

# JMETER - QUICK GUIDE

[http://www.tutorialspoint.com/jmeter/jmeter\\_quick\\_guide.htm](http://www.tutorialspoint.com/jmeter/jmeter_quick_guide.htm)

Copyright © tutorialspoint.com

## JMETER - OVERVIEW

Before going into the details of JMeter, let us first understand a few jargons 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 any changes need to be made before the application goes into production.
- **Load Test** – This test is basically used for testing 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 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. 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.

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
- Directory – LDAP
- Messaging Oriented service via JMS
- Service – POP3, IMAP, SMTP
- FTP Service

### JMeter Features

Following are some of the features of JMeter –

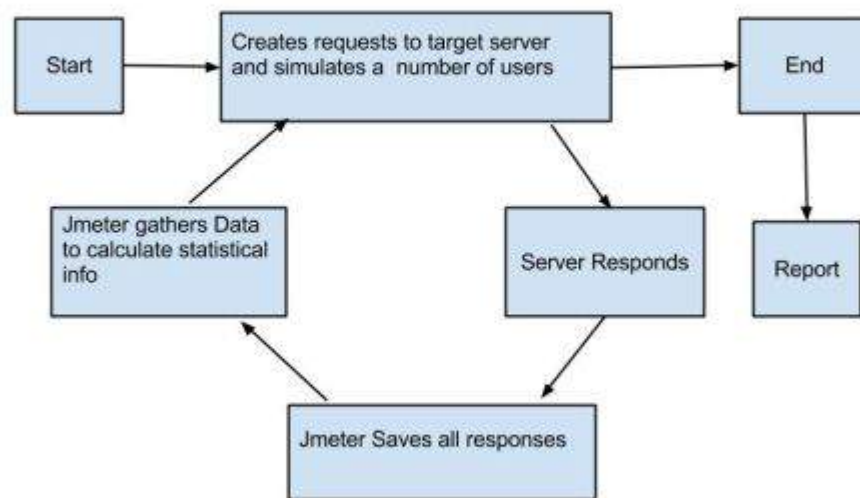
- Being an open source software, it is freely available.
- It has a simple and intuitive GUI.
- JMeter can conduct load and performance test for many different server types – Web - HTTP, HTTPS, SOAP, Database via JDBC, LDAP, JMS, Mail - POP3, etc.
- It is a platform-independent tool. On Linux/Unix, JMeter 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 store its test plans in XML format. This 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, etc.

Take a look at the following figure that depicts how JMeter works –



## JMETER - ENVIRONMENT

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

### System Requirement

<b>JDK</b>	1.6 or above.
<b>Memory</b>	No minimum requirement.
<b>Disk Space</b>	No minimum requirement.
<b>Operating System</b>	No minimum requirement.

### Step 1 – Verify Java Installation

First of all, verify whether you have Java installed in your system. Open your console and execute one of the following **java** commands based on the operating system you are working on.

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

Linux	Open Command Terminal	\$ java -version
Mac	Open Terminal	machine: ~ joseph\$ java -version

If you have Java installed in your system, you would get an appropriate output based on the OS you are working on.

OS	Output
Windows	java version "1.7.0_25" Java™ SE Runtime Environment <i>build1.7.0_25 – b15</i> Java HotSpot™ 64-Bit Server VM <i>build23.25 – b01, mixedmode</i>
Linux	java version "1.7.0_25" Java™ SE Runtime Environment <i>build1.7.0_25 – b15</i> Java HotSpot™ 64-Bit Server VM <i>build23.25 – b01, mixedmode</i>
Mac	java version "1.7.0_25" Java™ SE Runtime Environment <i>build1.7.0_25 – b15</i> Java HotSpot™ 64-Bit Server VM <i>build23.25 – b01, mixedmode</i>

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 the 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 as explained above.

### Step 3 – Download JMeter

Download the latest version of JMeter from [http://jmeter.apache.org/download\\_jmeter.cgi](http://jmeter.apache.org/download_jmeter.cgi). For this tutorial, we downloaded *apache-jmeter-2.9* and copied it into C:\>JMeter folder.

The directory structure should look like as shown 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

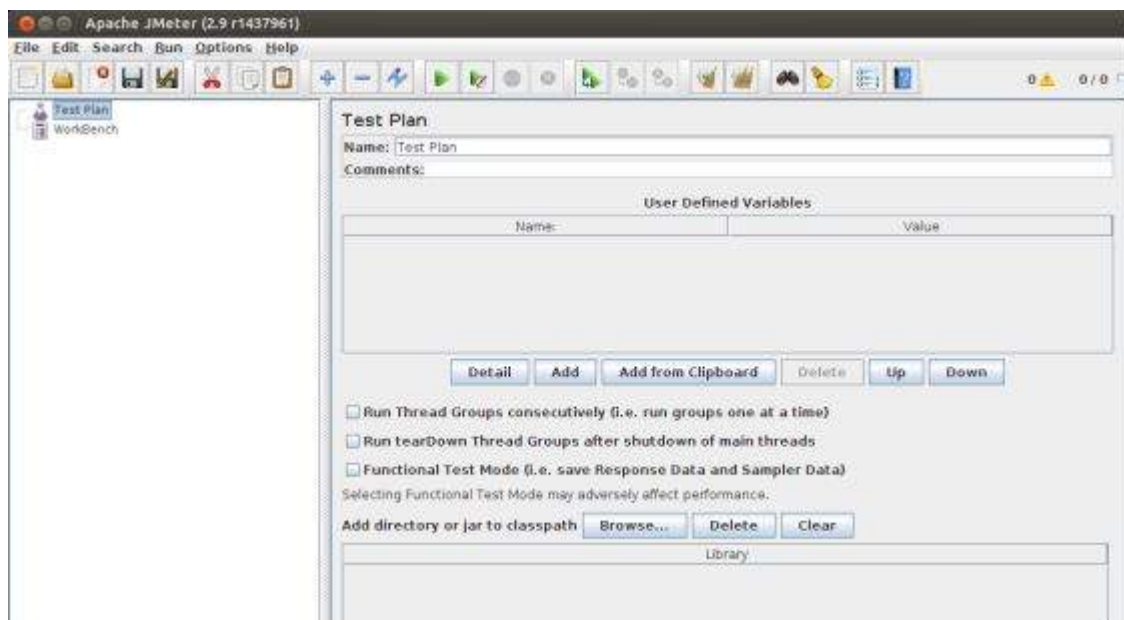
You can rename the parent directory *i. e. apache – jmeter – 2.9* if you want, but do not change any of the sub-directory names.

### Step 4 – Run JMeter

After downloading JMeter, go to the *bin* directory. In this case, it is **/home/manisha/apache-jmeter-2.9/bin**. Now click on the following –

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

After a short pause, the JMeter GUI should appear, which is a Swing application, as seen in the following screenshot –



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

## JMETER - BUILD TEST PLAN

### What is a Test Plan?

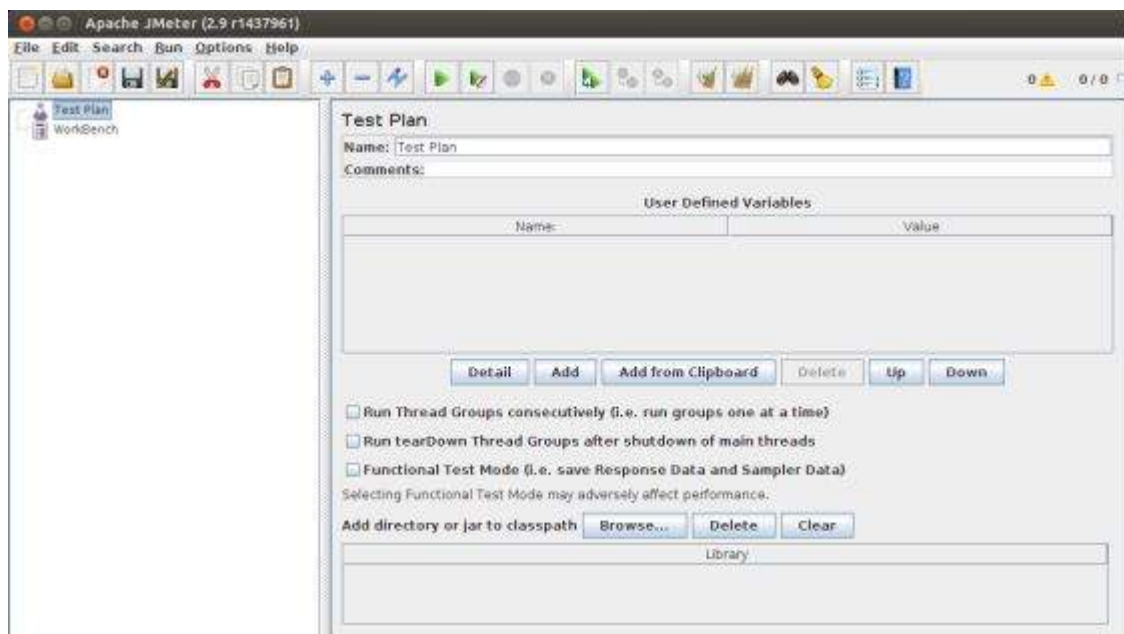
A Test Plan can be viewed as a container for running tests. It defines what to test and how to go about it. A complete test plan consists of one or more elements such as thread groups, logic controllers, sample-generating controllers, listeners, timers, assertions, and configuration elements. A test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



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

- **Test Plan node** – 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.

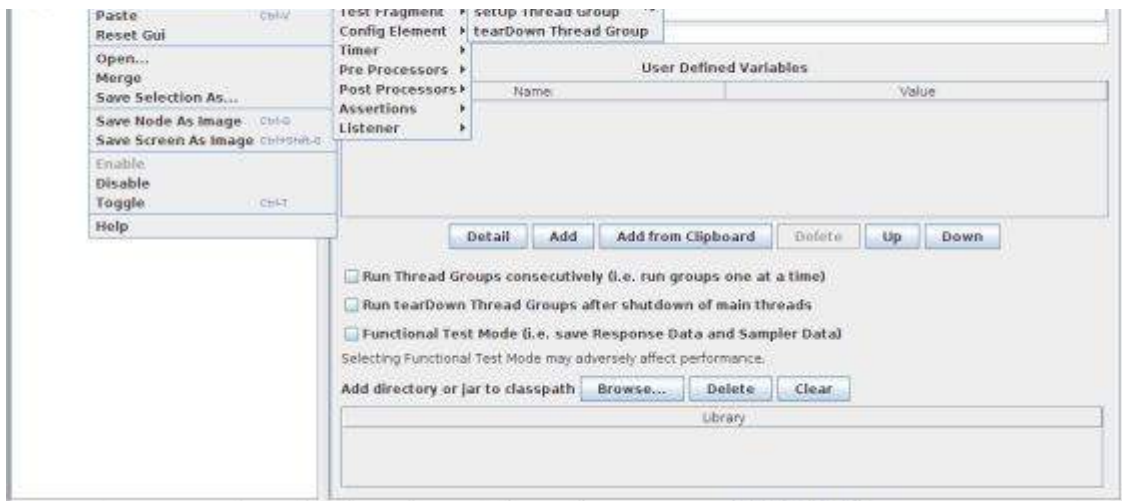
#### Step 2 – Add/Remove Elements

Elements (which will be discussed in the next chapter [Test Plan 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.

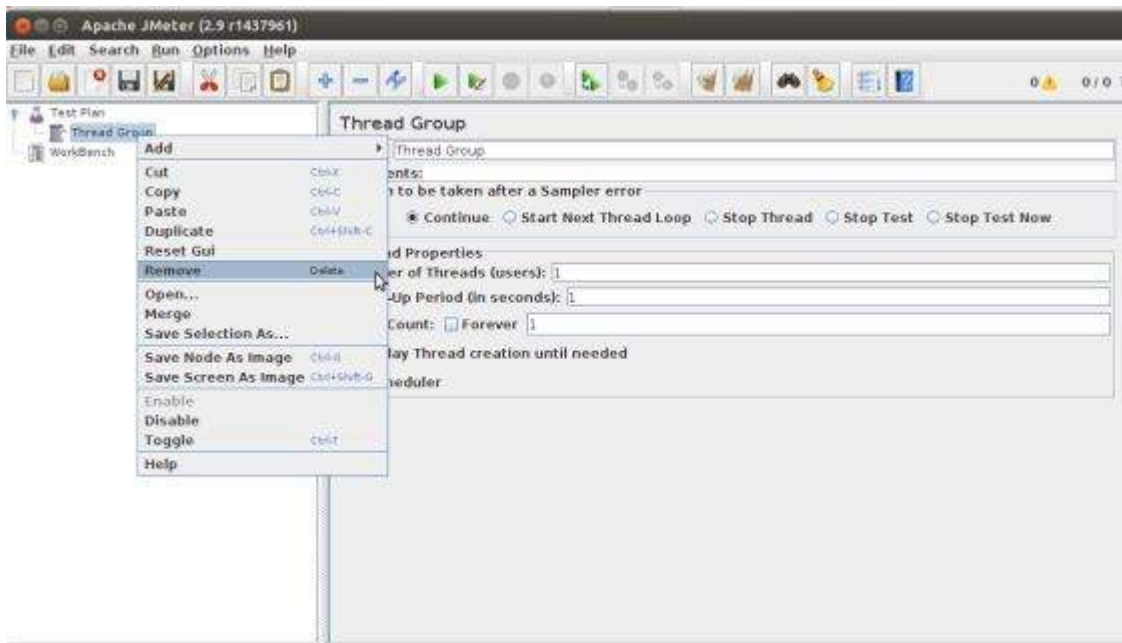
Alternatively, you can load an element from a file and add it by choosing the "merge" or "open" option.

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.

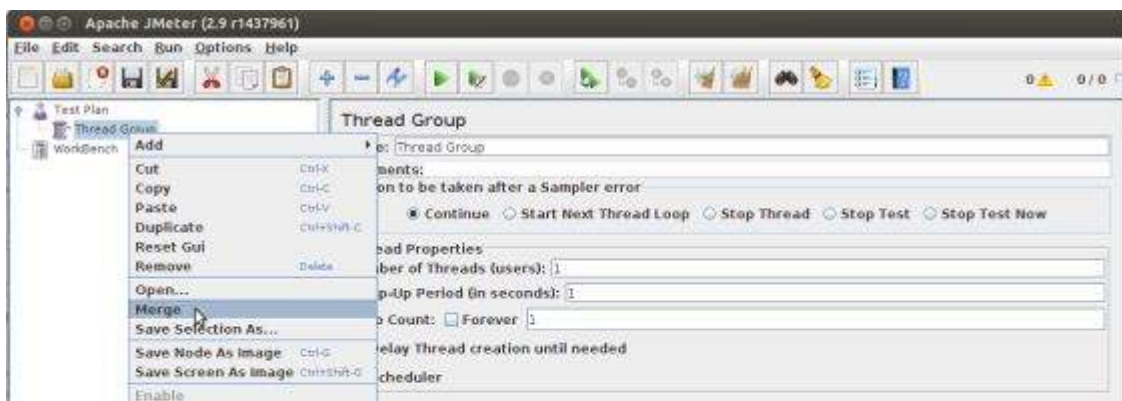


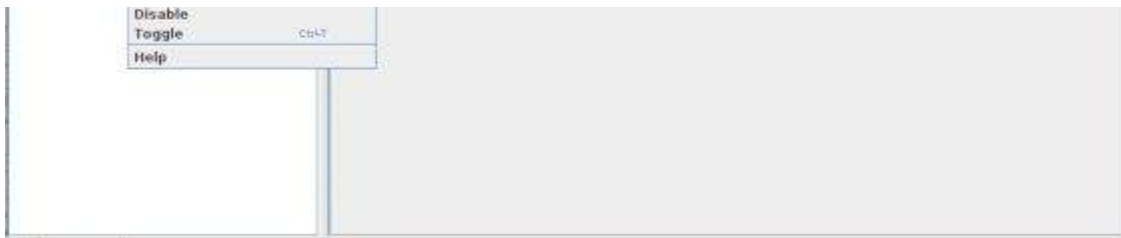
### Step 3 – Load and Save the Elements

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.





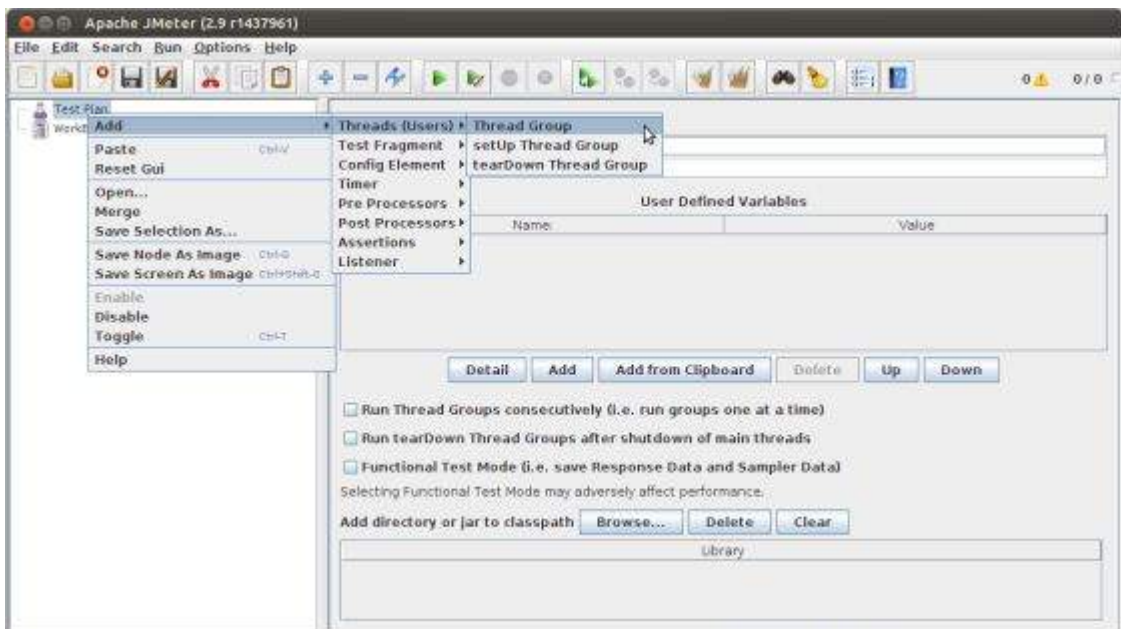
To save tree elements –

- 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.

