JMeter

**Performance Testing**is a type of [software testing](https://www.geeksforgeeks.org/software-testing-basics/)that ensures [software applications](https://www.geeksforgeeks.org/what-is-application-software/)perform properly under their expected workload. It is a testing technique carried out to determine system performance in terms of sensitivity, reactivity, and stability under a particular workload.

Performance testing aims to identify bottlenecks, measure system performance under various loads and conditions, and ensure that the system can handle the expected number of users or transactions.

# Why use performance testing?

* The objective of performance testing is to eliminate performance congestion.
* It uncovers what needs to be improved before the product is launched in the market.
* The objective of performance testing is to make software rapid.
* The objective of performance testing is to make software stable and reliable.
* The objective of performance testing is to evaluate the performance and scalability of a system or application under various loads and conditions.
* It helps identify bottlenecks, measure system performance, and ensure that the system can handle the expected number of users or transactions.
* It also helps to ensure that the system is reliable, stable, and can handle the expected load in a production environment.

# Why automate performance testing?

Performance testing automation is about establishing a repeatable and consistent process that checks reliability issues at different stages of the development and release cycle. For instance, you could run performance tests from CI/CD pipelines and nightly jobs, or manually trigger load tests and monitor their impact in real-time.

|  |
| --- |
| Performance Test Automation |
| Quick and effective: Executed by special software tools, automated testing requires significantly less time and effort. |
| Better long-term ROI: While requiring some upfront investment, testing automation proves to be more cost-efficient in the long run. |
| Transparent and meticulous: Testing automation gives better transparency, collaboration and visibility to the team. |

# Types of performance testing

**1. Load testing**

[Load testing](https://www.geeksforgeeks.org/software-testing-load-testing/)simulates a real-world load on the system to see how it performs under stress. It helps identify bottlenecks and determine the maximum number of users or transactions the system can handle. It checks the product’s ability to perform under anticipated user loads. The objective is to identify performance congestion before the software product is launched in the market.

**2. Stress testing**

[Stress testing](https://www.geeksforgeeks.org/stress-testing-software-testing/)is a type of load testing that tests the system’s ability to handle a high load above normal usage levels. It helps identify the breaking point of the system and any potential issues that may occur under heavy load conditions. It involves testing a product under extreme workloads to see whether it handles high traffic or not. The objective is to identify the breaking point of a software product.

**3. Spike testing**

[Spike testing](https://www.geeksforgeeks.org/spike-testing-software-testing/)is a type of load testing that tests the system’s ability to handle sudden spikes in traffic. It helps identify any issues that may occur when the system is suddenly hit with a high number of requests. It tests the product’s reaction to sudden large spikes in the load generated by users.

**4. Soak testing**

[Soak testing](https://www.geeksforgeeks.org/soak-testing-software-testing/)is a type of load testing that tests the system’s ability to handle a sustained load over a prolonged period. It helps identify any issues that may occur after prolonged usage of the system.

**5. Endurance testing**

[Endurance testing](https://www.geeksforgeeks.org/software-testing-endurance-testing/)is similar to soak testing, but it focuses on the system's long-term behaviour under a constant load. It is performed to ensure the software can handle the expected load over a long period.

**6. Volume testing**

In [Volume testing](https://www.geeksforgeeks.org/volume-testing/), a large number of data is saved in a database and the overall software system’s behaviour is observed. The objective is to check the product’s performance under varying database volumes.

**7. Scalability testing**

In [Scalability testing](https://www.geeksforgeeks.org/software-testing-scalability-testing/), the software application’s effectiveness is determined by scaling up to support an increase in user load. It helps in planning capacity additions to your software system.

# Performance test process

Conducting performance testing involves several steps to ensure that a software application can handle expected loads and perform well under stress.

# Performance testing matrixs

* **Response time:** The time that passes from the moment of server request until the last byte is received from the server is considered as response time. This performance testing metric is measured in kilobytes per second (KB/sec).
* **Requests per second:** When the client application forms an HTTP request and sends it to a server, a response is generated and sent back to the client. The total consistent requests processed per second is a key performance metric – requests per second (RPS).
* **User transactions:** The user transactions are a sequence of user actions via the interface of the software. By comparing the expected time with the transaction time (number of transactions per second), you can evaluate the load performance of the software application.
* **Virtual users per unit of time:** This performance testing metric helps in finding out whether the software performance meets the expected requirements. It helps the QA team in estimating the average load and software behaviour in different load conditions.
* **Error rate:** This metric evaluates the ratio of valid to invalid answers over time. Usually, the error occurs when the load exceeds its capacity. And the results are calculated in percentage.
* **Wait time:** This metric is also called average latency. It indicates how much time is passed from the moment a request is sent to the server until the first byte is received. Don’t confuse it with the response time – both consider different time frames.
* **Average load time:** A study indicated that over 40% of users expect to abandon a website if it takes over 3 seconds to load. This performance testing metric is the evaluation of the average time taken to deliver the request. It is one of the most important parameters to ensure the best product quality.
* **Peak response time:** This metric is similar to the average load time but the key difference is that the peak response time indicates the maximum time taken to fulfill a request. And it also shows that at least one of the components of software is problematic. That’s why this parameter is much more important than the average load time.
* **Concurrent users:** This metric is also known as load size and indicates the number of active users at any point.  It is one of the most widely used metrics to understand the software behaviour under a specific number of virtual users.
* **Transactions passed/failed:** This metric expresses the percentage of passed or failed requests against the total conducted tests. It is as critical as the load time for users and considered as one of the most evident metrics to ensure product performance.
* **Throughput:** Throughput shows the used bandwidth within the testing process. It indicates the maximum amount of data flowing through the network connection within a given amount of time. It is measured in KB/second.
* **CPU utilization:** This metric evaluates the time taken by the central processing unit to process the request at a given time.
* **Memory utilization:** This metric indicates the resources taken to process a request, in context to the physical memory on a specific device used for the tests.

