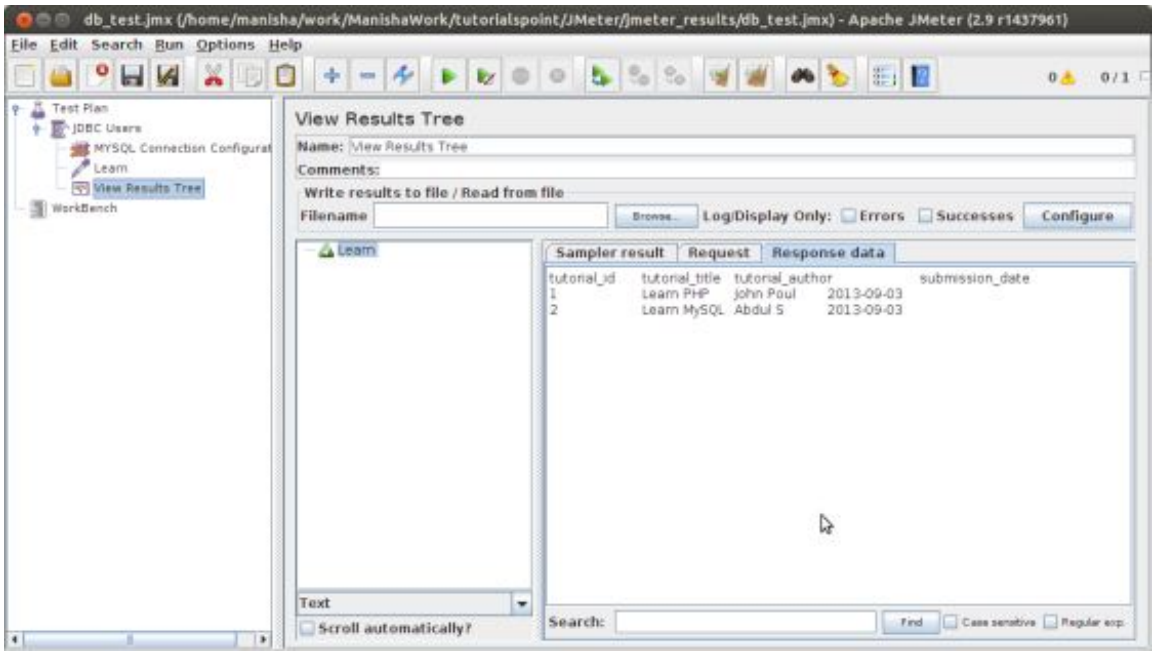
Save and Execute Test Plan

Now save the above test plan as *db_test.jmx*. Execute this test plan using **Run > Start** option.

Verify Output


db_test.jmx (/home/manisha/work/ManishaWork/tutorialspoint/JMeter/jmeter_results/db_test.jmx) - Apache JMeter (2.9 r1437961)
File Edit Search Run Options Help
Test Plan
JDBC Users
MYSQL Connection Configurat
Learn
View Results Tree
WorkBench
View Results Tree
Name: View Results Tree
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Learn
Sampler result
Request
Response data
Thread Name: JDBC Users 1-1
Sample Start: 2013-09-04 17:08:26 IST
Load time: 2
Latency: 1
Size in bytes: 125
Headers size in bytes: 0
Body size in bytes: 125
Sample Count: 1
Error Count: 0
Response code: 200
Response message: OK
Response headers:
com.mysql.jdbc.JDBC4Connection@33fca6d5
SampleResult fields:
ContentType: text/plain
DataEncoding: UTF-8
Text
Scroll automatically?
Raw
Parsed


db_test.jmx (/home/manisha/work/ManishaWork/tutorialspoint/JMeter/jmeter_results/db_test.jmx) - Apache JMeter (2.9 r1437961)
File Edit Search Run Options Help
Test Plan
JDBC Users
MYSQL Connection Configurat
Learn
View Results Tree
WorkBench
View Results Tree
Name: View Results Tree
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Learn
Sampler result
Request
Response data
[Select Statement] select * from tutorials_tbl where tutorial_title like 'learn%';
Text
Scroll automatically?
Raw
HTTP


db_test.jmx (/home/manisha/work/ManishaWork/tutorialspoint/JMeter/jmeter_results/db_test.jmx) - Apache JMeter (2.9 r1437961)
File Edit Search Run Options Help
Test Plan
JDBC Users
MYSQL Connection Configurat
Learn
View Results Tree
WorkBench
View Results Tree
Name: View Results Tree
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Learn
Sampler result
Request
Response data
tutorial_id tutorial_title tutorial_author submission_date
1 Learn PHP John Poul 2013-09-03
2 Learn MySQL Abdul S 2013-09-03
Text
Scroll automatically?
Search:
Find
Case sensitive
Regular exp.

In the last image you can see that 2 records are selected.

FTP TEST PLAN

In this chapter we will see how to test a FTP site using JMeter. Let us create a Test Plan to test the FTP site.

Rename Test Plan

Start the JMeter window by clicking on **/home/manisha/apache-jmeter-2.9/bin/jmeter.sh**. Click on the Test Plan node. Rename this Test Plan node as **TestFTPSite**.

Add Thread Group

Add one Thread Group, which is placeholder for all other elements like Samplers, Controllers, Listeners. Right click on **TestFTPSite(our Test Plan) > Add > Threads(Users) > Thread Group**. Thread Group will get added under the Test Plan (TestFTPSite) node.

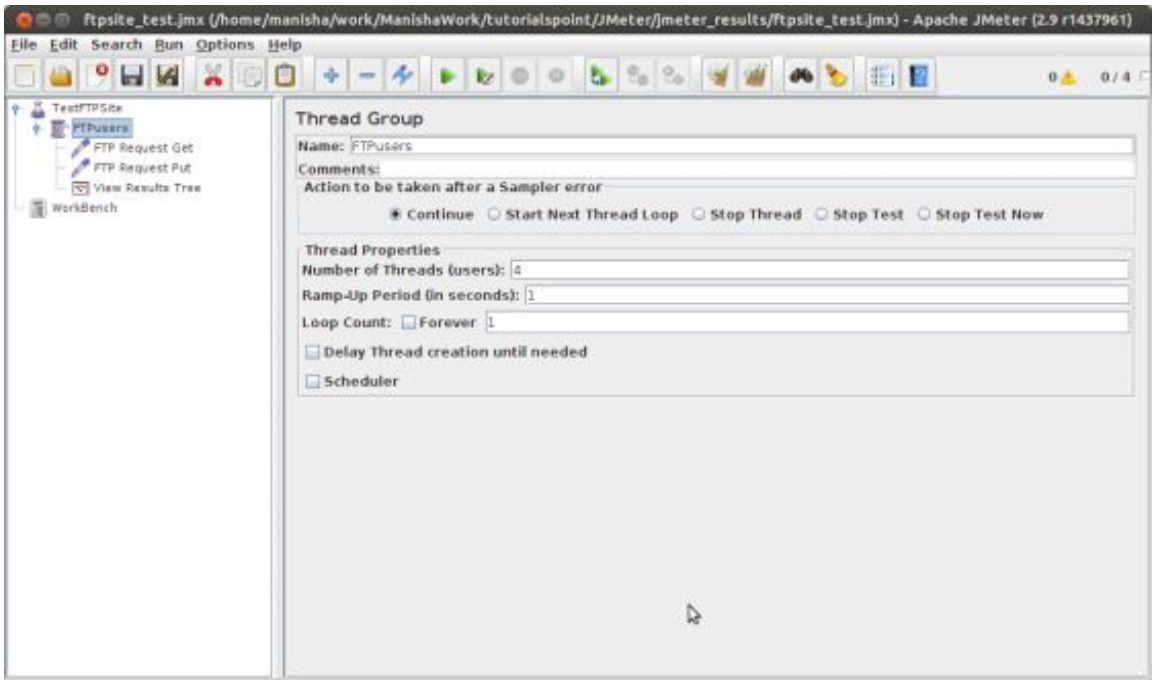
Next let us modify the default properties of the Thread Group to suit our testing. Following properties are changed:

Name: FTPusers

Number of Threads (Users): 4

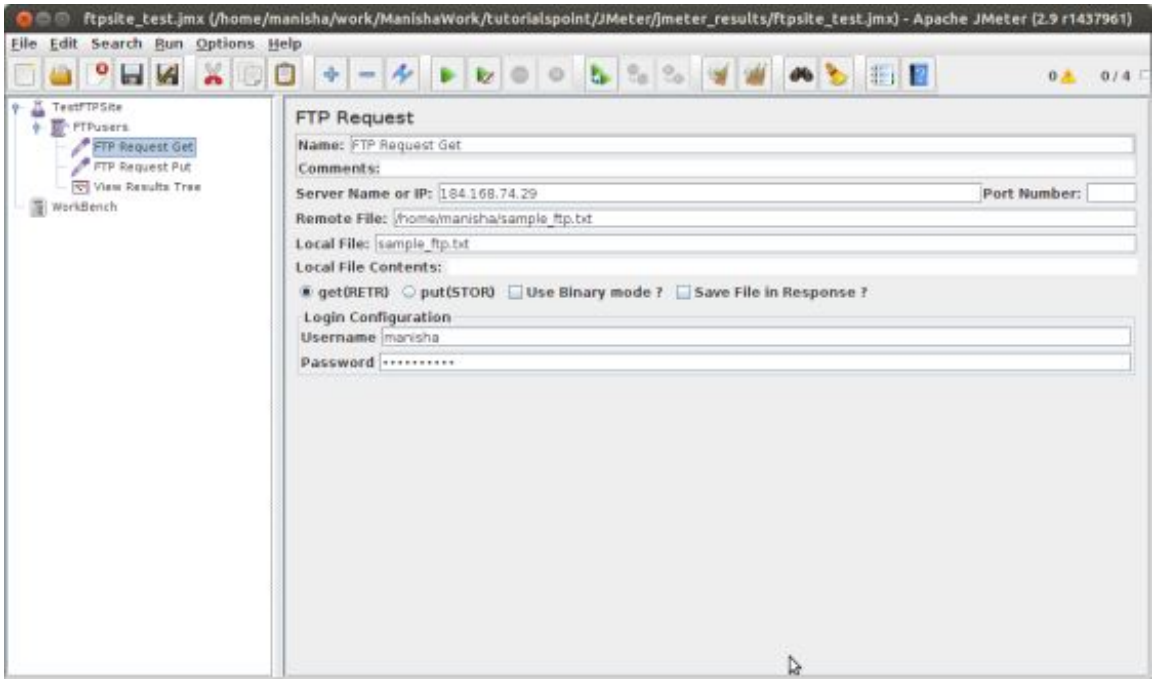
Ramp-Up Period: leave the the default value of 0 seconds.