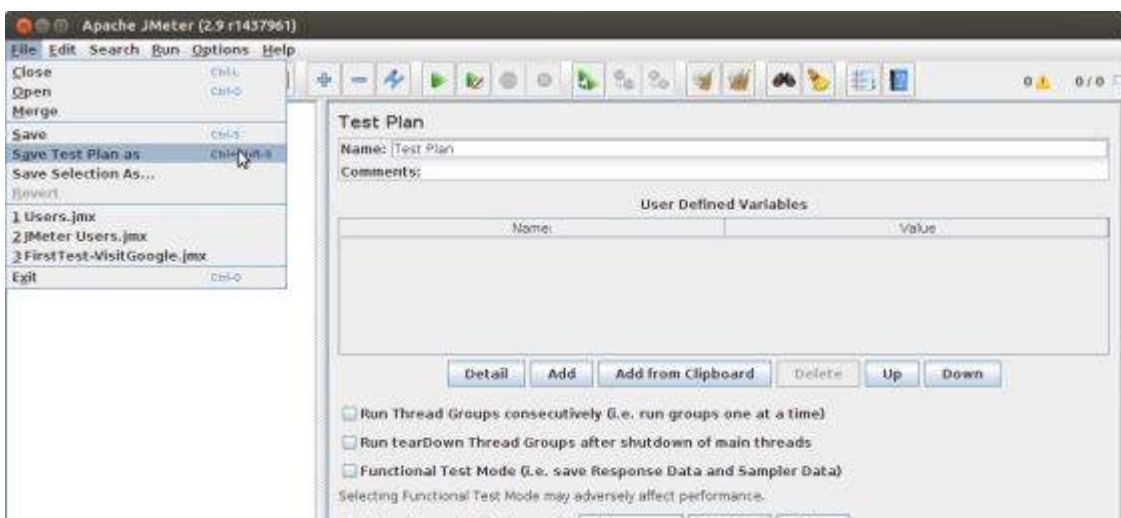
## Step 4 – Configuring the Tree Elements

Any element in the Test Plan can be configured using the controls present in JMeter's right-hand 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 –

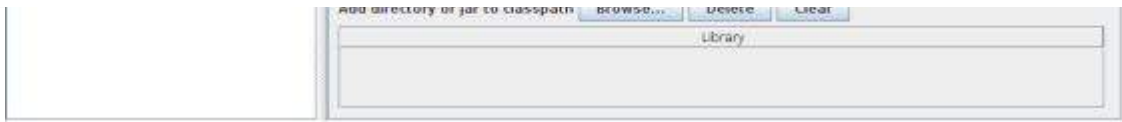


## Step 5 – Saving the Test Plan

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

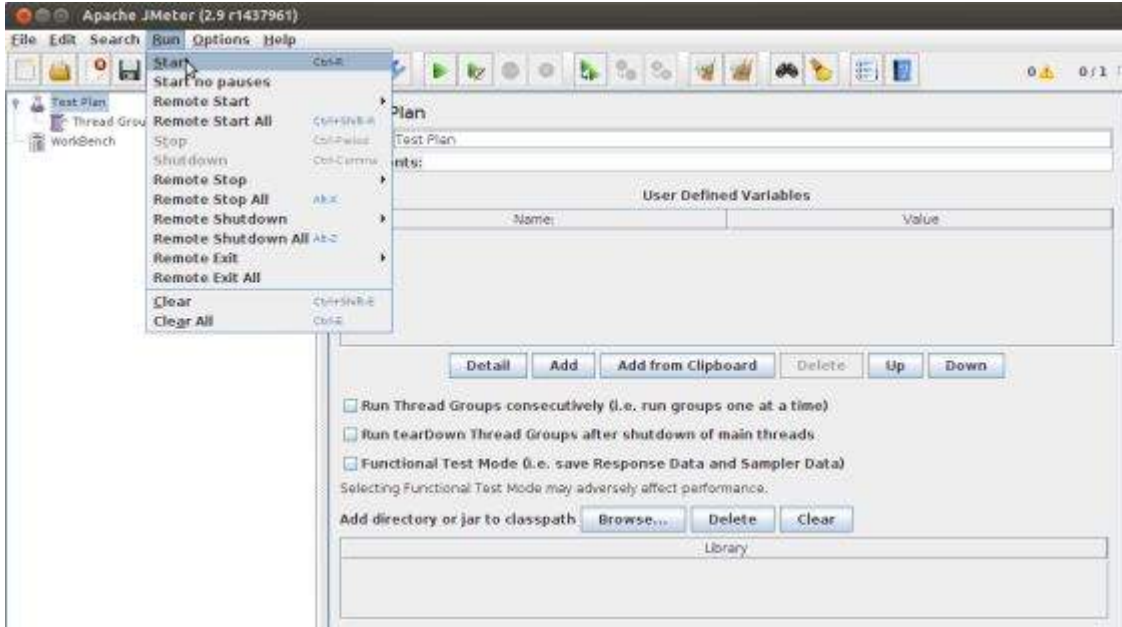






## Step 6 – Run the Test Plan

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.



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 discussed below. 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

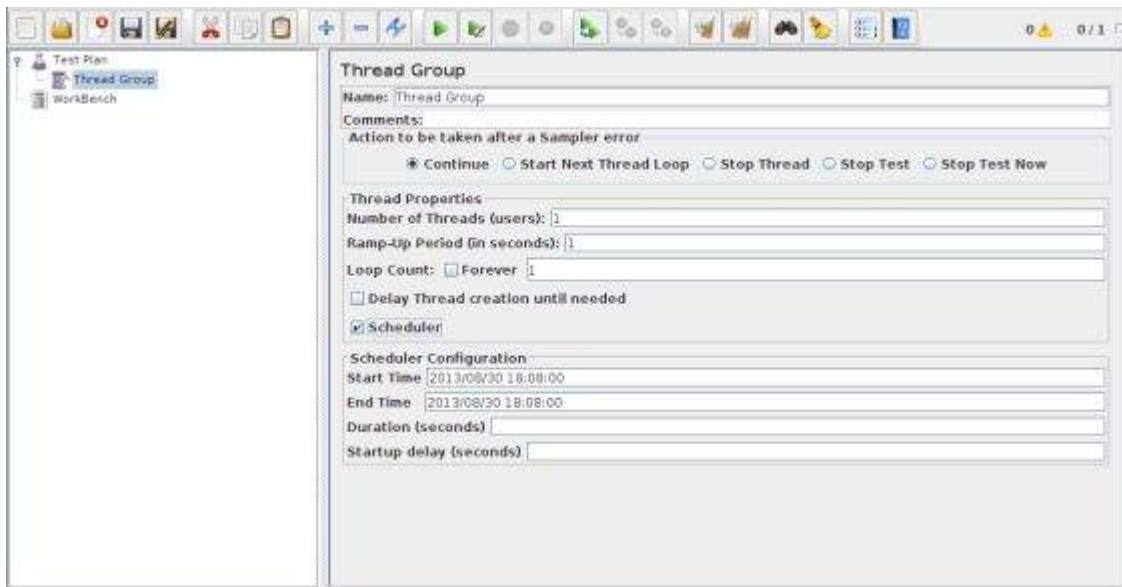
Thread Group elements are the beginning points of your test plan. As the name suggests, the thread group elements control the number of threads JMeter will use during the test. We can also control the following via the Thread Group –

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

The Thread Group Control Panel looks like this –







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*.

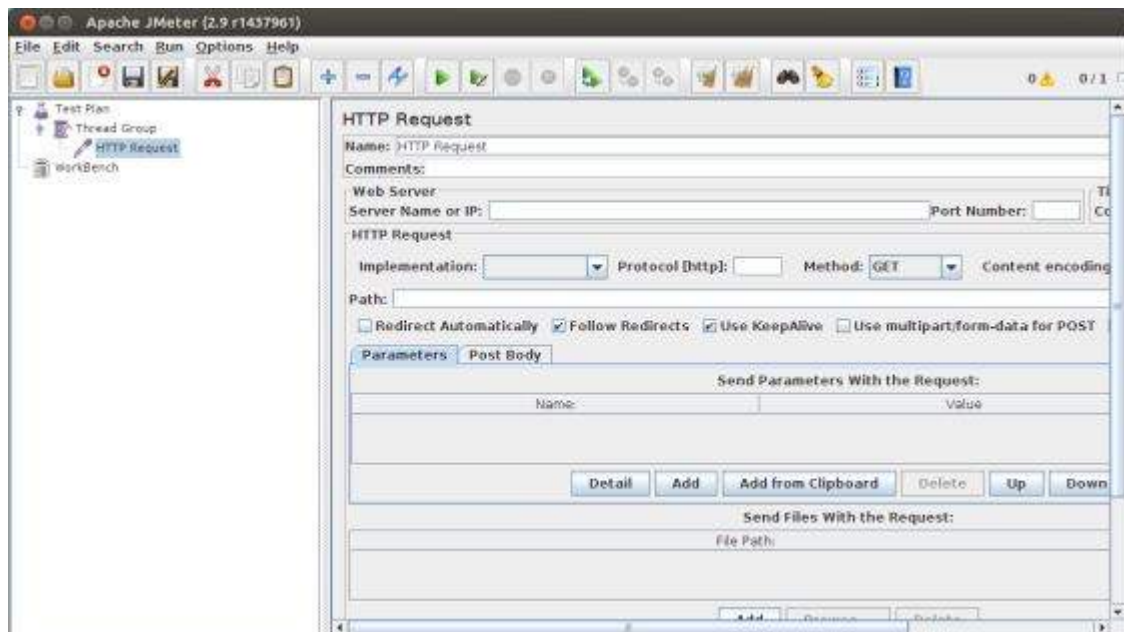
## Samplers

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

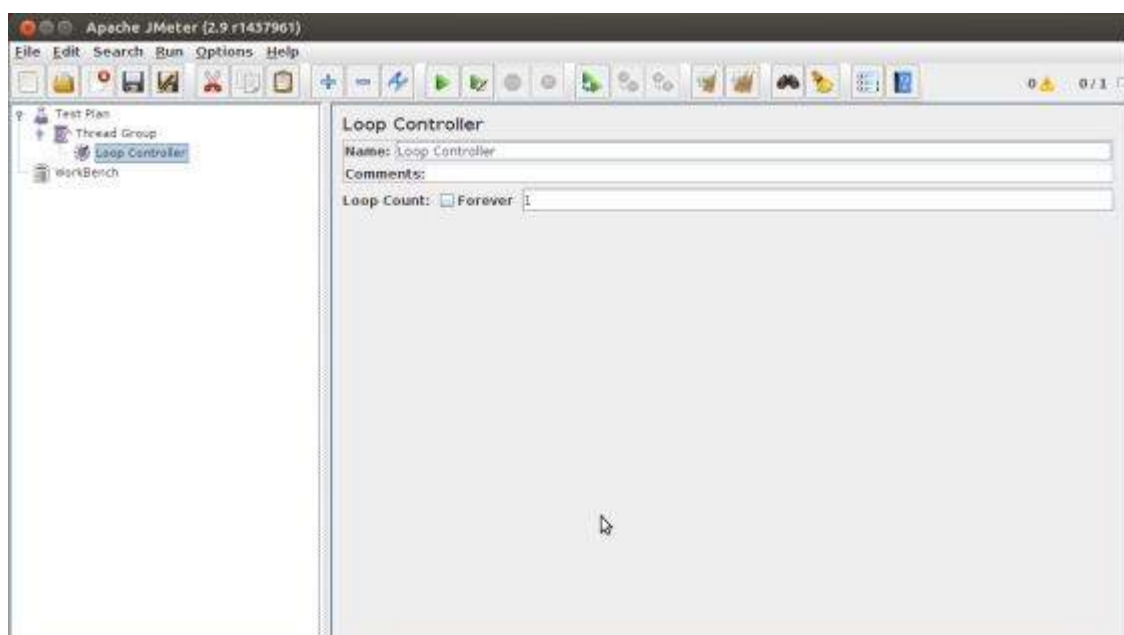
The following screenshot shows an HTTP Request Sampler Control Panel –



## Logic Controllers

Logic Controllers let you 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 – ForEach 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
- 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

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 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

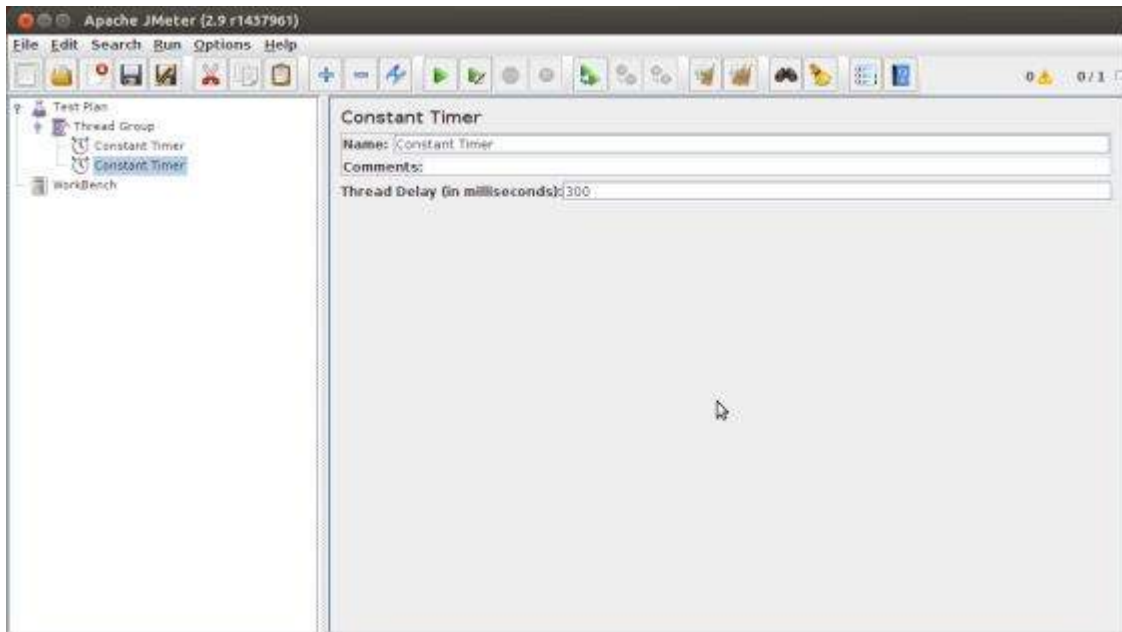
By default, a JMeter thread sends requests without pausing between each sampler. This may not be what you want. You can add a timer element which allows you to define a period to wait between each request.

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 –



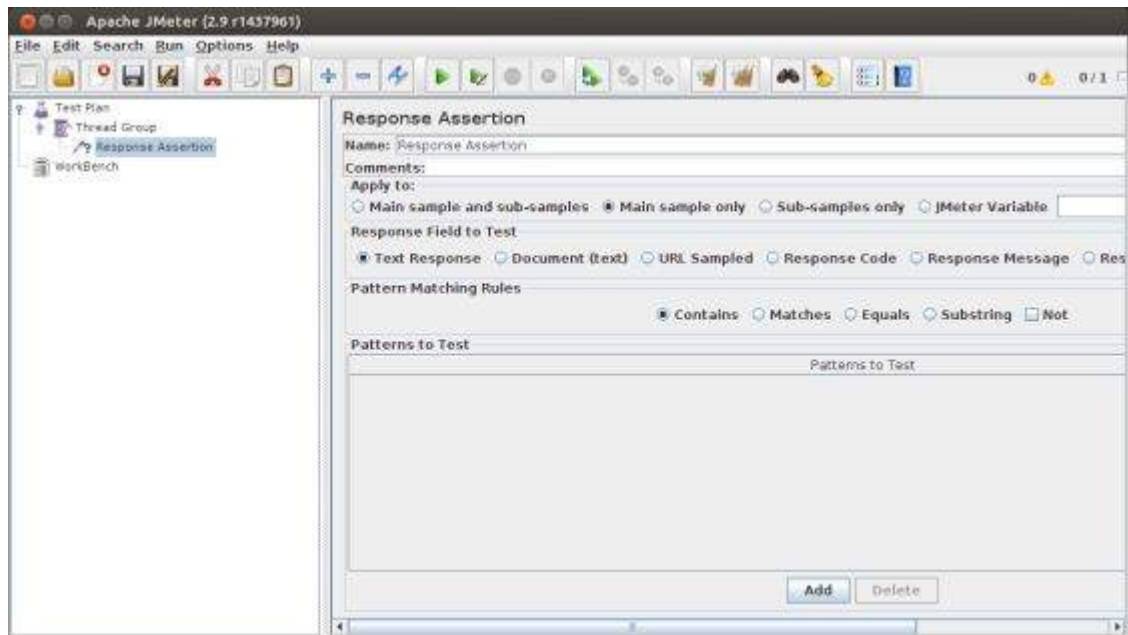
## Assertions

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

The following screenshot shows a Response Assertion Control Panel –



## 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
- 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-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 PreProcessor
- JSR223 PreProcessor
- RegEx 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 a 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 –

- Configuration elements
- Pre-Processors
- Timers
- Sampler
- Post-Processors *unlessSampleResultisnull*
- Assertions *unlessSampleResultisnull*
- Listeners *unlessSampleResultisnull*

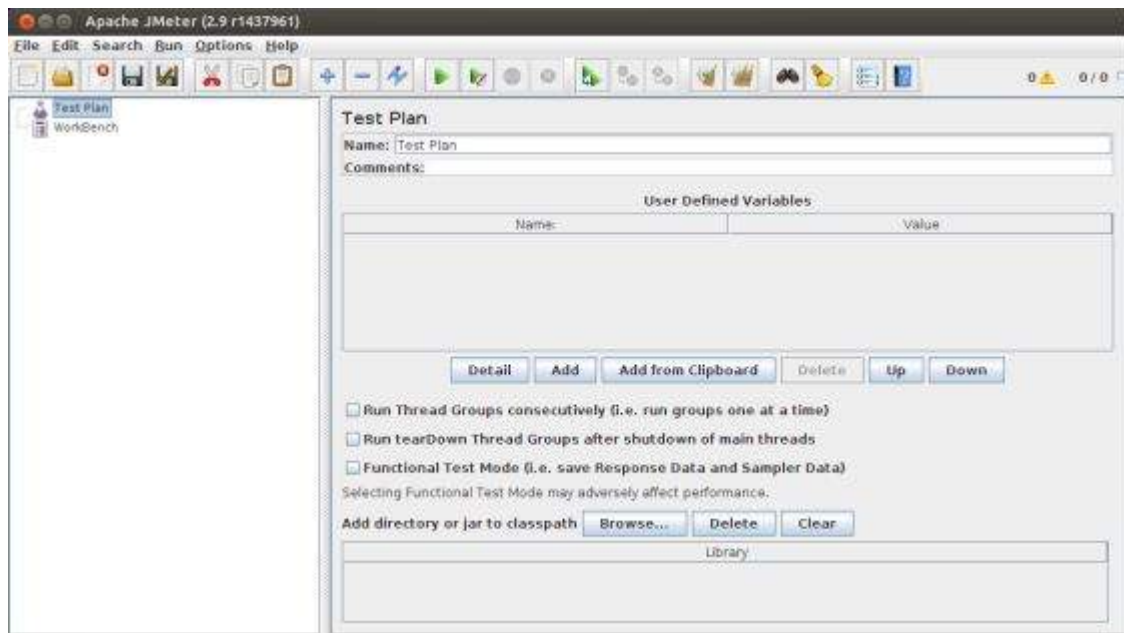
## JMETER - WEB TEST PLAN

Let us build a simple test plan which tests a web page. We write a test plan in Apache JMeter so that we can test the performance of the web page shown by the URL –

<http://www.tutorialspoint.com/>.