# Performance Testing Tools

1. **Apache JMeter**: is an open-source tool used for performance testing and load testing of applications. It simulates multiple users sending requests to a server, collecting performance metrics to analyse the application’s behaviour under different load conditions.
2. **Open STA (Open, Systems Testing Architecture)** is an open-source tool used for performance testing and measuring the load and stress on web applications. Its main function is to simulate multiple users accessing a site simultaneously to evaluate its performance and scalability.
3. **Load Runner**: LoadRunner is a performance testing tool used to simulate virtual users and analyze the behavior of applications under load. It helps identify performance bottlenecks by measuring system performance and response times under varying conditions.
4. **Web Load**: Web Load is a performance testing tool designed to test the scalability and reliability of web applications under various load conditions. The function by sending user requests to a web server, which processes the requests and sends back the appropriate web pages or data.
5. **Gatling**: Gatling is an open-source load testing tool designed to analyze and measure the performance of web applications. Its main function is to simulate a large number of users interacting with a website to identify performance bottlenecks and ensure the site can handle high traffic.
6. **BlazeMeter**: BlazeMeter is a cloud-based testing platform designed to simulate large-scale user load and measure application performance. It allows developers and testers to run continuous performance tests and analyze results to ensure their applications can handle high traffic and perform optimally under stress.

# About JMeter

jMeter is an Open Source testing software. It is 100% pure Java application for load and performance testing.

jMeter is designed to cover various categories of tests such as load testing, functional testing, performance testing, regression testing, etc., and it requires JDK 5 or higher.

# 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,
* 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.
* 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.

# 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.

# What is a Test Plan?

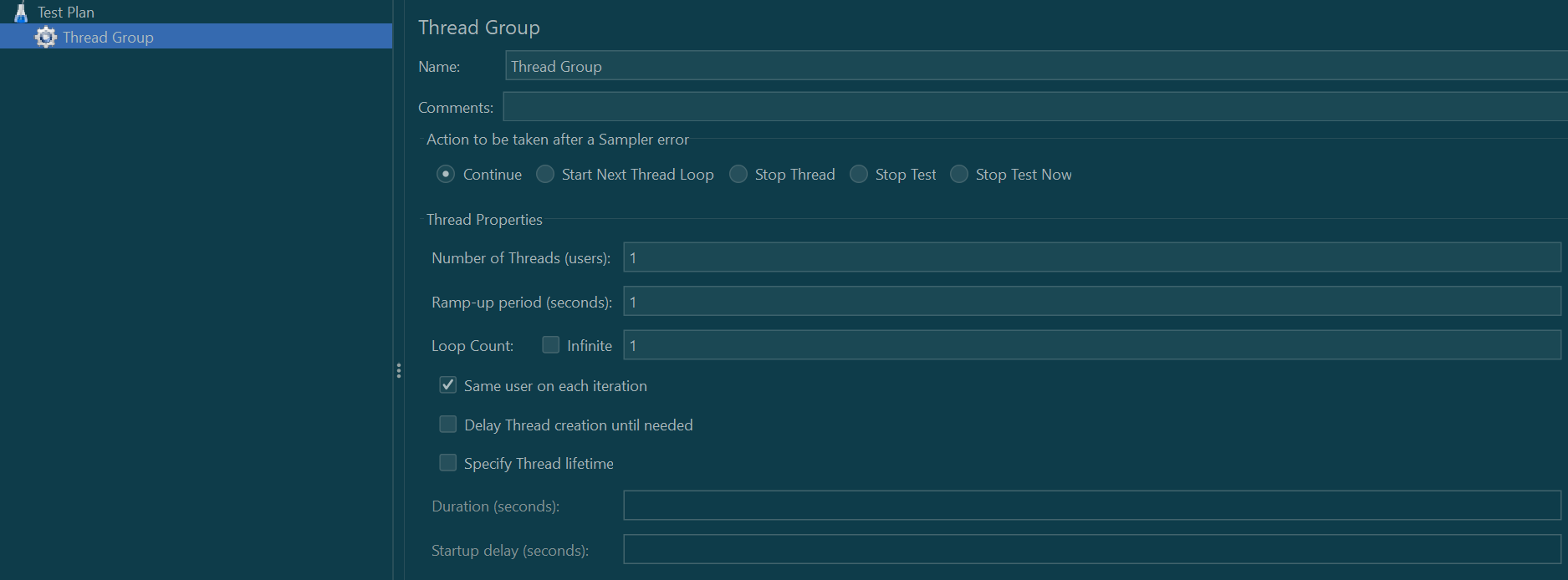
A Test Plan can be viewed as a container for running tests. It defines what to test and how to go about it. A complete test plan consists of one or more elements such as thread groups, logic controllers, sample-generating controllers, listeners, timers, assertions, and configuration elements. A test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

# JMeter TestPlan Elements

A JMeter Test Plan comprises of test elements. A Test Plan comprises of at least one Thread Group. Within each Thread Group, we may place a combination of one or more of other elements − Sampler, Logic Controller, Configuration Element, Listener, and Timer.

## 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.



It consists of:

* **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.

## 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 examples of samplers:

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

### 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 etc.

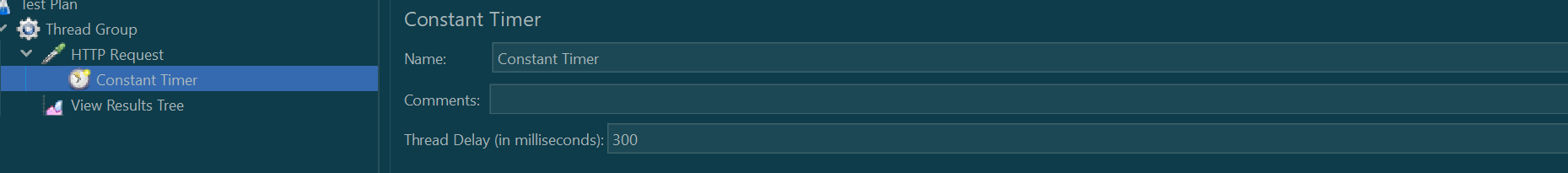
# Listeners

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.