Loop Count:1



Add Sampler- FTP Request

Now that we have defined our users, it is time to define the tasks that they will be performing. We will add FTP Request elements. We will add two FTP request elements, one which will retrieve a file and one which will put a file on the ftp site. Begin by selecting the FTPusers element. Click your right mouse button to get the Add menu, and then select **Add > Sampler > FTP Request**. Then, select the FTP Request element in the tree and edit the following properties as in the image below:



The following details are entered in the this element:

Name: FTP Request Get

Server Name or IP: 184.168.74.29

Remote File: /home/manisha/sample_ftp.txt

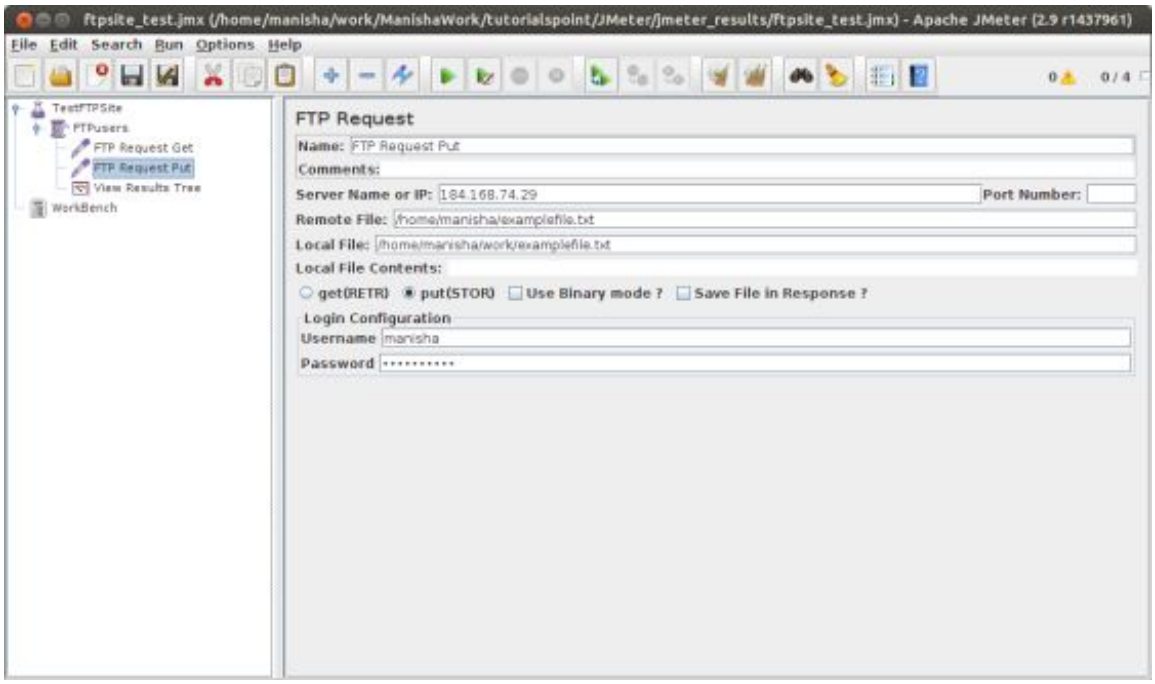
Local File:sample_ftp.txt

Select get(RETR)

Username:manisha

Password:manisha123

Now add another FTP request as above and edit the properties as in the image below:



The following details are entered in the this element:

Name: FTP Request Put

Server Name or IP: 184.168.74.29

Remote File: /home/manisha/examplefile.txt

Local File: /home/manisha/work/examplefile.txt

Select put(STOR)

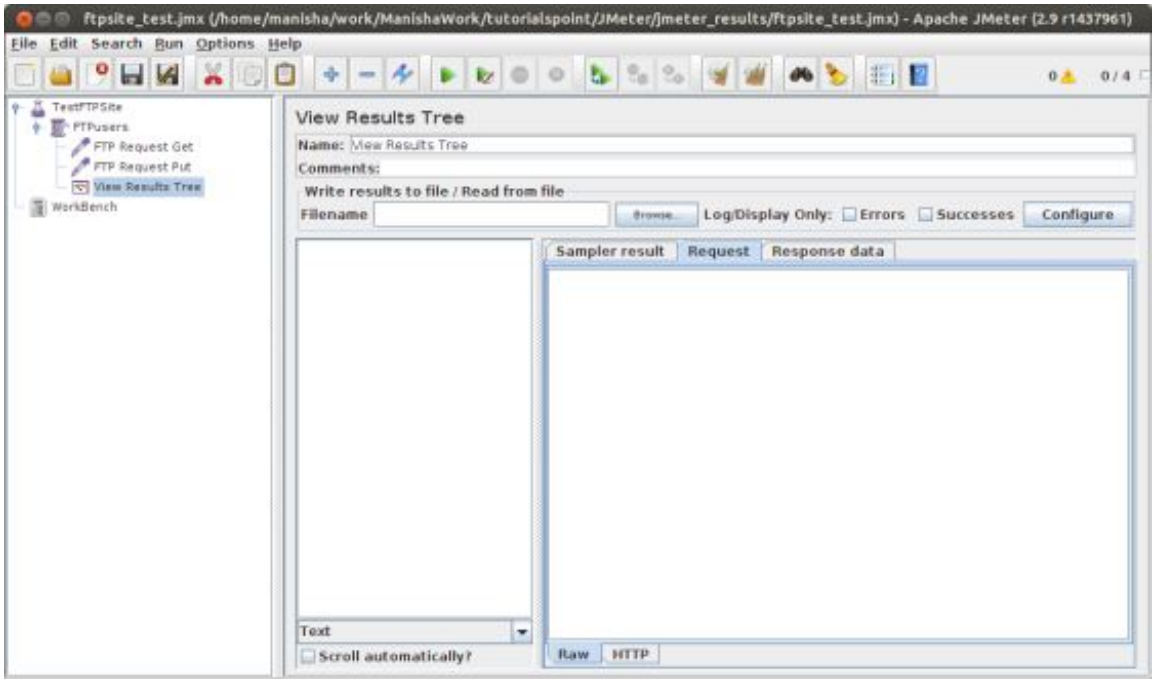
Username:manisha

Password:manisha123

Add Listener

The final element you need to add to your Test Plan is a Listener. This element is responsible for storing all of the results of your FTP requests in a file and presenting a visual model of the data.

Select the FTPusers element and add a View Results Tree listener (**Add > Listener > View Results Tree**).

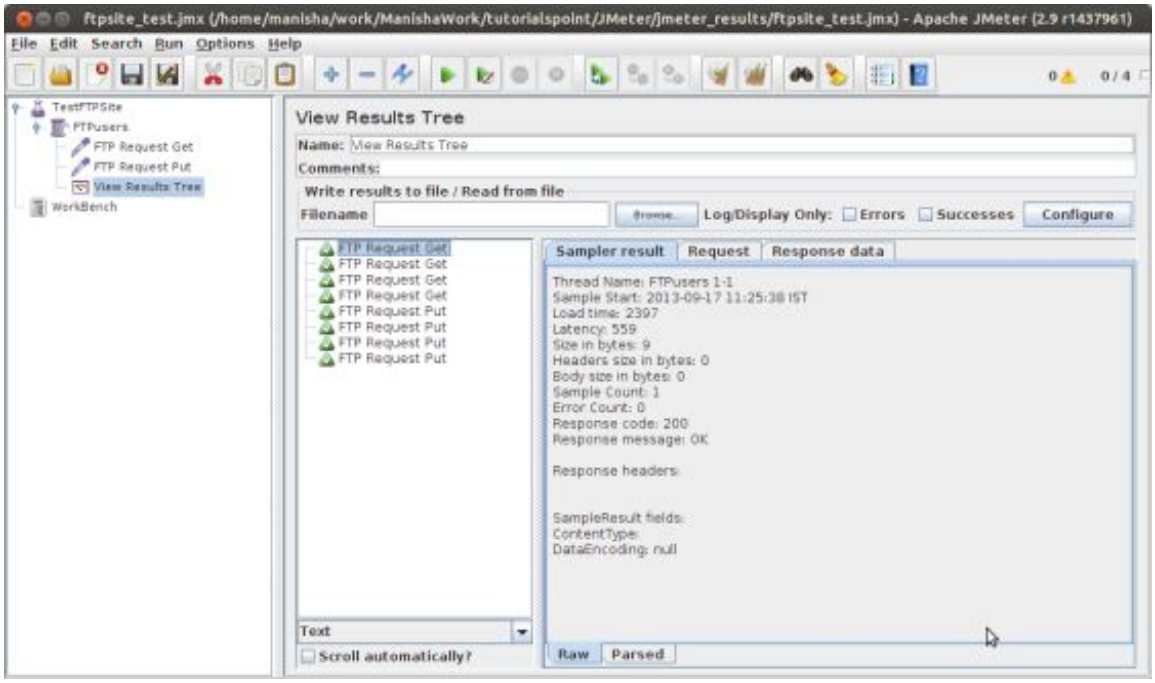


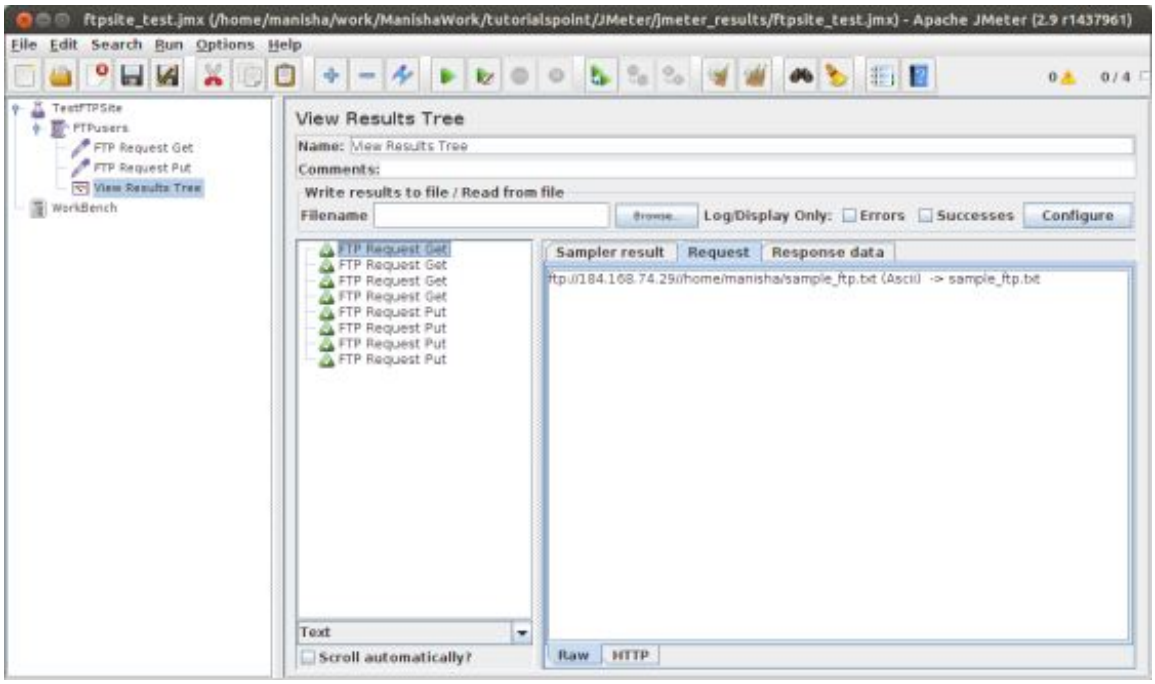
Run the Test Plan

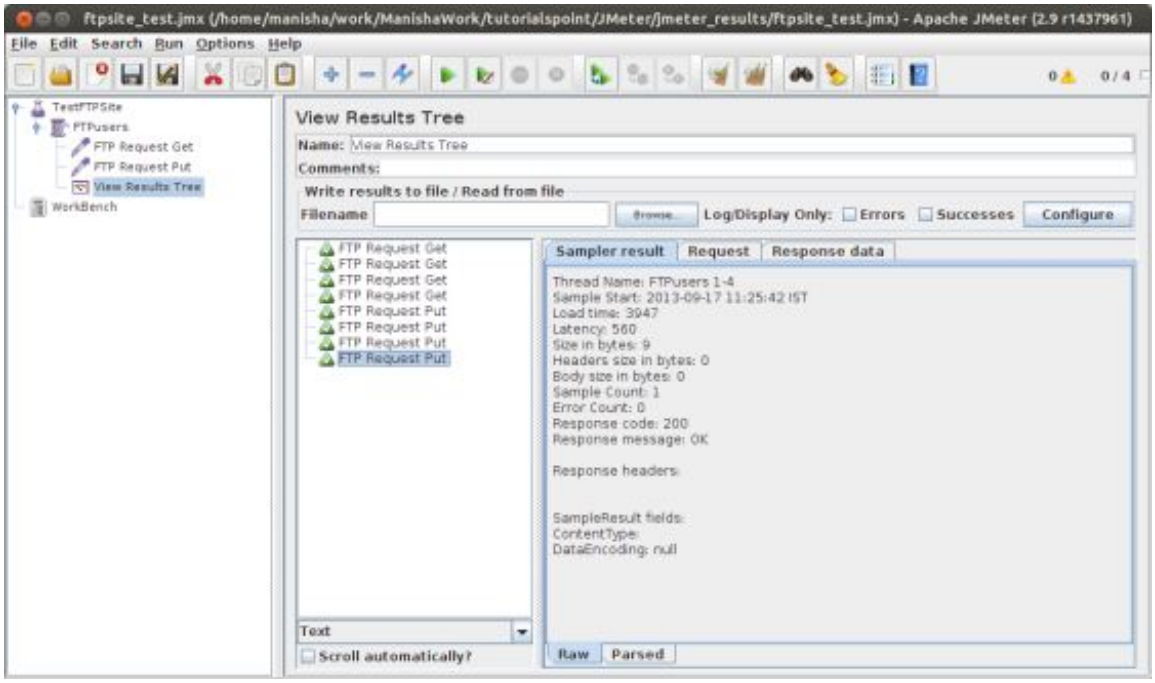
Now save the above test plan as *ftpsite_test.jmx*. Execute this test plan using **Run > Start** option.