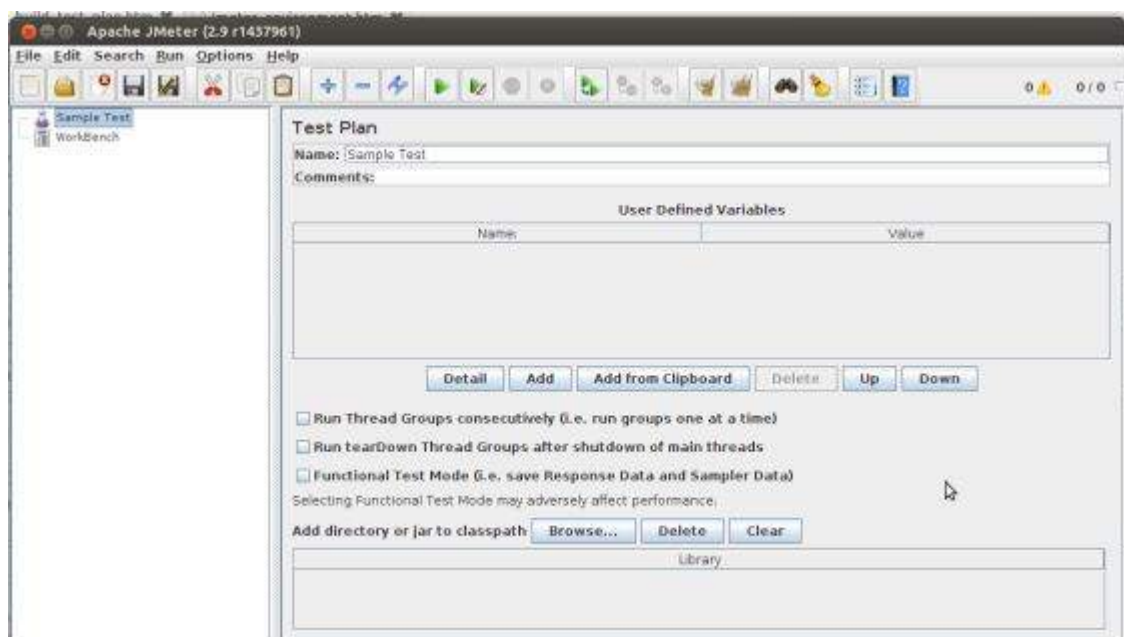
## Start JMeter

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



## Rename the Test Plan

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



## Add Thread Group

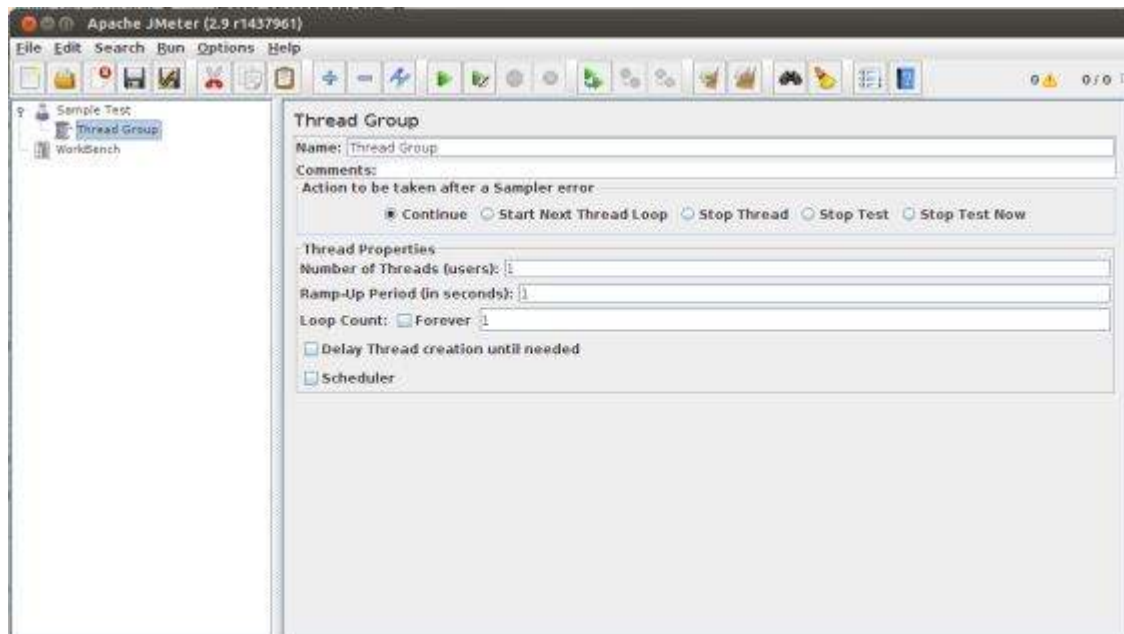
Now we add our first element in the window. We add one Thread Group, which is a placeholder for all other elements like Samplers, Controllers, and Listeners. We need one so we can configure number of users to simulate.

In JMeter, all the node elements are added by using the context menu.

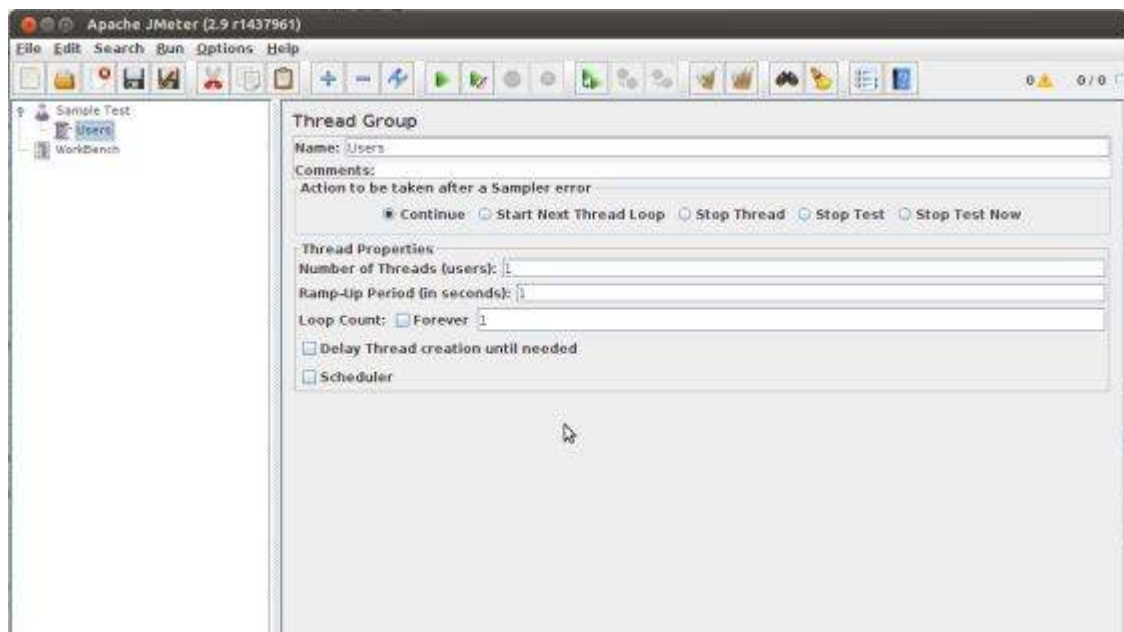
- Right-click the element where you want to add a child element node.
- Choose the appropriate option to add.



- Right-click on the Sample Test *ourTestPlan* > Add > Threads *Users* > Thread Group. Thus, the Thread Group gets added under the Test Plan *SampleTest* node.

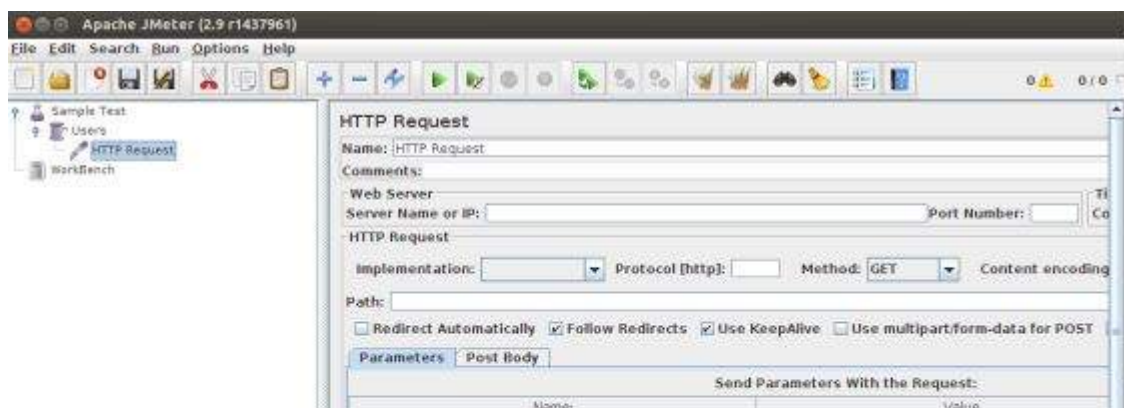


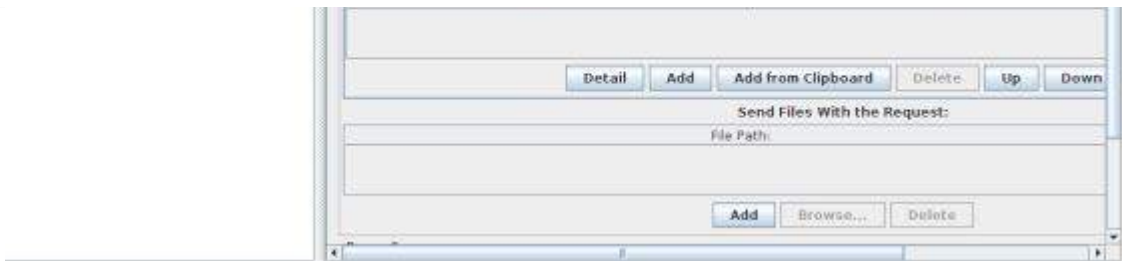
- Name the Thread Group as *Users*. For us, this element means users visiting the TutorialsPoint Home Page.



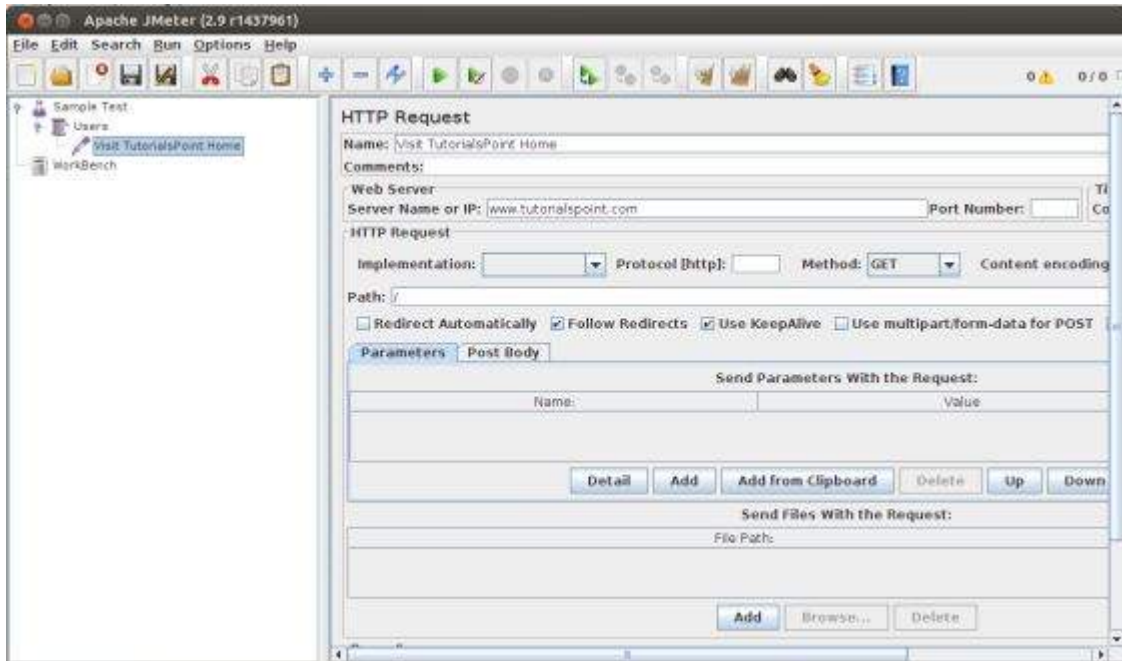
## Add Sampler

We need 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.





It will add one empty HTTP Request Sampler under the Thread Group *Users* node. Let us configure this node element –



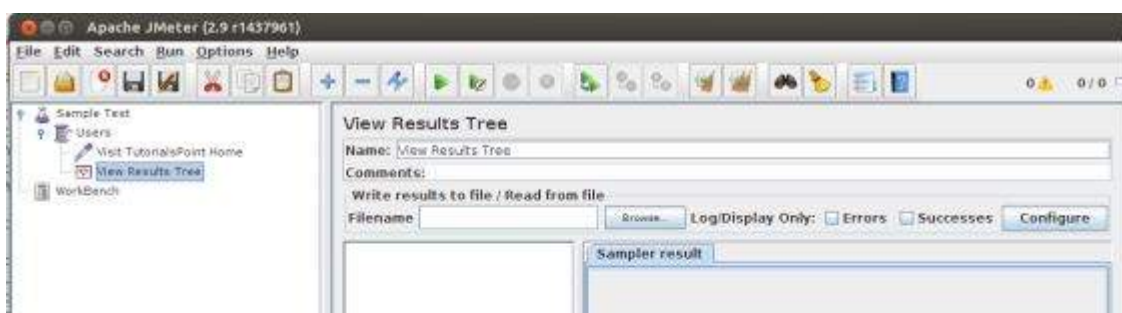
- **Name** – We will change the name to reflect the action what we want to achieve. We will name it as **Visit Tutorialspoint Home Page**
- **Server Name or IP** – Here, we have to type the web server name. In our case it is **www.tutorialspoint.com**. *http://partisnotwrittenthisisonlythenameoftheserveroritsIP*
- **Protocol** – We will keep this blank, which means we want HTTP as the protocol.
- **Path** – We will type path as */ slash*. It means we want the root page of the server.

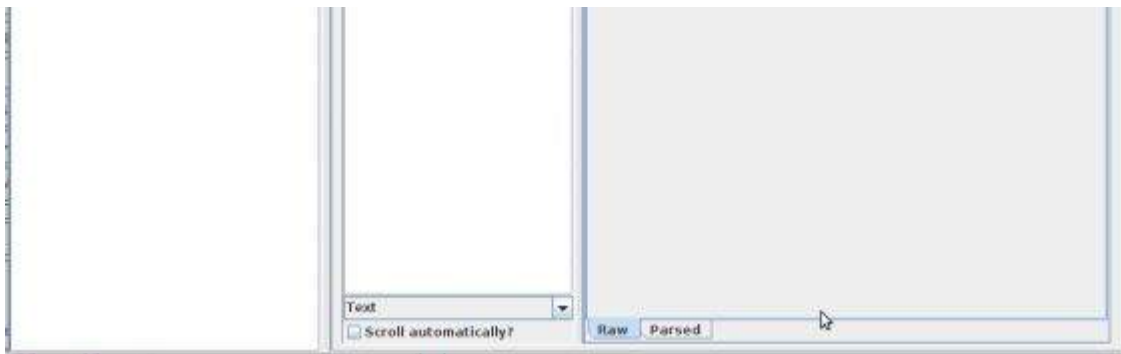
## Add Listener

We will now add a listener. Let us add View Results Tree Listener under the Thread Group *User* node. It will ensure that the results of the Sampler will be available to view in this Listener node element.

To add a listener –

- Open the context menu
- Right-click the Thread Group *Users*
- Choose Add > Listener > View Results Tree option



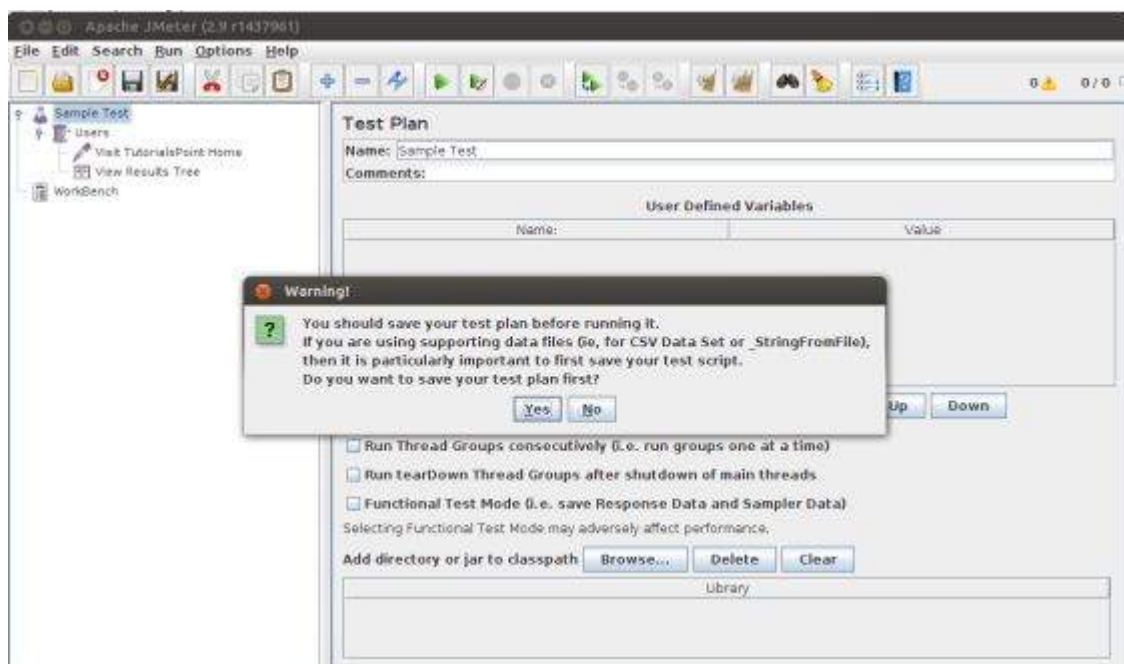


## Run the Test Plan

Now with all the setup, let us execute the test plan. With the configuration of the Thread Group *Users*, we keep all the default values. It means JMeter will execute the sampler only once. It is similar to a single user, only once.