# 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 that allows you to define a period to wait between each request.

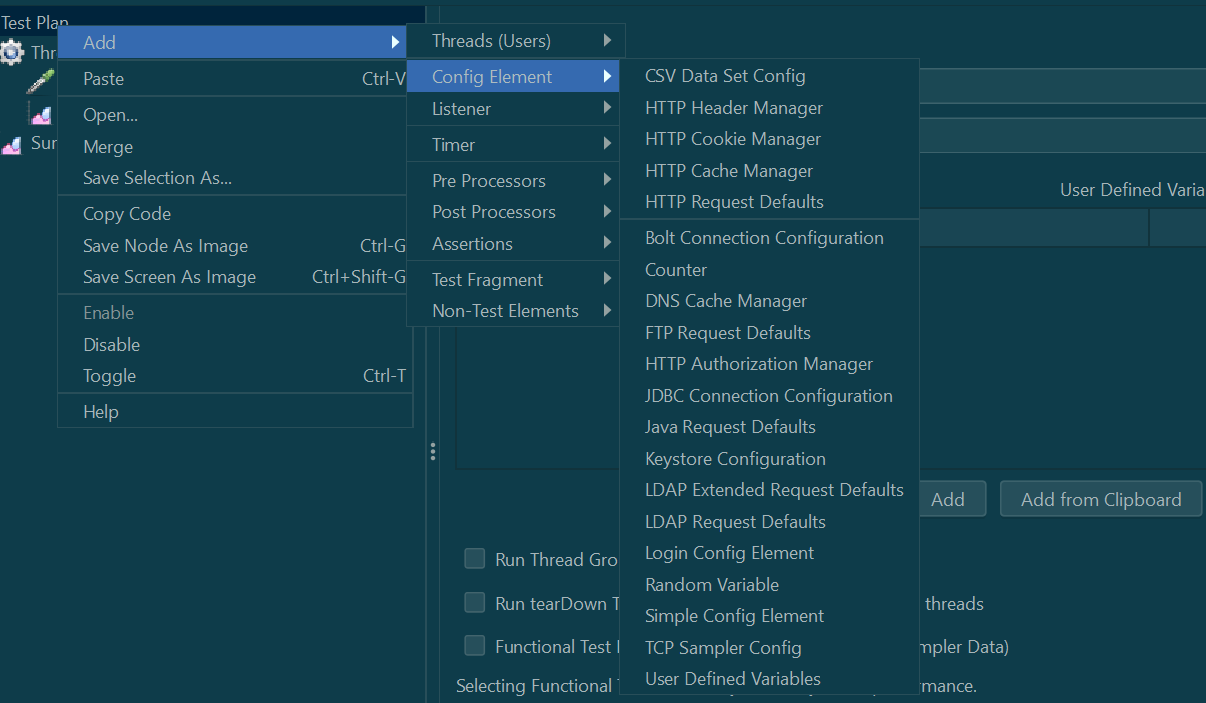


# Assertions

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.

# 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.



# Execution order of test Elements

Following is the execution order of the test plan elements −

* Configuration elements
* Pre-Processors
* Timers
* Sampler
* Post-Processors (unless SampleResult is null)
* Assertions (unless SampleResult is null)
* Listeners (unless SampleResult is null)

# Working with JMeter

**Step 1: Launch JMeter**

Extract the JMeter archive to your desired location and launch it by running the JMeter Windows Batch file from the ‘bin’ directory.

**Step 2: Create a Test Plan**

Open JMeter, navigate to**File -> New Test Plan** and provide a name for your test plan to create a new testing project within the JMeter interface.

**Step 3: Add a Thread Group**

Right-click on the Test Plan, and select**Add -> Threads (Users) -> Thread Group i**n the tree view.

Configure the Thread Group by setting the number of threads, ramp-up period, and loop count for simulating user behavior and load in the performance test.

**Step 4: Add Sampler (HTTP Request)**

Inside the Thread Group, right-click and choose **Add -> Sampler -> HTTP Request** to add an HTTP Request Sampler.

Configure the sampler by providing the server details such as server name, port, and path to simulate HTTP requests in the performance test.

**Step 5: Add Listeners**

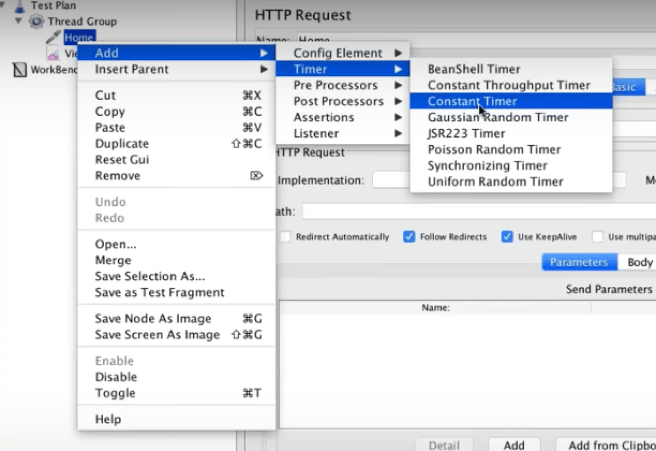
Within the Thread Group, right-click and select **Add -> Listener.** Common listeners, such as View Results Tree, Summary Report, and Response Times Over Time, can be added. Configure listeners to display pertinent information about the test results.

**Step 6: Add Timers**

When users click on your website or app, they naturally have pauses and delays. These can be simulated with [Timers](https://www.blazemeter.com/blog/jmeter-timer).

Constant timers are the most common. They determine how many milliseconds to wait between requests.

Right-click your HTTP Request, select “Add”, select “Timer”, then select “Constant Timer”



**Step 7: Run the Test**

Save your test plan with a meaningful name using the “**File -> Save Test Plan**” option.

Click on the “**Run**” menu, then select “**Start**,” or press the “**Play**” button to initiate the test. Monitor the test progress in the listeners to analyze the performance metrics.

# JMeter test Plan and Workbench

**What is a Test Plan?**

Test Plan is where you add elements required for your JMeter Test.

It stores all the elements (like ThreadGroup, Timers etc) and their corresponding settings required to run your desired Tests.

