**Introduction to JMeter**

Today, in the digital era, where websites and their users are increasing drastically, **seamless user experience** is a must-have for organizations because **unnecessary delays in the response** of the website or application distract the user's attention.

Now we have greater expectations of the software we use than ever before. This is the number one reason why performance testing has become so important.

Research suggests that just a 1 second delay in page load time results in 7% fewer conversions, 11% fewer page views, and a 16% decrease in customer satisfaction. And this translates to real dollars – if your site earns $100,000 per day, you’re losing $2.5 million every year due to this 1 second.

Therefore, organizations prefer doing performance and load tests on application before launching them or after every major release.

***What is performance testing?***

***Performance testing is a non-functional*** type of testing which is used to evaluate ***application under test*** (*AUT*) performance in terms of its responsiveness as well as the behaviour of AUT under the various workload.

An application is performant if it lets the end-user carry out a given task without undue delay or irritation.

An application which is performing well always have a **good response time**. So, responsiveness of an application plays a very important role just like it's user interface and functionality because nowadays users are very demanding and want everything quick.

**A slow website results into a bad user experience and have negative financial impact.** Even delay of second, for longer period of time, may result into huge revenue loss. Thus, performance test of websites comes into picture. To performance test a website there are tools available, for example: ***JMeter, LoadRunner, WebLoad, LoadView*** and many more.

***Importance of Performance Testing***

* *Helps in evaluating potential bottlenecks of an AUT*.
* *Slowness of an application or web service can be evaluated under heavy load.*
* *We can find out how many parallel users an AUT can handle.*
* *Helps in finding out impact of changes in each release in terms of performance*.

***Note: An application or software system which is when tested is called AUT or Application Under Test***..

***What is JMeter and Its Brief History?***

[***Apache JMeter***](https://jmeter.apache.org/) **is an open-source, pure Java platform software** which is designed to **load test functional behaviour and measure performance**.

Initially, JMeter was introduced for load and performance test web applications, but later on its scope has widened and can perform load and performance test on Web Pages, Web Applications and static or dynamic resources Like ***Database, Rest Webservices, LDAP, Java Objects*** and more.

**Features of JMeter**

***Key features of JMeter includes:***

* ***License****: Since* ***JMeter is open source****, it is free and easily available*.
* ***Graphical User Interface****: Simple, user friendly and easy to learn as compared to other performance testing tools*
* ***Server/ Protocol Suppor****t: JMeter has ability to load and performance test different applications/server/protocols. A few protocols inclucdes* ***HTTP, HTTPS, FTP, SOAP/REST, Database via JDBS, LDAP, JMS, SMTP(S), POP(3) and IMAP(S), Native Commands/ Shell Scripts and TCP*.**
* ***Platform****: JMeter is pure java software. Therefore, it is platform-independent and supports all environment.*
* ***Simulation****: Simulate multiple users by using virtual users or unique users in order to generate heavy traffic on web server or services*.
* ***Supports Distributed Testing****: It has master slave for distributed testing where master will distribute tests among all slaves and slaves will execute scripts against your server.*
* ***Test Result Visualisation****: Test result can b view in different formats like Graph, Table, Tree, and Report, etc.*
* ***Reporting:****By default, JMeter provides XML and CVS Report Formats only. We can use Jmeter and ANT together to obtain HTML report as per the requirement.*
* ***Testing Types****: Apart from just Performance, Load, Stress testing Jmeter works well for Functional, Regression and Soak/Endurance testing too*.
* ***Record and Playback:****Record user scenario/action in Firefox Browser and play scripts*
* ***Framework****: Multi-threading framework allows concurrent and simultaneous sampling of different functions by many or separate thread groups*.
* ***Installation****: No Complex installation required – Just run JMeter.bat on windows / run Jmeter.sh on Linux*
* ***Knowledge****: Jmeter does not require extensive programming knowledge. Only prior knowledge Java language is preferred.*

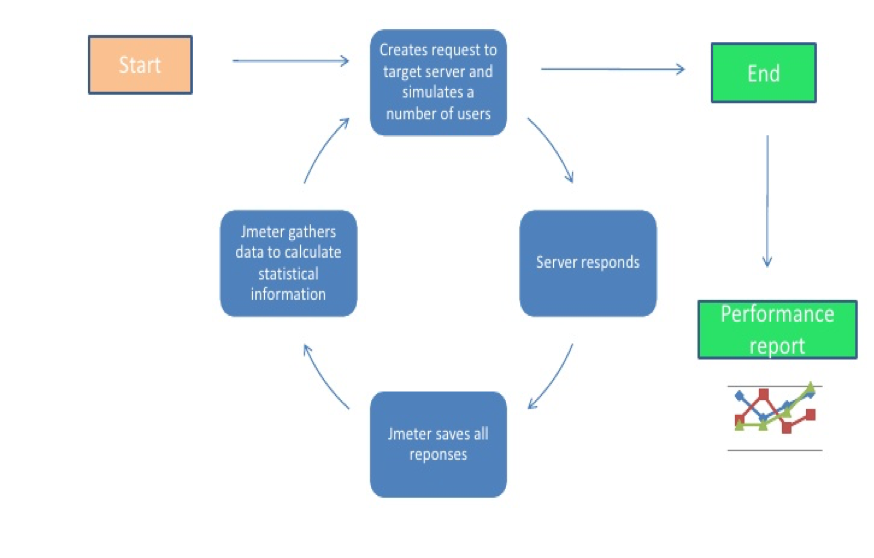
**How JMeter Works**

**JMeter simulates a number of users sending request to the Application under Test**.  As soon as JMeter simulates requests, server responds and Jmeter starts collecting data. Jmeter saves all the responses and based on the server response it return statistics. These statistics shows performance of the AUT in the form of various formats as per the requirements.

Thus, with the help of JMeter we can simulate load on server, network or objects which are coming from different machines to implement real world scenario.

***JMeter Workflow***

When we start load or perform test of an application, JMeter creates requests to target server and simulates number of users sending requests to the target server. As soon as server starts responding to the requests, JMeter starts saving all the responses. On the basis of data/response JMeter gathers data to calculate statistical information. Finally, using this statistical information JMeter prepares a report which tells about performance of the AUT.

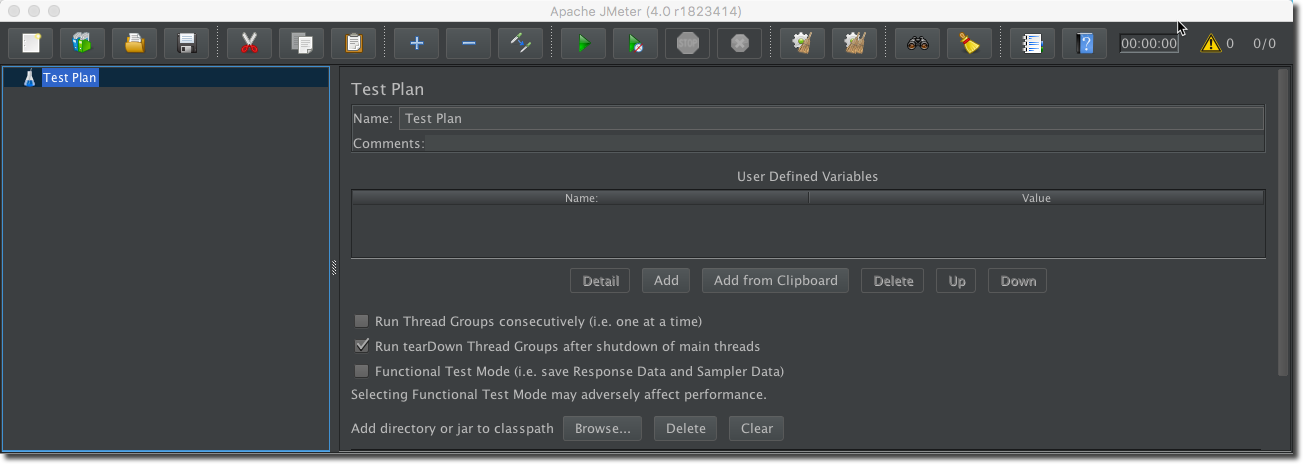


So, this was a brief about what is JMeter and how is it useful for us. In the upcoming section we will be installing this open source, user friendly software and slowly we will try to get hands on this performance testing tool.

How to launch Jmeter

To launch ***JMeter on Windows just double click the jmeter.bat*** file or go to ***command prompt*** and type ***<file path>/apache-jmeter-4.0/bin/jmeter.bat*** and wait for a few seconds JMeter GUI will be launched.

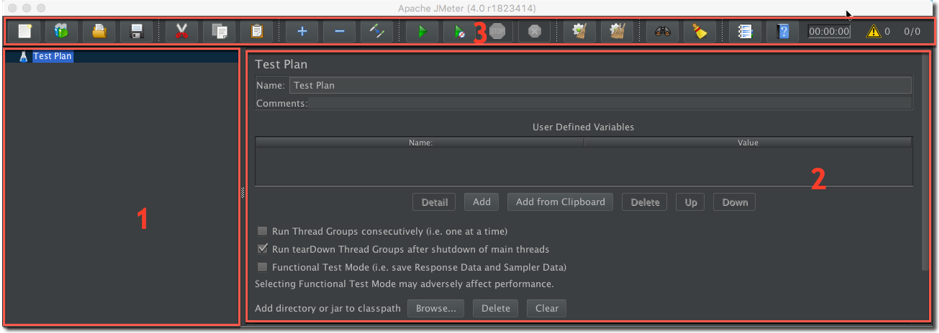
# Getting Familiar with JMeter GUI



JMeter is primarily divided into three major parts:

* ***Left Pane***: Left pane is the place where the test you want to execute resides.
* ***Configuration Window***: In this window we set the configuration and control the behaviour of the tests that we want to execute.
* ***Menu Bar***: It is an intuitive menu bar from where you may perform all the functions.

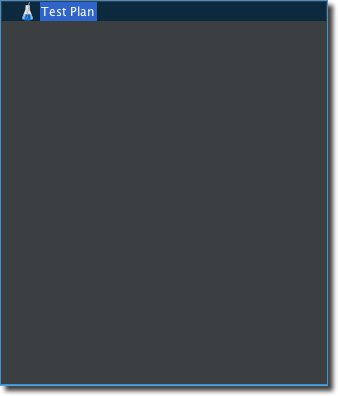
The ***options menu*** has been removed from the GUI in JMeter 4.0. So, the user may perform nearly all the functions using the buttons present in the ***Menu Bar***.



## Left Pane

Left pane in JMeter consists of a node known as ***Test Plan***. As the name suggests, Test Plan can be thought of as a container that consists of test scenarios as well as test data.

Similarly, JMeter Test Plan also can be thought of as a container that consists of series of steps that will execute when we run a test plan. Test plan can consist of one or more elements like ***Thread Groups, Logic Controllers, Configuration Elements, Samplers, Timers, Listeners and Assertions***.

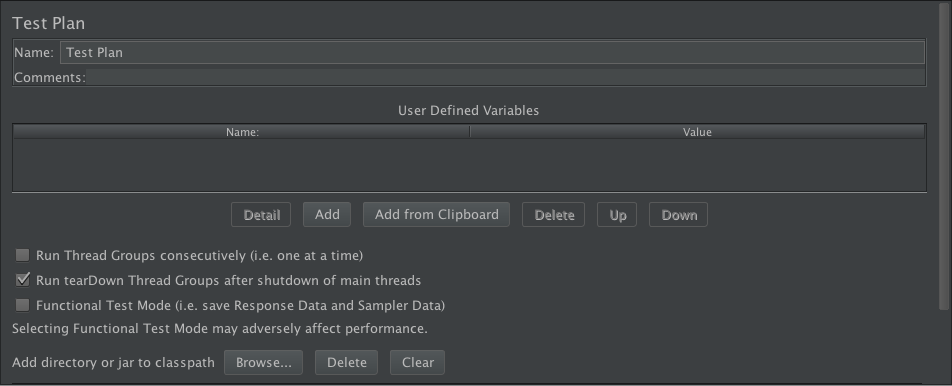


***Note*: Only one test plan is executed at a time in JMeter.**

## Configuration Pane

You can **configure** your Test Plan and its element on this window. A Test Plan’s name, user defined variables and its properties can be configured here. Configuring a Test Plan helps you to control its properties and the behaviour in which you would like to run your test Plan as per your requirement.

By default the below screen is visible when we launch JMeter.

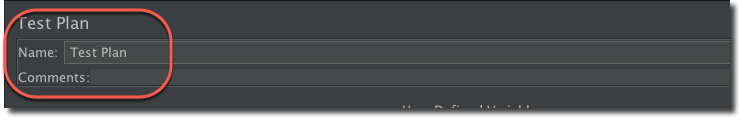


The above screen consists of three major parts:

* ***Name***
* ***User Defined Variables***
* ***Test Plan Properties.***

#### **Name**

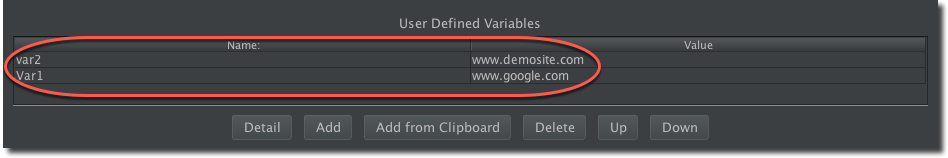
This shows the name of your test plan. You may change the name of your test plan here. We will be discussing how to create, name and save a Test Plan in our next tutorial.



#### **User- Defined Variables**

User-defined Variables are nothing but ***name-value pairs***.

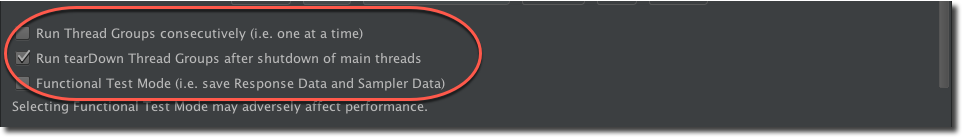
You can add a variable by clicking on ***Add button***. You can add multiple name-value pairs either by clicking ***Add button*** every time or by using ***Add from Clipboard*** button. The following screen shows the added variables along with their values.



#### **Test Plan Properties**

There are three major configuration properties of a Test Plan which you may use to control test plan’s behaviour as per requirement. The three properties are:

* ***Run Thread Groups Consecutively(i.e. one at a time)***
* ***Run tearDown Thread Groups after shutdown of main threads***
* ***Functional Test Mode(i.e. save Response Data and Sampler Data)***



***Run Thread Groups Consecutively (i.e. one at a time)***

A Test Plan can have one or more Thread Group. It should have at least one Thread Group (which is the minimum requirement) but can have more than one too. If a Test Plan has more than one thread group then they will execute one after the other if this checkbox is checked. If this checkbox remains unchecked then all the thread groups will execute in parallel.

***Run teardown Thread Groups after shutdown of main threads***

TearDown threads execute after the test has finished executing its regular thread group. If this checkbox is checked, then this thread will run after the test execution. The tearDown feature is generally used for reporting or cleaning operations. ***Example***, If you want your logs to clean automatically after the execution of a Test Plan or want your reports to be in particular format then you may use this property of the Test Plan.

#### **Functional Test Mode (i.e. save Response Data and Sampler Data)**

If this checkbox is selected, then sampler requests and response data are saved in the listeners. Do not worry about the listeners and other technical terms here. We will discuss everything in the later tutorial and this tutorial is just to get you familiar with the GUI of JMeter. This checkbox allows you to verify that the test is working as expected.

***Note: A Thread Group is a child element of a Test Plan. Each Thread Group represents a use-case. (We will discuss about Thread Group in our upcoming tutorials)***

## Menu Bar

This is the upper most bar present on the JMeter GUI. It has many buttons which helps us to perform various function just by clicking the respective button.

We can perform many operations using this above menu bar but discussing all of them here is not necessary. Hence we will only be discussing the most important one. If any other function comes later in the course, we will detail it there.

MenubAr

The below images shows the mostly used icons followed by their working.