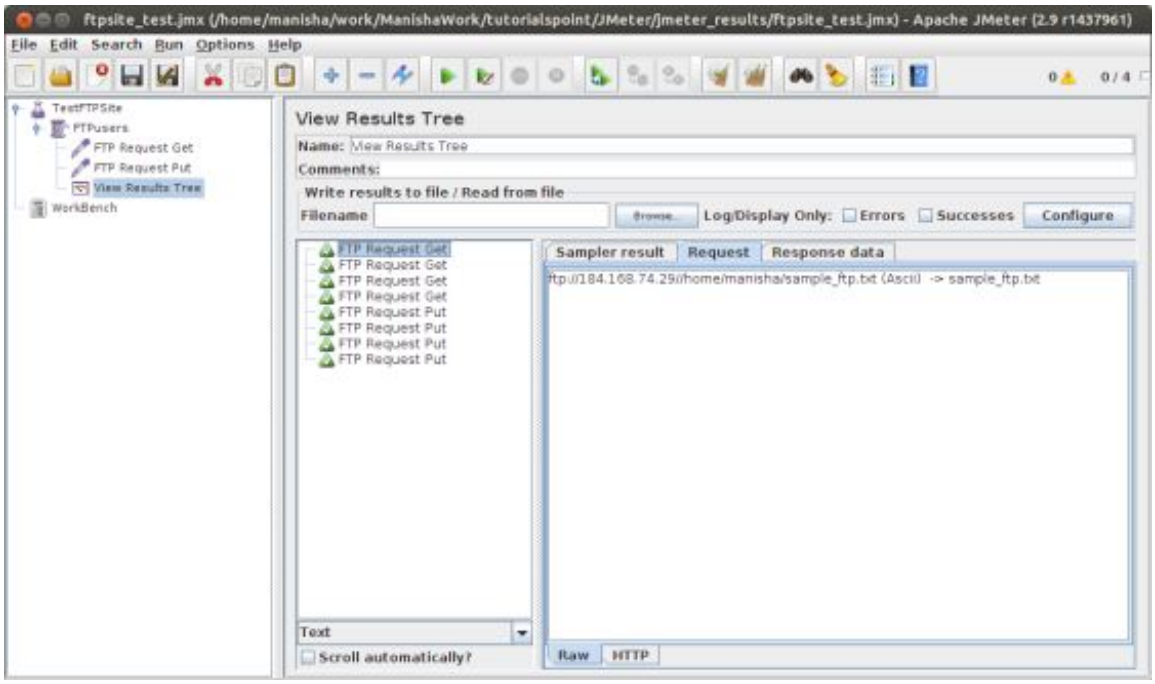
View Output

The following output can be seen in the listener.


ftpsite_test.jmx (/home/manisha/work/ManishaWork/tutorialspoint/JMeter/jmeter_results/ftpsite_test.jmx) - Apache JMeter (2.9 r1437961)
File Edit Search Run Options Help
TestFTPSite
FTPusers
FTP Request Get
FTP Request Put
View Results Tree
WorkBench
View Results Tree
Name: View Results Tree
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
FTP Request Get
FTP Request Get
FTP Request Get
FTP Request Get
FTP Request Put
FTP Request Put
FTP Request Put
FTP Request Put
Sampler result
Request
Response data
Thread Name: FTPusers 1-1
Sample Start: 2013-09-17 11:25:38 IST
Load time: 2397
Latency: 559
Size in bytes: 9
Headers size in bytes: 0
Body size in bytes: 0
Sample Count: 1
Error Count: 0
Response code: 200
Response message: OK
Response headers:
SampleResult fields:
ContentType:
DataEncoding: null
Text
Scroll automatically?
Raw
Parsed


ftpsite_test.jmx (/home/manisha/work/ManishaWork/tutorialspoint/JMeter/jmeter_results/ftpsite_test.jmx) - Apache JMeter (2.9 r1437961)
File Edit Search Run Options Help
TestFTPSite
FTPusers
FTP Request Get
FTP Request Put
View Results Tree
WorkBench
View Results Tree
Name: View Results Tree
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
FTP Request Get
FTP Request Get
FTP Request Get
FTP Request Get
FTP Request Put
FTP Request Put
FTP Request Put
FTP Request Put
Sampler result
Request
Response data
ftp://184.168.74.29//home/manisha/sample_ftp.txt (Ascii) -> sample_ftp.txt
Text
Scroll automatically?
Raw
HTTP





You can see that four requests are made for each FTP request. We see that the test is successful. The retrieved file for GET request is stored in the bin folder. In our case it would be **/home/manisha/apache-jmeter-2.9/bin/**. For PUT request the file is uploaded at the path **/home/manisha/**.

WEBSERVICE TEST PLAN

In this chapter, we will learn how to create a Test Plan to test a WebService. For our test purpose, we have created a simple webservice project and deployed it on the Tomcat server locally.

Create Webservice Project

To create a webservice project we have used Eclipse IDE. First write the Service Endpoint Interface **HelloWorld** under the package **com.tutorialspoint.ws**. The contents of the HelloWorld.java are as below:

```
package com.tutorialspoint.ws;

import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
```

```
import javax.jws.soap.SOAPBinding.Style;

//Service Endpoint Interface
@WebService
@SOAPBinding(style = Style.RPC)
public interface HelloWorld{

@WebMethod String getHelloWorldMessage(String string);

}
```

This service has a method **getHelloWorldMessage** which takes String paramater.

Next create the implementation class **HelloWorldImpl.java** under the package **com.tutorialspoint.ws**.

```
package com.tutorialspoint.ws;

import javax.jws.WebService;

@WebService(endpointInterface="com.tutorialspoint.ws.HelloWorld")
public class HelloWorldImpl  implements HelloWorld  {
 @Override
 public String getHelloWorldMessage(String myName){
  return("Hello "+myName+" to JAX WS world");
 }

}
```

As a next step let's publish this web service locally by creating the Endpoint publisher and expose the service on the server.

The publish method takes two parameters:

- Endpoint URL String.
- Implementor object, in this case the HelloWorld implementation class, which is exposed as a Web service at the endpoint identified by the URL mentioned in the parameter above.

The contents of HelloWorldPublisher.java is a follows :

```
package com.tutorialspoint.endpoint;

import javax.xml.ws.Endpoint;

import com.tutorialspoint.ws.HelloWorldImpl;

public class HelloWorldPublisher {

 public static void main(String[] args){
  Endpoint.publish("http://localhost:9000/ws/hello", new HelloWorldImpl());
 }
}
```

Next modify the web.xml as below:

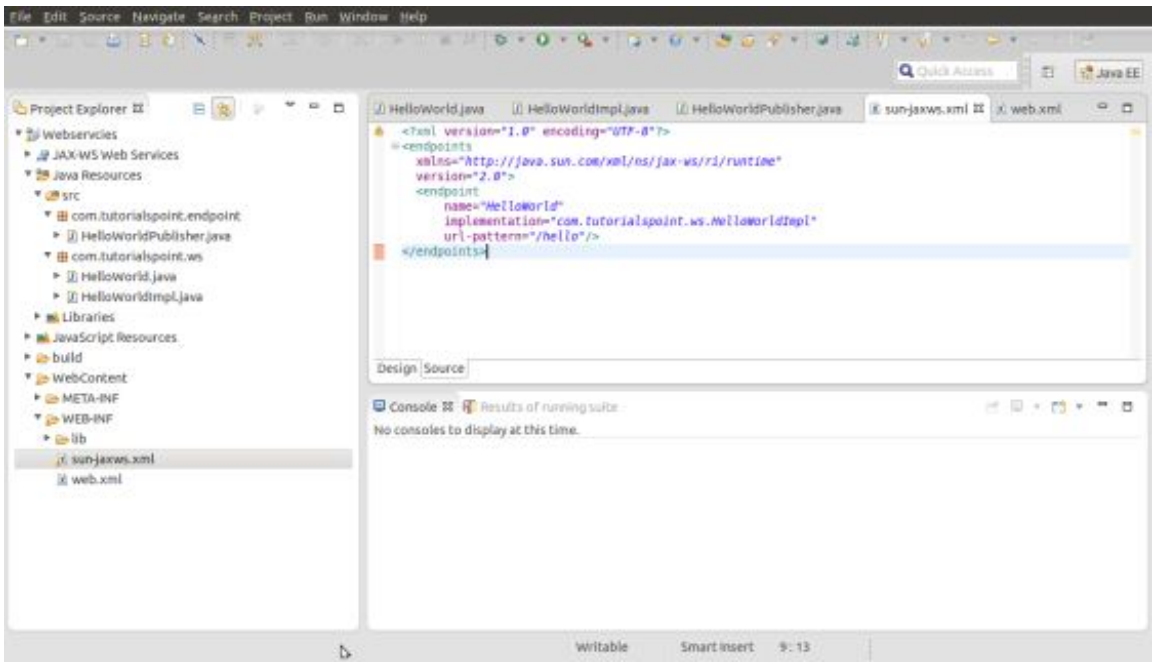
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems,
Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">
<web-app>
    <listener>
        <listener-class>
                com.sun.xml.ws.transport.http.servlet.WSServletContextListener
        </listener-class>
    </listener>
    <servlet>
        <servlet-name>hello</servlet-name>
        <servlet-class>
         com.sun.xml.ws.transport.http.servlet.WSServlet
```

```
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>hello</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>120</session-timeout>
    </session-config>
</web-app>
```

To deploy this application as webservice we would need another configuration file **sun-jaxws.xml**, the contents of this file are as below:

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints
  xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime"
  version="2.0">
  <endpoint
      name="HelloWorld"
      implementation="com.tutorialspoint.ws.HelloWorldImpl"
      url-pattern="/hello"/>
</endpoints>
```

Now that all the files are ready the directory structure would look like as in the image below:



Now create a WAR file of this application. Choose the **project > right click > Export > WAR file**. Save this as **hello.war** file under the **webapps** folder of Tomcat server. Now start the Tomcat server. Once the server is started, you should be able to access the webservice with the following URL:

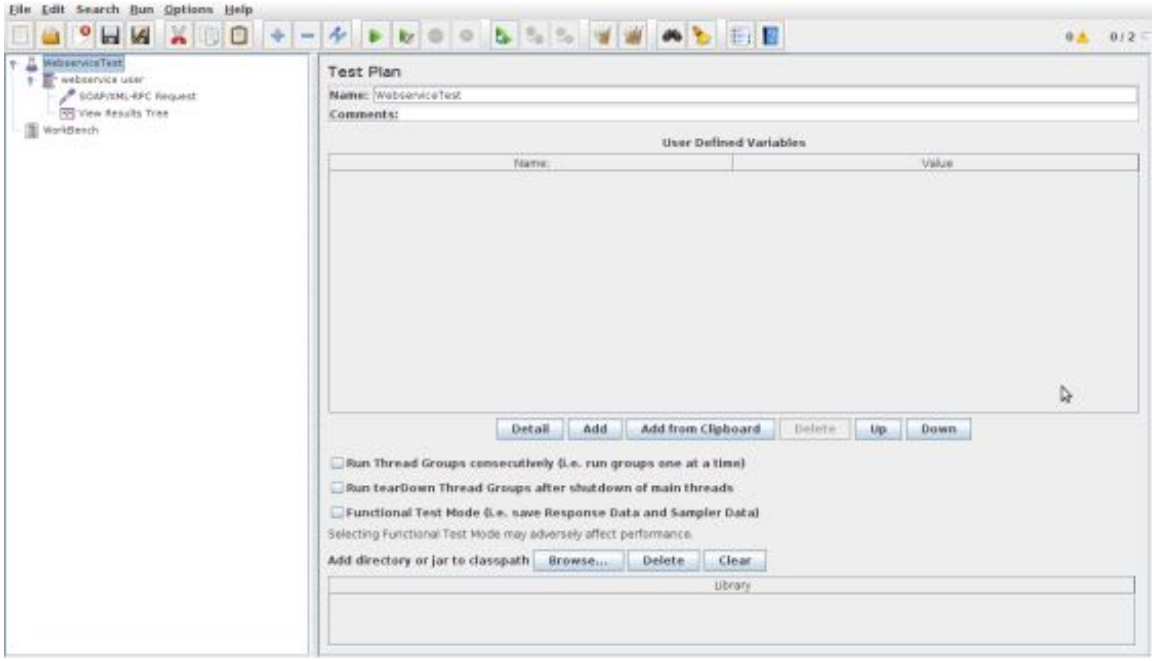
http://localhost:8080/hello/hello

Create JMeter Test plan

Now let's create a test plan to test the above webservice.