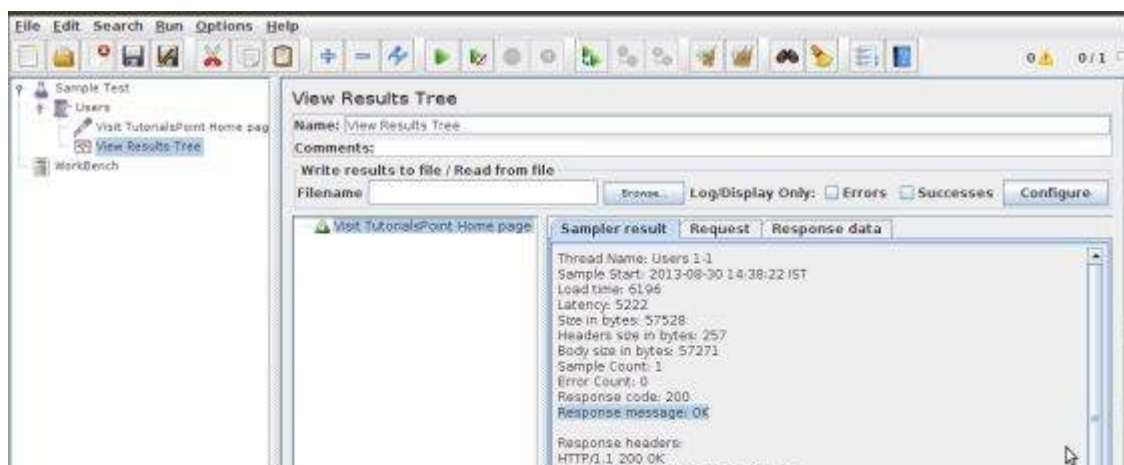
This is similar to a user visiting a web page through browser, with JMeter sampler. To execute the test plan, Select Run from the menu and select Start option.

Apache JMeter asks us to save the test plan in a disk file before actually starting the test. This is important if you want to run the test plan multiple times. You can opt for running it without saving too.



## View the Output

We have kept the setting of the thread group as single thread *oneuseronly* and loop for 1 time *runonlyonetime*, hence we will get the result of one single transaction in the View Result Tree Listener.





# JMETER - 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 use the MYSQL database server. You can use any other database for testing. For installation and table creation in MYSQL 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* as shown below –

```
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**.

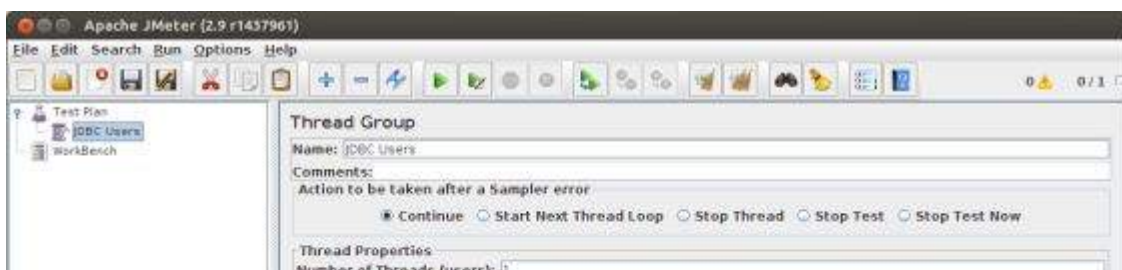
## Create JMeter Test Plan

Let us start the JMeter from **/home/manisha/apache-jmeter-2.9/bin/jmeter.sh**.

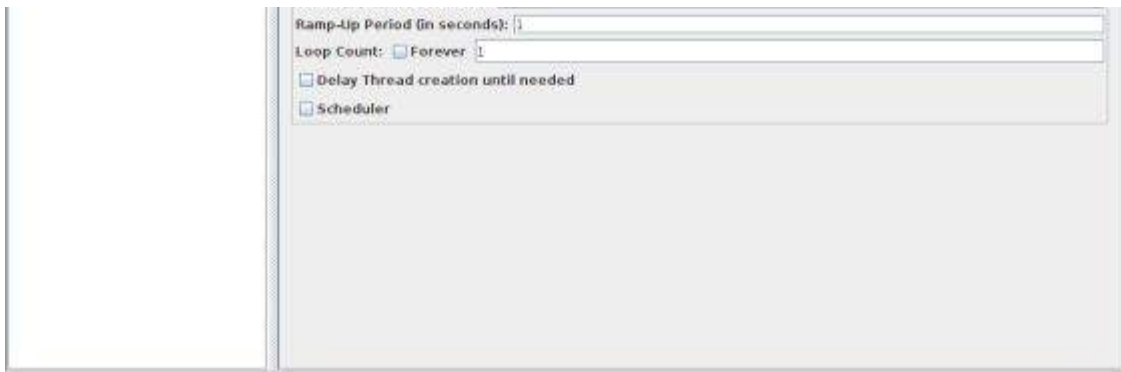
## Add Users

To create a Thread group,

- Right-click on Test Plan.
- Select Add > Threads *Users* > Thread Group.
- Thus, thread group gets added under the Test Plan node.
- Rename this Thread Group as *JDBC Users*.







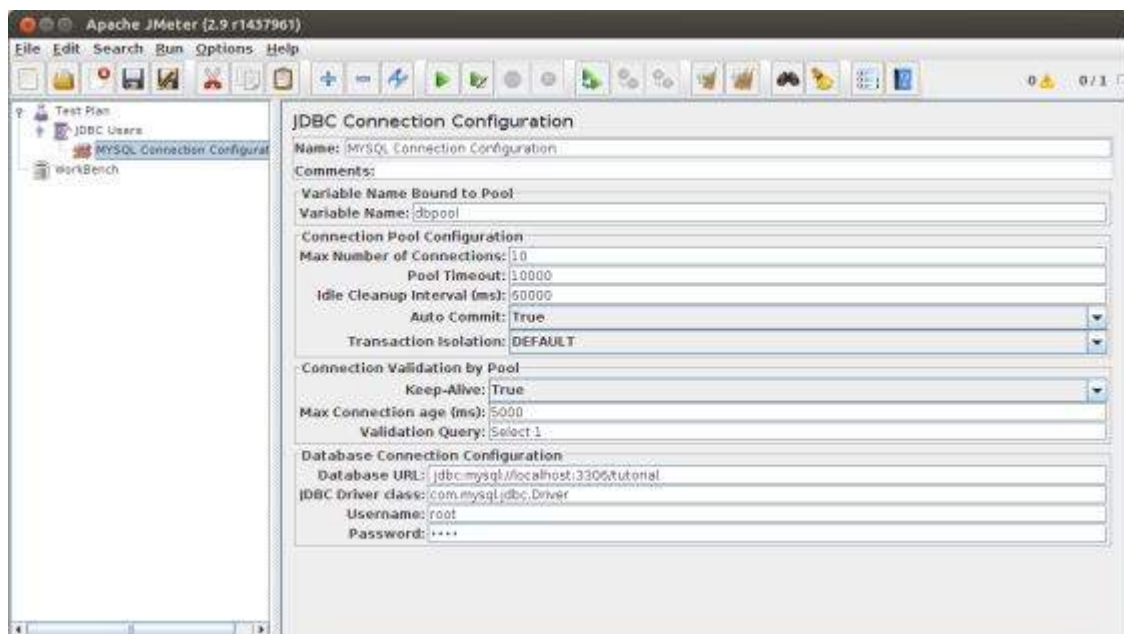
We will not change the default properties of the Thread Group.

## Adding JDBC Requests

Now that we defined our users, it is time to define the tasks that they will be performing. In this section, 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 *weareusingMySQLdatabasecalledtutorial* –
  - Variable name bound to pool. This needs to identify the configuration uniquely. 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`.

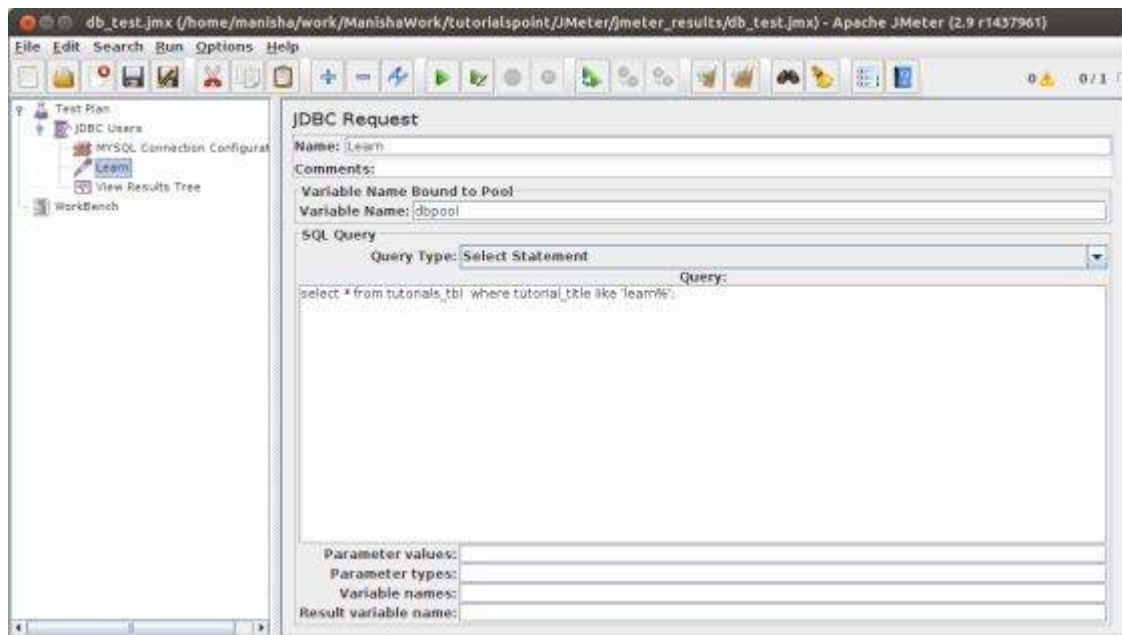
The other fields on the screen are left as defaults as shown below –



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
- Select **Add > Sampler > JDBC Request**.

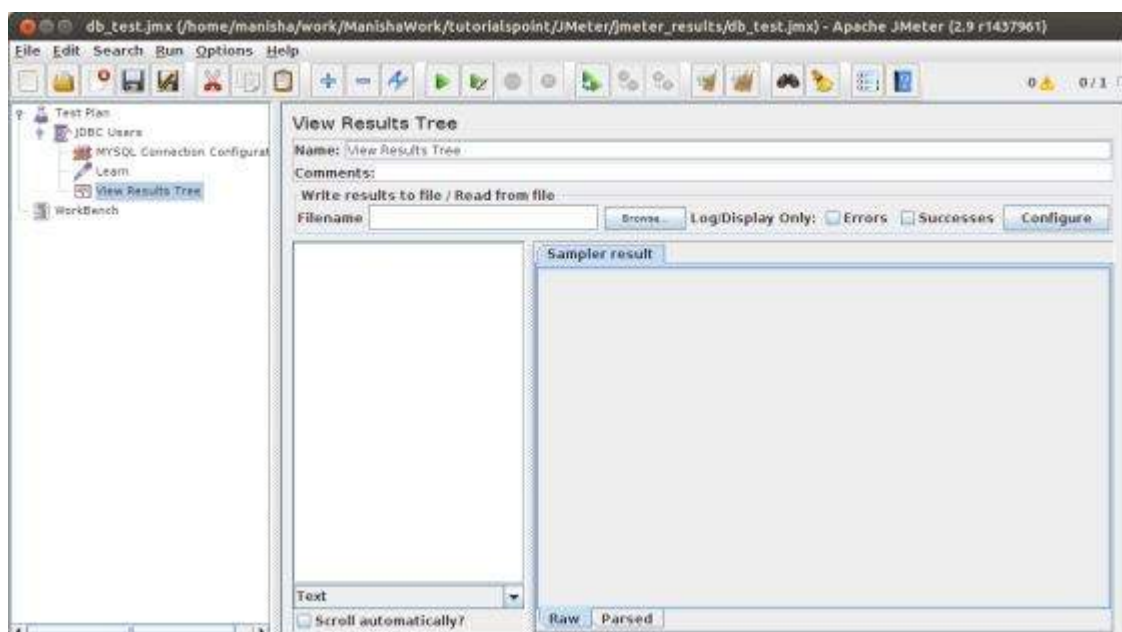
- Select this new element to view its Control Panel.
- Edit the properties as shown below –
  - Variable name bound to pool. This needs to uniquely identify the configuration. It is used by the JDBC Sampler to identify the configuration to be used. Named it as *test*.
  - Name – Learn.
  - Enter the Pool Name – *test sameasintheframeworkelement*.
  - 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