FirstMenu

* ***New***: Using New menu item you can create a new Test Plan.
* ***Open***: To open an existing Test Plan you may use open button present in the menu bar.
* ***Save***: To save the Test plan or its elements you can simply click this button. As you will click this button a prompt appears

SecondMenu

* ***Start***:  Start icon used to execute the test. After creating a test plan and adding elements to it you may simply click this button to start the execution of your test.
* ***Start with no pauses***: Like Start we can use Start with no pauses to execute the test. Difference between both is that in case of Start No Pause timer configured in the thread group are skipped and the thread will run without any manual pauses.
* ***Stop***: As the name suggests, you can use Stop to abruptly stop the test that is running.
* ***Shutdown***: Shutdown also stops the execution but gracefully. When you shut down the running test it does not stop it immediately but allow the running threads to wind down.

ThirdMenu

* ***Clear All***: You can clear the log window by clicking Clear All button.
* ***Elapsed time of current running test***: Shows the time taken to execute the test.
* ***Show Number of Errors in log***: On clicking this menu option you will get the logs at the bottom of the configuration window.
* ***Running thread/total number of threads***:  this option tells you how many threads are currently running out of the total number of threads.

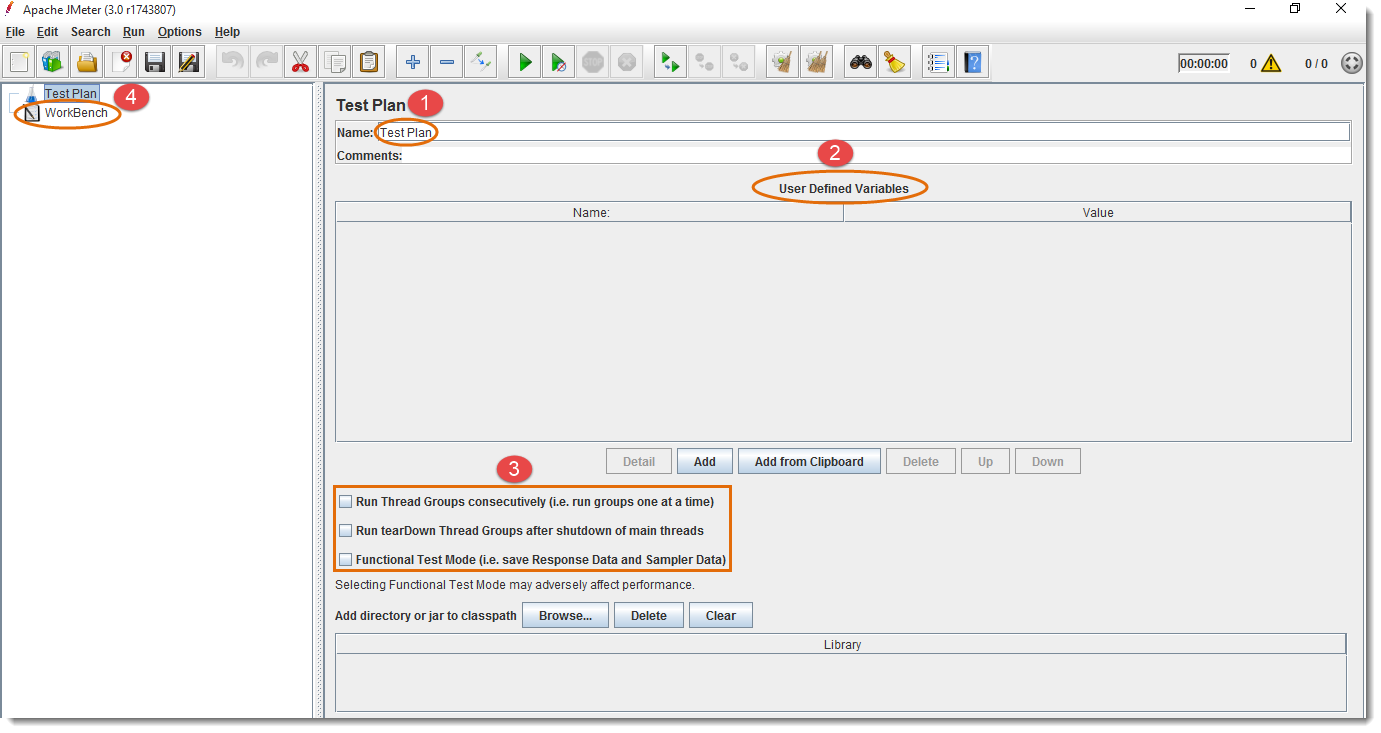
Hence, in this tutorial we learned about ***Getting Familiar with JMeter GUI*** and got to know in detail what is a test plan and what are the properties of a test plan. In the next tutorial we will be learning  about how to perform some of the basic operations in JMeter.

# Build JMeter Test Plan

A ***Test Plan*** can be considered **as a root node** that has potentially many levels of additional elements / nodes that forms a hierarchy, which *JMeter* executes when we run test plan.

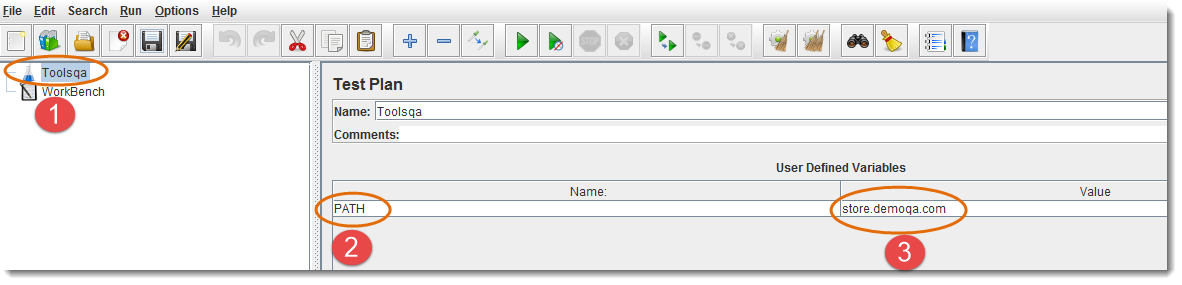
A test plan can consist of one or more elements like ***Thread Groups, Logic Controllers, Configuration Elements, Timers, Listeners and Assertions.*** T

here should be at least one thread group in every test plan. We can add or remove elements from test plan easily.



1. ***Rename****your test plan*
2. ***User Defined Variables provides flexibility when you have to repeat any value in several parts of the test plan****. You will be able to change the value from a single point, and it will reflect everywhere.*
3. *Optional – or you can use according to your needs*
4. *WorkBench is used to store test elements temporary for copy/paste. It is also used for Non-Test Elements like HTTP(S) Test Script Recorder, HTTP Mirror Server and Property Display. If you need to save it along with the test plan, then you should select the option “****Save WorkBench****” from WorkBench control panel.*

After renaming test plan and defining variable, JMeter will look like as follows:



1. *New test plan name will also reflect on left side*
2. *E.g. Variable name is set to “****PATH****”*
3. *E.g. Variable value is set to URL (without http://): “****store.demoqa.com****”*

*Now, this variable can be used in any part of the test plan elements like this ${PATH}*

Look at the following example.

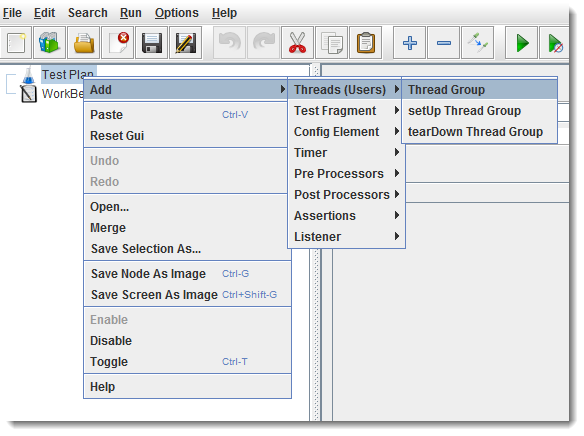
TestPlan_3

**Add and Remove Test Plan Elements**

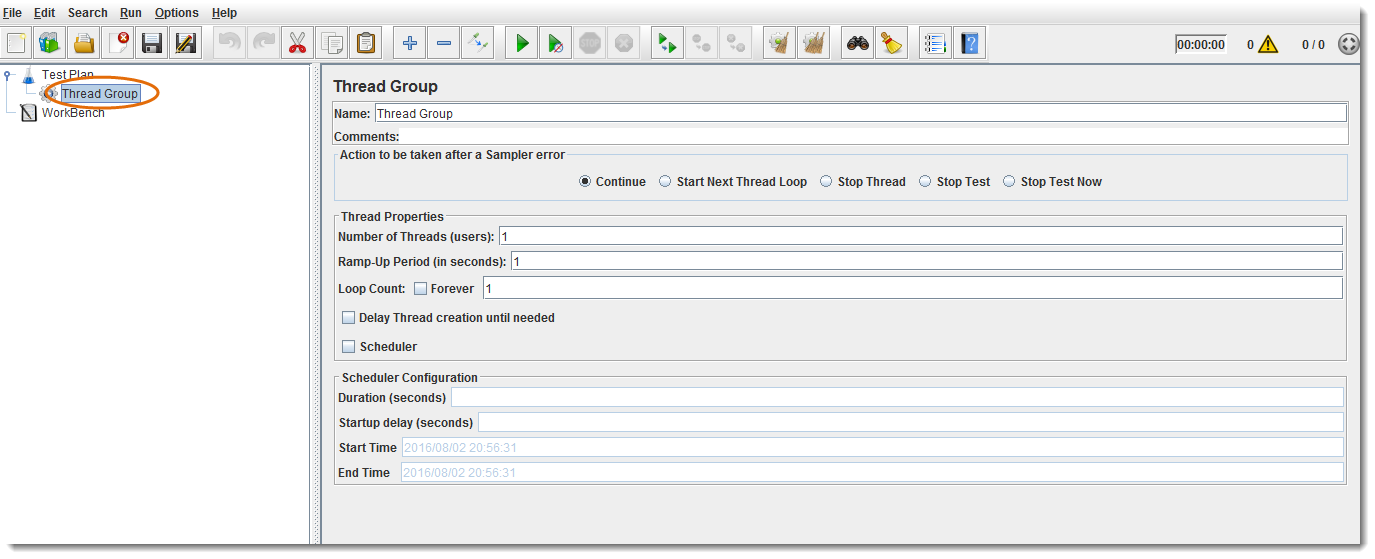
Once the *Test Plan for JMeter* is created, the next thing we should be learning is to ***Add and Remove Elements to JMeter Test Plan.***

***Steps to Add Elements to JMeter Test Plan***

1. *Select test plan node or any element*
2. *Right click the selected element*
3. *Mouse hover on “****Add****” option, then Elements list will be displayed*
4. *Mouse hover on desired list element, and select appropriate option by clicking it*



After adding Thread Group element, you can see the following screen:

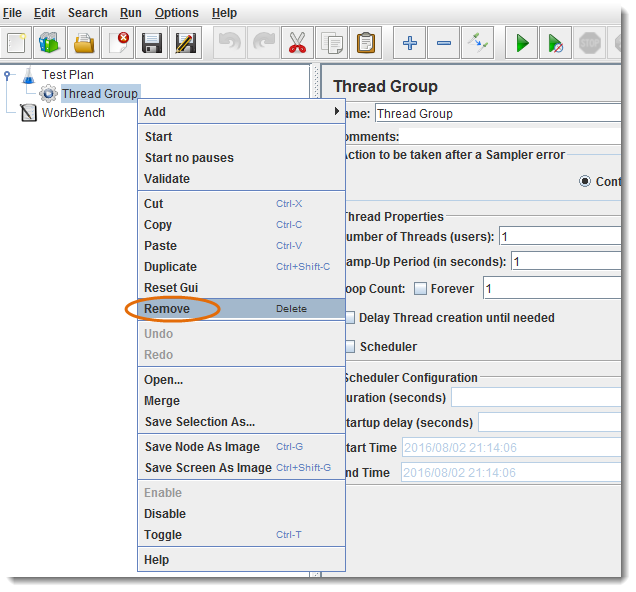


***Note****:****Elements can also be loaded from a file and added by selecting the “merge” or “open” option***

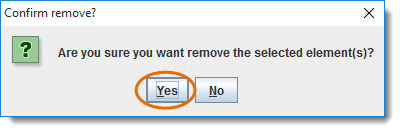
***Steps to Remove Elements from JMeter Test Plan***

When you add number of Elements to Test Plan, you may like to remove one which is not required any more. That can be done by following the below steps:

1. *Select the desired element*
2. *Right click on the element*
3. *Choose the “****Remove****” option*



Click ***“Yes”*** on the remove confirmation popup

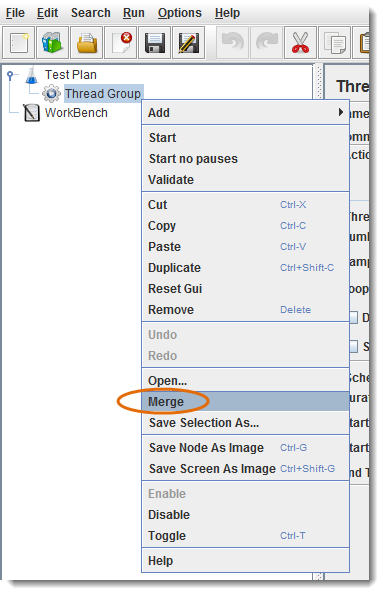


**Load and Save JMeter Test Plan Elements**

Above we learned to add and remove elements. Now it is the time to ***Load and Save Elements to JMeter Test Plan.***

***Steps to Load Elements to JMeter Test Plan Tree***

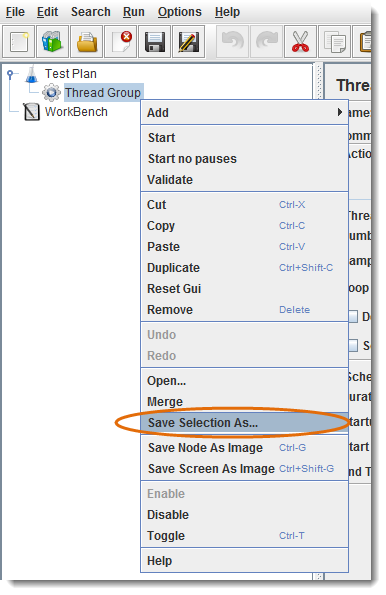
1. *Select and right click on any****Tree Element,****in which you want to add the loaded element*
2. *Select****“Merge”****option*
3. *Choose the****.jmx****file where you saved the elements*
4. *Elements will be Merged into the tree*
5. *Don’t forget to****Save****test plan / element*



***Steps to Load Elements to JMeter Test Plan Tree***

1. *Select and Right click on the element*
2. *Choose “****Save Selection As****” option*
3. *Save file on desired location*

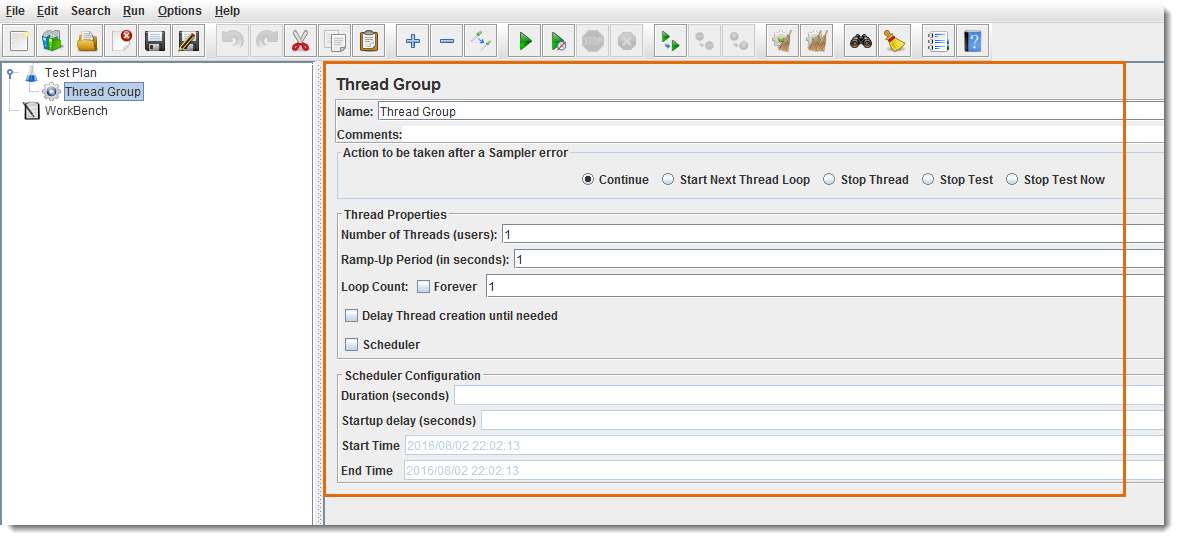
*JMeter will save the element, and all the child elements beneath it.*



***Steps to Configure Tree Elements***

Elements can be configured by using controls present on JMeter’s right hand side frame. The controls allow us to configure behaviour of the selected element. The configuration varies from element to element.