Rename Test Plan

Start the JMeter window by clicking on **/home/manisha/apache-jmeter-2.9/bin/jmeter.sh**. Click on the Test Plan node. Rename this Test Plan node as **WebserviceTest**.



Add Thread Group

Add one Thread Group, which is placeholder for all other elements like Samplers, Controllers, Listeners. Right click on **WebserviceTest(our Test Plan) > Add > Threads(Users) > Thread Group**. Thread Group will get added under the Test Plan (WebserviceTest) node.

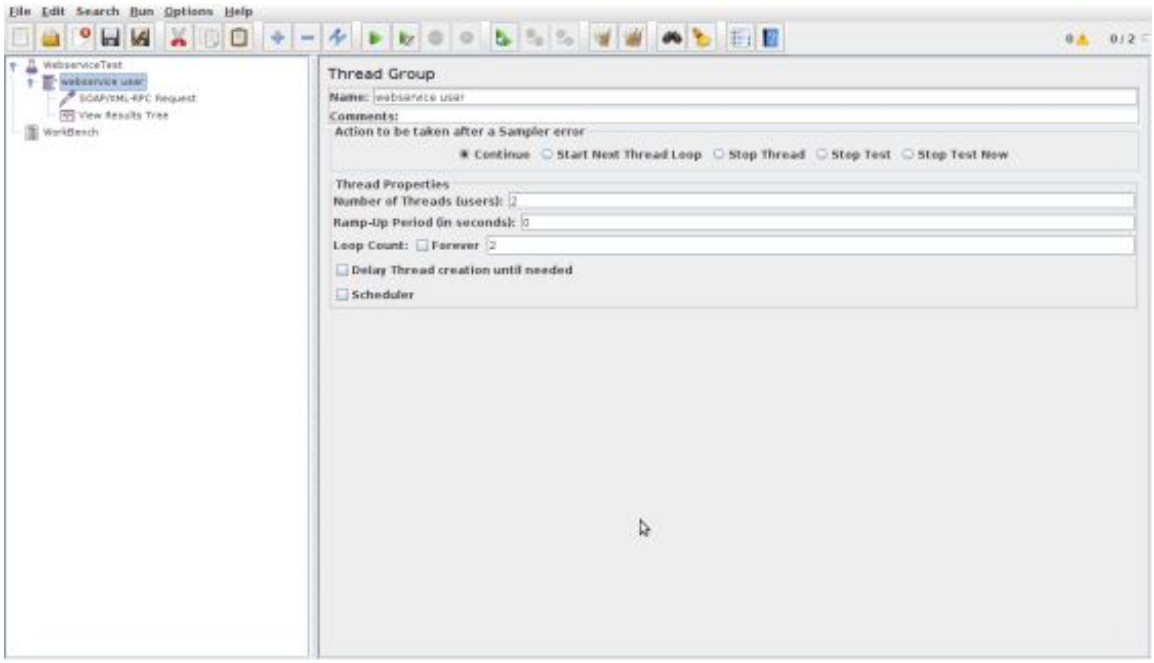
Next let us modify the default properties of the Thread Group to suit our testing. Following properties are changed:

Name: webservice user

Number of Threads (Users): 2

Ramp-Up Period: leave the the default value of 0 seconds.

Loop Count:2



Add Sampler-SOAP/XML -RPC Request

Now that we have defined our users, it is time to define the tasks that they will be performing. We will add SOAP/XML-RPC Request element. Click your right mouse button to get the Add menu, and then select **Add > Sampler > SOAP/XML-RPC Request**. Then, select the SOAP/XML-RPC Request element in the tree and edit the following properties as in the image below:

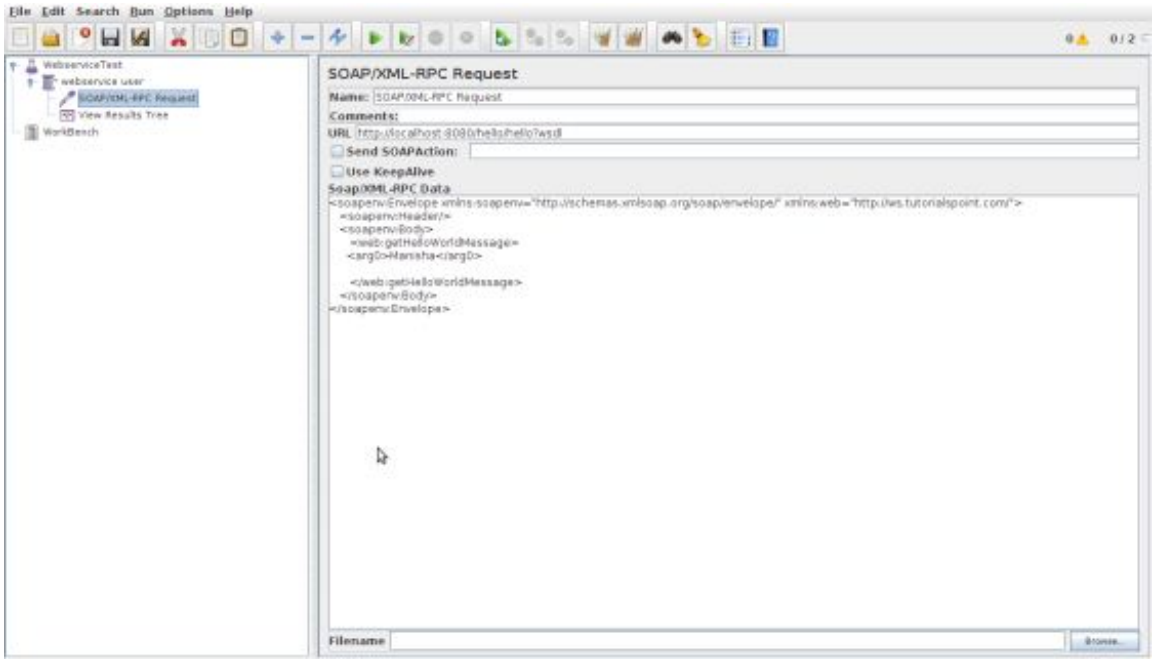
The following details are entered in the this element:

Name: SOAP/XML-RPC Request

URL: http://localhost:8080/hello/hello?wsdl

Soap/XML-RPC Data: Enter the below contents

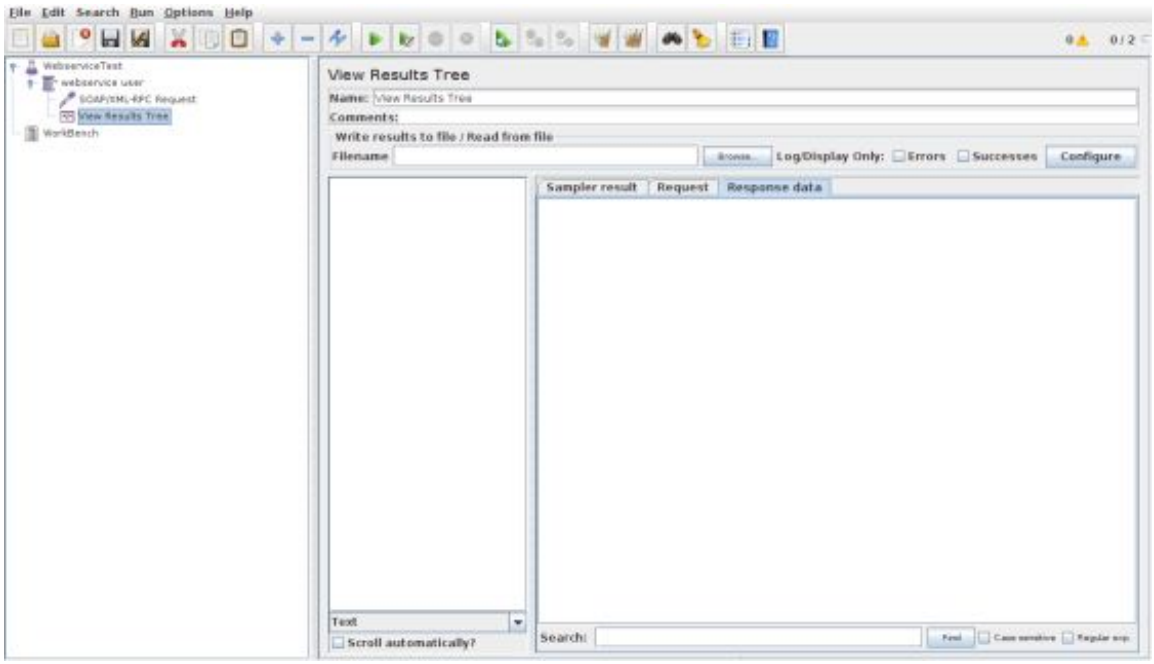
```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:web="http://ws.tutorialspoint.com/">
   <soapenv:Header/>
   <soapenv:Body>
      <web:getHelloWorldMessage>
     <arg0>Manisha</arg0>

      </web:getHelloWorldMessage>
   </soapenv:Body>
</soapenv:Envelope>
```



Add Listener

The final element you need to add to your Test Plan is a Listener. This element is responsible for storing all of the results of your HTTP requests in a file and presenting a visual model of the data.

Select the webservice user element and add a View Results Tree listener (**Add > Listener > View Results Tree**).