- Add a View Results Tree listener (**Add > Listener > View Results Tree**).

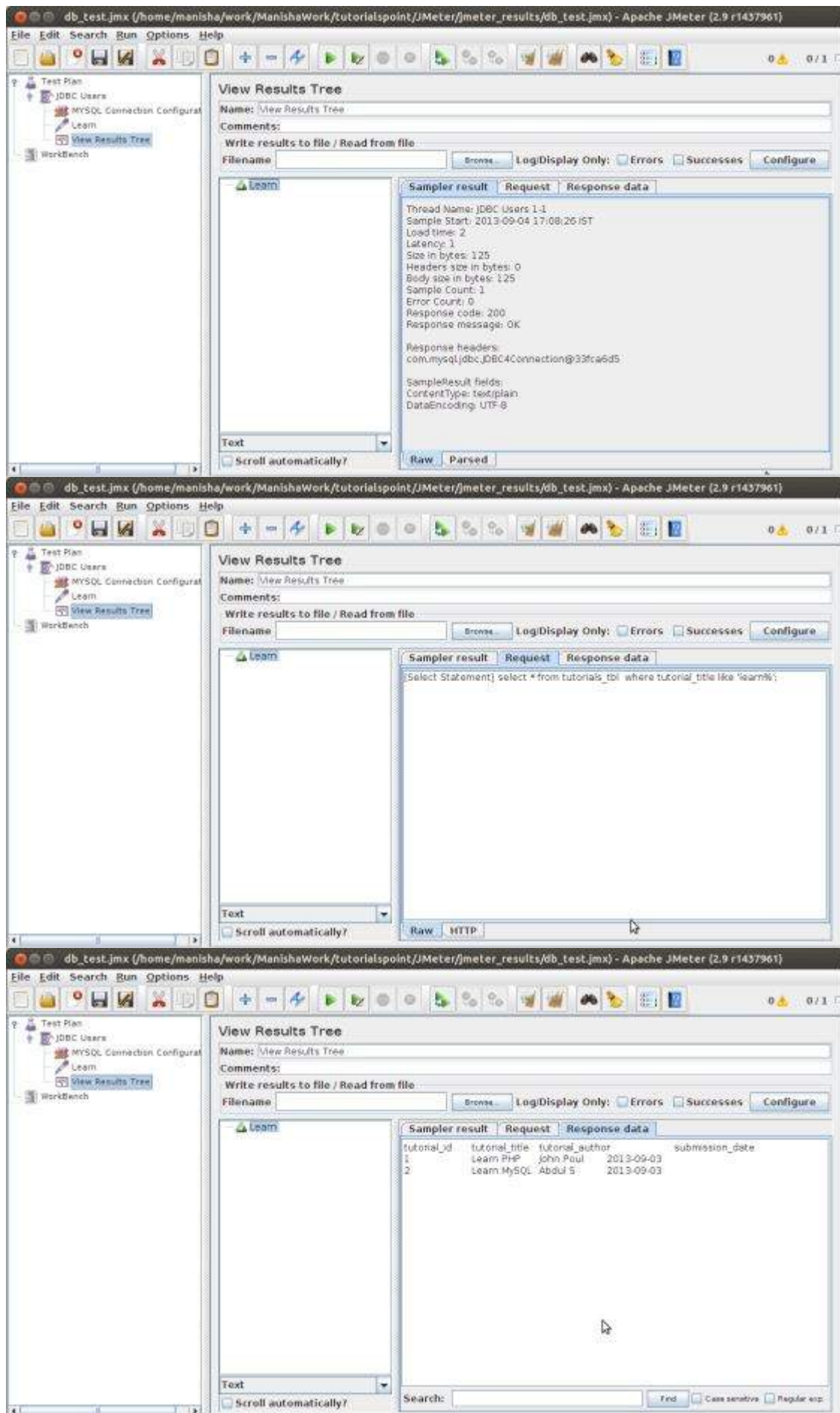


## Save and Execute Test Plan



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

## Verify the Output



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

# JMETER - 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.

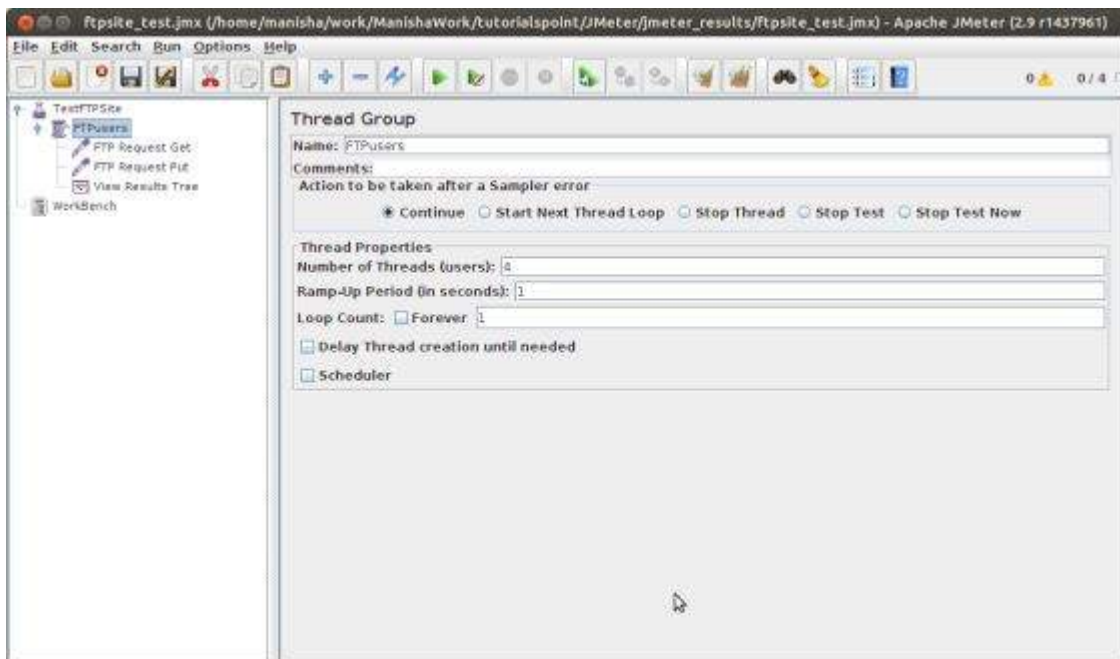
## Rename Test Plan

- Open the JMeter window by clicking `/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, and Listeners.

- Right click on TestFTPSite *ourTestPlan*
- Select Add > ThreadsUsers > Thread Group. Thread Group will get added under the Test Plan *TestFTPSite* node.
- Modify the default properties of the Thread Group to suit our testing as follows –
  - **Name** – FTPUsers
  - **Number of Threads Users** – 4
  - **Ramp-Up Period** – leave the the default value of 0 seconds.
  - **Loop Count** – 1

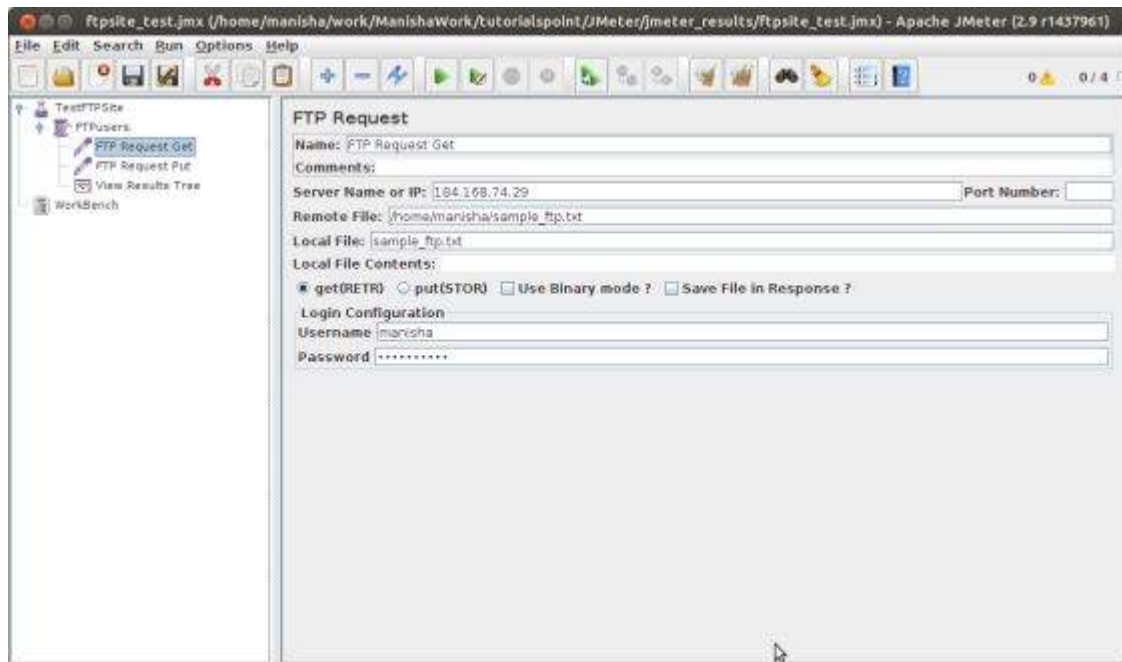


## Add Sampler – FTP Request

Now that we have defined our users, it is time to define the tasks that they will be performing. Add FTP Request elements. We add two FTP request elements, one which retrieves a file and the other which puts a file on the ftp site.

- Select the FTPUsers element.
- Right-click the mouse button to get the Add menu
- Select Add > Sampler > FTP Request.

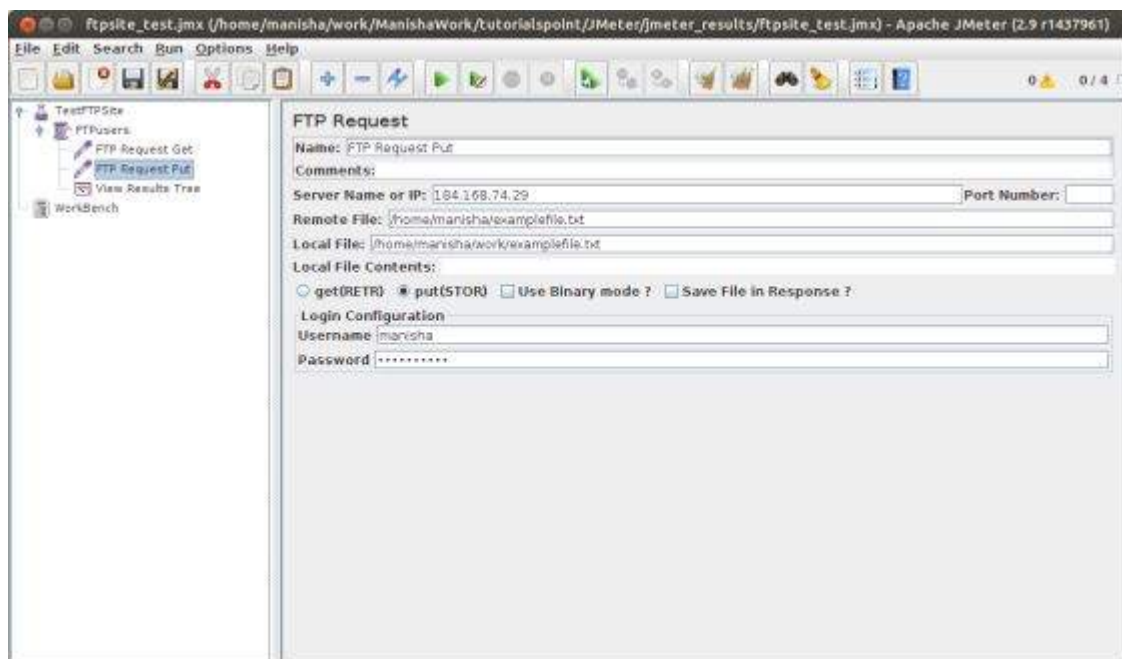
- Select the FTP Request element in the tree.
- Edit the following properties as shown below –



The following details are entered in 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 getRETR
- **Username** – manisha
- **Password** – manisha123

Now add another FTP request as above and edit the properties as shown in the following screenshot –



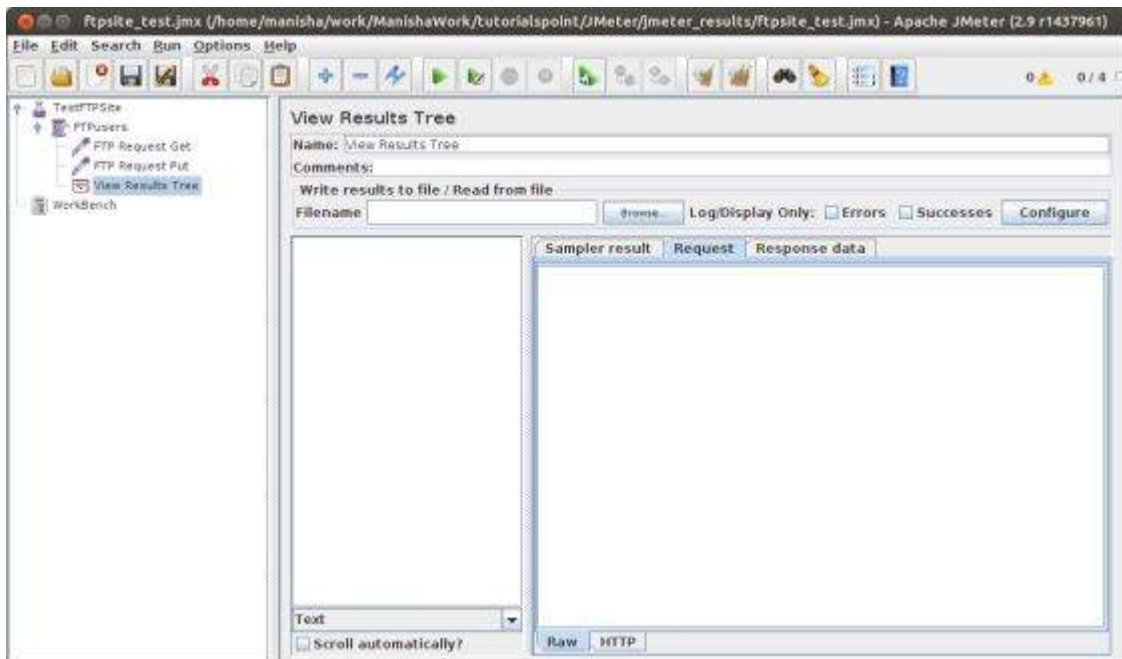
The following details are entered in this element –

- **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.
- Add a View Results Tree listener by selecting Add > Listener > View Results Tree.

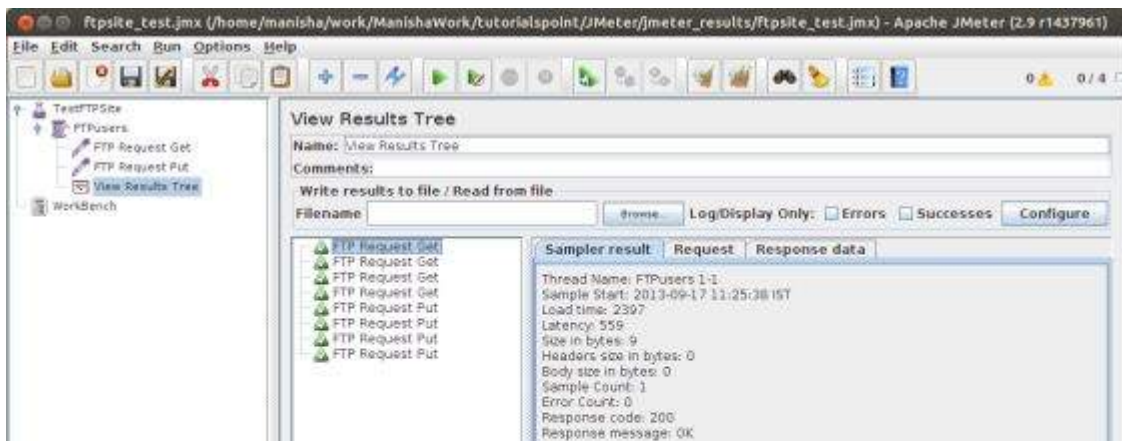


## Run the Test Plan

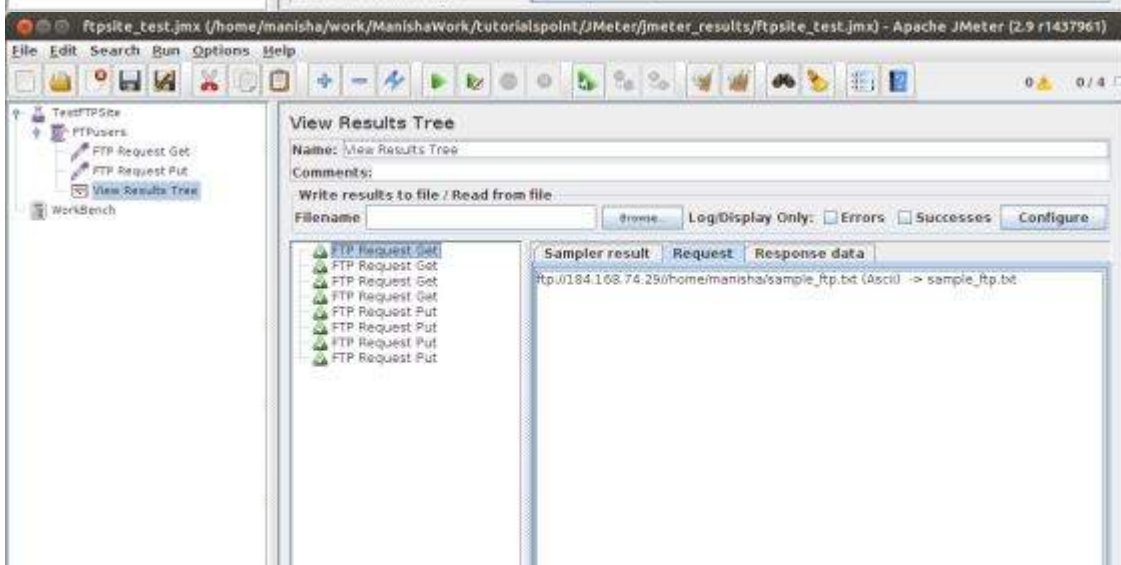
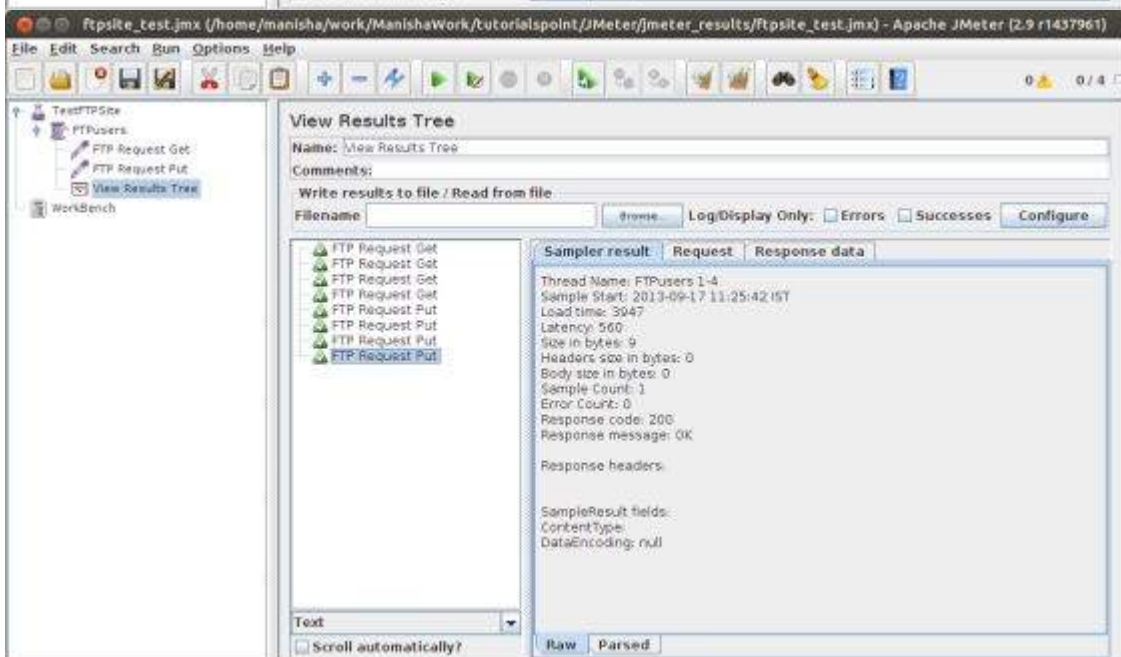
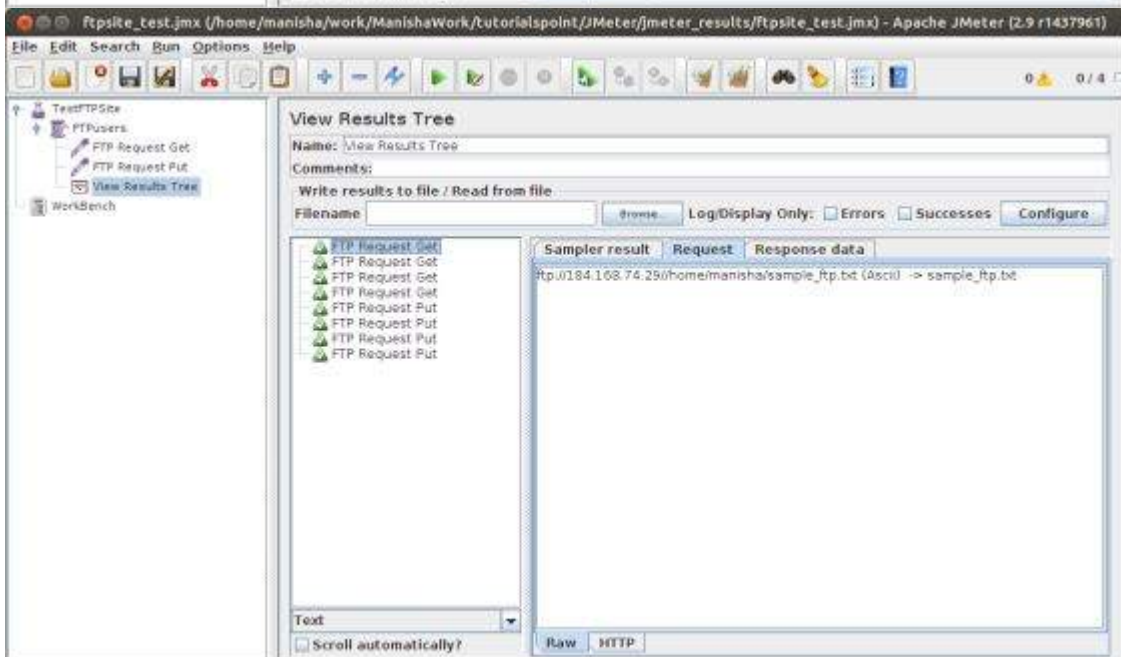
Now save the above test plan as *ftpsite\_test.jmx*. Execute this test plan using **Run > Start** option.

## View the Output

The following output can be seen in the listener.









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

## JMETER - 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 follows –