* *E.g. Thread Group can be configured for Number of Threads, Ramp-Up Period, and Scheduler etc., as shown below:*

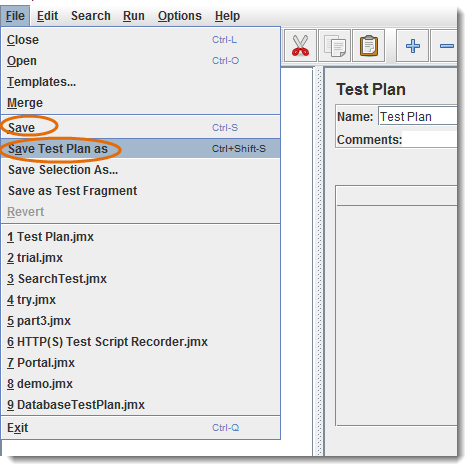


**JMeter Actions on Test Plan**

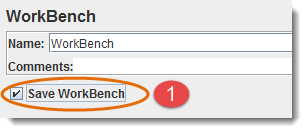
Till now we are done with *Creating a Test Plan, Adding a Element and Configuring a Tree.* Let's just move forward with different operations on Test Plan like *Save, Run & Stop.*

***Steps to Save JMeter Test Plan***

It’s always better to ***Save Test Plan*** before executing it. Test Plan can be saved by choosing ***“Save”*** or ***“Save Test Plan As”*** from ***File*** menu.

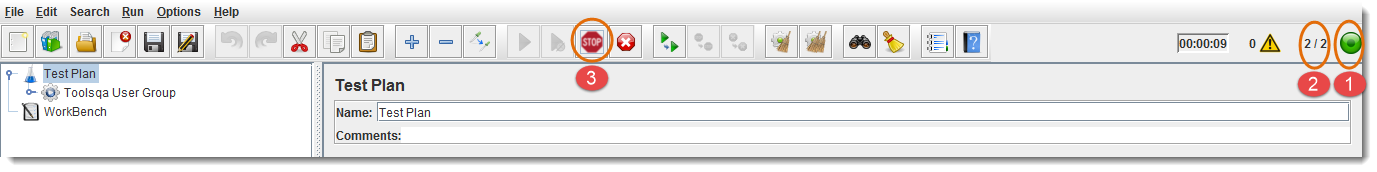


***Note****:* ***WorkBench will not be saved along with the Test Plan, you should select the option “Save WorkBench” from WorkBench control panel otherwise your WorkBench data will be lost.***



***Steps to Run JMeter Test Plan***

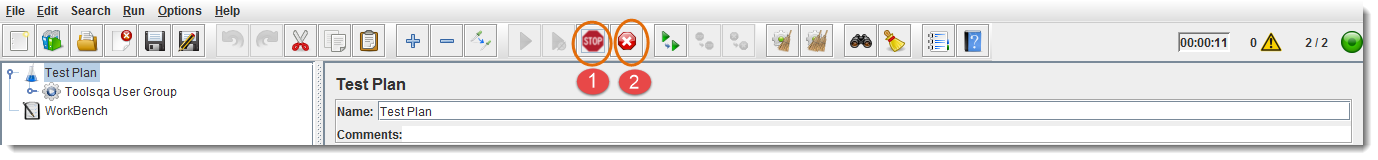
Test plan can be run from the ***“Run”*** menu item or by clicking ***Green Play*** button.



1. *Green button shows that test plan is running.*
2. *Number of active threads / Total number of threads.*
3. *Stop button is enabled when test plan is running.*

***Steps to Stop to JMeter Test Plan***

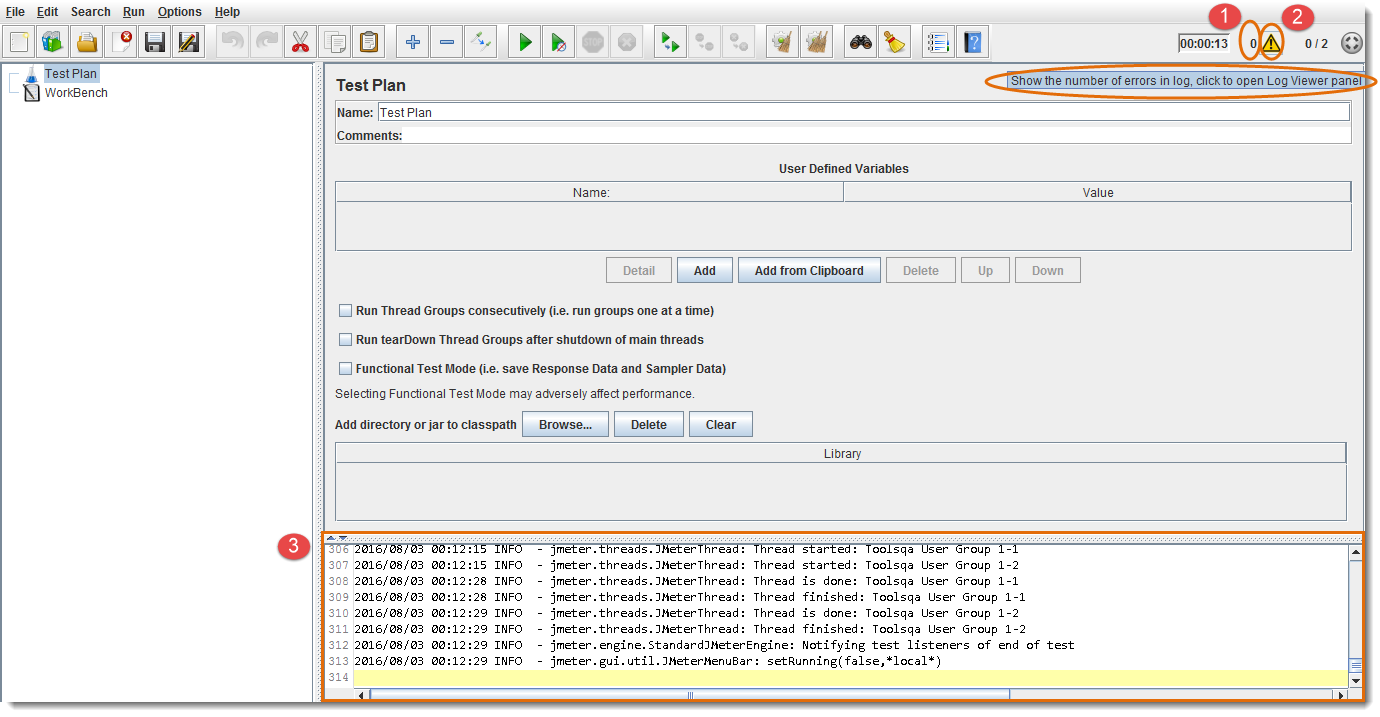
1. ***Stop****(Control + '.') It stops the threads immediately if possible. Many samplers are interruptible so active samples can be terminated early.*
2. ***Shutdown****(Control + ',') It requests the threads to stop at the end of any ongoing task. It will not interrupt any active samples.*



***Steps to Check JMeter Test Plan Execution Logs***

JMeter stores test run details, warnings and errors to the ***jmeter.log*** file by default, you can access Jmeter logs for debugging purpose.

1. *It shows the total number of errors in log*
2. *Click yellow triangle button to view log panel*
3. *Logs are displaying in log panel*



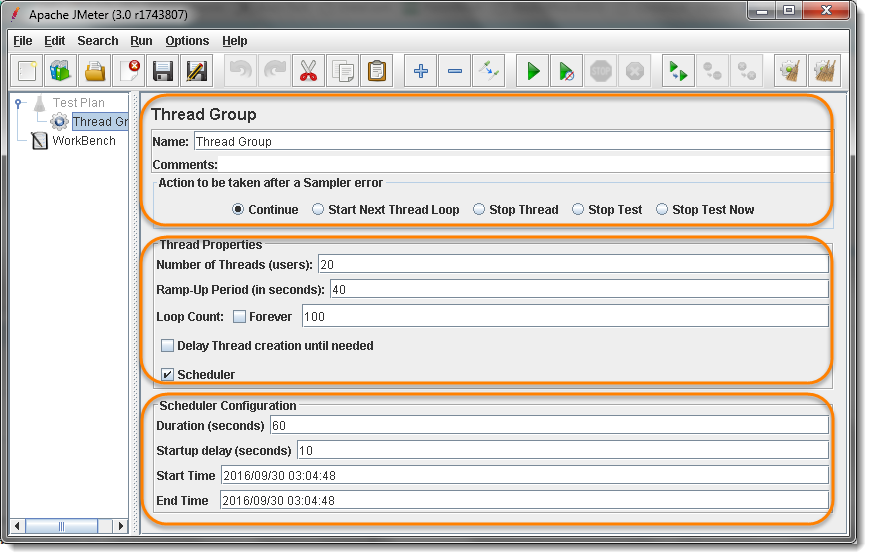
In this chapter of Thread Group in JMeter Test Plan, we will be covering the following topics:

* ***What is Thread Group***
* ***How to Create a Thread Group***
* ***Component of Thread Group***

# What is Thread Group in JMeter?

**A *Thread Group* is a set of threads executing the same scenario. Set the number of iterations in the configuration**.

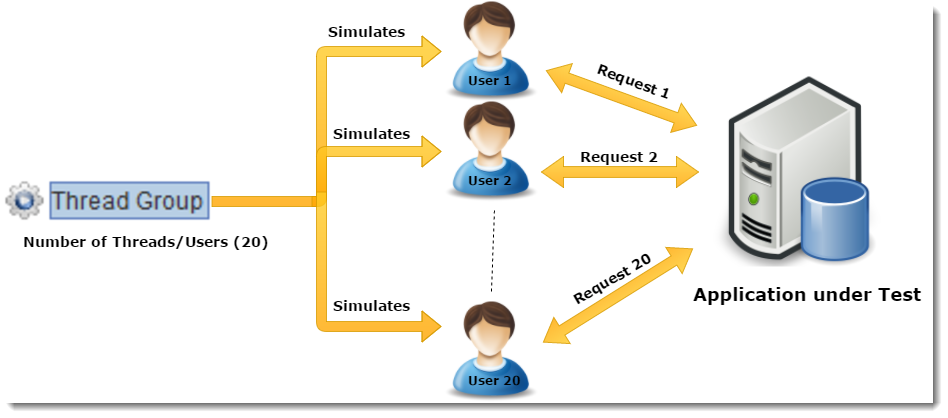
Thread behaviour is defined according to ramp up and destroyed once the number of iterations per thread has elapsed. This is how a Thread Group Element looks:



***Thread Group elements*** are the initial steps of [***JMeter Test Plan***](https://toolsqa.com/jmeter/build-jmeter-test-plan/).

**A number of threads (*users*) can be defined in a Thread Group. Each thread simulates a real user requesting to the server under a test.**

If you set the number of threads as 20; JMeter will create and simulate 20 virtual users during the load test. A diagram here can help us understand it better

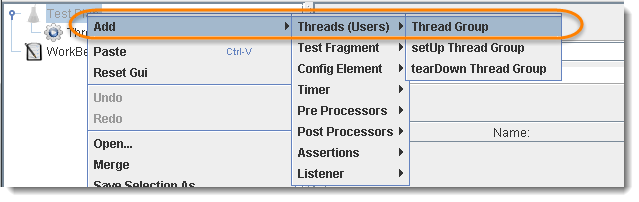


**How to Create a Thread Group Element in JMeter Test Plan?**

In the previous chapter we learnt that how to create a test plan. Thread Group is a element of test plan. To create a Thread Group it is required to have the Test Plan created.

* Start ***JMeter***
* Select ***Test Plan*** on the tree
* Add ***Thread Group***

Open the thread group panel by Right Click on Test Plan and then going to ***Add >> Threads >> Thread Group***. As shown in the image below:

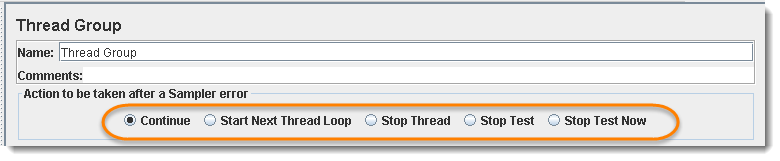


**Components of Thread Group**

The Thread Group Panel holds the following components −

***1) Action to be taken after a Sampler Error***

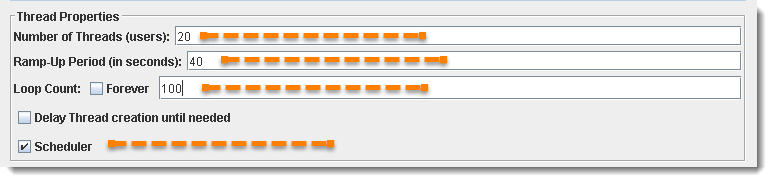
If JMeter catch any sampler error during test execution, you can tell it how to react in that scenario from the following available options.



* ***Continue***, *to ignore error and move to the next element in the tree*
* ***Start Next Thread Loop*** *to stop current Thread and Start Next*
* ***Stop Thread***, *to stop the current Thread execution.*
* ***Stop Test***, *to stop entire test execution.*
* ***Stop Test Now***, *the entire test will be stopped abruptly.*

Default is Continue.

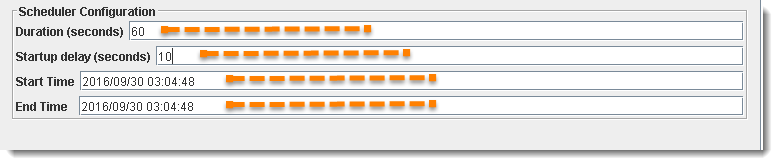
***2) Thread Properties***



* ***Number of Threads****(users): Simulates the number of users or connections to your server application*.
* ***Ramp-Up Period****(in seconds): Tells JMeter how long to take ramp-up to the full number of threads chosen. For Example: If you set “****Number of Threads****” to “****20****”, and “****Ramp-Up Period to 40 seconds****", then JMeter will wait till 40 seconds to make all threads up and running. That means each thread will start 2 seconds late after the previous thread was started*.
  + *Formula: Ramp-Up Period / Number of Threads i.e. 40 / 20 = 2 (seconds)*
* ***Loop Count****: the number of times the test to be executed. If you need to run the test forever, then select the “****Forever****” check box.*
* ***Scheduler****: To schedule test execution. Scheduler Configuration bottom panel will get enabled when you select this checkbox. The schedule feature is also very helpful in soak/endurance testing*.

***3) Scheduler Configuration***

You can configure test ***start time, end time, duration*** and ***start up delay*** of your load test plan using Scheduler Configuration section. To enable this area of configuration, Scheduler check box must be selected from the above Section of Thread Properties.