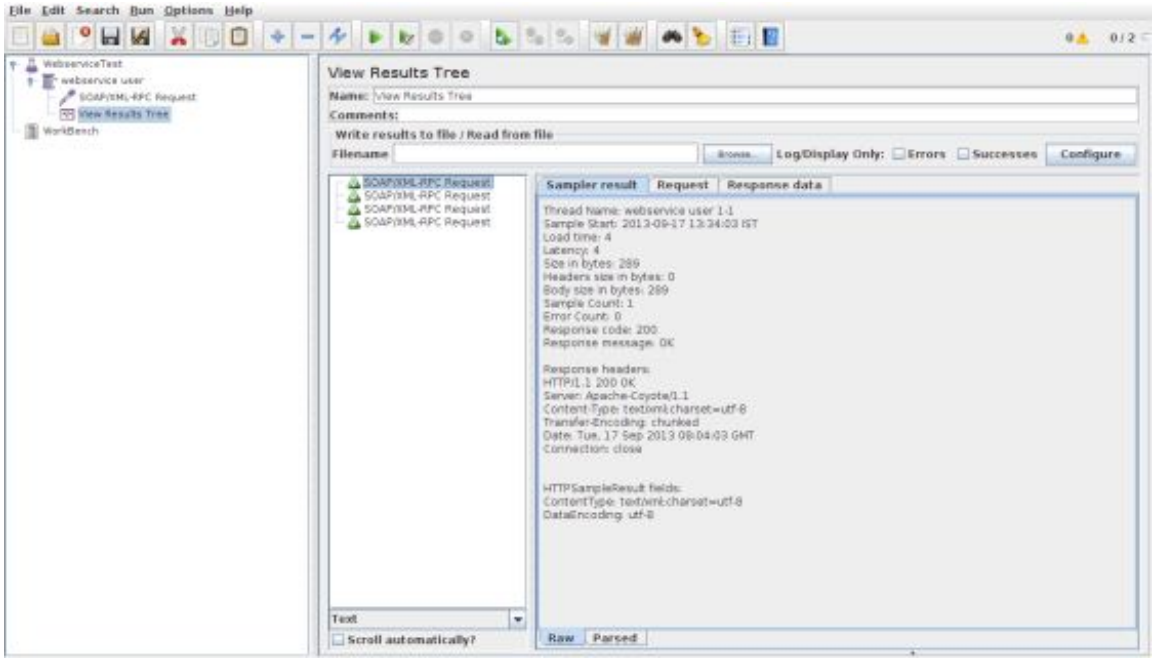


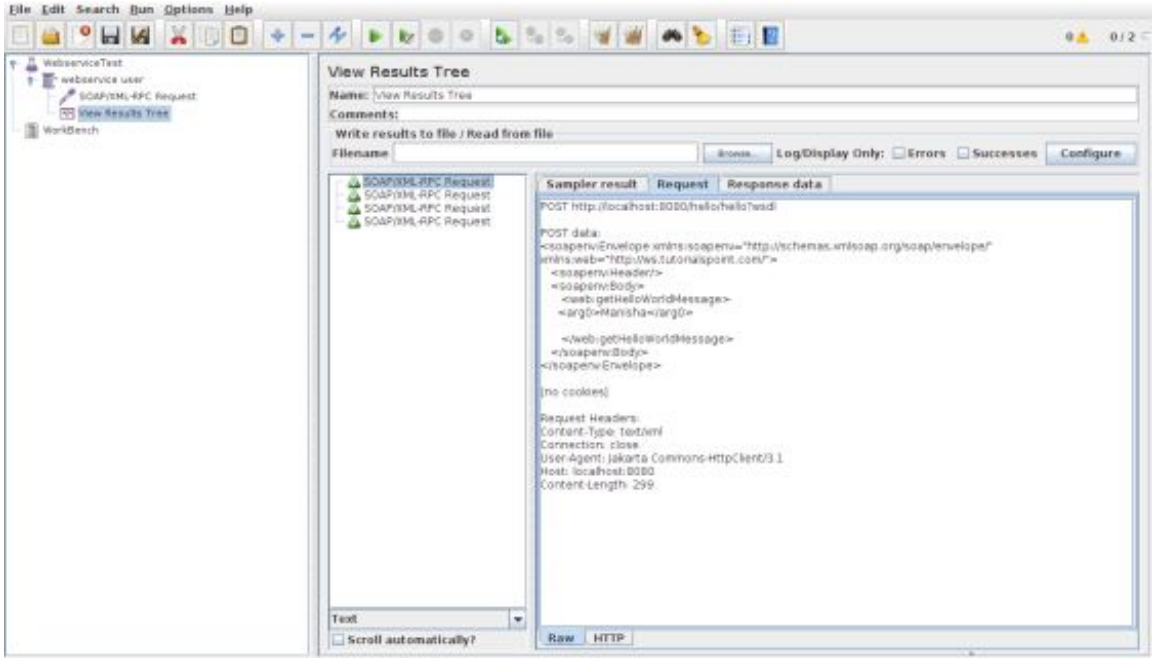
Run the Test Plan

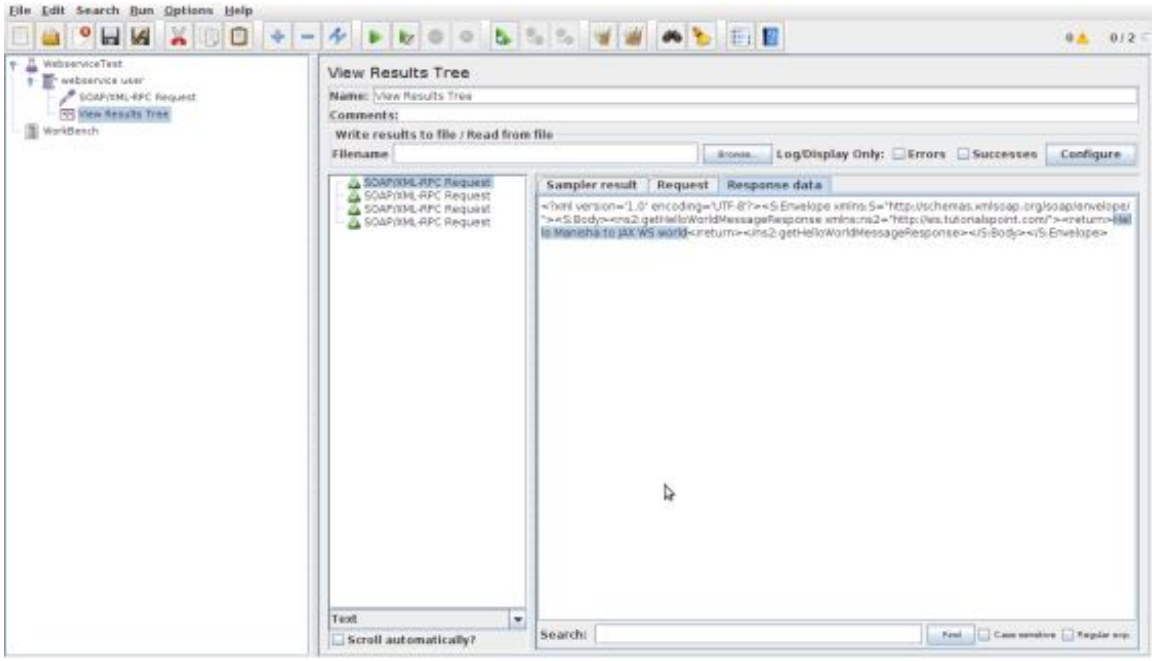
Now save the above test plan as *test_webservice.jmx*. Execute this test plan using **Run > Start** option.

View Output

The following output can be seen in the listener.







In the last image you can see the response message "Hello Manisha to JAX WS world".

JMS TEST PLAN

In this chapter we will learn how to write a simple test plan to test JMS (Java Messaging Service). Those who are not aware of JMS please make yourself familiar with JMS before reading this chapter. JMS supports two types of messaging:

- **Point-to-Point messaging** : Queue messaging is generally used for transactions where the sender expects a response. Messaging systems are quite different from normal HTTP requests. In HTTP, a single user sends a request and gets a response.
- **Topic messaging** : Topic messages are commonly known as pub/sub messaging. Topic messaging is generally used in cases where a message is published by a producer and consumed by multiple subscribers.

Let us see a test example for each of these. Pre-requisites for testing JMS are:

- We will be using Apache ActiveMQ for our example. Though there are other JMS servers like IBM WebSphere MQ (formerly MQSeries), Tibco, etc. Download the binaries from the Apache ActiveMQ

website.

- Unzip the archive, go to the decompressed directory and run the following command from the command console to start the ActiveMQ server:

```
.\bin\activemq  start
```

You can verify if the ActiveMQ server has started by visiting the admin interface at the following address http://localhost:8161/admin/. If it asks for authentication enter the userid and password as *admin*. The screen would be similar as below:

- Now copy the activemq-all-x.x.x.jar (XXX depending on the version) from the ActiveMQ unzipped directory to **/home/manisha/apache-jmeter-2.9/lib**.

With the above setup let's build the test plan for:

- JMS Point-to-Point Test Plan
- JMS Topic Test Plan

MONITOR TEST PLAN

In this chapter we will discuss about how to create Test plan using JMeter, to monitor webservers. Uses of monitor tests are:

- Monitors are useful for a stress testing and system management.
- Used with stress testing, the monitor provides additional information about server performance.
- Monitors makes it easier to see the relationship between server performance and response time on the client side.
- As a system administration tool, the monitor provides an easy way to monitor multiple servers from one console.

We would need Tomcat 5 or above version for monitoring. For our test purpose we will monitor Tomcat 7.0.42 server. You can test any servlet container that supports JMX (Java Management Extension). Let us write a test case to monitor the Tomcat server. But before that let us first set up our tomcat server.

Setup Tomcat Server

We'll start with "open" the Tomcat service status. To do this,edit the configuration file for users **<TOMCAT_HOME>/conf/tomcat-users.xml**. This file contains a tomcat-users section (commented) like this:

```
<tomcat-users>
```

```
<!--
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
  <user username="role1" password="tomcat" roles="role1"/>
-->
</tomcat-users>
```

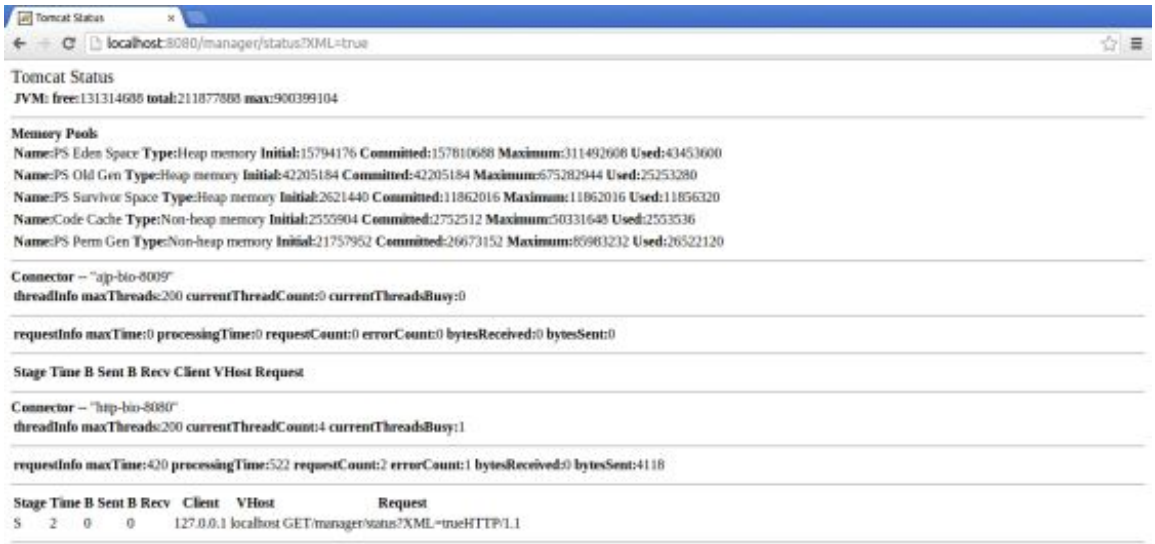
We need to change this section to add the admin roles, manager, manager-gui and assign the user "admin". The revised file is as follows:

```
<tomcat-users>

  <role rolename="manager-gui"/>
  <role rolename="manager-script"/>
  <role rolename="manager-jmx"/>
  <role rolename="manager-status"/>
  <user username="admin" password="admin" roles="manager-gui,manager-script,manager-
jmx,manager-status"/>

</tomcat-users>
```

Now start the tomcat server <TOMCAT_HOME>/bin/startup.sh for linux and <TOMCAT_HOME>/bin/startup.bat for windows. Once started, check that the Tomcat supervision works by entering the below link in your browser:

```
http://localhost:8080/manager/status?XML=true
```

An authentication window appears in the browser, enter the tomcat log in and password associated (in our case it is admin). Then, the browser shows the execution status of Tomcat as below:



From the above image we can note few things:

- In the URL, note that *XML = true* (note the case sensitivity) allows a clean display of the supervisory Tomcat necessary for the JMeter functioning.
- Also note that there are default two connectors. The AJP connector used in general coupled with the mod_jk Apache HTTPD front module and the HTTP connector which is commonly used connector for direct access to Tomcat via port 8080.