```
package com.tutorialspoint.ws;

import javax.ws.WebMethod;
import javax.ws.WebService;
import javax.ws.soap.SOAPBinding;
import javax.ws.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 a String parameter.

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

```
package com.tutorialspoint.ws;

import javax.ws.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");
    }

}
```

Let us now 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 are as 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());
    }
}

```

Modify the web.xml contents as shown 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 a webservice, we would need another configuration file **sun-jaxws.xml**. The contents of this file are as follows –

```

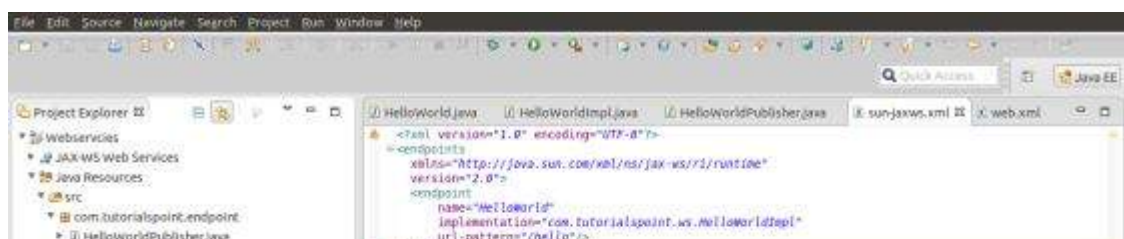
<?xml version="1.0" encoding="UTF-8"?>
<endpoints xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime" version="2.0">

    <endpoint name="HelloWorld" implementation="com.tutorialspoint.ws.HelloWorldImpl"
url-pattern="/hello"/>

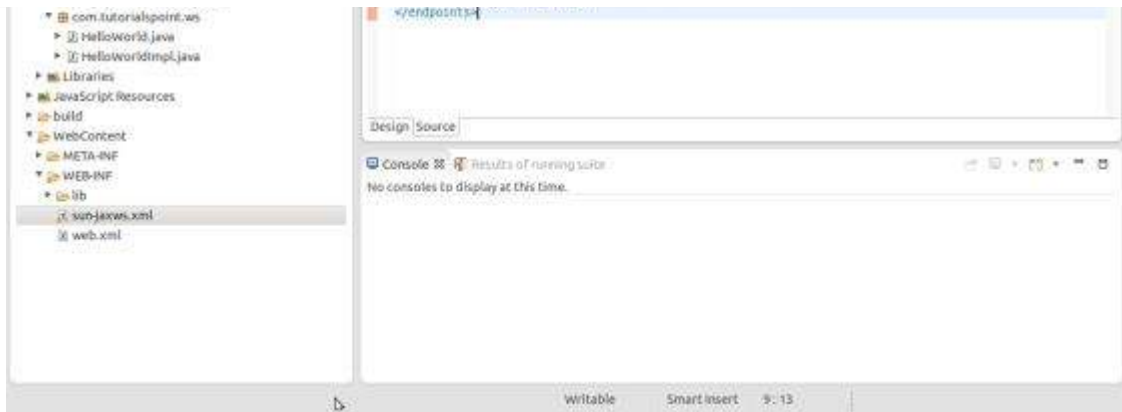
</endpoints>

```

Now that all the files are ready, the directory structure would look as shown in the following screenshot –







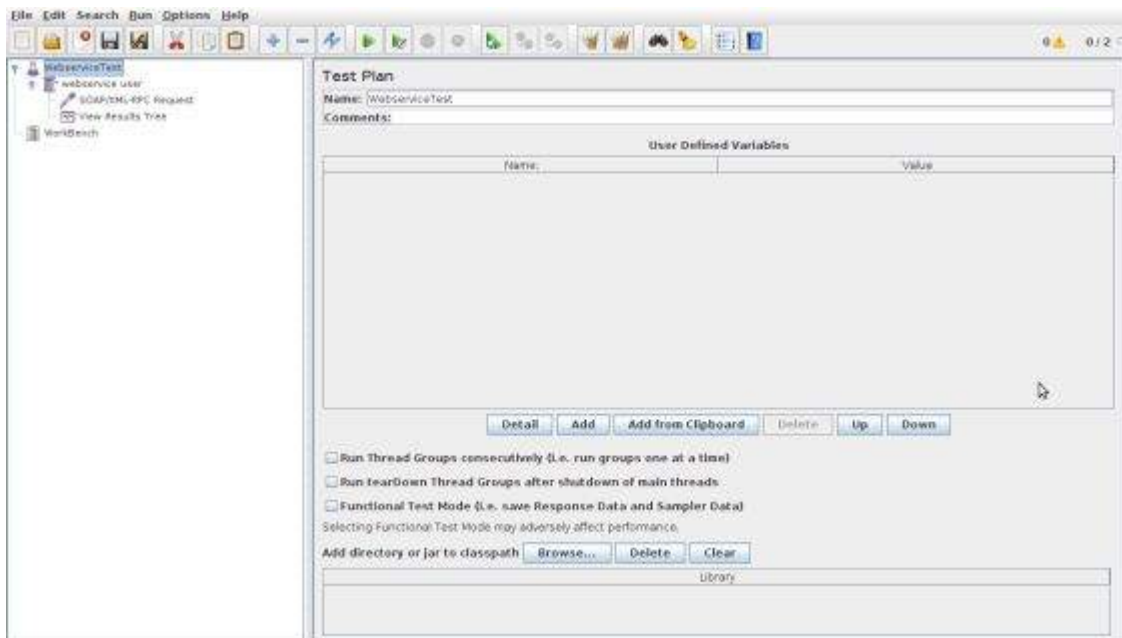
- Now create a WAR file of this application.
- Choose the project > right click > Export > WAR file.
- Save this as **hello.war** file under the **webapps** folder of Tomcat server.
- Now start the Tomcat server.
- Once the server is started, you should be able to access the webservice with the URL – <http://localhost:8080/hello/hello>

## Create JMeter Test plan

Now let us create a test plan to test the above webservice.

### Rename the Test Plan

- Open the JMeter window by clicking `/home/manisha/apache-jmeter2.9/bin/jmeter.sh`.
- Click the Test Plan node.
- Rename this Test Plan node as **WebServiceTest**.



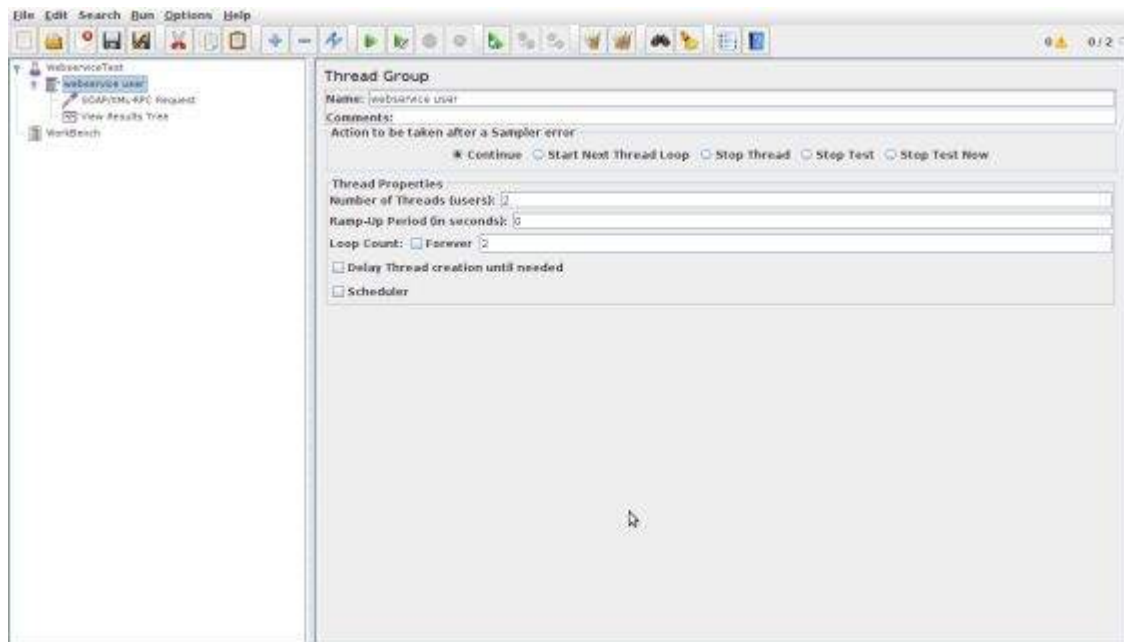
## Add Thread Group

Add one Thread Group, which is placeholder for all other elements like Samplers, Controllers, and Listeners.

- Right click on **WebServiceTest** > Add > Threads > 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 the users, it is time to define the tasks that they will be performing.

We will add SOAP/XML-RPC Request element –

- Right-click mouse button to get the Add menu.
- Select Add > Sampler > SOAP/XML-RPC Request.
- Select the SOAP/XML-RPC Request element in the tree
- Edit the following properties as in the image below –
- The following details are entered in 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/"

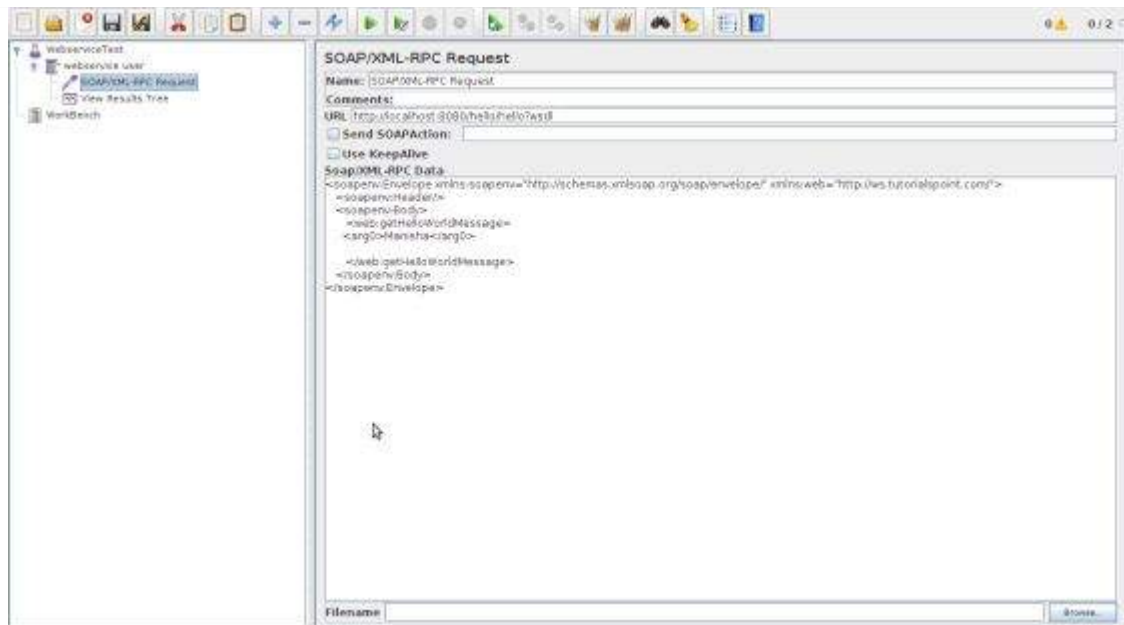
  <soapenv:Header/>

  <soapenv:Body>

    <web:getHelloWorldMessage>
      <arg0>Manisha</arg0>
    </web:getHelloWorldMessage>

  </soapenv:Body>

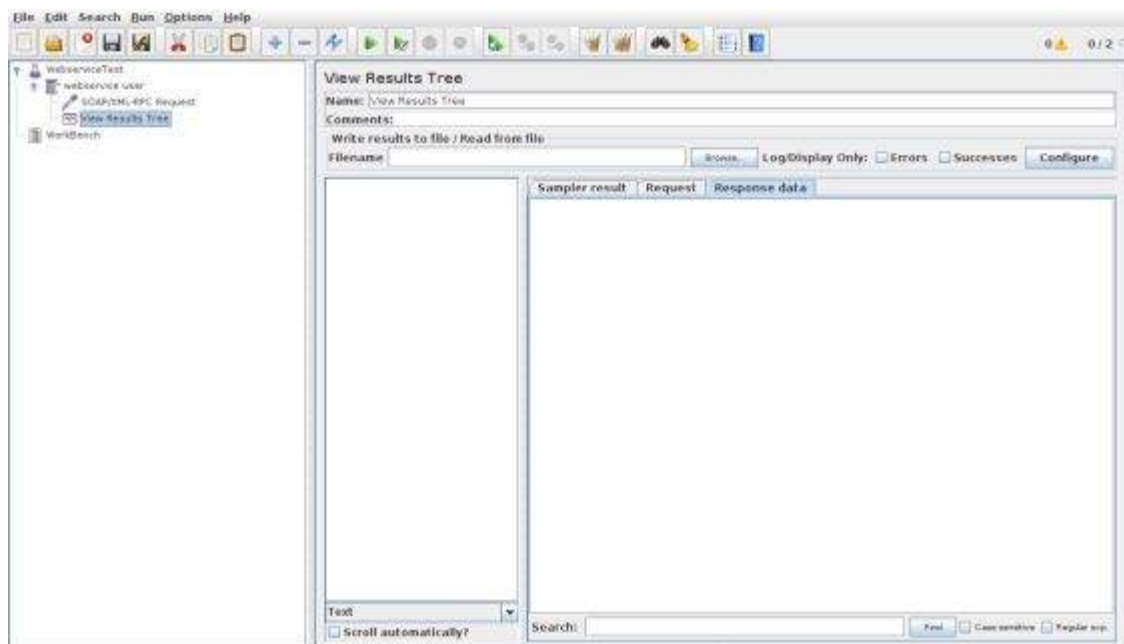
</soapenv:Envelope>
```



## 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.
- Add a View Results Tree listener by selecting Add > Listener > View Results Tree.

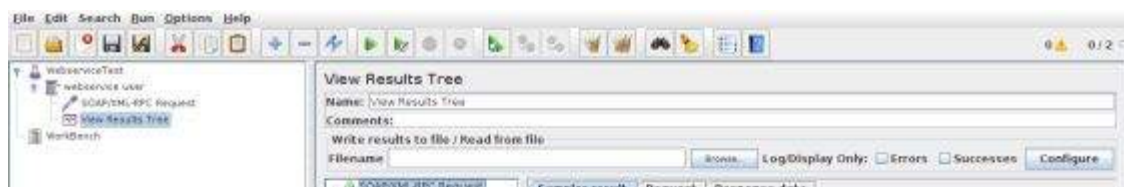


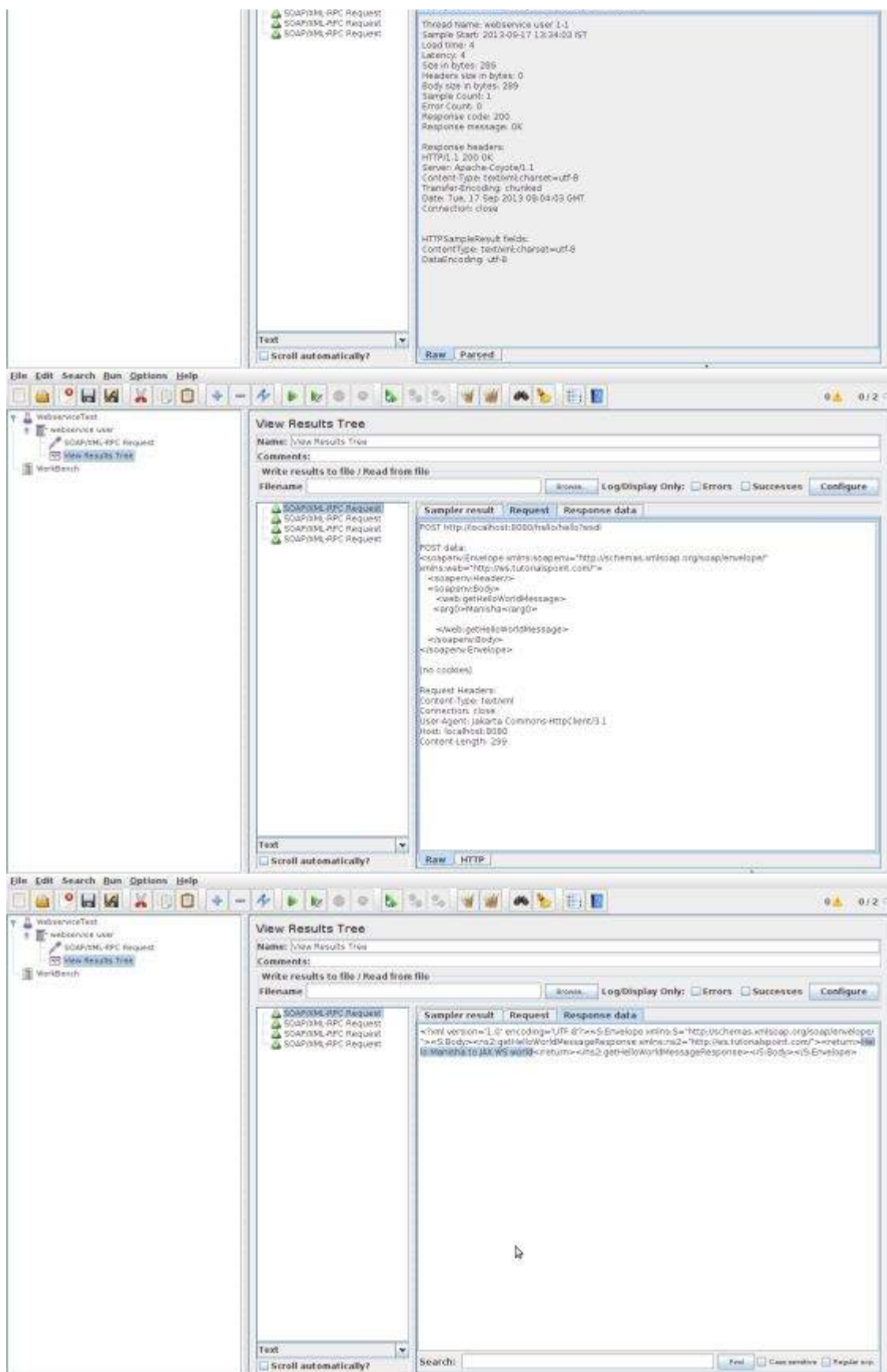
## Run the Test Plan

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

## View the 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".

## JMETER - JMS TEST PLAN

In this chapter, we will learn how to write a simple test plan to test Java Messaging Service *JMS*. 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.

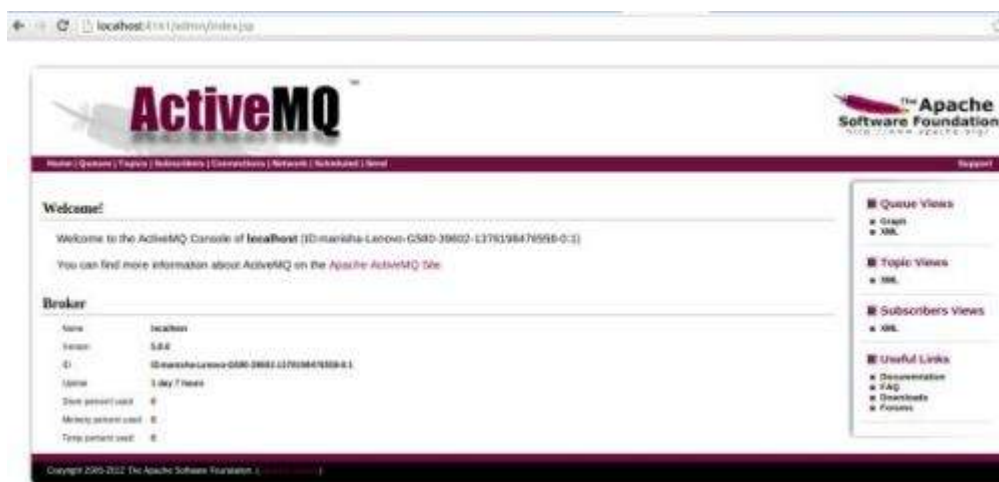
- **Topic messaging** – Topic messages are commonly known as pub/sub messaging. Topic messaging is generally used in cases where a message is published by a producer and consumed by multiple subscribers.

Let us see a test example for each of these. The pre-requisites for testing JMS are –

- We use Apache ActiveMQ in the example. There are various JMS servers like IBM WebSphere MQ *formerly MQSeries*, Tibco, etc. Download it from [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, then enter the userid and password as *admin*. The screen is similar as shown below –



- Now copy the `activemq-all-x.x.x.jar` *XXXdependingontheversion* from the ActiveMQ unzipped directory to **/home/manisha/apache-jmeter-2.9/lib**.

With the above setup, let us build the test plan for –

- [JMS Point-to-Point Test Plan](#)
- [JMS Topic Test Plan](#)

## JMETER - MONITOR TEST PLAN

In this chapter, we will discuss how to create a Test Plan using JMeter to monitor web servers. The uses of monitor tests are as follows –

- Monitors are useful for a stress testing and system management.
- Used with stress testing, the monitor provides additional information about server performance.
- Monitors make 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 need Tomcat 5 or above for monitoring. For our test purpose, we will monitor Tomcat 7.0.42 server. You can test any servlet container that supports Java Management Extension *JMX*. Let us write a test case to monitor the Tomcat server. Let us first set up our tomcat server.