Thread Group configuration panel is as follows:

* ***Start Time****: This plans the test to start at scheduled time. Pre-Condition is that the JMeter should be running on given date and time in "****Start Time****" field*.
* ***End Time:****This command JMeter to end the test at the mentioned time. End time override and stop execution in between. Means End Time is maximum allowed time to finish execution of the test plan. JMeter ends the execution immediately as soon as End Time is occur*.
* ***Duration****(seconds): This tell the JMeter to execute the test for the specific duration of time. If the duration is set to 60 secs, JMeter will keep the execution on for 60 secs and ends it once the time is elapsed. It also ignores or override the End Time and All threads has completed its test or not*.
* ***Startup delay****(seconds): This tells JMeter to wait for specified time before starting the test. If the StartUp time is set to 10 secs, JMeter will not start loading the Users till the time 10 secs are over*.

# ***Logic Controller*** let you handle the order of processing Samplers/Requests in a Thread. Logic Controllers will decide “When & How” to send a request to a web server.

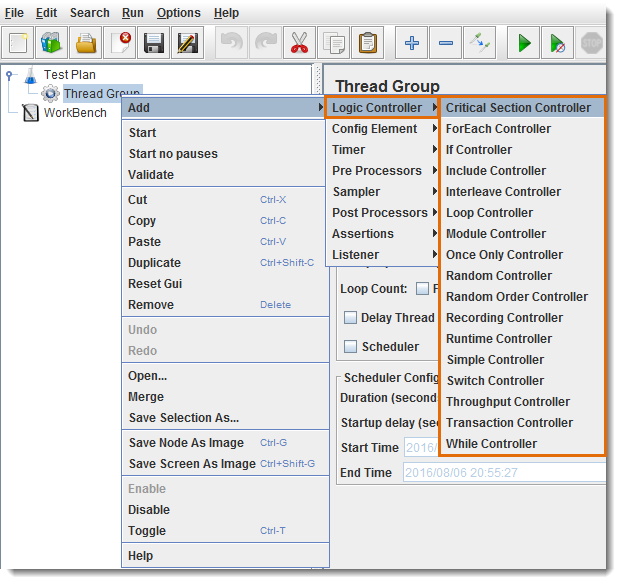
***JMeter provides several Logic Controller, which are as follows:***

* ***Critical Section Controller***
* ***ForEach Controller***
* ***If Controller***
* ***Include Controller***
* ***Interleave Controller***
* ***Loop Controller***
* ***Module controller***
* ***Once Only Controller***
* ***Random Controller***
* ***Random Order Controller***
* ***Recording Controller***
* ***Runtime Controller***
* ***Simple Controller***
* ***Switch Controller***
* ***Throughput Controller***!
* ***Transaction Controller***
* ***While Controller***

:

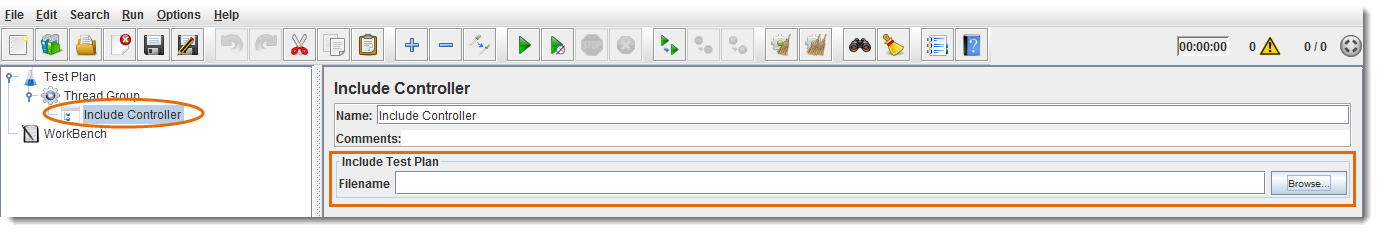
**1: Include Controller**

Include Controller is made to use an external test plan. This controller allows the usage of multiple test plans in JMeter.



**2: Interleave Controller**

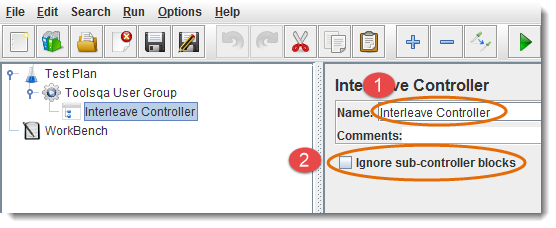
Interleave Controller will select only one samplers/requests stored in it, to run in each loop of the thread. It will execute the samplers sequentially.



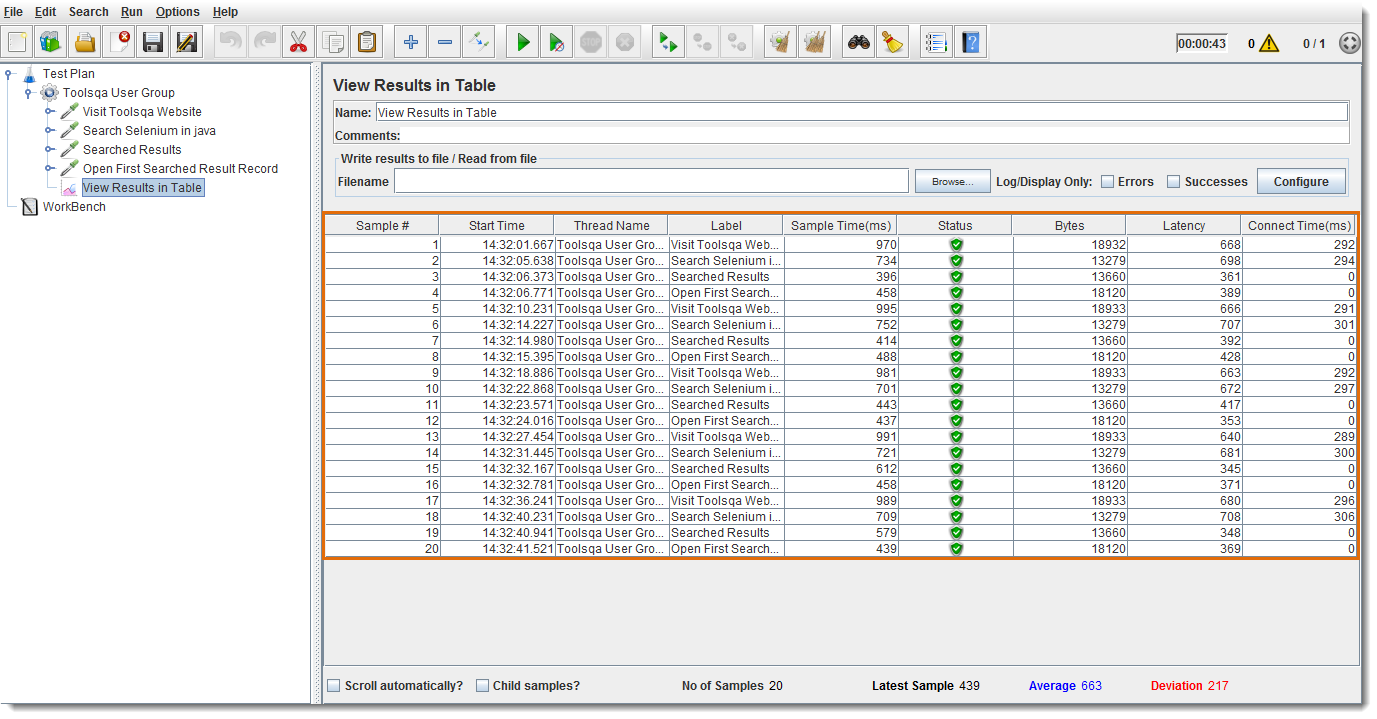
1. *You can give descriptive name for the controller (optional)*
2. *If checkbox is selected, the interleave controller will consider sub-controllers as single request elements and permit only one request per controller at a time (optional)*

Let’s consider an example of Interleave Controller, where [***Thread Group***](https://toolsqa.com/jmeter/thread-group-in-jmeter-test-plan/) contains *“****Number of Threads****” to “****1****”, “****Loop Count****” to 5, and a total of “****4****” Samplers (Http requests).*

Total ***1 \* 5 \* 4 = “20”*** requests will be sent to web server under test without using any Logic Controller. You can see details in the below figure:



Total “***5***” requests will be sent to web server under test by using Interleave Controller in sequential order i.e. 1 request per loop. You can see details in the below figure:

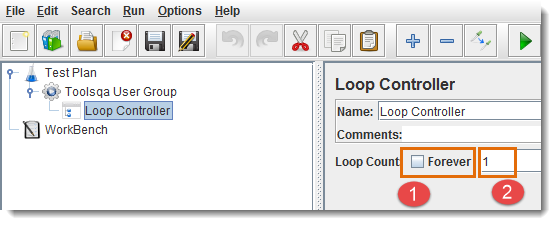


1. *You can see “****4****” Http Requests/Samplers under Interleave Controller node on the left panel.*
2. *Only 5 Http Requests are being sent to the web server (one request for each loop)*

**3: Loop Controller**

Loop Controller will run the samplers/requests stored in it for the definite number of times or forever (*if forever checkbox is selected*).

* *Select Forever checkbox to run tests for infinite times*
* *Enter loop count number to run tests for a fixed number of times*

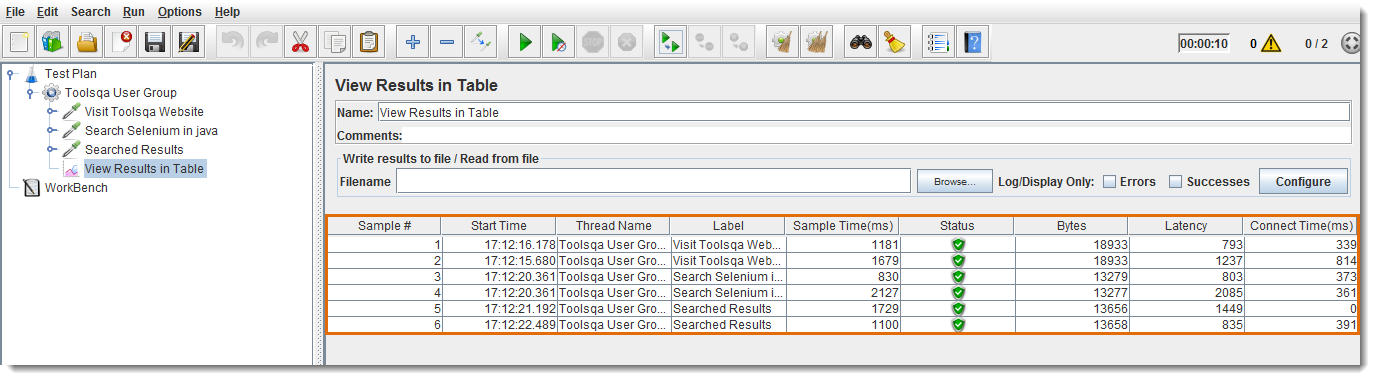


***For Example****, Thread Group loop count is set to “****2****”, Loop Controller loop count is set to “****2****”, and “****3****” requests are present under Loop Controller node, then JMeter will send a total of “****30****” Http Requests to the web server under test.*

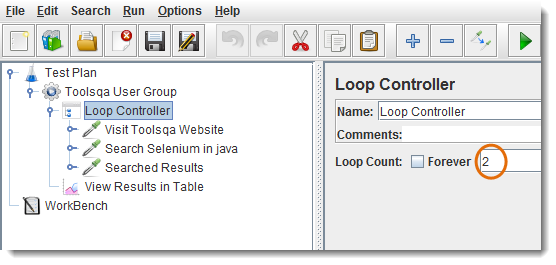
*Total Requests (12) = Thread Group loop count (2)  \*  Loop Controller loop count (2)  \*  Number of Requests inside Loop Container (3)*

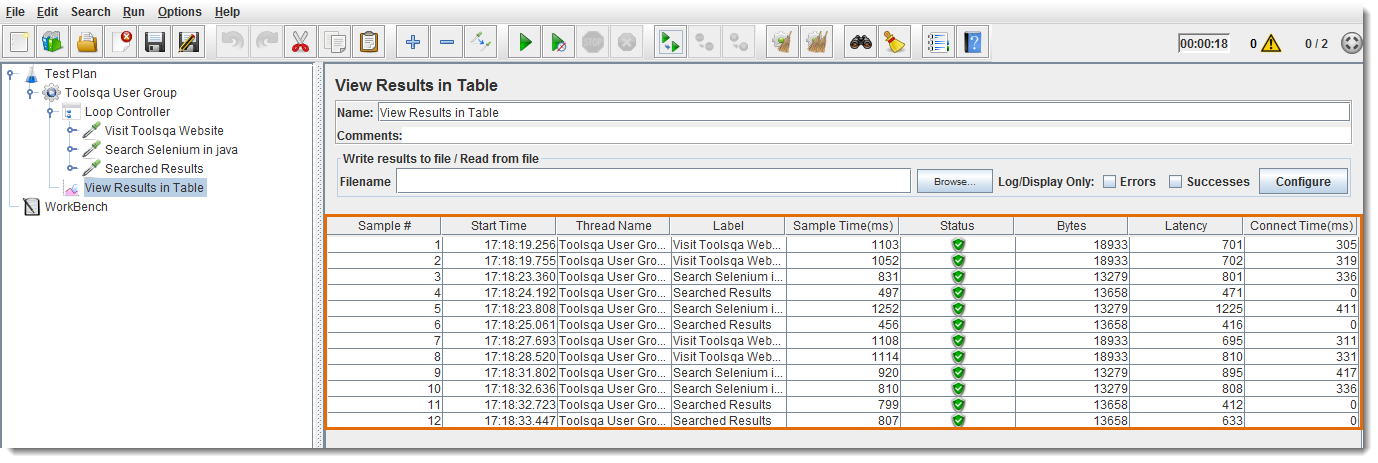
Let’s compare the requests sent with or without Loop Controller:

Without Loop Controller, “***6***” requests are sent to the web server:



After adding Loop Controller with Loop Count to “***2***”, a total of “***12***” Http requests are sent to the web server.

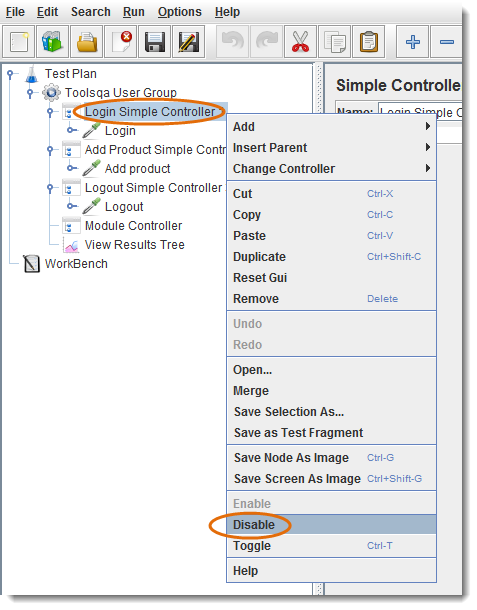




**4: Module Controller**

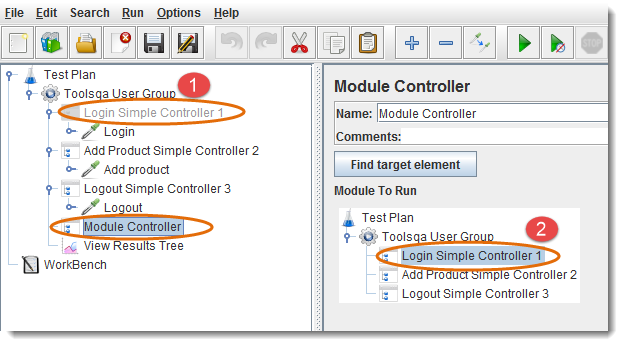
Module controller adds modularity to the JMeter Test Plan. Normally we construct test plan consists of small units of functionality like Login, Add Product, Logout. The functionality can be stored inside Controllers as modules e.g. “***Simple Controller***” can be used to store each module inside it. In order to run modules by only Module controller, don’t forget to disable Simple Controller.