**What is WorkBench?**

The WorkBench simply provides a place to store test elements **temporarily**. WorkBench has no relation with Test Plan. JMeter will **not save** the contents of the WorkBench. It only saves the contents of the[Test Plan](https://www.guru99.com/test-planning.html)branch

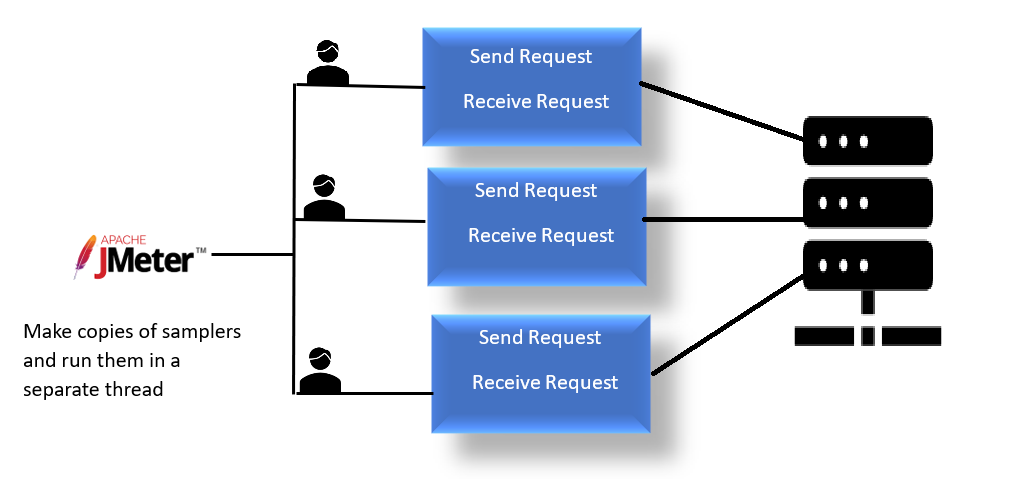
Since JMeter 4: Workbench has been dropped from UI, you can now use Non Test Elements as immediate children of Test Plan.

# Performance and Load testing using JMeter

**JMeter Performance Testing** is[Testing](https://www.guru99.com/software-testing.html)method performed using Apache JMeter to test the performance of a web application. JMeter for performance testing helps to test both static and dynamic resources, helps to discover concurrent users on website and provides variety of graphical analysis for performance testing. JMeter performance testing includes load test and stress test of web application.

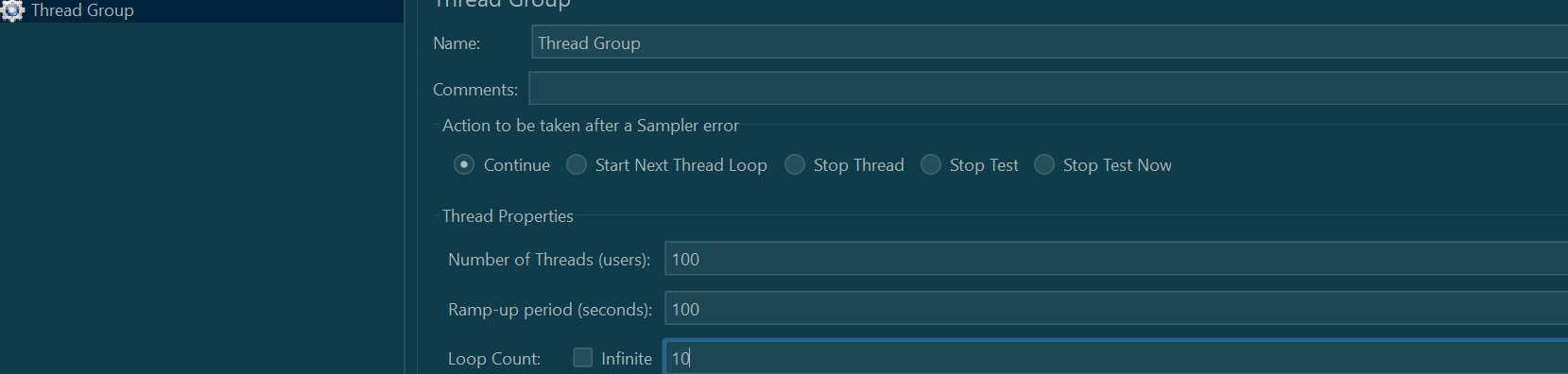
* **Load**Testing: Modeling the expected usage by simulating multiple user access the[Web services](https://www.guru99.com/web-services-tutorial.html)concurrently.
* **Stress** Testing: Every web server has a maximum load capacity. When the load goes beyond the limit, the web server starts responding slowly and produce errors. The purpose of the [Stress Testing](https://www.guru99.com/stress-testing-tutorial.html) is to find the maximum load the web server can handle.

## Process



Step 1: Add Thread group

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



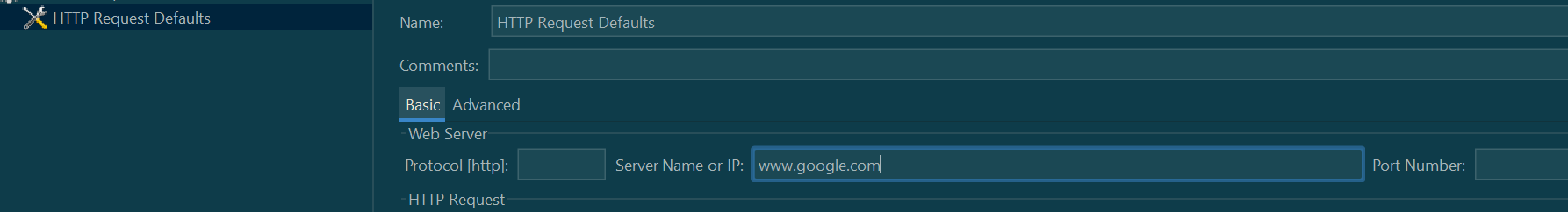
* **Number of Threads**: 100 (Number of users connects to the target website: 100)
* **Loop Count**: 10 (Number of time to execute testing)
* **Ramp-Up Period**: 100

Note- Thread count is the number of users and loop count is number of times each user will iterate through

Step 2 – Adding JMeter elements

* **HTTP request Default**

This element can be added by right-clicking on the Thread Group and selecting: **Add**->**Config Element**->**HTTP Request Defaults.**



* **HTTP Request**

Right-click on Thread Group and select: **Add**-> **Sampler**-> **HTTP Request**.