## Setup Tomcat Server

We start with opening 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* as shown –

```
<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 following link in your browser –

<http://localhost:8080/manager/status?XML=true>

An authentication window appears in the browser. Enter the tomcat login and password associated *inourcaseitisadmin*. Then, the browser shows the execution status of Tomcat as below –

Tomcat Status

localhost:8080/manager/status?XML=true

Tomcat Status

JVM: free:131314688 total:211877888 max:900398104

Memory Pools

Name:PS Eden Space Type:Heap memory Initial:15794176 Committed:157810688 Maximum:311492608 Used:43453600

Name:PS Old Gen Type:Heap memory Initial:42205184 Committed:42205184 Maximum:675282944 Used:25253280

Name:PS Survivor Space Type:Heap memory Initial:2621440 Committed:11862016 Maximum:11862016 Used:11856320

Name:Code Cache Type:Non-heap memory Initial:2555904 Committed:2752512 Maximum:50331648 Used:2553536

Name:PS Perm Gen Type:Non-heap memory Initial:21757952 Committed:26673152 Maximum:90983232 Used:26522128

Connector -- "ajp-bio-8009"

threadInfo maxThreads:200 currentThreadCount:0 currentThreadsBusy:0

requestInfo maxTime:0 processingTime:0 requestCount:0 errorCount:0 bytesReceived:0 bytesSent:0

Stage Time B Sent B Recv Client VHost Request

Connector -- "http-bio-8080"

threadInfo maxThreads:200 currentThreadCount:4 currentThreadsBusy:1

requestInfo maxTime:420 processingTime:522 requestCount:2 errorCount:1 bytesReceived:0 bytesSent:4118

Stage Time B Sent B Recv Client VHost Request

5 2 0 0 127.0.0.1 localhost GET /manager/status?XML=true HTTP/1.1

From the above screenshot, we can note a few things –

- In the URL, note that *XML = true* *notethecasesensitivity* 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

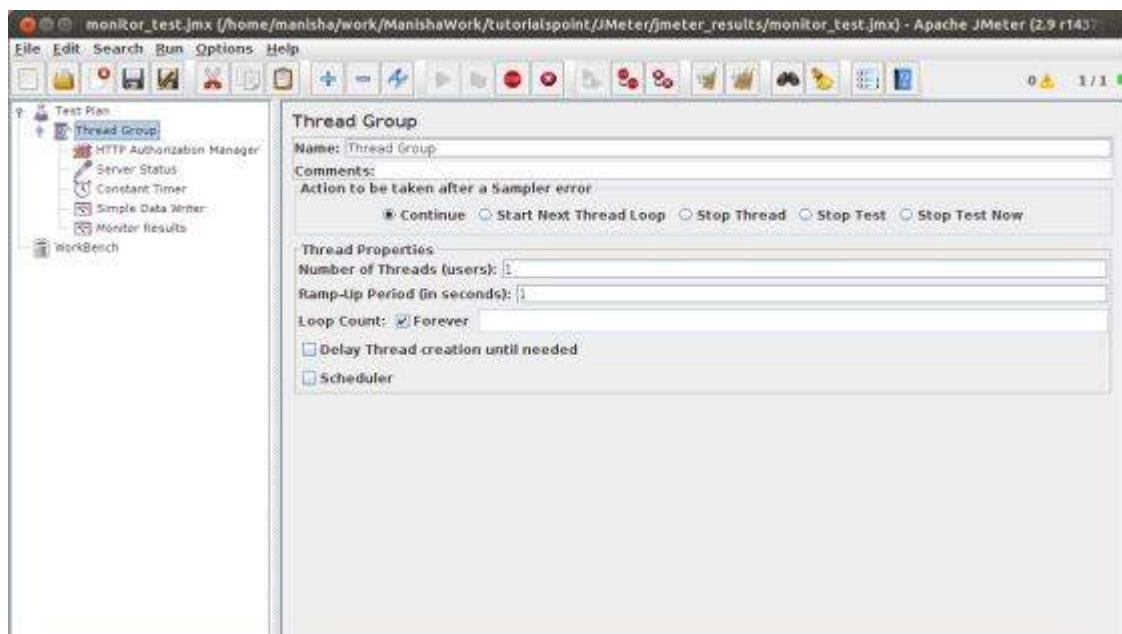
Let us monitor the Tomcat server by writing a test plan –

## Rename Test Plan

- Open the JMeter window by clicking /home/manisha/apache-jmeter2.9/bin/jmeter.sh.
- Click the Test Plan node.
- Add a thread group as explained in the next step.

## Add Thread Group

- Right-click on **Test Plan > Add > ThreadsUsers > Thread Group**. Thread Group will get added under the Test Plan node.
- Change the loop count to forever *orsomelargenumber* so that enough samples are generated.

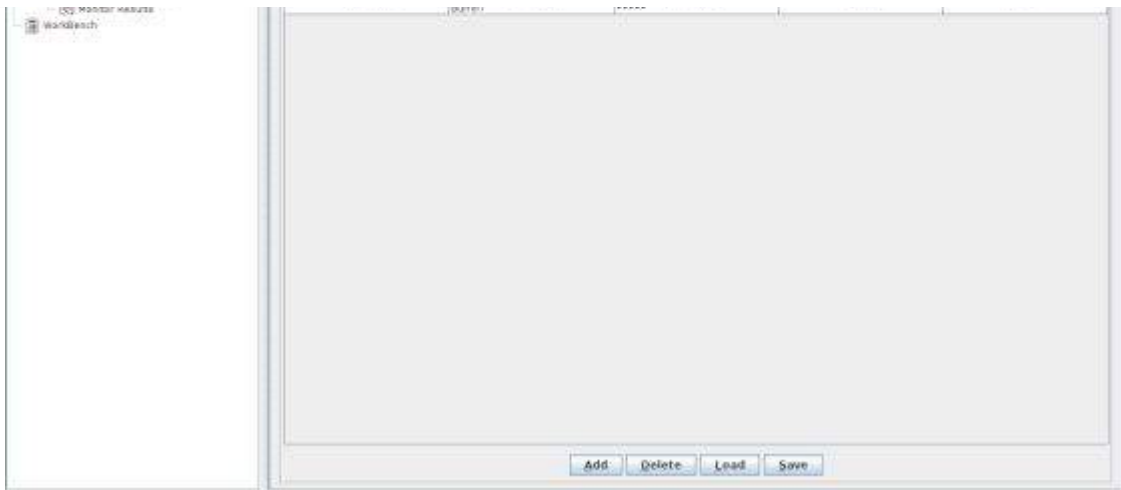


## HTTP Authorization Manager

- Add HTTP Authorization Manager to the Thread Group element by selecting 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.
- Edit the following details –
  - **Username** – admin *dependingontheconfigurationintomcat – users.xmlfile*
  - **Password** – admin *dependingontheconfigurationinthetomcatusers.xmlfile*
  - The other fields are left empty.



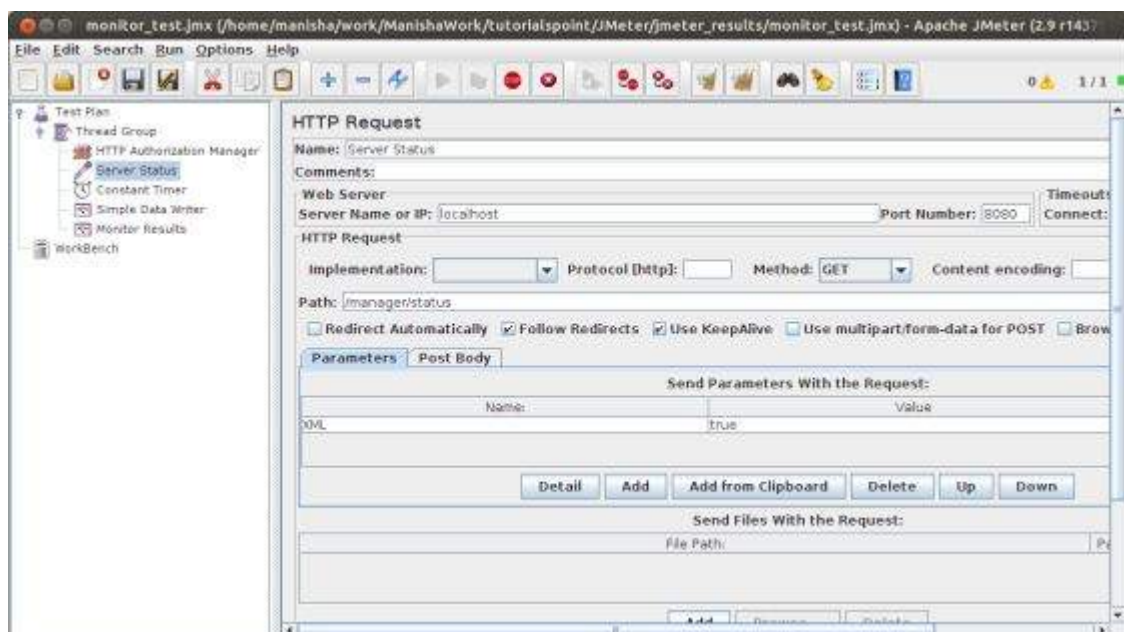




## Add Sampler-HTTP Request

Now that we have defined our users, it is time to define the tasks that they will be performing. We add HTTP Request element.

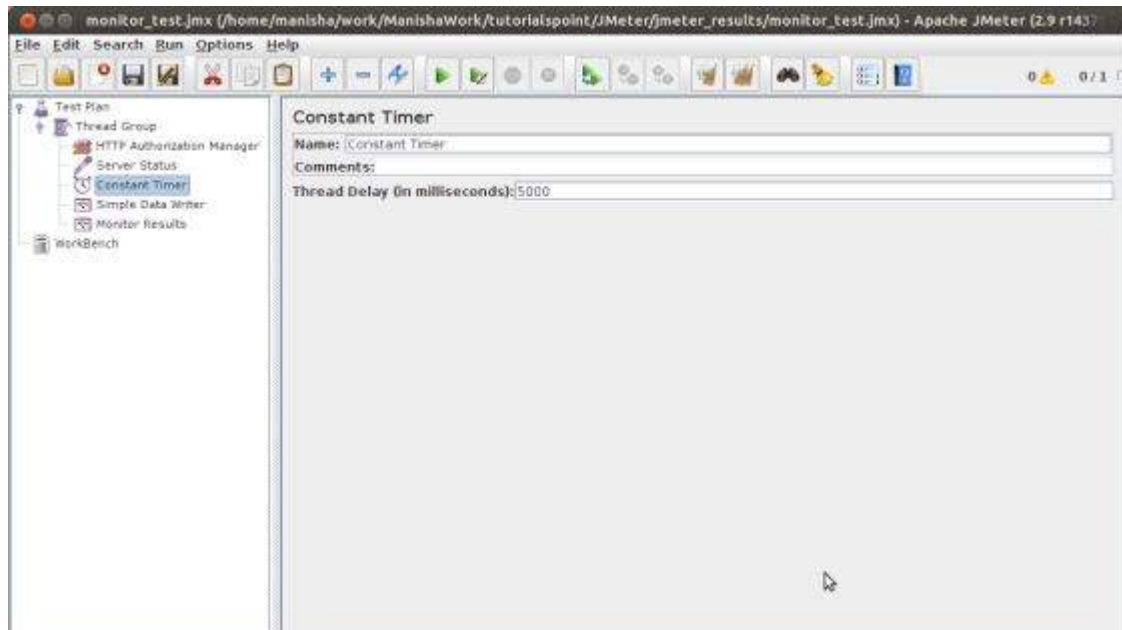
- Right click the mouse button to get the Add menu.
- Select Add > Sampler > HTTP Request.
- Then, select the HTTP Request element in the tree.
- Edit the following properties as in the image below –
- The following details are entered in this element –
  - **Name** – Server Status
  - **Server Name 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 a Constant Timer

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

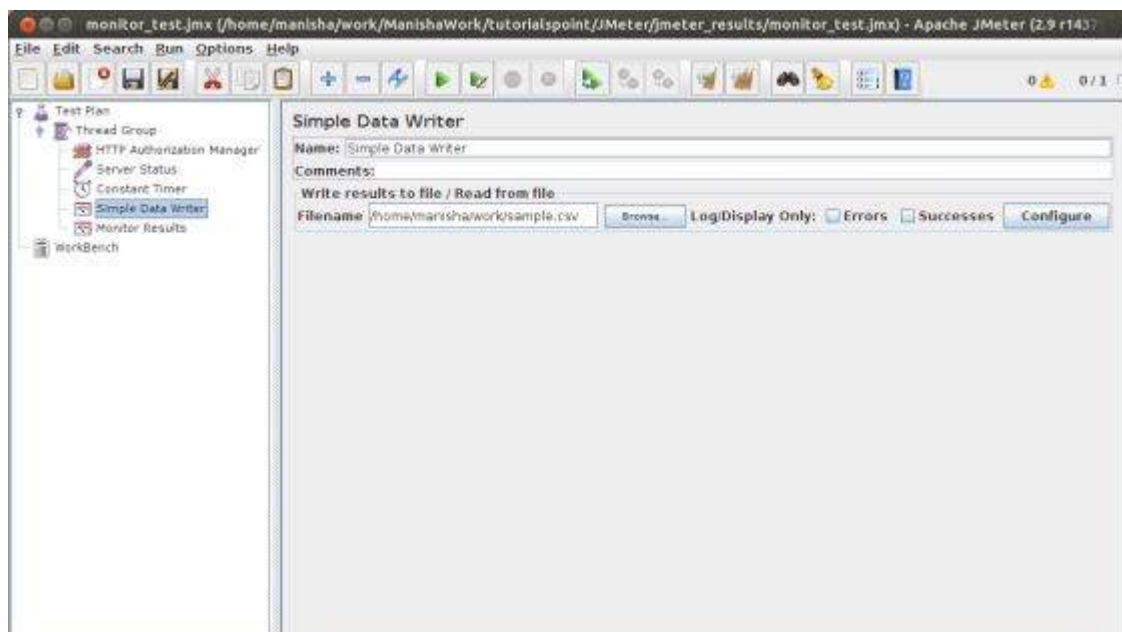
Enter 5000 milliseconds in the *Thread Delay* box. In general, using intervals shorter than 5 seconds may add stress to your server. Find out what is an acceptable interval before you deploy the monitor in your production environment.



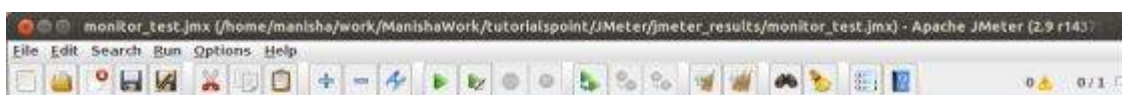
## Add Listener

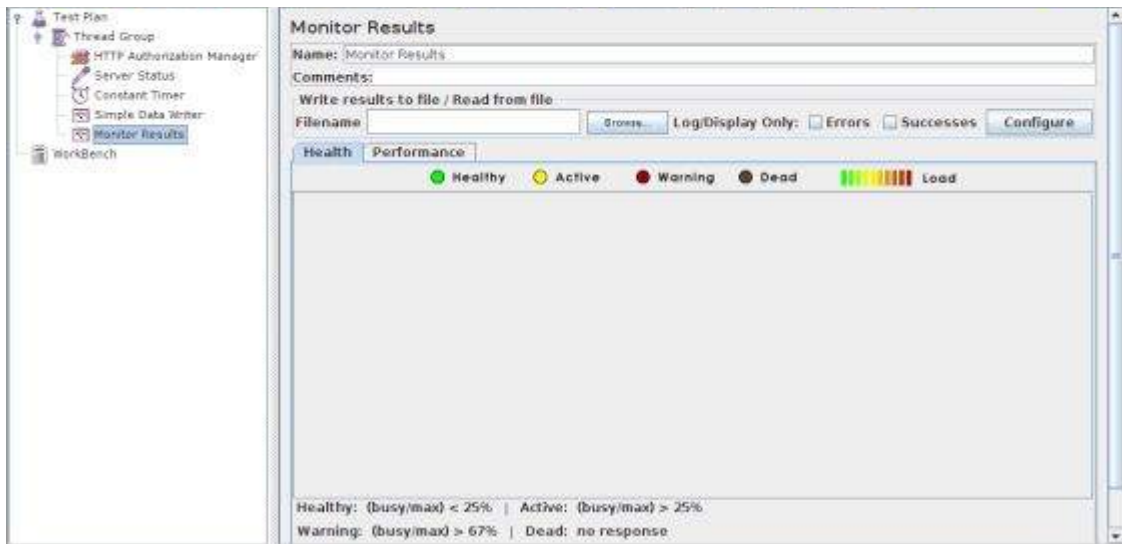
The final element you need to add to your Test Plan is a Listener. We add two types of listeners. One that stores results in a file and second that shows the graphical view of the results.

- Select the thread group element.
- Add a Simple Data Writer listener Add > Listener > Simple Data Writer.
- Specify a directory and filename of the output file *inourcase, itis/home/manisha/work/sample.csv*



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



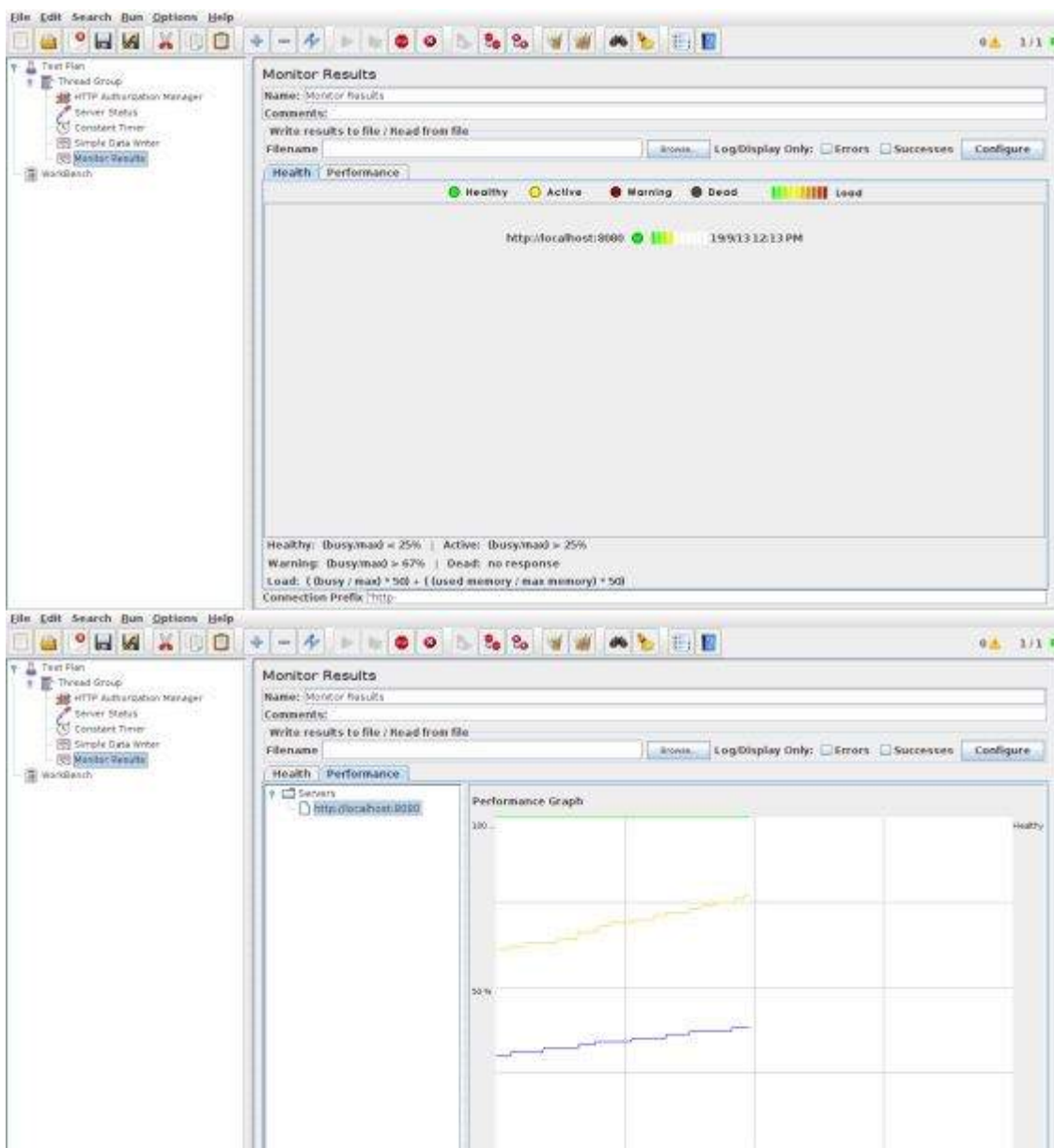


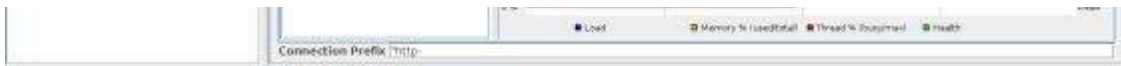
## Run the Test Plan

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

## View the 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 Optimizelt 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.

## JMETER - 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.

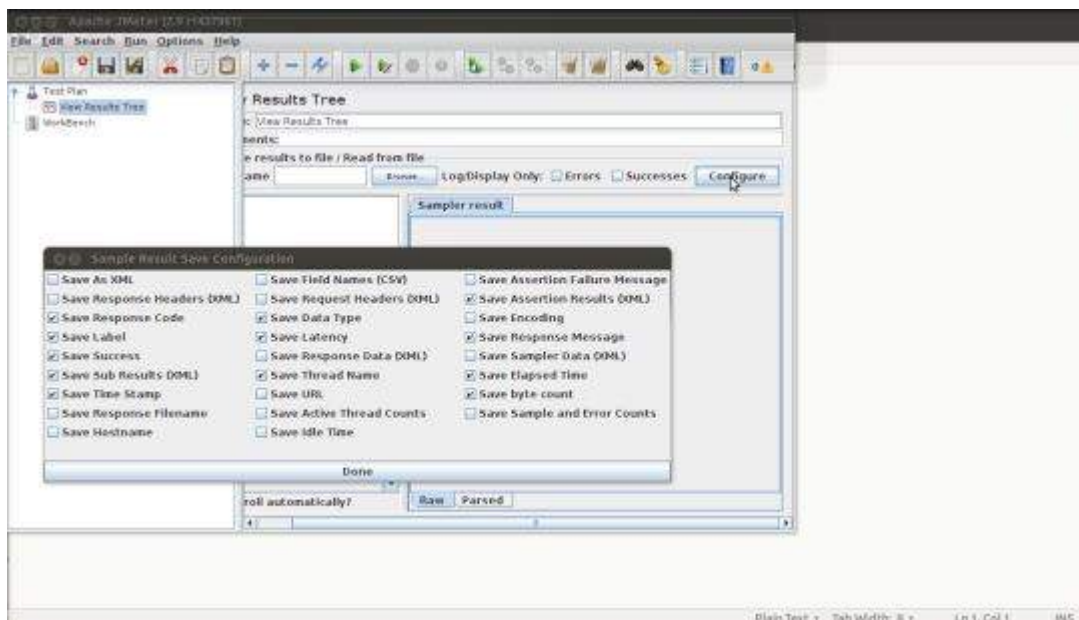
### Default Configuration

The default items to be saved can be defined in one of the following two ways –

- 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=
```