*Select Controller    >   Right Click   >   Select “****Disable****” Option*



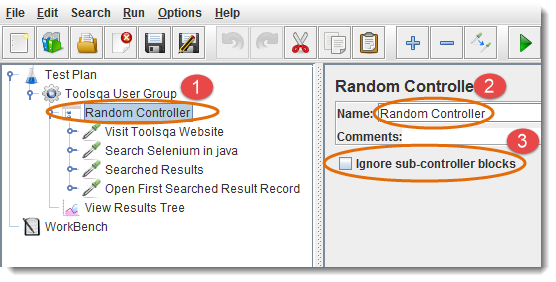
1. *Simpler Controller has been disabled, in order to run this fragment by “****Module  Controller****” only.*
2. *Select any listed Module e.g. “****Login Simple Controller 1****”*
3. *Run*

In our example, only the selected Module will run:



**5: Random Controller**

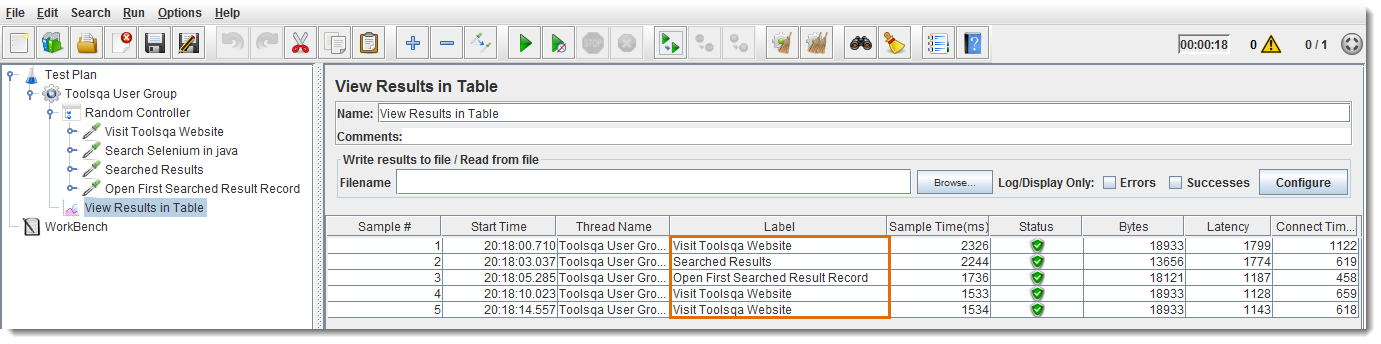
Random Controller is similar to “**Interleave Controller**” except that it makes samplers/requests to run in **random** order in each loop. For Example, you have defined “**4**” Http requests under “**Random Controller**”, then these requests will be sent in random order and their order of execution will be decided on rum-time.



1. *Random Controller contains “****4****” samplers/requests in the left panel*
2. *You can give descriptive name for the controller (optional)*
3. *If checkbox is selected, the Random Controller will consider sub-controllers as   single request elements and permit only one request per controller at a time (optional)*

Let’s consider an example of Random Controller, where Thread Group contains “***Number of Threads***” to “***1***”, “***Loop Count***” to 5, and a total of “***4***” Samplers (*Http requests*).

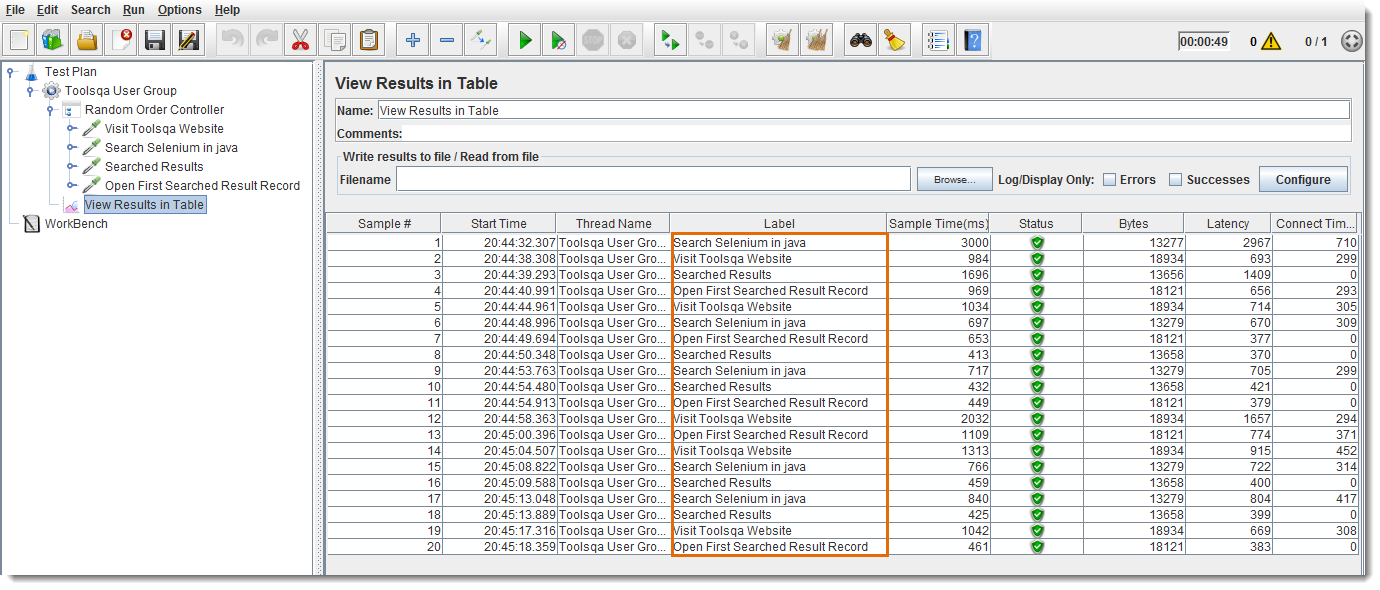
* *Total 1 \* 5 \* 4 = “****20****” requests will be sent to web server under test without using any Logic Controller.*
* *Total “****5****” requests will be sent to web server under test by using “****Random Controller****”, but requests will be sent in random order. As you can see in the following figure:*



**6: Random Order Controller**

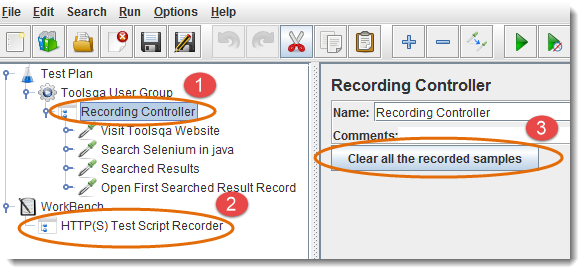
Random Order controller is similar to “***Simple Controller***”, it will make all samplers to run in random order in each thread loop.

Let’s consider an example of Random Order Controller, where Thread Group contains “***Number of Threads***” to “***1***”, “***Loop Count***” to 5, and a total of “***4***” Samplers (*Http requests*). A total of *1 \* 5 \* 4  = “****20****”* requests will be sent to the web server under test. The order of execution will be random, as you can see in the following figure:



**7: Recording Controller**

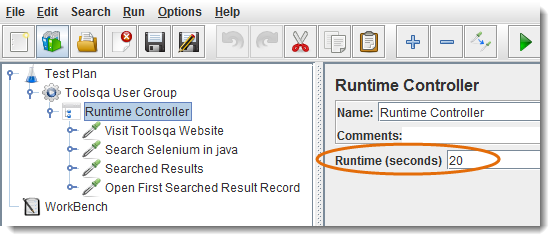
Recording Controller is a place holder where proxy server can save recorded requests. It has no effects on test execution. As you can see in the following figure:



1. *Recording Controller node has all the recorded steps/requests*
2. *For recording, “****HTTP(S) Test Script Recorder****” by default use “****Recording   Controller****”, to store recorded steps*
3. *Click to clean all the steps recorded inside “****Recording Controller****”*

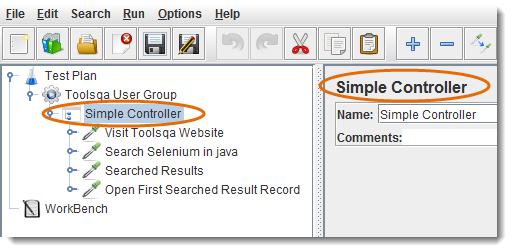
**8: Runtime Controller**

Runtime Controller controls the execution of its samplers/requests for the given time. *For example, if you have specified Runtime(seconds) to “****20****”, JMeter will run your test for 20 seconds only.*



**9: Simple Controller**

Simple Controller helps in organizing and storing the Samplers and other Logic Controllers. It doesn’t offer any other functionality like other controllers.

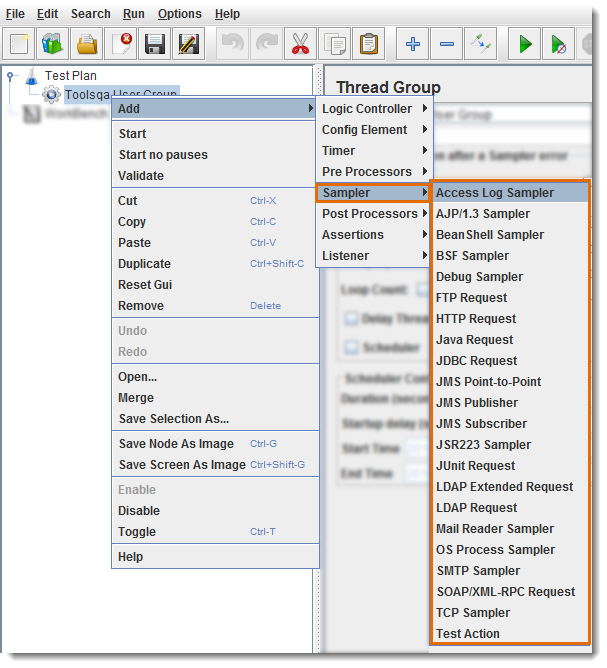


# Samplers in JMeter allows JMeter to send different types of requests to a server. **Samplers are the actual requests**, JMeter sends to the web server under test.

Each sampler (*except Test Action*) generates one or more sample results. The sample results have various attributes (*success/fail, elapsed time, data size etc.*) and can be viewed in the various listeners. Some important Samplers available are as follows:

* ***FTP Request***
* ***HTTP Request***
* ***Java Request***
* ***SMTP Sampler***
* ***BSF Sampler***
* ***JDBC Request***
* ***SOAP/XML RPC-Request***

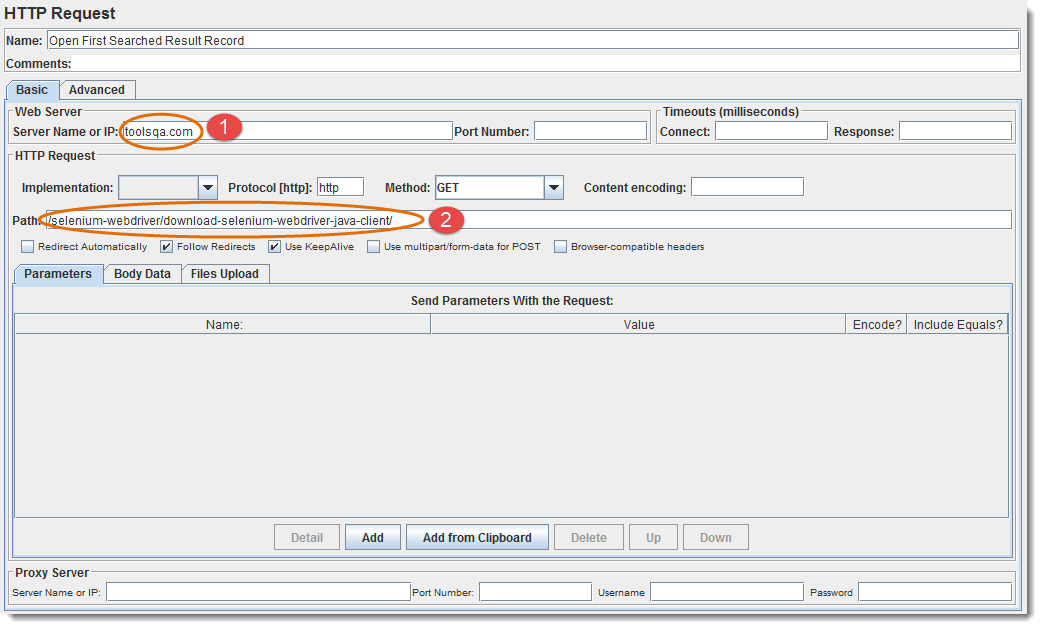
***Complete list of available samplers can be seen in the following figure:***



Let’s have a look on some of the common Samplers:

**1:  HTTP Request**

HTTP Request Sampler is used to send HTTP/HTTPS requests to the web server.

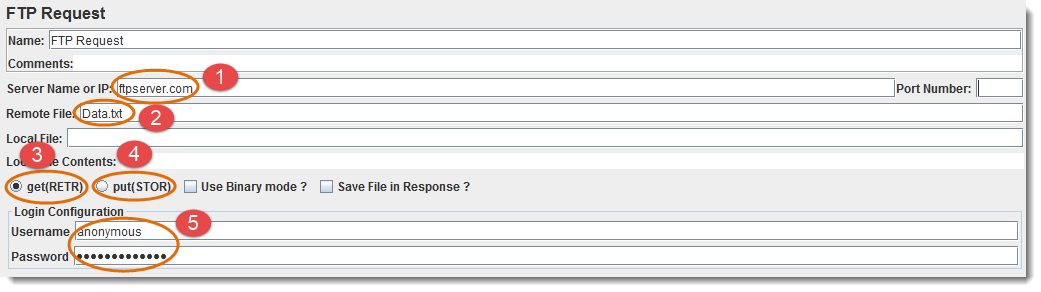


1. *Enter Domain name / IP address of the web server. Do not include the “****http://****” prefix*.
2. *Enter Path to resource*.

***Note****: If you are going to send multiple requests to the same web server, consider using an****HTTP Request Defaults Configuration Element****so you do not have to enter the same information for each HTTP Request. Or, instead of manually adding HTTP Requests, you may want to use JMeter's****HTTP(S) Test Script Recorder****to create them. This can save you time if you have a lot of HTTP requests or requests with many parameters*.

**2:  FTP Request**

FTP Request Sampler allows to send an FTP "***retrieve/download file***" or "***upload file***" request to an FTP server.