In HTTP Request Control Panel, the Path field indicates which **URL request** you want to send to Google server.

For example, if you enter “*calendar*” in Path field. JMeter will create the URL request [http://www.google.com/calendar](https://www.google.com/calendar) to Google server

Step 3: Adding Graph result

JMeter can show the test result in Graph format.

Right click Test Plan, **Add**-> **Listener**-> **Graph Results**

Step 4: Run the test and get the test result.

Press **the Run** button (Ctrl + R) on the Toolbar to start the software testing process. You will see the test result display on Graph in the real time.



The **Throughput** is the most important parameter. It represents the ability of the server to handle a heavy load. The **higher** the Throughput is, the **better** is the server performance.

In this test, the throughput of Google server is 253/minute. It means Google server can handle 253 requests per minute.

The **deviation** is shown in red – it indicates the deviation from the average. The **smaller** the **better**.

# Constant, Gaussian Random, Uniform

What are timers?

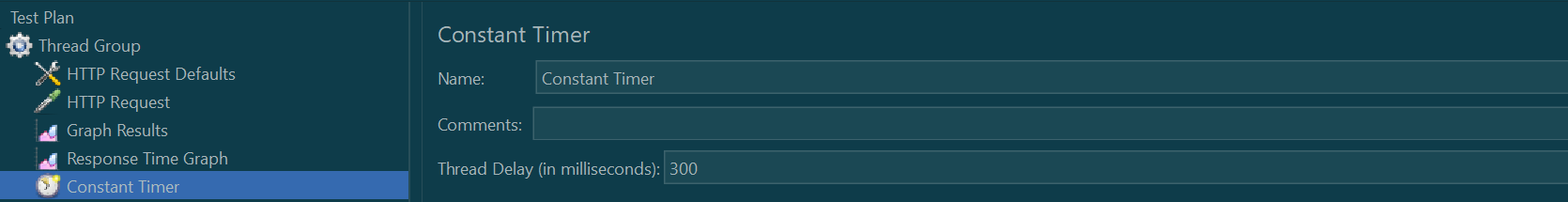
By default, JMeter sends the request without pausing between each request. In that case, JMeter could overwhelm your test server by making too many requests in a short amount of time.

Timers allow JMeter to delay each request that a thread makes. A timer can solve the server overload problem.

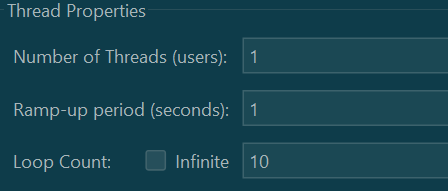
Also, in real life visitors do not arrive at a website all at the same time, but at different time intervals. So Timer will help mimic the real-time behaviour.

## Constant Timer

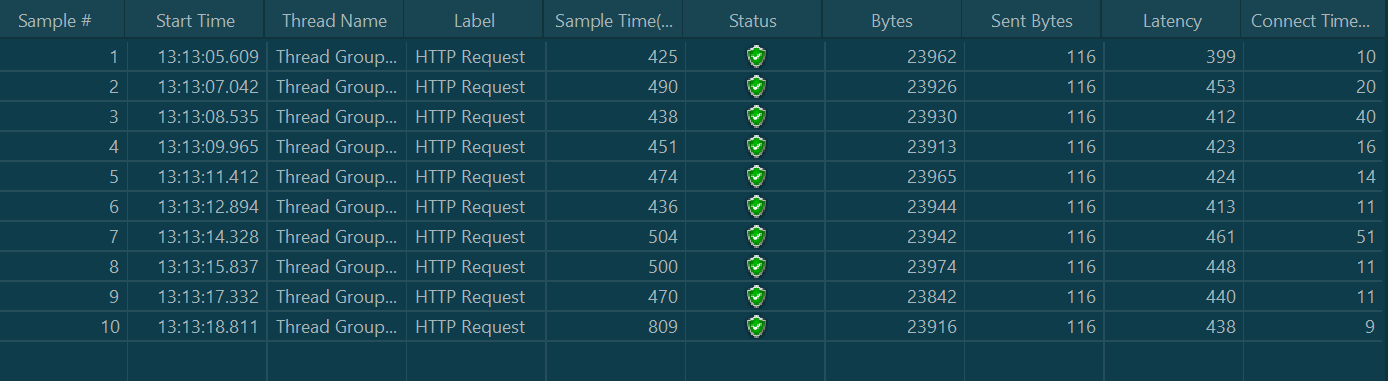
Constant timer delays each user request for the **same** amount of time.



Let us create for thread group



We will set the constant timer for 1000ms



Listener added 🡪 View result in a table

* **Start time**is 13:13:05.609
* **Sample Time** of Sample 1 is 425
* **Constant Timer**: 1000 ms (as configured)
* **End Time** of this sample is = 13:13:05.609+ 425+ 1000 = 13:13:07.042
* So the Sample 2 should start at the time is 13:13:07.042 ( As shown in the above figure)

The **delay** of each sample is **1000** ms

## Gaussian Random Timer

[Gaussian](http://hyperphysics.phy-astr.gsu.edu/hbase/math/gaufcn.html) random timer delays each user request for a **random** amount of time.



|  |  |
| --- | --- |
| Attribute | Description |
| Name | Descriptive name for this timer that is shown in the tree |
| Deviations (milliseconds) | A **parameter** of Gaussian Distribution Function |
| Constant Delay Offset (milliseconds) | **Additional** value in milliseconds |

So total delay will be:

**Return value of Gaussian distribution function**

# Uniform Random Timer

Uniform random timer delays each user request for a random amount of time.