Write JMeter Test Plan

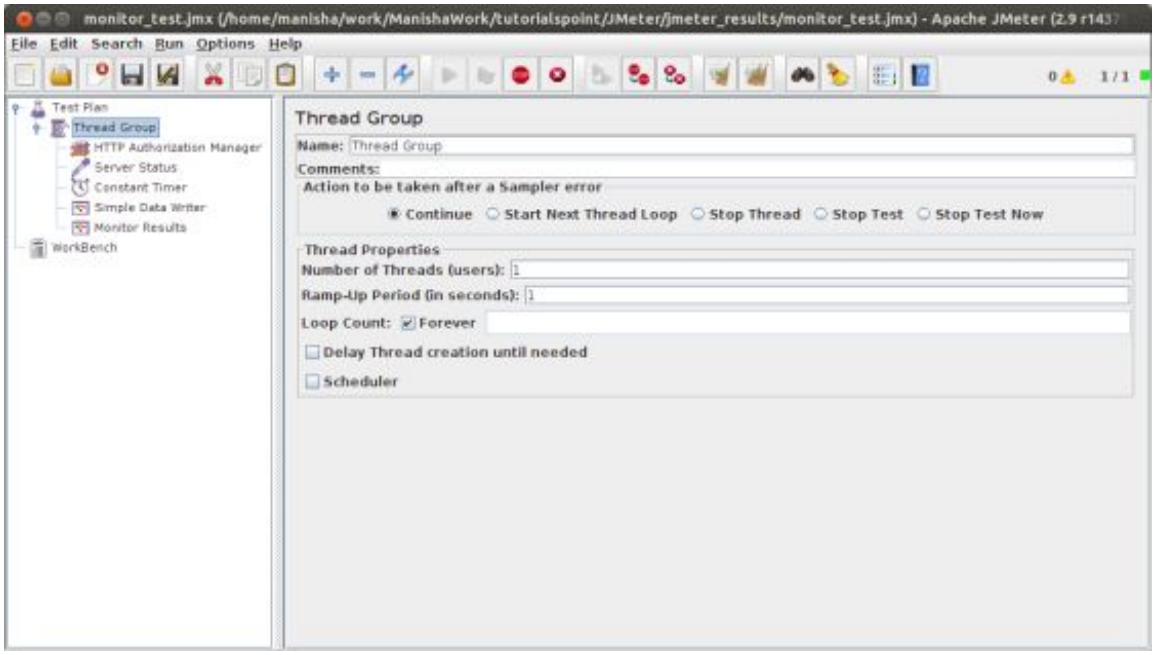
Let us monitor the Tomcat server by writing a test plan as below:

Rename Test Plan

Start the JMeter window by clicking on **/home/manisha/apache-jmeter-2.9/bin/jmeter.sh**. Click on the Test Plan node. Add a thread group as explained in the next step.

Add Thread Group

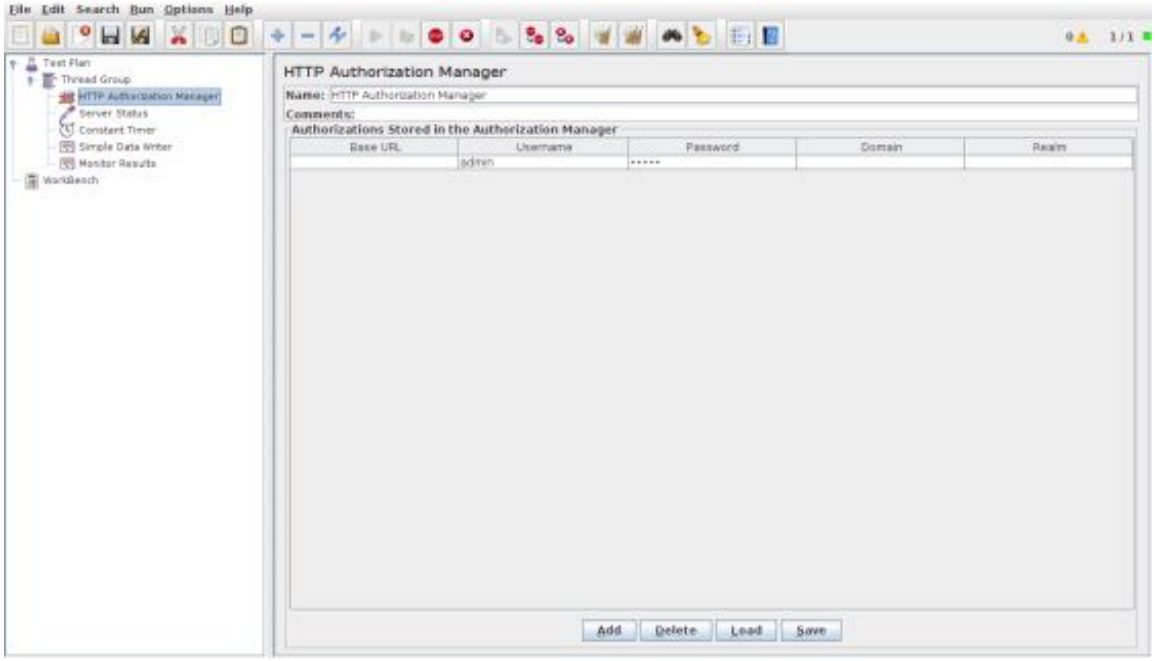
Add one Thread Group. Right click on **Test Plan > Add > Threads(Users) > Thread Group**. Thread Group will get added under the Test Plan node. Change the loop count to forever (or some large number) so that enough samples are generated.



HTTP Authorization Manager

Add the HTTP Authorization Manager to the Thread Group element **Add > Config element > HTTP Authorization Manager**. This element manages authentication requested by the browser to see the Tomcat server status. Select the **HTTP Authorization Manager** and edit the following details:

- **Username** : admin (depending on the configuration in tomcat-users.xml file)
- **Password** : admin (depending on the configuration in the tomcat-users.xml file)
- The other fields are left empty.

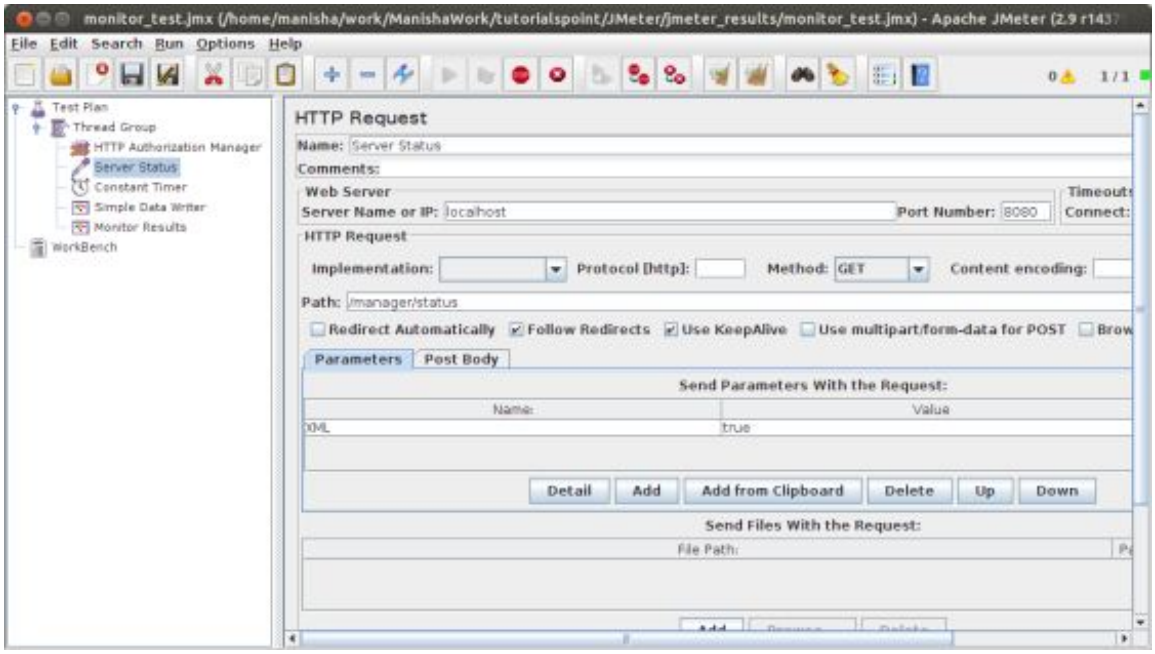


Add Sampler-HTTP Request

Now that we have defined our users, it is time to define the tasks that they will be performing. We will add HTTP Request element. Click your right mouse button to get the Add menu, and then select **Add > Sampler > HTTP Request**. Then, select the HTTP Request element in the tree and edit the following properties as in the image below:

The following details are entered in the this element:

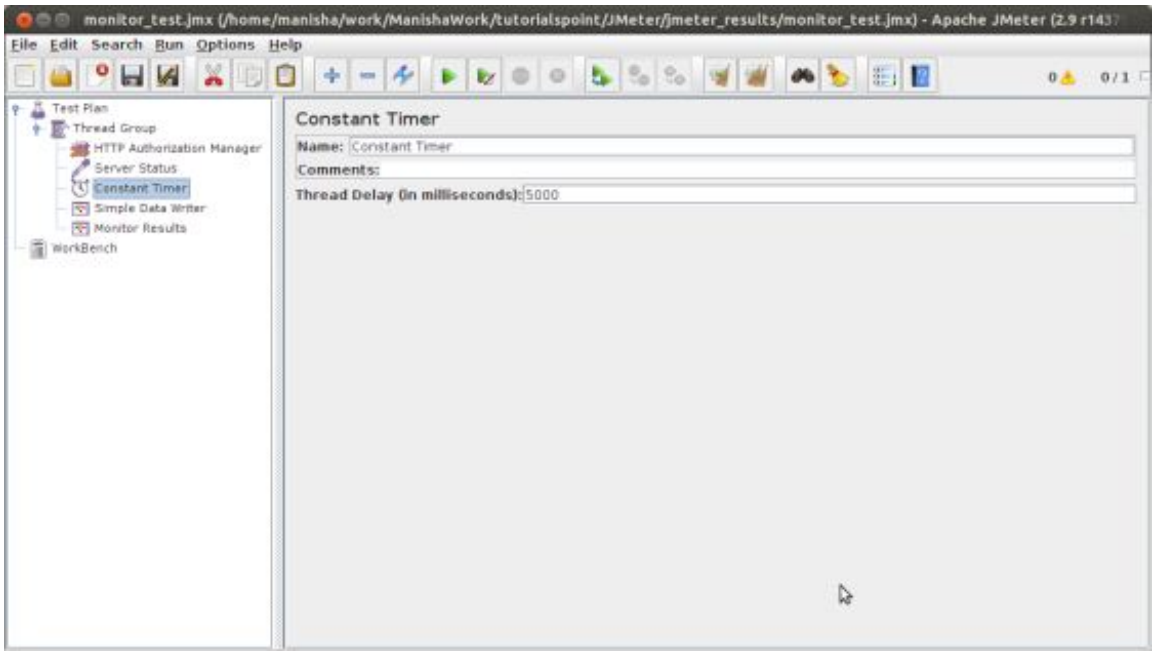
- **Name** : Server Status
- **Server Name or IP** : localhost
- **Port** : 8080
- **Path** : /manager/status
- **Parameters** : Add a request parameter named "XML" in uppercase. Give it a value of "true" in lowercase.
- **Optional Tasks** : Check "Use as Monitor" at the bottom of the sampler.



Add Constant Timer

To request the status of the server periodically, add an element Constant Timer which will allow a time interval between each request. Add a timer to this thread group **Add > Timer > Constant Timer**.

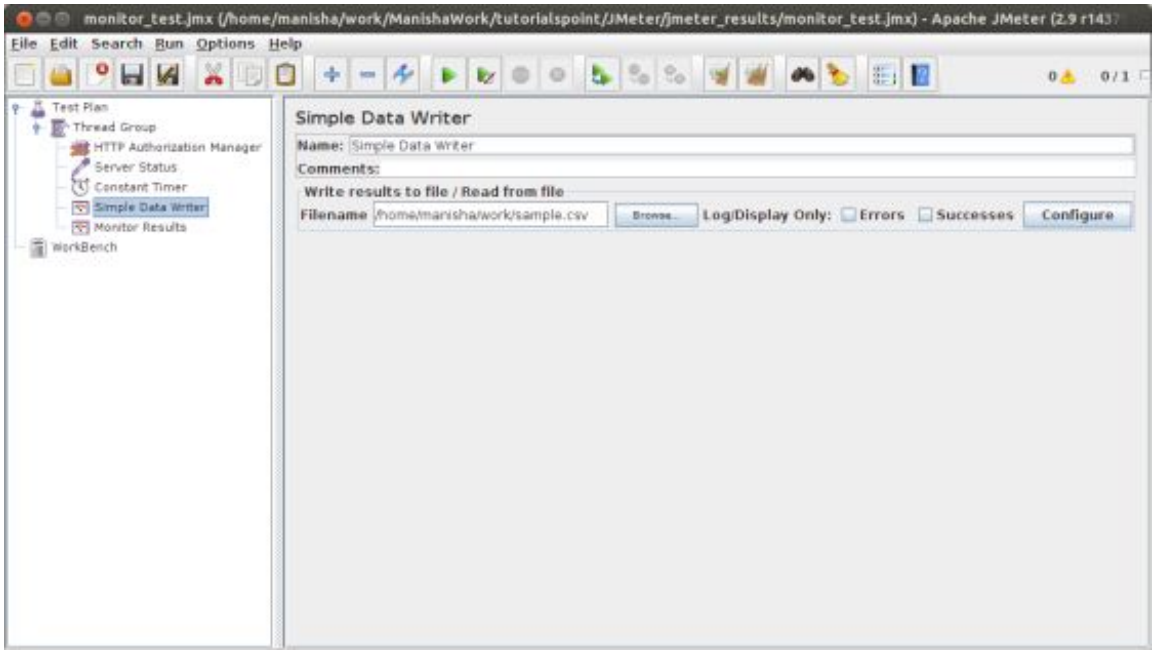
Enter 5000 milliseconds in the *Thread Delay* box. In general, using intervals shorter than 5 seconds will add stress to your server. Find out what is an acceptable interval before you deploy the monitor in your production environment.