- By using the Config popup as shown in the following screenshot –



JMeter creates results of a test run as JMeter Text Logs *JTL*. 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 at 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 minimize the amount of memory needed, use the Simple Data Write with 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 <i>1, unless multiplesamplesareaggregated</i>	1
ErrorCount	number of errors <i>0or1, unless multiplesamplesareaggregated</i>	0
Hostname	where the sample was generated	LaptopManisha
IdleTime	number of milliseconds of 'Idle' time <i>normally0</i>	
Variables	if specified	

## Saving Response Data

The response data can be saved in the XML log file if required. However it does not allow to save large files and images. 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 the Listener GUI Data

JMeter is capable of saving any listener as a PNG file. To do so,

- Select the listener in the left panel by selecting Edit > Save As Image. A file dialog appears.
- Enter the desired name.
- Save the listener.

# JMETER - 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 make sure you 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 –

Function Type	Name	Comment
Information	threadNum	Get thread number.
Information	samplerName	Get the sampler name <i>label</i> .
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 <i>ordisplay</i> a message <i>andreturnthevalue</i> .
Information	logn	Log <i>ordisplay</i> a message <i>emptyreturnvalue</i> .
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 <i>MozillaRhino</i> .
Scripting	jexl, jexl2	Evaluate a Commons Jexl expression.
Properties	property	Read a property.
Properties	P	Read a property <i>shorthandmethod</i> .
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. <code>\n &amp; \t</code> .
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.
- The variables cannot be nested; i.e. `{Var{N}}` does not work.
- The `__V` variable function *versionsafter2.2* can be used to do this – `{__V(Var{N})}`.
- This type of replacement is possible without functions, but it is less convenient and less

intuitive.

## Where to Use Functions And Variables

Functions and variables can be written into any field of any test component.

The following functions should work well in a 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. Hence, *split* and *regex* and the variable evaluation functions will not work. The *threadNum* function will not work and it 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 example –

```
${__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 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, which you can copy-paste into the test plan wherever you need to.

## 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.

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.

## JMETER - 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 of 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. Hence the '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 extracts text from the current page using a Regular Expression to identify the text pattern that a desired element conforms with.

First we write an HTML page which a list of people and their email IDs. We 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 >3</td>
            <td >Manisha</td>
            <td >manisha@domain.com</td>
        </tr>

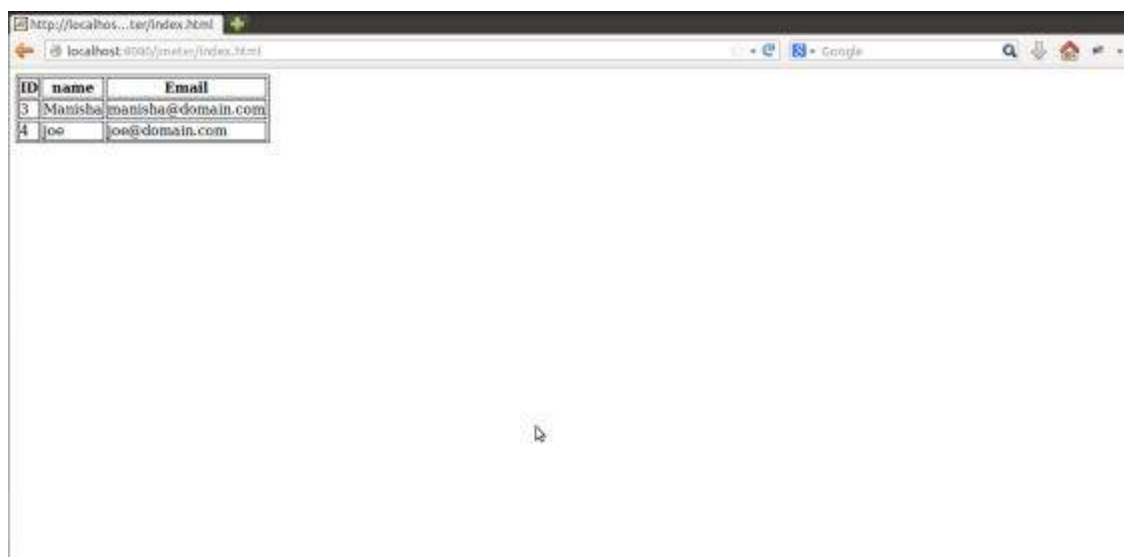
        <tr>
            <td >4</td>
            <td >joe</td>
            <td >joe@domain.com</td>
        </tr>

    </table>

</body>

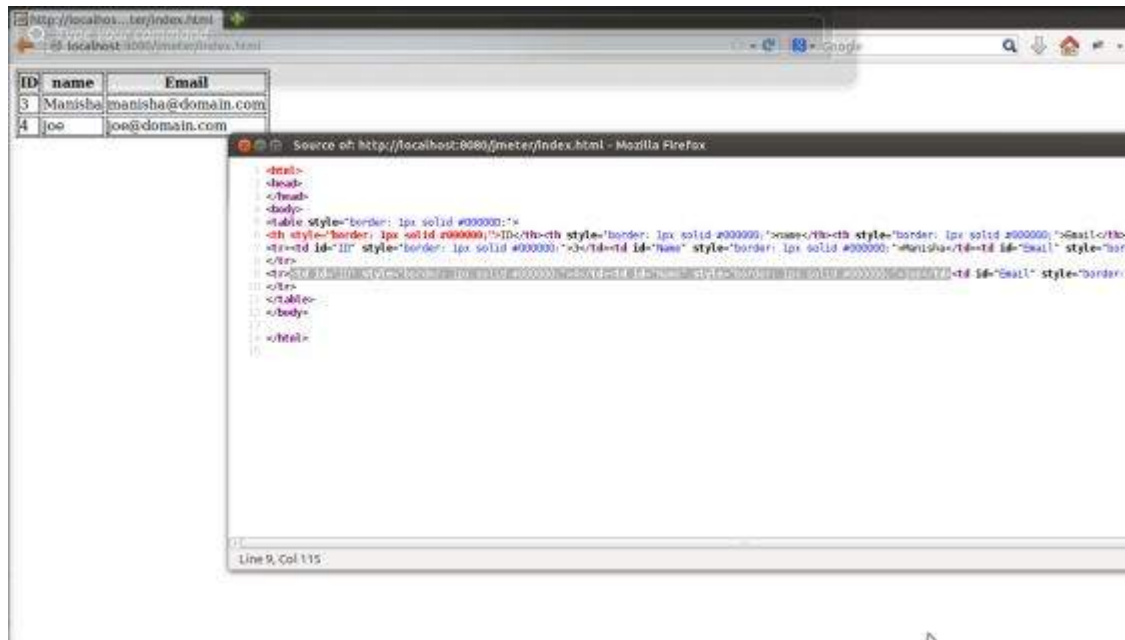
</html>
```

On deploying it on the tomcat server, this page would look like as shown in the following screenshot –



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.

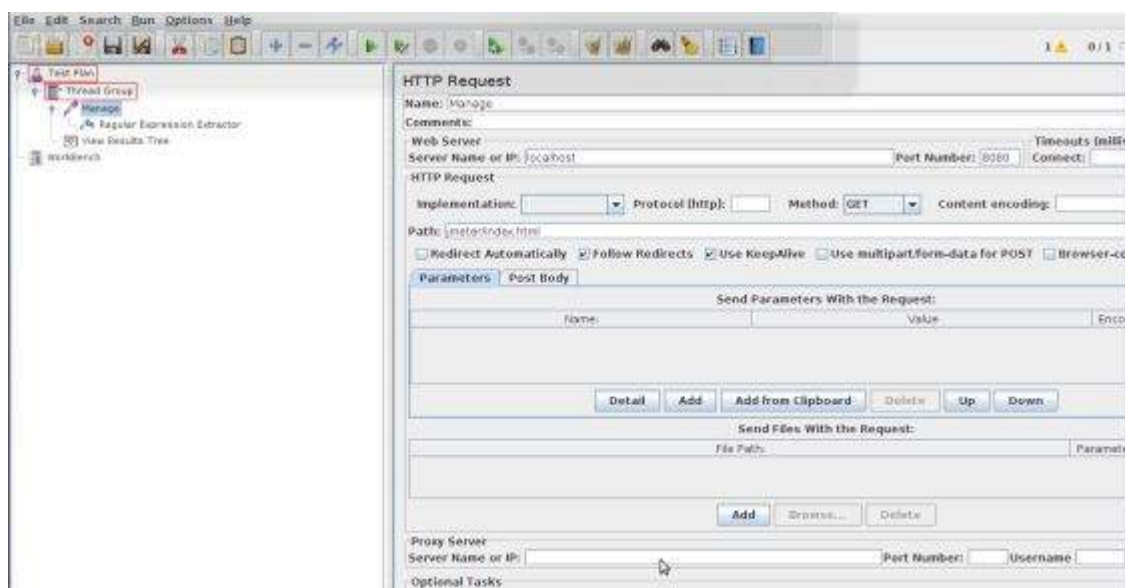
As can be seen in the following snapshot, the ID of the second person is surrounded by `<td>` and `</td>`, and it is the second row of data having this pattern. We can use this to match the exact pattern that we want to extract information from. As we want to extract two pieces of information from this page, the person ID and the person name, the fields are defined as follows –

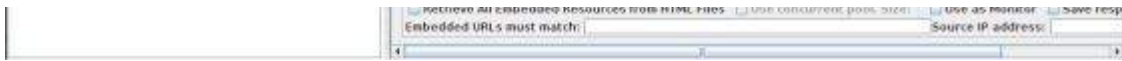


Start JMeter, add a Thread group **Test Plan > Add> ThreadsUsers> Thread Group**.

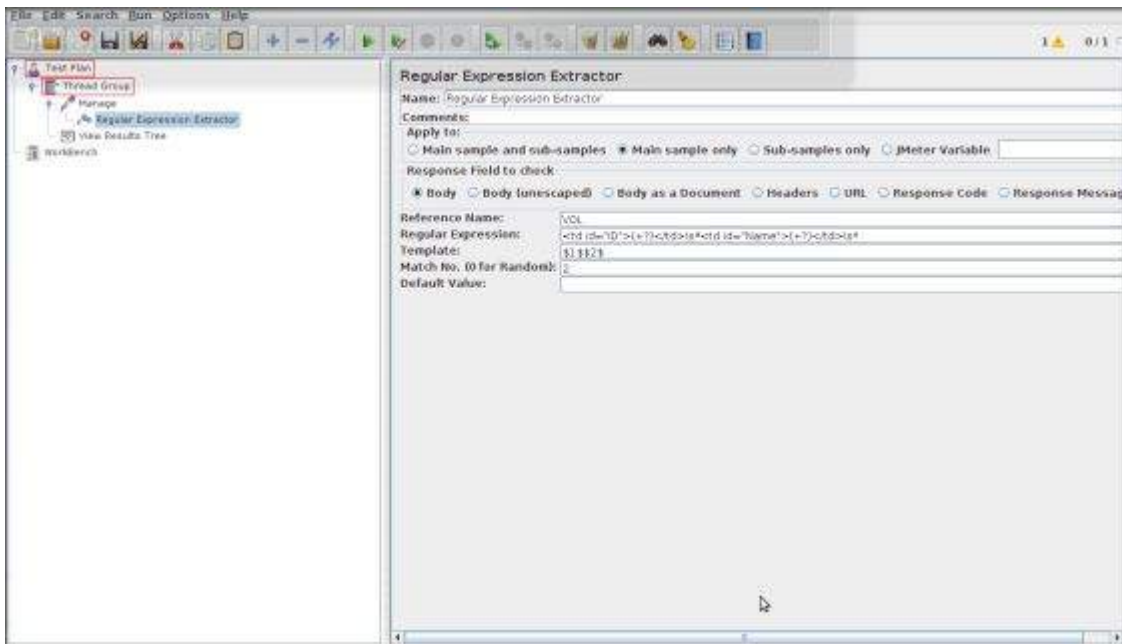
Next add a sampler HTTP Request, select the test plan, right click **Add > Sampler > HTTP Request** and enter the details as shown below –

- **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**.



The following table provides a description of the fields used in the above screenshot –

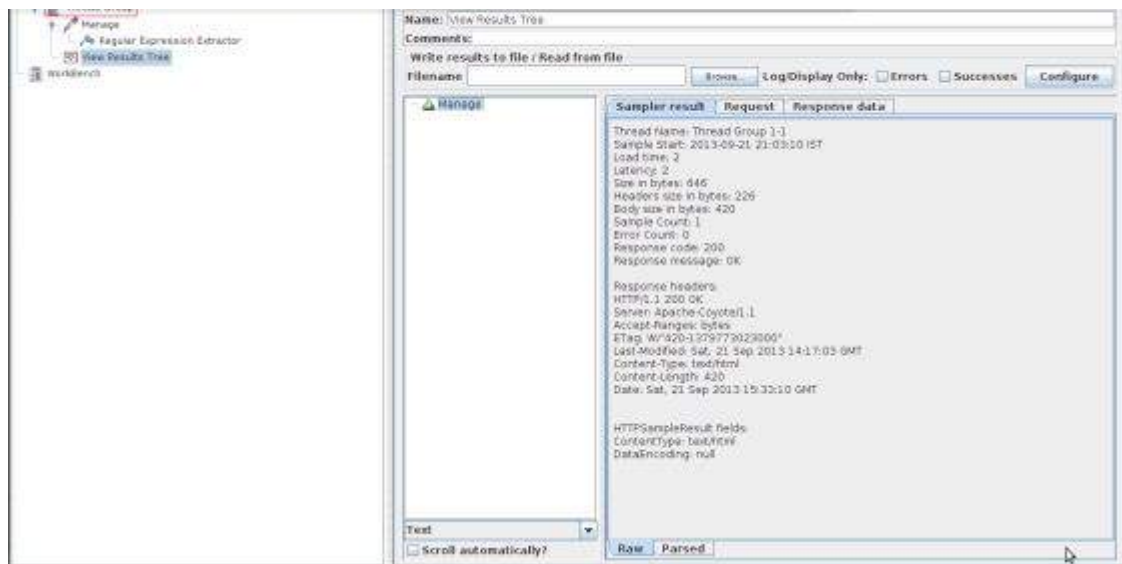
Field	Description
Reference Name	The name of the variable in which the extracted test will be stored <i>refname</i> .
Regular Expression	The pattern against which the text to be extracted will be matched. The text groups that will be extracted are enclosed by the characters ' <i>and</i> '. We use ' <i>.+?</i> ' to indicate a single instance of the text enclosed by the <code>&lt;td&gt;..&lt;/td&gt;</code> tags. In our example the expression is – <code>&lt;td &gt;+?&lt;/td&gt;\\s*</code>
Template	Each group of extracted text placed as a member of the variable <i>Person</i> , following the order of each group of pattern enclosed by ' <i>and</i> '. Each group is stored as <code>refname_g#</code> , where <code>refname</code> is the string you entered as the reference name, and <code>#</code> 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 is maintained in <code>Person_g1</code> , while the Name value is stored in <code>Person_g2</code> .
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 <i>ForEach Controller</i> .
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 the Thread Group and select **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 shown in the following screenshot –







## JMETER - 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.
- Here 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 `CSVDataSet` 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.
  - Don't forget to erase the local path from CSV Data Set Config if used.
  - Clean the Files tab prior to every test run.

Loading [MathJax]/jax/output/HTML-CSS/jax.js