**Parameters**

|  |  |
| --- | --- |
| Attribute | Description |
| Name | Descriptive name for this timer that is shown in the tree |
| Random Delay Maximum | Maximum random number of milliseconds to delay. |
| Constant Delay Offset (milliseconds) | **Additional** value in milliseconds |

**The total delay is the sum of the random value and the offset value.**

# Assertions in Jmeter

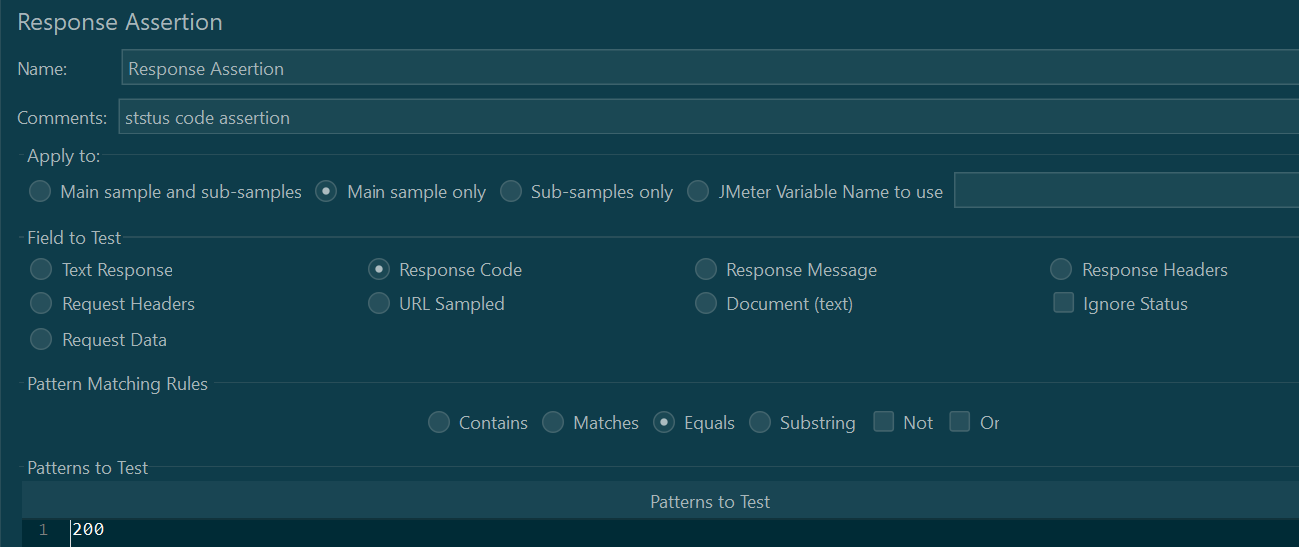
Assertion help verify that your server under test returns the **expected** results.

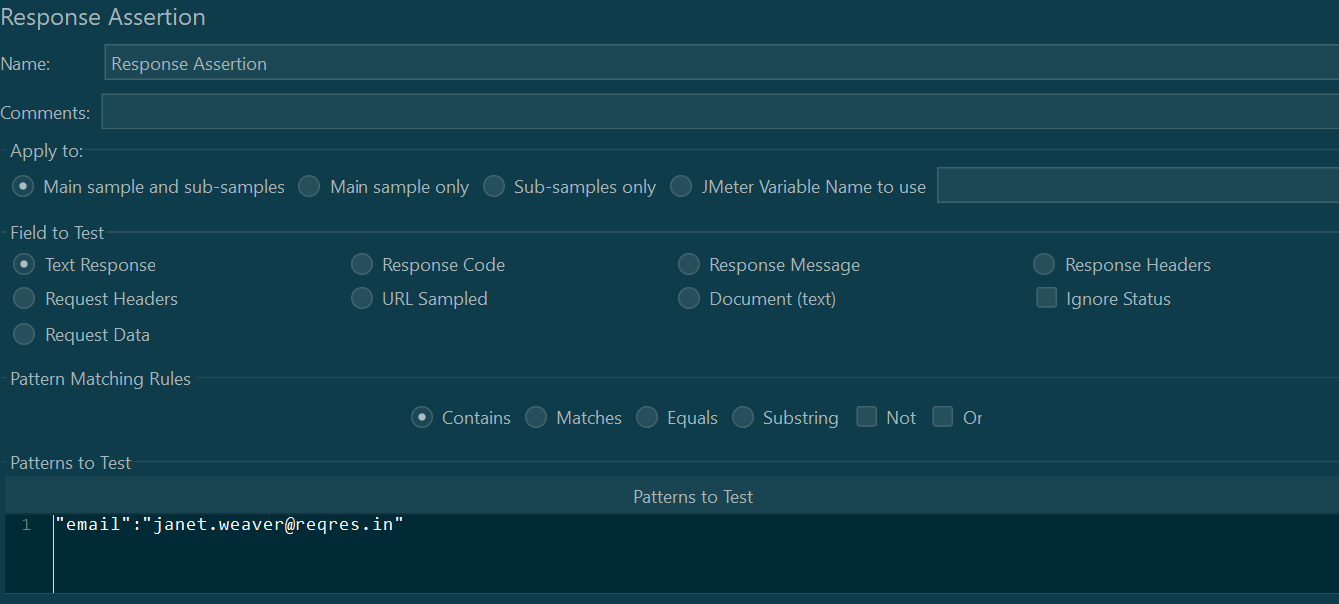
## Types of assertions

* Response Assertion
* Duration Assertion
* Size Assertion
* XML Assertion
* HTML Assertion

### Response assertions

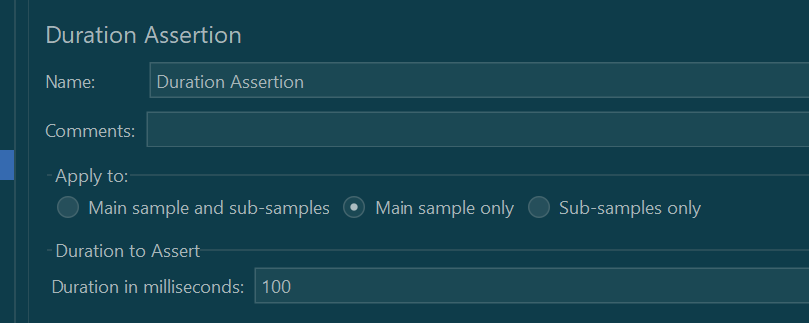
The response assertion lets you add pattern strings to be compared against various fields of the server response.