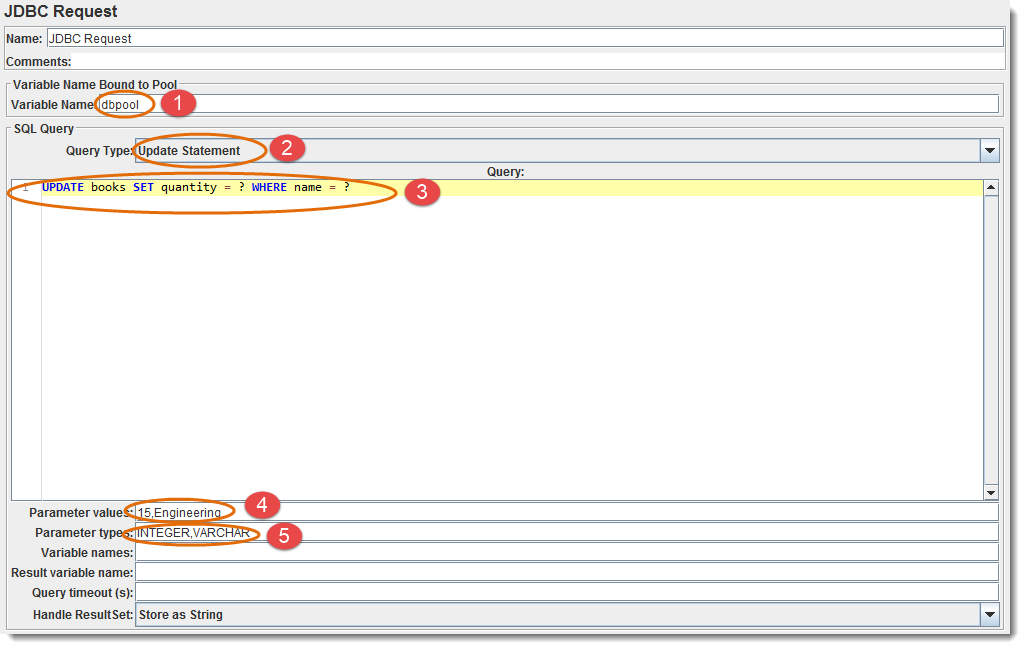


1. *Domain name / IP address of FTP server.*
2. *File name to retrieve*
3. *Select if you want to download a file*
4. *Select if you want to upload a file*
5. *Credentials of FTP account User*

***Note****: If you are going to send multiple requests to the same FTP server, consider using a****FTP Request Defaults Configuration Element****so you do not have to enter the same information for each FTP Request Generative Controller. When downloading a file, it can be stored on disk (Local File) or in the Response Data, or both.*

**3:  JDBC Request**

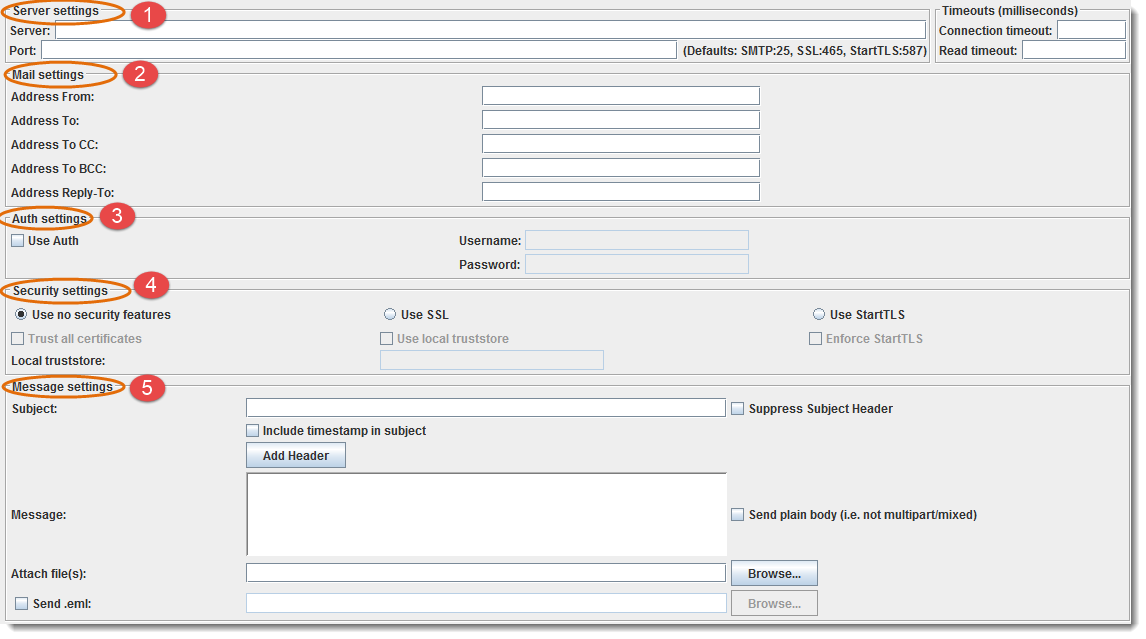
JDBC Request Sampler is useful for database testing. It sends JDBC request consist of SQL query to a database.



1. ***Variable Name that the connection pool is bound to. The variable name   should be same as of the 'Variable Name' defined in a JDBC Connection Configuration*.**
2. *Select an appropriate query type from dropdown*
3. *Write your SQL query without semicolon (;)*
4. *Enter values to update (comma-separated)*
5. *Enter columns data types (comma-separated)*

**4:  SMTP Sampler**

SMTP Sampler is used to test a Mail Server where it sends email messages by using SMTP/SMTPS protocol.



1. *Server settings to set Mail Server Name and Port*
2. *Mail settings to send an email*
3. *Authentication Settings*
4. *Security settings*
5. *Message settings to enter Subject, Message Body and attach any file*

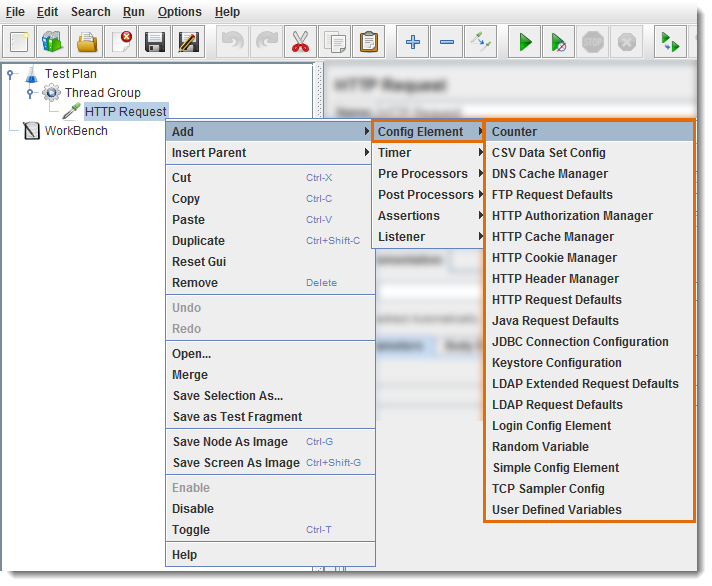
# Configuration Elements

***Configuration Element*** enable us to declare variables, so [***Samplers***](https://toolsqa.com/jmeter/samplers/) can use data through these variables.

Configuration element is accessible from only inside the tree node where you defined it. Also, if a configuration element is defined inside a tree node, it will have high precedence than the same configuration element which is defined in a parent node.

***List of available Configuration Elements is as follows:***

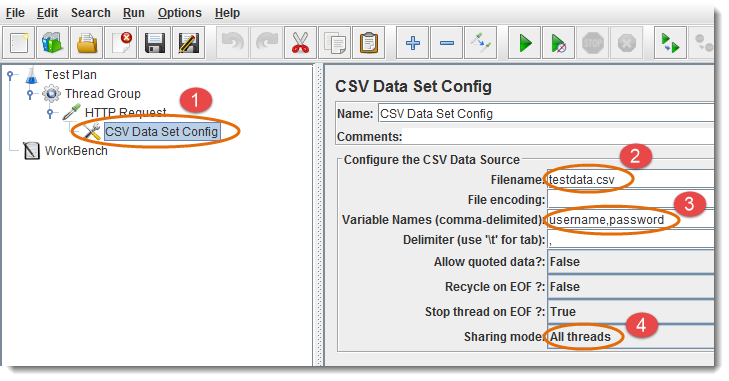
* ***Counter***
* ***CSV Data Set Config***
* ***DNS Cache Manager***
* ***FTP Request Defaults***
* ***HTTP Authorization Manager***
* ***HTTP Cache Manager***
* ***HTTP Cookie Manager***
* ***HTTP Header Manager***
* ***HTTP Request Defaults***
* ***Java Request Defaults***
* ***JDBC Connection Configuration***
* ***Keystore Configuration***
* ***LDAP Extended Request Defaults***
* ***LDAP Request Defaults***
* ***Login Config Element***
* ***Random Variable***
* ***Simple Config Element***
* ***TCP Sampler Config***
* ***User Defined Variables***

:

## 1: CSV Data Set Config:

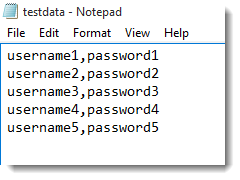
**CSV Data Set Config element is used to read data from a text or CSV format file.**

For Example, If you need to perform load test of login scenario with 100 unique users. Prepare the data in CSV file with 100 user records with username and password, and you can use this file data in every thread through variables in your requests/samplers by using this configuration element.



1. Add “***CSV Data Set Config***” Element where it will be used
2. You can give file path or filename of your text/CSV file. If file is stored in the same location as your “***Test Plan***” is, then give only filename.
3. Give variable names for text/CSV file column values, use these variable as “***$username***” / “***$password***” in samplers
4. Select sharing mode

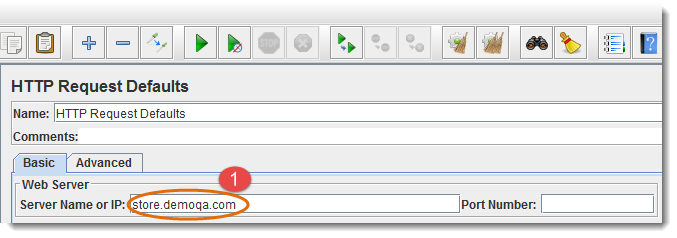
Following is the sample text file which contains user credentials for login scenario:



## 2: HTTP Request Defaults:

**HTTP Request Defaults Configuration Element let us set default values to be used in HTTP Request Samplers**.

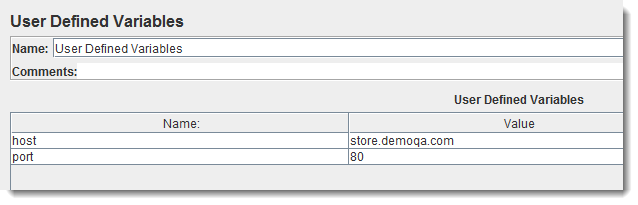
***For Example:*** You need to send 100 HTTP Requests to the same server, then you can utilize HTTP Request Default element with your “**Server Name or IP**”. Now, there is no need to give “**Server Name or IP**” in all 100 HTTP Requests explicitly. The requests will get “**Server Name or IP**” from the HTTP Request Default, just give the relative path of the web pages in samplers/requests.



1. Give Server Name or IP without (http://) prefix.

## 3: User Defined Variables (UDV):

**User Defined Variables Element let us use default variables and values in the test plan. If you need to use UDV in only one Sampler, then define it under that Sampler. If you need to use UDV in multiple parts, then define it at the start of Test Plan**.

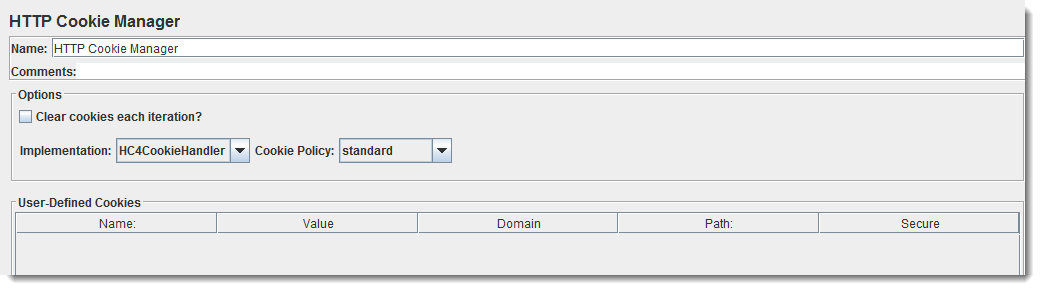


## 4: HTTP Cookie Manager

**HTTP Cookie Manager is used to store cookies which targeted server sends in the response of your Http request. The saved cookies can be used in other samplers/requests of the Test Plan.**

1. You can also add User Defined Cookies, these cookies will be shared to all the Threads.
2. Cookies can be seen using the “***View Results Tree***” Listener.
3. Such Cookies usually have Expiration time far in the future.

HTTP Cookies Manager Panel looks like below figure:



# Timers

**JMeter sends requests without applying any delay between each sampler/request**.

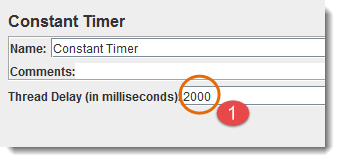
If you perform load/stress testing on your server without any delay, it will be overloaded. Then, it won’t be able to give you realistic results and fail to simulate real world user traffic experience. ***JMeter Timers*** are the solution to all these problems. Timer element can be added in a test plan to apply wait between each sampler/request.

***The JMeter Timers available are as follows:***

* ***BeanShell Timer***
* ***BSF Timer***
* ***Constant Throughput Timer***
* ***Constant Timer***
* ***Gaussian Random Timer***
* ***JSR223 Timer***
* ***Poisson Random Timer***
* ***Synchronizing Timer***
* ***Uniform Random Timer***

## 1:   Constant Timer:

***Constant Timer*** **element delays each request in a Thread Group for the same amount of time.**

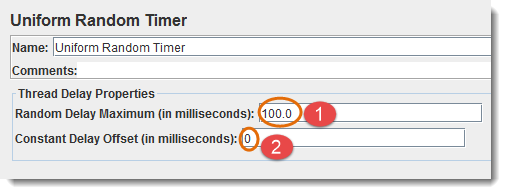


1. Enter Thread Delay value in milliseconds. We have entered delay of “***2000***” ms (which is equals to 2 seconds) in between each user request.

## 2:  Uniform Random Timer:

***Uniform Random Timer*** **element is used to delay each request for a random period of time.**

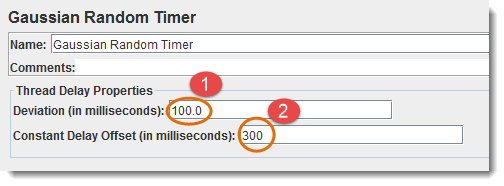
***Total amount of delay   =   Random Delay Maximum    +    Constant Delay Offset***



1. Enter any Maximum Random Delay value in milliseconds.
2. nter constant Delay Offset value in milliseconds in addition to random delay.

## 3:  Gaussian Random Timer:

***Gaussian Random Timer*** element is used to delay each user request for a random period of time. It has a random deviation around the Constant Delay Offset based on Gaussian curve distribution.



For Example:

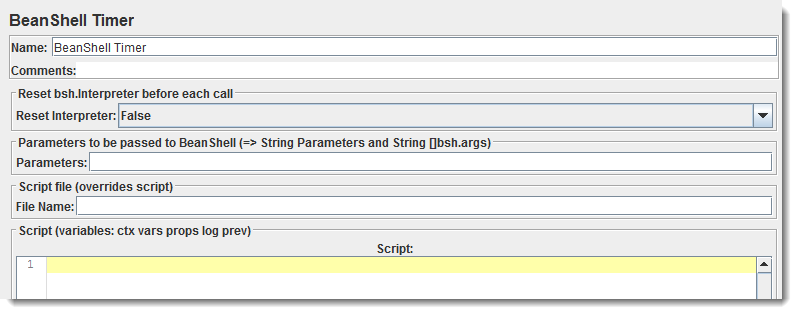
1. Deviation Value: 100 milliseconds
2. Constant Delay Offset Value: 300 milliseconds

Approximately 68% of the delays will be between (200 - 400) ms

* ***Constant Delay Offset   -   Deviation    =  200 ms***
* ***Constant Delay Offset   +   Deviation   =  400 ms***

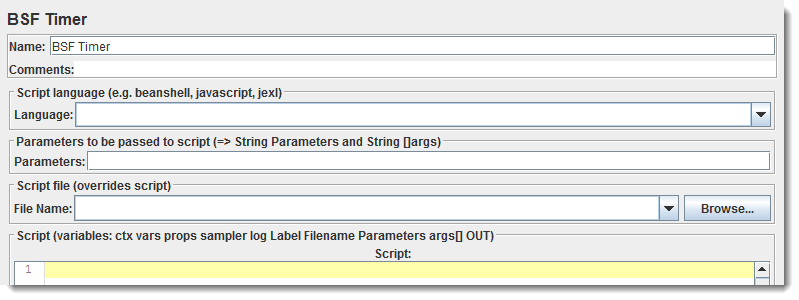
## 4:  BeanShell Timer:

***BeanShell Timer*** element can be used to generate delay between each user request. For BeanShell scripting, go to this URL: [***http://www.beanshell.org/***](https://www.beanshell.org/)



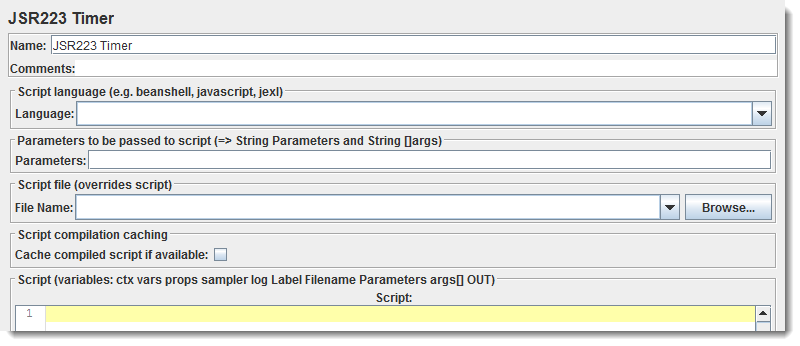
## 5:  BSF Timer:

***BSF*** Timer element can be used to generate delay using a BSF scripting language.



## 6: JSR223 Timer:

***JSR223 Timer*** element can be used to generate delay using a JSR223 scripting language.

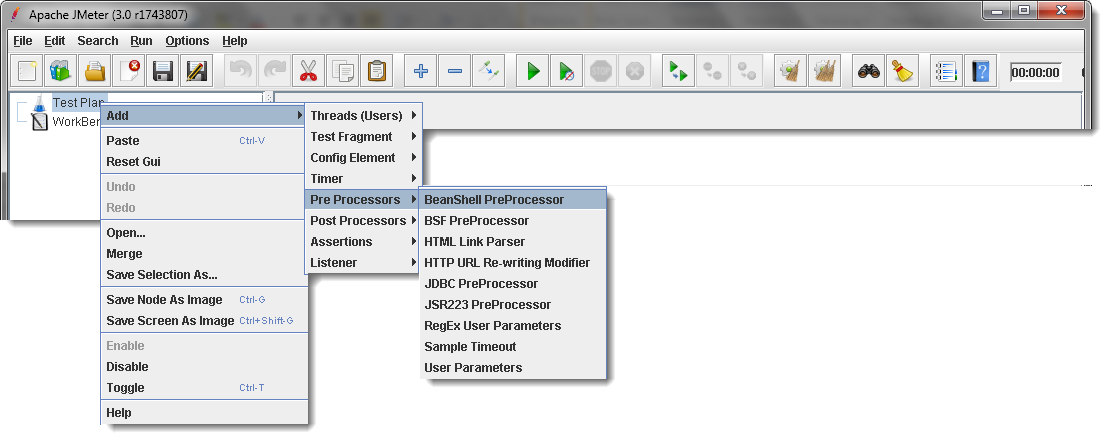


# Pre-Processor

***Pre-Processor* element is defined to alter the settings of Samplers in their scope**. It will always execute before the actual sampler request.