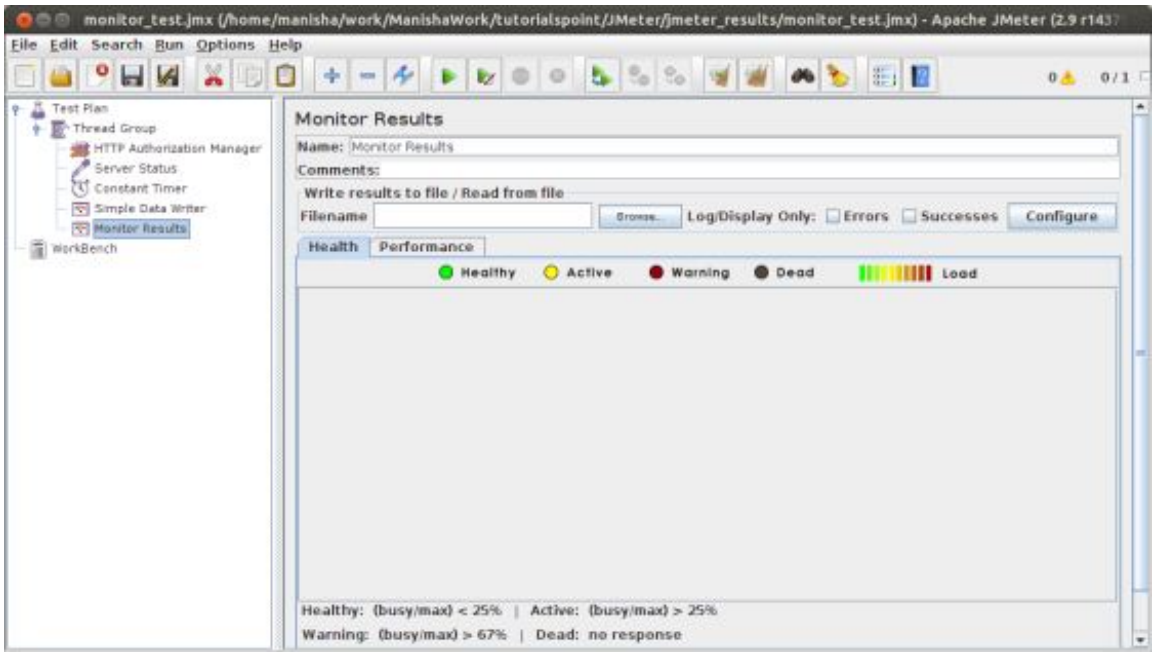
Add Listener

The final element you need to add to your Test Plan is a Listener. We will two types of listeners. One that stores results in a file and second that shows the graphical view of the results.

Select the thread group element and add a Simple Data Writer listener **Add > Listener > Simple Data Writer**. Next, specify a directory and filename of the output file (in our case it is /home/manisha/work/sample.csv)



Let us add another listener, by selecting the test plan element **Add > Listener > Monitor Results**.

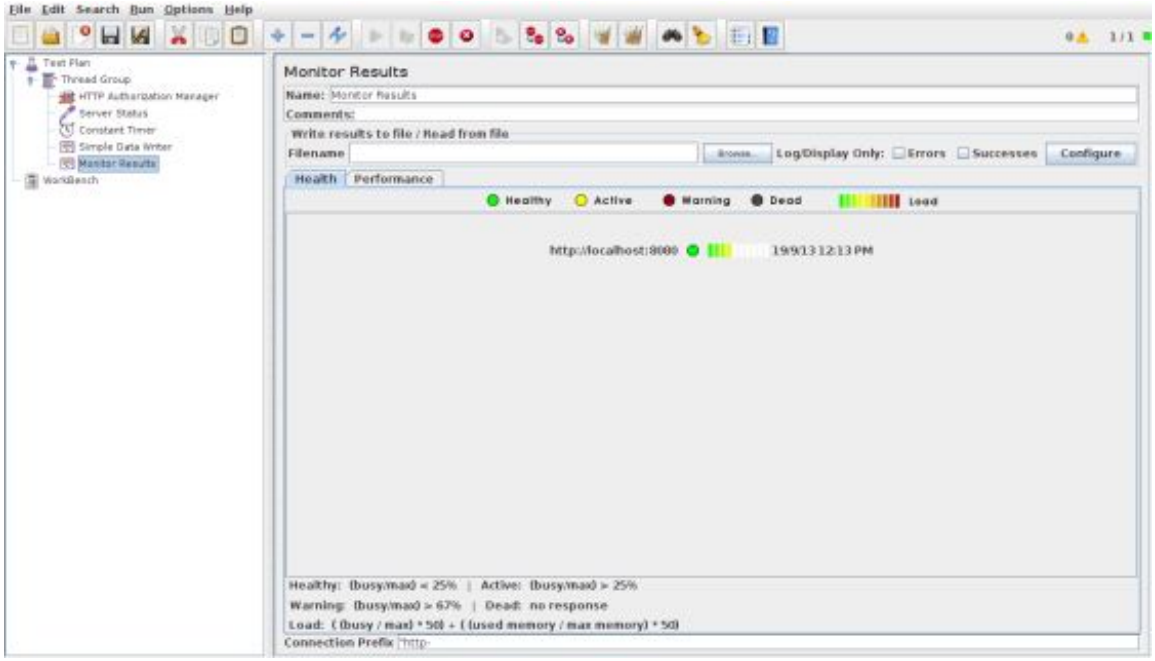


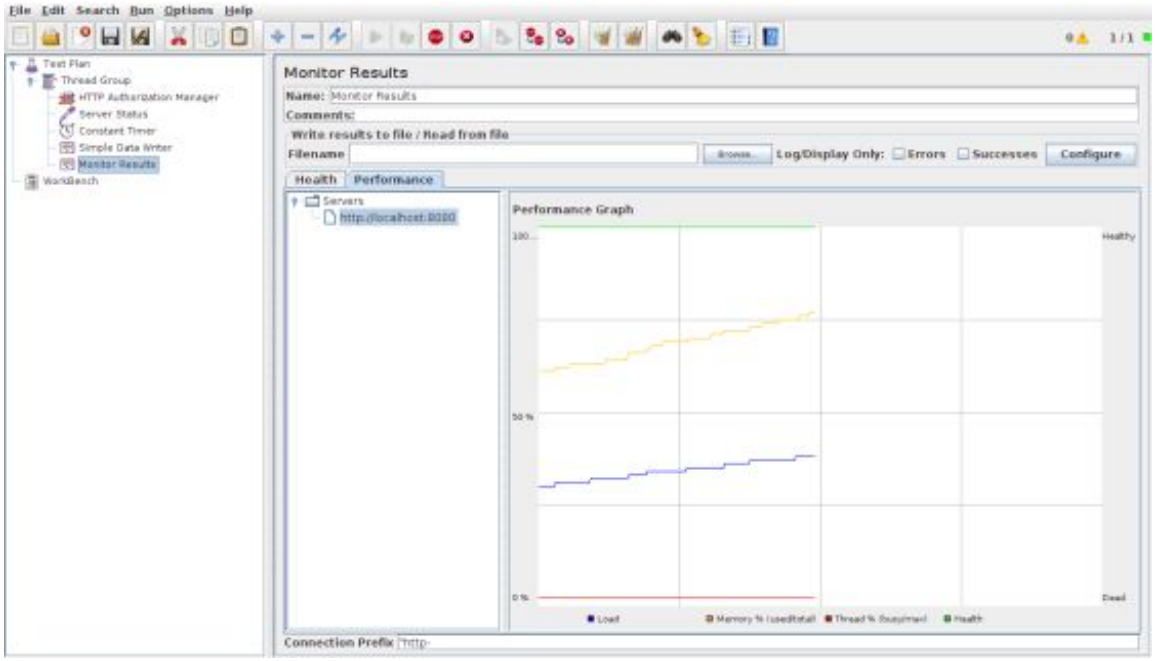
Run the Test Plan

Now save the above test plan as *monitor_test.jmx*. Execute this test plan using **Run > Start** option.

View Output

Results will be saved in /home/manisha/work/sample.csv file. You can also see a graphical result in the Monitor result listener as in the image below.





Note the graph has captions on both sides of the graph. On the left is percent and the right is dead/healthy. If the memory line spikes up and down rapidly, it could indicate memory thrashing. In those situations, it is a good idea to profile the application with Borland OptimizeIt or JProbe. What you want to see is a regular pattern for load, memory and threads. Any erratic behavior usually indicates poor performance or a bug of some sort.

LISTENERS

Listeners provide access to the information JMeter gathers about the test cases while JMeter runs. The results or information gathered by listeners can be shown in the form of:

- tree
- tables
- graphs
- log file

All listeners write the same raw data to the output file when one is specified.

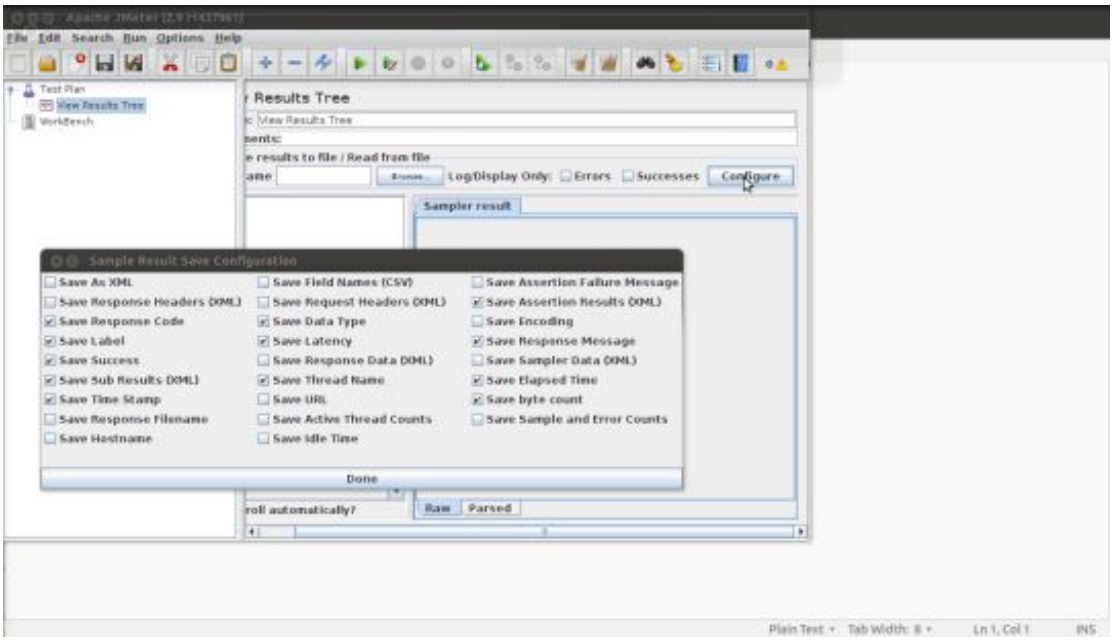
Default Configuration

The default items to be saved can be defined:

- in the **jmeter.properties** (or user.properties) file. This file is present in the **/bin** folder of JMeter.To change the default format, find the following line in jmeter.properties:

```
jmeter.save.saveservice.output_format=
```

- or by using the Config popup as shown in the image below:



JMeter creates results of a test run as JTL (JMeter Text Logs). These are normally called JTL files, as that is the default extension - but any extension can be used.

If multiple tests are run using the same output file name, then JMeter automatically appends new data to the end of the file.