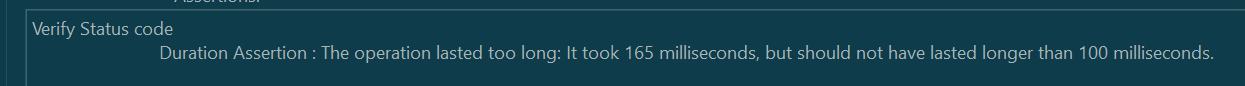




### Duration Assertion

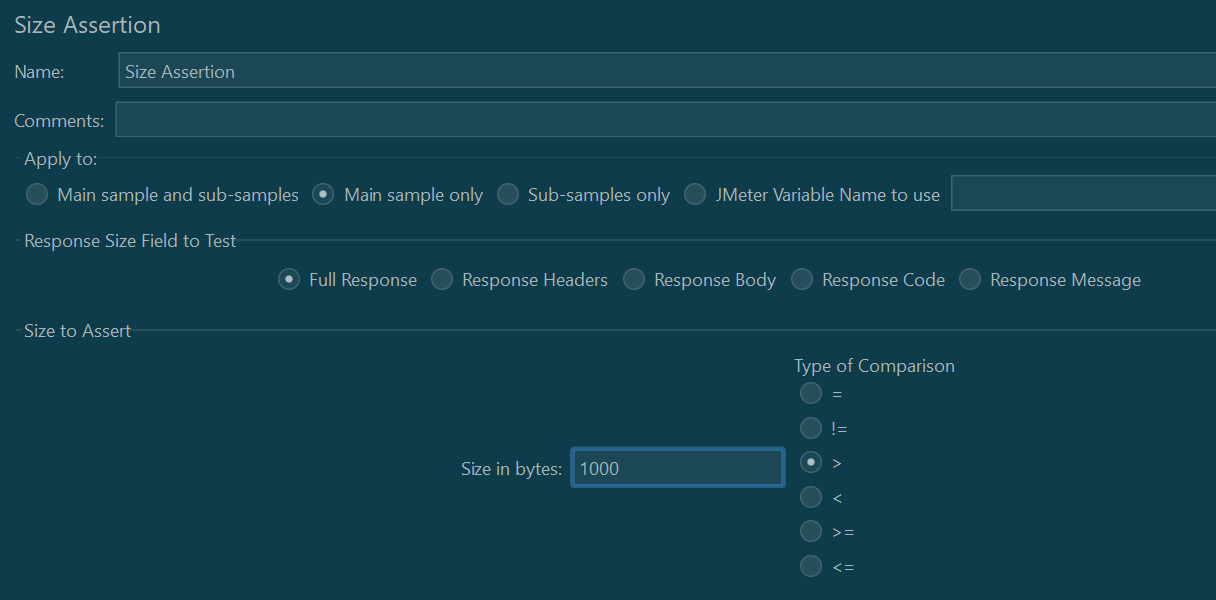
The Duration Assertion tests that each server response was received within a given amount of time. Any response that takes longer than the given number of milliseconds (specified by the user) is marked as a failed response.

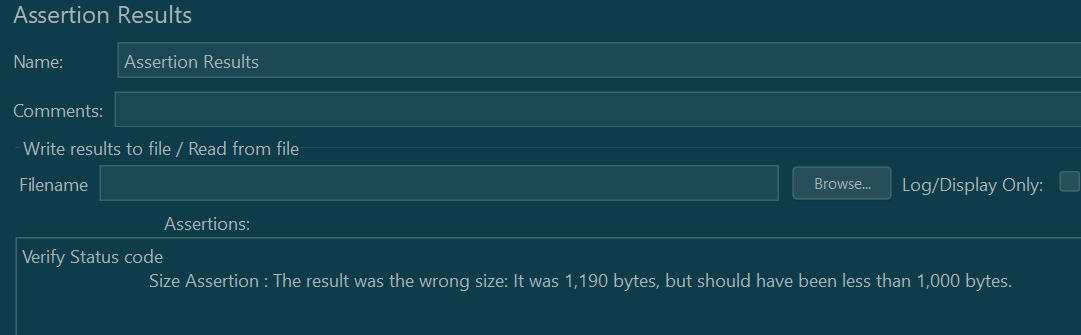




### Size Assertion

The Size Assertion tests that each server response contains the expected number of byte in it. You can specify that the size be equal to, greater than, less than, or not equal to a given number of bytes.





### XML Assertion

The [XML](https://en.wikipedia.org/wiki/XML) Assertion tests that the response data consists of a formally correct XML document.

### HTML Assertion

The HTML Assertion allows the user to check the HTML syntax of the response data. It means the response data must be met the HTML syntax.

# Logic Controller

Logic Controllers let you define the order of processing requests in a Thread. It lets you control “when” to send a user request to a web server. For example, you can use Random Controllers to send HTTP requests to the server randomly

Logic Controllers determine the order in which user request is executed.

## Loop Controller

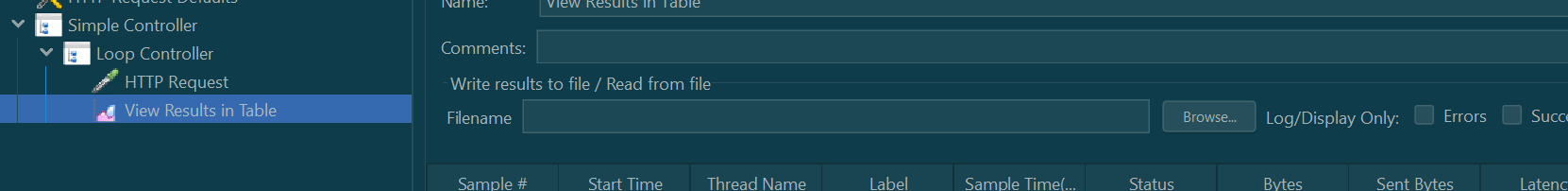
Loop Controller makes the user request run a specified number of times or run forever

* 1. Add Thread group
  2. Add loop controller
  3. Configure loop controller
  4. Add HTTP Request
  5. Add View results in table

Loop will run for number of times mentioned in the thread group x count mentioned in the loop controller

## 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.



## Transaction Controller

A Transaction Controller measures the response time of a group of requests (or samplers in JMeter speak) as a single transaction. This can be extremely useful for measuring the response time of related requests.