***Pre-Processors elements list is as follows:***

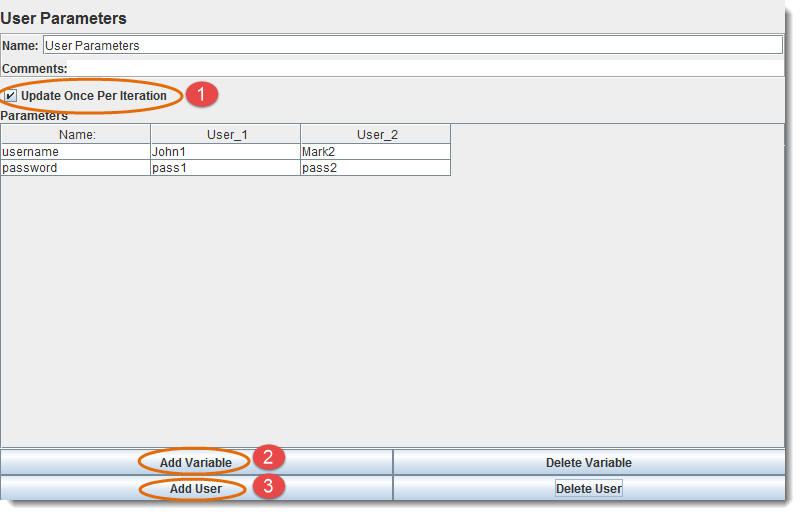
* ***BeanShell PreProcessor***
* ***BSF PreProcessor***
* ***HTML Link Parser***
* ***HTTP URL Re-writting Modifier***
* ***JDBC PreProcessor***
* ***JSR223 PreProcessor***
* ***RegEx User Parameters***
* ***Sample Timeout***
* ***User Parameters***



## 1:  User Parameters:

User Parameter element is used to define variables and its values, specific to individual threads.

User Parameter values will be defined for the current sampler only, unlike User Defined Variables in test plan which are available to all the samplers in the scope.

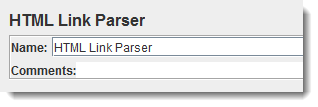


1. Update Once Per Iteration Checkbox:  A flag to specify whether the User Parameters element should update its variables only once per iteration.
2. Add Variable: New empty row can be added to define variable
3. Add User: New column can be added for user/thread

## 2:  HTML Link Parser:

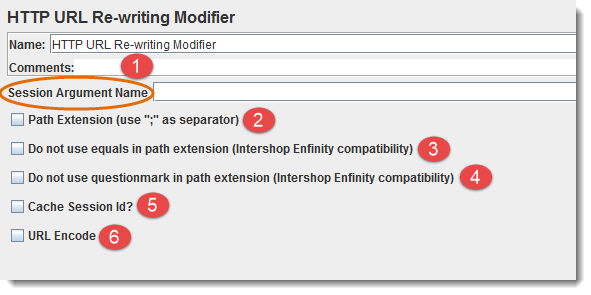
HTML Link Parser is used to parse HTML response from the test server and to extract links/forms.

If we use this preprocessor with a sampler, that URL test sample will be examined to see if it matches any of the links/forms which extracted from the previous response. If this is the case then it will replace the URL test sample values from the matching link/form.



## 3:  HTTP URL Re-writing Modifier:

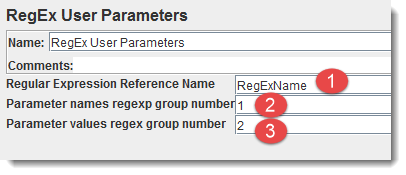
HTTP URL Re-writing Modifier is quite similar to the “***HTML Link Parser***”. However, it is more efficient and easy to use because it has a specific goal to achieve. For Instance: If your web application uses URL rewriting to store session Ids instead of cookies, you can define this element at the Thread Group level similar to the HTTP Cookie Manager.  You are required to mention Session Id parameter name, then it will find it on the page and will pass argument to every request of that Thread Group.



1. Session Argument parameter name is used to fetch value (href/form) from previous response.
2. If you web application rewrite URLs by appending a semi-colon and session Id parameter, then select it.
3. If your web application rewrite URLs without an "***=***" sign between parameter name and its value, then select it.
4. It will prevent query string in the path extension.
5. Check "***Cache Session Id***" checkbox, if session Id is to be saved for later use, when there is no session Id present.
6. URL Encode value while writing parameter.

## 4:  RegEx User Parameters

RegEx User Parameters element let us stipulate dynamic values for HTTP parameters extracted from other HTTP Request with the help of Regular Expressions. RegEx User Parameter are specific to individual threads in a test plan.



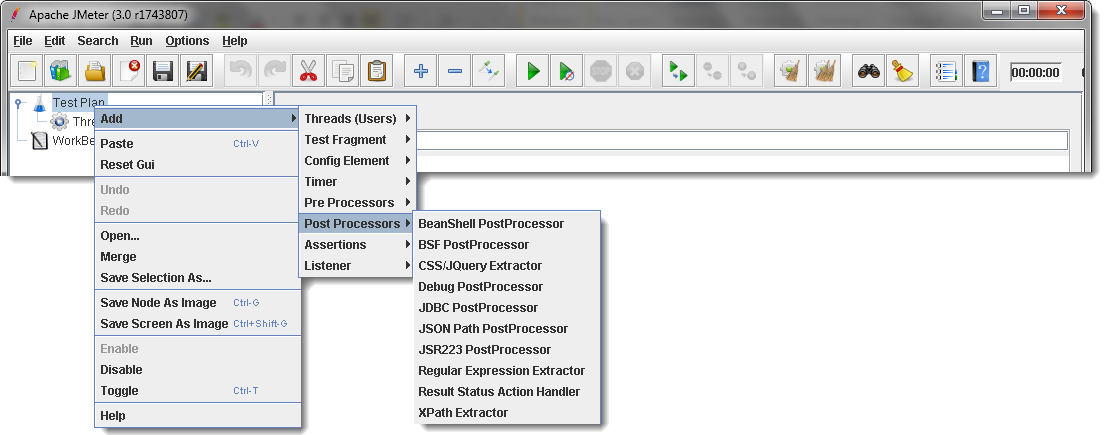
# Post-Processor

***Post-Processor*** **will be executed when a Sampler Request finishes its execution.**

If you need to use post-processor for a particular Sampler, then add it as a child of that Sampler. It is used to process response data from server and extract specific value for later use.

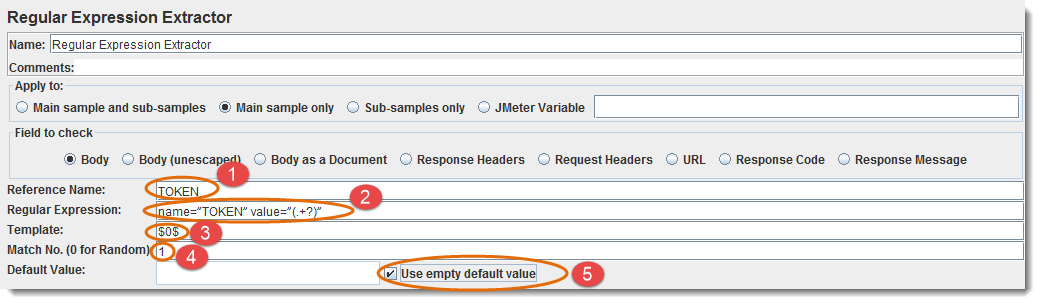
***Post-Processor elements list is as follows:***

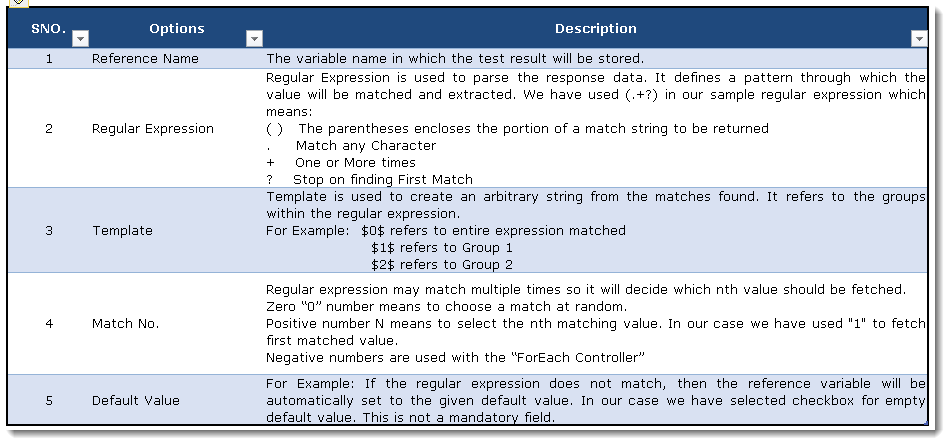
* ***BeanShell PostProcessor***
* ***BSF PostProcessor***
* ***CSS/JQuery Extractor***
* ***Debug PostProcessor***
* ***JDBC PostProcessor***
* ***JSON Path PostProcessor***
* ***JSR223 PostProcessor***
* ***Regular Expression Extractor***
* ***Result Status Action Handler***
* ***XPath Extractor***



## 1:  Regular Expression Extractor

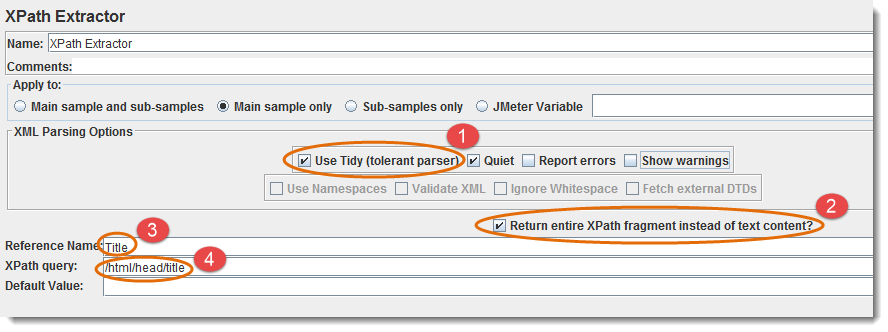
Regular Expression Extractor is used to extract values from the response of test server using a Perl-Type Regular Expression. Being a Post-Processor, it will execute after each Sampler request. It will extract the desired value by using the regular expression and generate template string, then it will store the extracted value in the specified variable name.





## 2:  XPath Extractor:

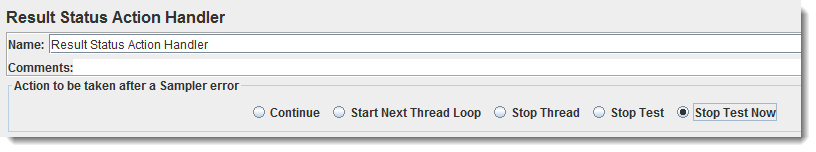
XPath Extractor element is used to extract values from structured response XML or (X)HTML by using XPath query language



1. Use Tidy Checkbox:  If selected use Tidy to parse HTML response into XHTML.
2. If selected then fragment will be returned instead of just text. It means you will get the Title name along with its tags in our case.
3. The variable name in which the test result will be stored.
4. XPath Query: Element query in XPath language and it can also return more than one match.

## 3:   Result Status Action Handler:

Result Status Action Handler element can be used to stop the thread or the test if the relevant sampler gets failed.



# Assertion

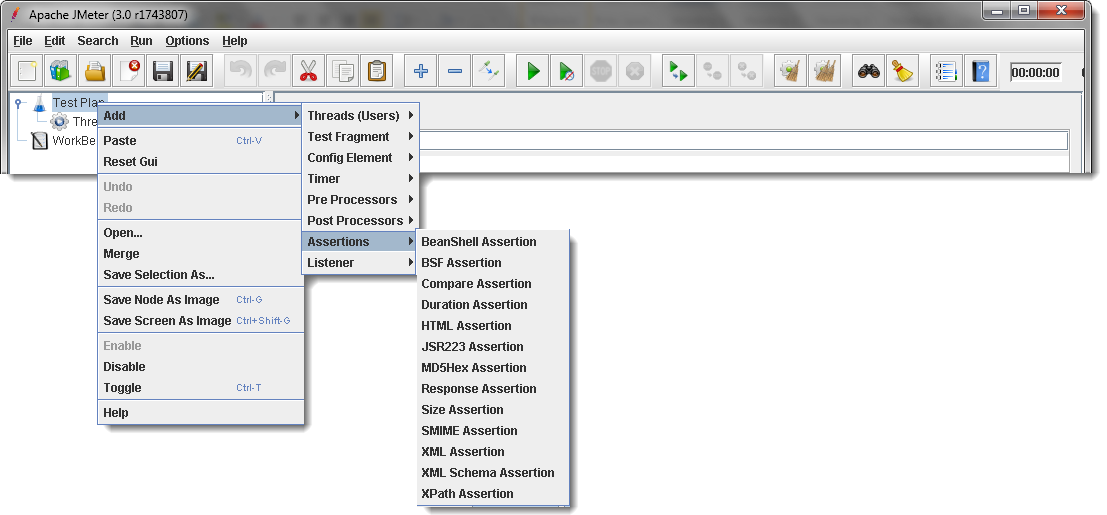
***Assertion*** in JMeter is used to validate response of the request, that you have sent to the server.

Assertion is a process where you verify expected result with the actual result of the request at run time. If you need to apply assertion on a particular Sampler, then add it as a child of that Sampler.

You can view assertion results by adding “***Assertion Listener***” to the Thread Group. Failed assertions will be displayed in other listeners as well.