The listener can record results to a file but not to the UI. It is meant to provide an efficient means of recording data by eliminating GUI overhead.

When running in:

- **GUI mode**: use the listener Simple Data Writer
- **non-GUI mode**: the -l flag can be used to create a data file.

Listeners can use a lot of memory if there are a lot of samples. To minimise the amount of memory needed, use the Simple Data Writer, and use the CSV format.

CSV Log format

The CSV log format depends on which data items are selected in the configuration. Only the specified data items are recorded in the file. The order of appearance of columns is fixed, and is as follows:

Field	Description	Value Example
timeStamp	in milliseconds since 1/1/1970	1354223881017
elapsed	in milliseconds	1858
label	sampler label	HTTP Request

responseCode	e.g. 200, 404	200
responseMessage	e.g. OK	OK
threadName		Thread Group 1-1
dataType	e.g. text	text
success	true or false	true
failureMessage	if any	
bytes	number of bytes in the sample	34908
grpThreads	number of active threads in this thread group	1
allThreads	total number of active threads in all groups	1
URL		http://tutorialspoint.com
Filename	if Save Response to File was used	
latency	time to first response	132
encoding		utf-8
SampleCount	number of samples (1, unless multiple samples are aggregated)	1
ErrorCount	number of errors (0 or 1, unless multiple samples are aggregated)	0
Hostname	where the sample was generated	LaptopManisha
IdleTime	number of milliseconds of 'Idle' time (normally 0)	
Variables	if specified	

Saving response data

The response data can be saved in the XML log file if required. But it has constraints, when the file is large and also images cannot be included. In such cases use the Post-Processor Save_Responses_to_a_file . This generates a new file for each sample, and saves the file name with the sample. The file name can then be included in the sample log output. The data will be retrieved from the file if necessary when the sample log file is reloaded.

Loading (reading) response data

To view an existing results file, you can use the File "Browse..." button to select a file. If necessary, just create a dummy testplan with the appropriate Listener in it.

Saving Listener GUI data

JMeter is capable of saving any listener as a PNG file. To do so, select the listener in the left panel,
Edit > Save As Image.

A file dialog will appear. Enter the desired name and save the listener.

FUNCTIONS

JMeter Functions and User Variables

- JMeter functions are special values that can populate fields of any Sampler or other element in a test tree. A function call looks like this:

```
${__functionName(var1,var2,var3)}
```

- *_functionName* matches the name of a function. For example **${__threadNum}**.
- If a function parameter contains a comma, then be sure to escape this with "\" as shown below.

```
${__time(EEE\, d MMM yyyy)}
```

- Variables are referenced as:

```
${VARIABLE}
```

List of Functions

Following table lists a group of functions loosely grouped into types:

Type of function	Name	Comment
Information	threadNum	get thread number
Information	samplerName	get the sampler name (label)
Information	machineIP	get the local machine IP address
Information	machineName	get the local machine name
Information	time	return current time in various formats
Information	log	log (or display) a message (and return the value)
Information	logn	log (or display) a message (empty return value)
Input	StringFromFile	read a line from a file
Input	FileToString	read an entire file
Input	CSVRead	read from CSV delimited file
Input	XPath	Use an XPath expression to read from a file
Calculation	counter	generate an incrementing number
Calculation	intSum	add int numbers
Calculation	longSum	add long numbers
Calculation	Random	generate a random number
Calculation	RandomString	generate a random string
Calculation	UUID	generate a random type 4 UUID
Scripting	BeanShell	run a BeanShell script
Scripting	javaScript	process JavaScript (Mozilla Rhino)
Scripting	jexl, jexl2	evaluate a Commons Jexl expression
Properties	property	read a property
Properties	P	read a property (shorthand method)

Properties	setProperty	set a JMeter property
Variables	split	Split a string into variables
Variables	V	evaluate a variable name
Variables	eval	evaluate a variable expression
Variables	evalVar	evaluate an expression stored in a variable
String	regexFunction	parse previous response using a regular expression
String	escapeOroRegexpChars	quote meta chars used by ORO regular expression
String	char	generate Unicode char values from a list of numbers
String	unescape	Process strings containing Java escapes (e.g. \n & \t)
String	unescapeHtml	Decode HTML-encoded strings
String	escapeHtml	Encode strings using HTML encoding
String	TestPlanName	Return name of current test plan

- There are two kinds of functions:
    - user-defined static values (or variables)
    - built-in functions
- User-defined static values allow the user to define variables to be replaced with their static value when a test tree is compiled and submitted to be run.
- Note that variables cannot currently be nested; i.e **${Var${N}}** does not work.
- The __V (variable) function (versions after 2.2) can be used to do this: ${__V(Var${N})}.
- This type of replacement is possible without functions, but was less convenient and less intuitive

Where can functions and variables be used?

- Functions and variables can be written into any field of any test component.
- The following functions should work OK on the test plan:
    - intSum
    - longSum
    - machineName
    - BeanShell
    - javaScript
    - jexl
    - random
    - time
    - property functions
    - log functions

> *Functions which are used on the Test Plan have some restrictions. JMeter thread variables will have not been fully set up when the functions are processed, so variable names passed as parameters will not be set up, and variable references will not work, so split() and regex() and the variable evaluation functions won't work. The threadNum() function won't work (and does not make sense at test plan level).*

Referencing variables and functions

- Referencing a variable in a test element is done by bracketing the variable name with '${' and '}'.
- Functions are referenced in the same manner, but by convention, the names of functions begin with "__" to avoid conflict with user value names.
- Some functions take arguments to configure them, and these go in parentheses, comma-delimited. If the function takes no arguments, the parentheses can be omitted. For eg:

```
${__BeanShell(vars.put("name"\,"value"))}
```

- Alternatively, you can define your script as a variable, e.g. on the Test Plan:

```
SCRIPT     vars.put("name","value")
```

- The script can then be referenced as follows:

```
${__BeanShell(${SCRIPT})}
```

The Function Helper Dialog

The Function Helper Dialog is available from JMeter's **Options** tab.

- Using the Function Helper, you can select a function from the pull down, and assign values for its arguments. The left column in the table provides a brief description of the argument, and the right column is where you write in the value for that argument. Different functions take different arguments.
- Once you have done this, click the “Generate" button, and the appropriate string is generated for you to copy-paste into your test plan wherever you like.

Pre-defined Variables

Some variables are defined internally by JMeter. They are:

- COOKIE_cookiename - contains the cookie value
- JMeterThread.last_sample_ok - whether or not the last sample was OK - true/false. Note: this is updated after PostProcessors and Assertions have been run.
- START variables

Pre-defined Properties

Some built-in properties are defined by JMeter. These are listed below. For convenience, the START properties are also copied to variables with the same names.

- START.MS - JMeter start time in milliseconds
- START.YMD - JMeter start time as yyyyMMdd
- START.HMS - JMeter start time as HHmmss
- TESTSTART.MS - test start time in milliseconds

Please note that the START variables / properties represent JMeter startup time, not the test start time. They are mainly intended for use in file names etc.

REGULAR EXPRESSIONS

Regular expressions are used to search and manipulate text, based on patterns. JMeter interprets forms of regular expressions or patterns being used throughout a JMeter test plan, by including the pattern matching software Apache Jakarta ORO.

With the use of regular expressions, we can certainly save a lot of time and achieve greater flexibility as we create or enhance a Test Plan. Regular expressions provide a simple method to get information from pages when it is impossible or very hard to predict an outcome.

A standard usage example of using expressions is to get a session ID from the server response. If the server returns a unique session key we can easily get it using expressions in our load script.

To use regular expressions in your test plan, you need to use the Regular Expression Extractor in JMeter. You can place regular expressions in any component in a Test Plan.

It is worth stressing the difference between **contains** and **matches**, as used on the Response Assertion test element:

- **contains** means that the regular expression matched at least some part of the target, so 'alphabet' "contains" 'ph.b.' because the regular expression matches the substring 'phabe'.
- **matches** means that the regular expression matched the whole target. So 'alphabet' is "matched" by 'al.*t'.

Suppose you want to match the following portion of a web-page:

```
name="file" value="readme.txt"
```

and you want to extract readme.txt. A suitable regular expression would be:

```
name="file" value="(.+?)">
```

The special characters above are:

- **(** and **)** - these enclose the portion of the match string to be returned
- **.** - match any character
- **+** - one or more times
- **?** - stop when first match succeeds

Create JMeter Test Plan

Let us understand the use of Regular expressions in the Regular Expression Extractor—a Post-Processor Element by writing a test plan. This element will extract text from the current page using a Regular Expression to identify the text pattern that a desired element conforms with.

First we will write an HTML page which a list of people and their email id's. Deploy it to our tomcat server. The contents of html (index.html) are as follows:

```
<html>
<head>
</head>
<body>
<table style="border: 1px solid #000000;">
<th style="border: 1px solid #000000;">ID</th><th style="border: 1px solid
#000000;">name</th><th style="border: 1px solid #000000;">Email</th>
<tr><td >manisha@domain.com</td></tr>
<tr><td >joe@domain.com</td></tr>
```

```
</table>
</body>
</html>
```

On deploying it on tomcat server, this page would look like as in the snapshot below:

In our test plan we will select the person in the first row of the person table seen in the person list page above. To capture the ID of this person, let us first determine the pattern where we will find the person in the second row. As can be seen in the following snapshot, the ID of the second person is surrounded by <td > and </td >, and it is the second row of data having this pattern. We can use this to match the exact pattern that we want to extract information from. As we want to extract two pieces of information from this page, the person ID and the person's name, the fields are defined as follows:

Start JMeter, add a Thread group **Test Plan > Add> Threads(Users)> Thread Group**.

Next add a sampler HTTP Request, select the test plan right click **Add > Sampler > HTTP Request** and enter the details as below:

- **Name:** Manage
- **Server Name or IP:** localhost
- **Port Number:** 8080
- **Protocol:** We will keep this blank, which means we want HTTP as the protocol.
- **Path:** jmeter/index.html

Next, add a Regular Expression Extractor. Select the HTTP Request Sampler (Manage), right click **Add > Post Processor > Regular Expression Extractor**.

Details of the above snapshot are as below:

Field	Description
Reference Name	The name of the variable in which the extracted test will be stored (refname).
Regular Expression	The pattern against which the text to be extracted will be matched. The text groups that will extracted are enclosed by the characters '(' and ')'. We use '.+?' to indicate a single instance of the text enclosed by the <td..>..</td> tags. In our example the expression is : <td >(+?)</td>\s*
Template	Each group of text extracted will be placed as a member of the variable Person, following the order of each group of pattern enclosed by '(' and ')'. Each group is stored as refname_g#, where refname is the string you entered as the reference name, and # is the group number. 1 to refers to group 1, 2 to refers to group 2, etc. 0 refers to whatever the entire expression matches. In this example, the ID we extract will be maintained in Person_g1, while the Name value will be stored in Person_g2.
Match No.	Since we plan to extract only the second occurrence of this pattern, matching the second

	volunteer, we use value 2. Value 0 would make a random matching, while a negative value needs to be used with the ForEach Controller.
Default	If the item is not found, this will be the default value. This is an optional field. You may leave it blank.

Add a listener to capture the result of this Test Plan. Right click on Thread Group choose **Add > Listener > View Results Tree** option to add the listener.

Save the test plan as *reg_express_test.jmx* and run the test. The output would be a success as in the snapshot below:

BEST PRACTICES

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.
- Below are some suggestion to reduce resource requirements:
    - Use non-GUI mode: jmeter -n -t test.jmx -l test.jtl.
    - Use as few Listeners as possible; if using the -l flag as above they can all be deleted or disabled.
    - Disable the “View Result Tree” listener as it consumes a lot of memory and can result in the console freezing or JMeter running out of memory. It is, however, safe to use the “View Result Tree” listener with only “Errors” checked.
    - Rather than using lots of similar samplers, use the same sampler in a loop, and use variables (CSV Data Set) to vary the sample. Or perhaps use the Access Log Sampler.
    - Don't use functional mode.
    - Use CSV output rather than XML.
    - Only save the data that you need.
    - Use as few Assertions as possible.
    - Disable all JMeter graphs as they consume a lot of memory. You can view all of the real time graphs using the JTLs tab in your web interface.
    - Do not forget to erase the local path from CSV Data Set Config if used.
    - Clean the Files tab prior to every test run.