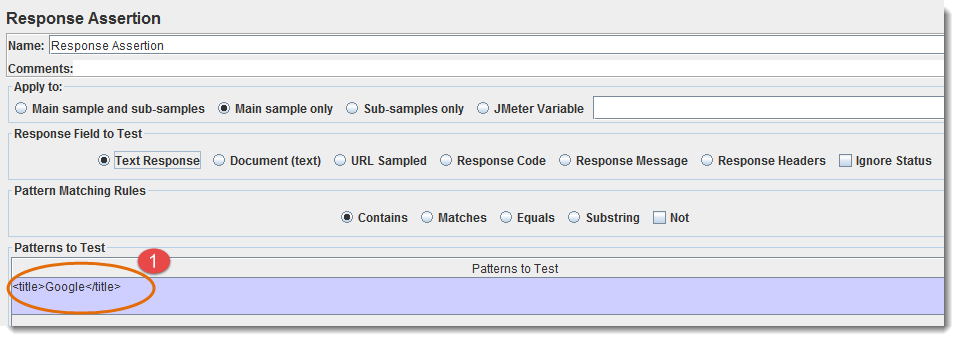
***List of Assertions are as follows:***

* ***BeanShell Assertion***
* ***BSF Assertion***
* ***Compare Assertion***
* ***Duration Assertion***
* ***HTML Assertion***
* ***JSR223 Assertion***
* ***MD5Hex Assertion***
* ***Response Assertion***
* ***Size Assertion***
* ***SMIME Assertion***
* ***XML Assertion***
* ***XML Schema Assertion***
* ***XPath Assertion***



## 1:  Response Assertion:

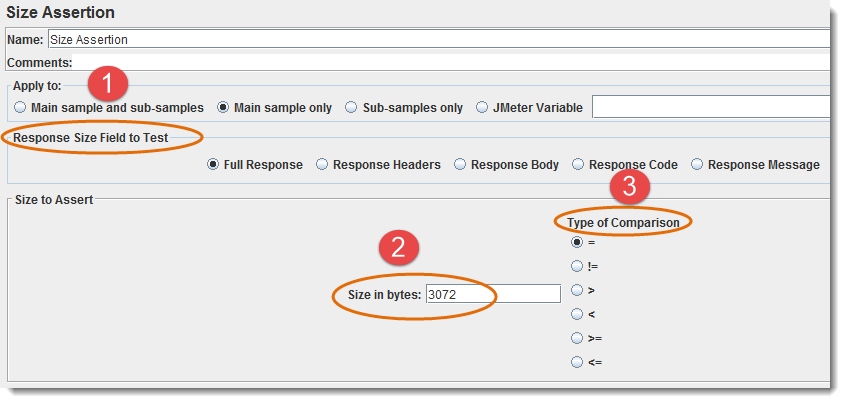
Response Assertion can be used to add and compare pattern strings against one or many values of server response. For Example, when you send a request to the URL: https://www.google.com and get the server response. Here, you can verify this response by using Response Assertion. You can insert "*<title>Google</title>*" as a "***Pattern to Test***" field value in Response Assertion. If response doesn’t contain this string, it will fail the sampler.



1. Add the pattern string according to your expected response

## 2:  Size Assertion

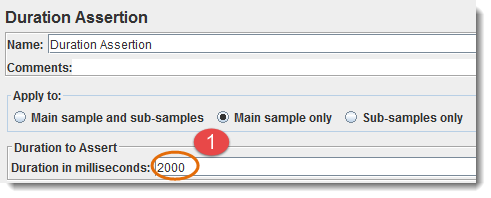
Size Assertion is used to verify the server response contains expected number of Bytes or not.



1. Select Response Size Field to Test
2. Enter expected response size in bytes
3. Choose type of comparison

## 3:  Duration Assertion

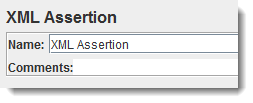
Duration Assertion is used to verify that server response is received within a specified time constraint or not. It will mark the sample request as failed, if response takes longer time duration than the given amount of time.



1. Enter duration period of response in milliseconds

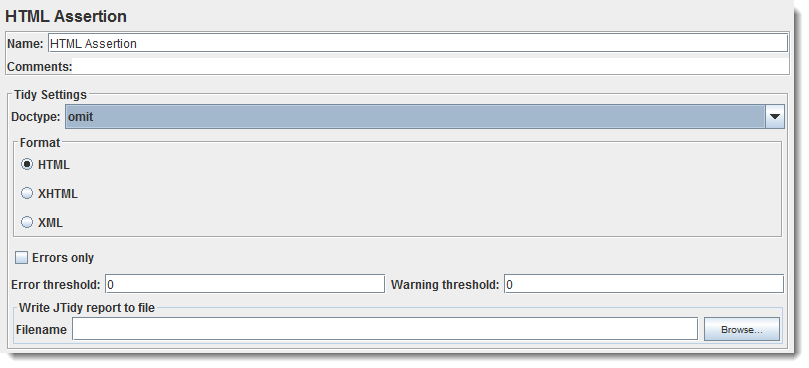
## 4:  XML Assertion

XML Assertion is used to verify that the server response data comprises of a correct XML document or not.



## 5:  HTML Assertion

HTML Assertion is used to verify that the response contains correct HTML syntax or not using JTidy (HTML Syntax Checker). It will fail the test in case of improper HTML syntax response.



# Listener

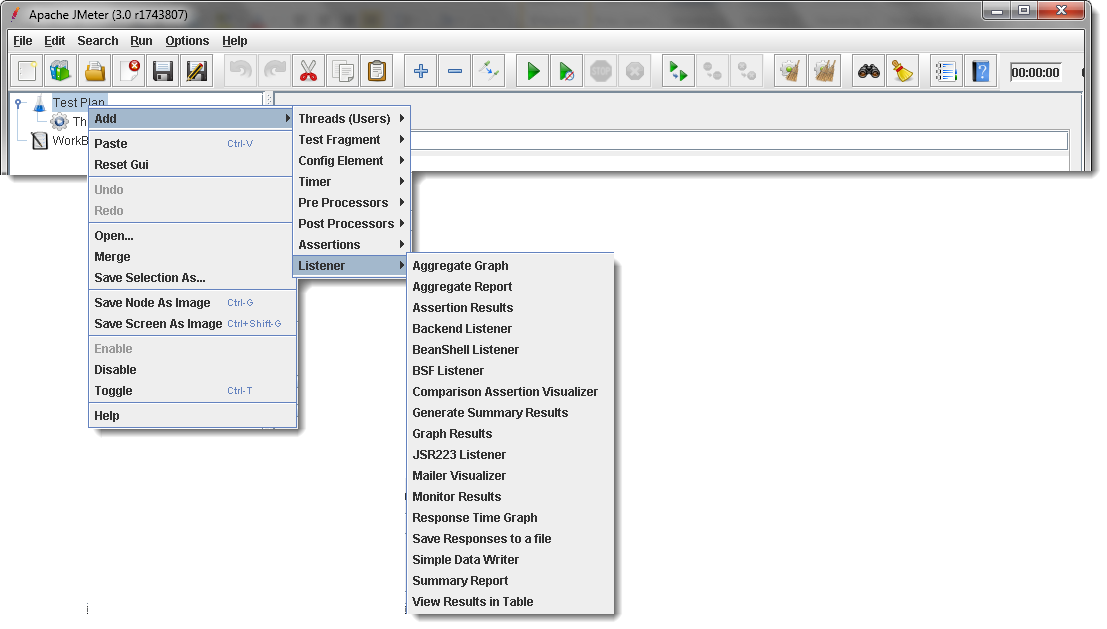
***Listener*** allows us to view and analyze the Sampler request and response in the form of tables, graphs, trees etc.

You can also save results in a file and read saved results later. You can access the sampler result in an ongoing load/stress test without waiting till end.

Listener can be added anywhere in the test plan. It will gather data only from elements defined in its scope.

***List of listeners is as follows:***

* ***Aggregate Graph***
* ***Aggregate Report***
* ***Assertion Results***
* ***Backend Listener***
* ***BeanShell Listener***
* ***BSF Listener***
* ***Comparison Assertion Visualizer***
* ***Generate Summary Results***
* ***Graph Results***
* ***JSR223 Listener***
* ***Mailer Visualizer***
* ***Monitor Results***
* ***Response Time Graph***
* ***Saves Responses to a file***
* ***Simple Data Writer***
* ***Summary Report***
* ***View Results in Table***
* ***View Results Tree***



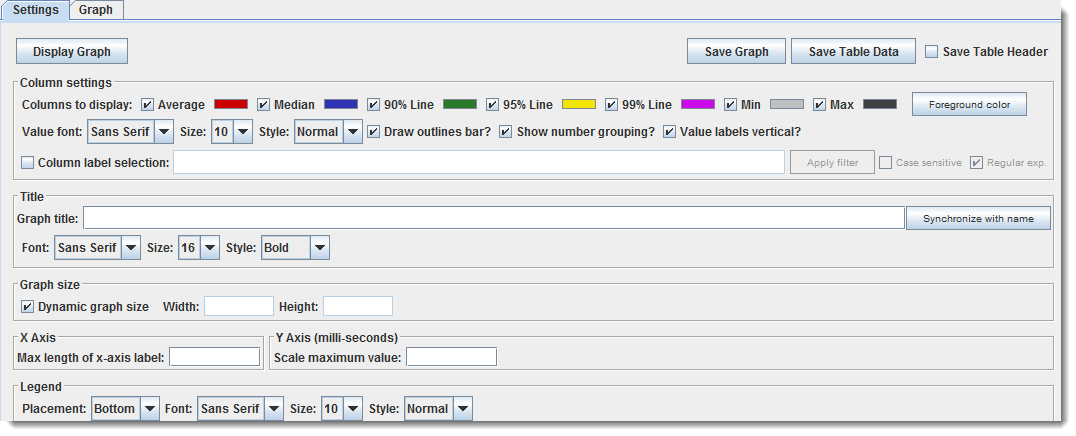
## 1:  Aggregate Graphs:

Aggregate graphs allow us to generate bar graphs easily and let us select graph display settings. The generated graph and table data can be saved explicitly in the form of PNG and CSV.



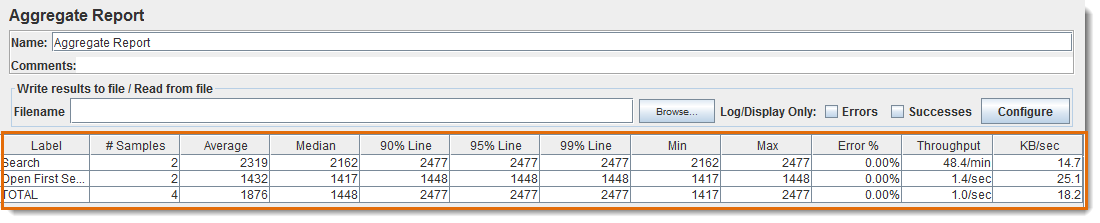
1. Sample Result Save Configuration
2. Settings for the aggregated graph (as you can see in the below figure)
3. Graphical Representation

Aggregate Graph Settings Screen:



## 2:  Aggregate Report

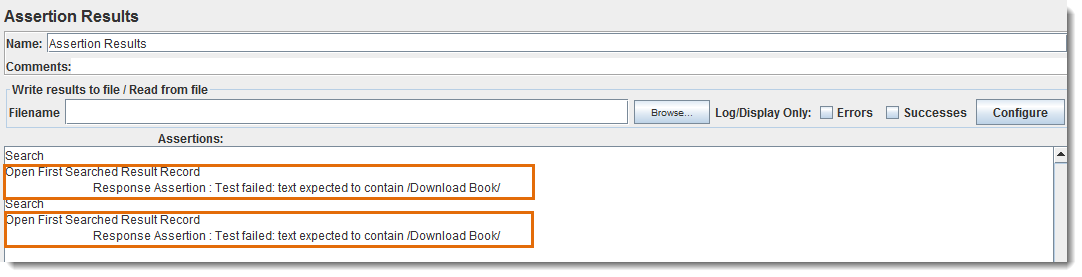
Aggregate report shows a separate table row for each differently named sampler request in the test.



## 3:  Assertion Results:

Assertion Results displays the results of assertions applied on the Sampler. You can see in the below figure, if any assertion fails it will look like this.

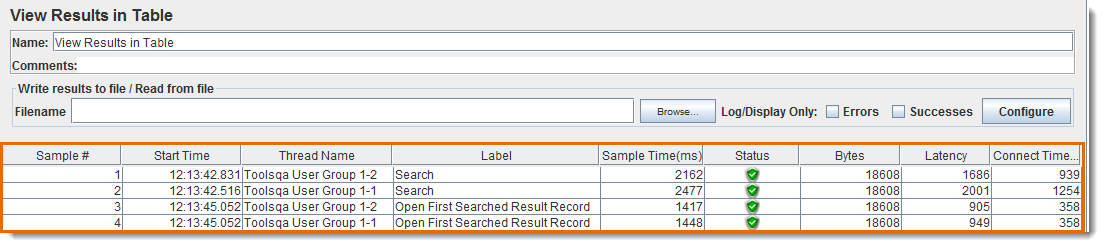
Please Note that Assertion Results **SHOULD NOT BE USED** during Load/Stress test as it consumes a lot of resources like memory and CPU. It is recommended to use it with functional testing or debugging purposes.



## 4:  View Results In Table:

View Results in table creates and displays a row for every sample/request result separately.

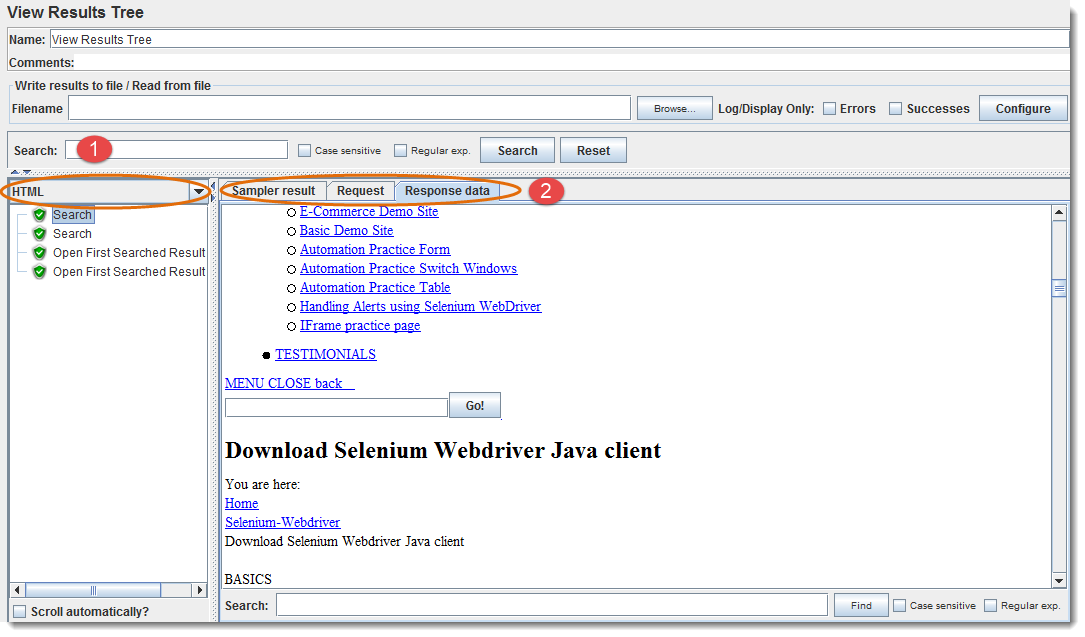
Please Note that View Results In Table **SHOULD NOT BE USED** during Load/Stress test as it consumes a lot of resources like memory and CPU. It is recommended to use it with functional testing or debugging purposes.



## 5:  View Results Tree:

View Results Tree displays a tree consists of all the Sampler responses along with their requests.

Please Note that View Results Tree **SHOULD NOT BE USED** during Load/Stress test as it consumes a lot of resources like memory and CPU. It is recommended to use it with functional testing or debugging purposes.



1. Select desired response format view from available options in dropdown like Text, RegExp Tester, CSS/JQuery Tester, XPath Tester, HTML, HTML (download resources), Document, JSON
2. Select tab Sampler result, Request and Response data