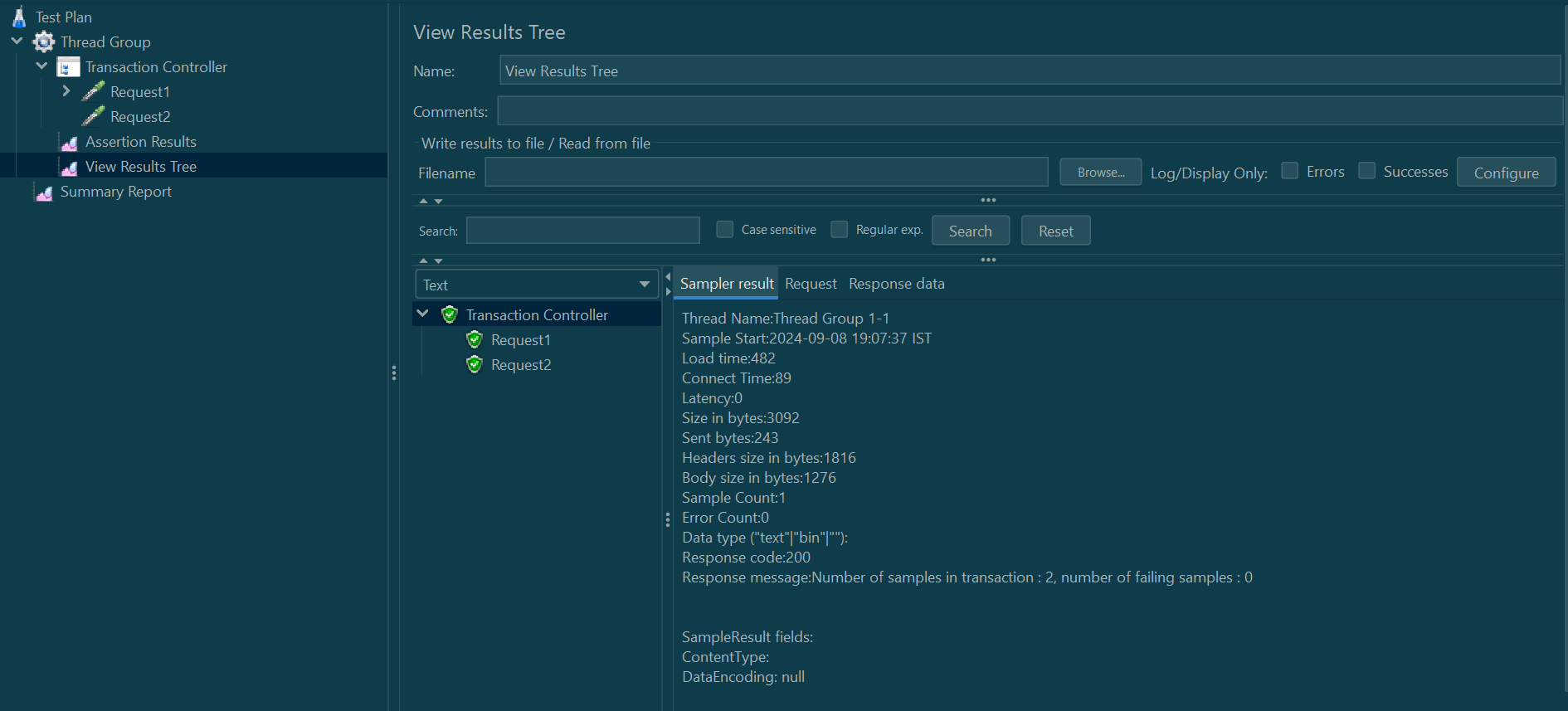
The Transaction Controller works by grouping requests together under a single parent element. The parent element is the Transaction Controller, and the child elements are the requests that make up the transaction. When the transaction is executed, JMeter measures the response time of the entire transaction, which includes the response time of each individual request.

A transaction controller can include child transaction controllers. There is no limit to how deep your transaction hierarchy can go.

Transaction controller is used for:

**Measuring the response time of a transaction:** By grouping related samplers together under a single transaction, the Transaction Controller allows testers to measure the response time of the entire transaction.

**Simplify and provide a logical structure to the test plan:** Grouping related requests together makes your test plan simpler and more logical, making it easier to read and manage.



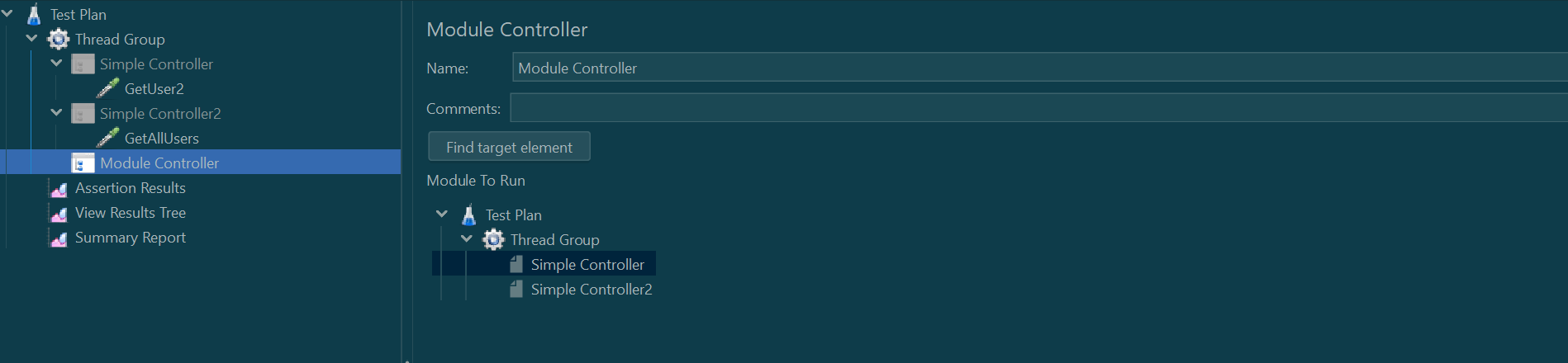
## Module Controller

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, we should disable Simple Controller.

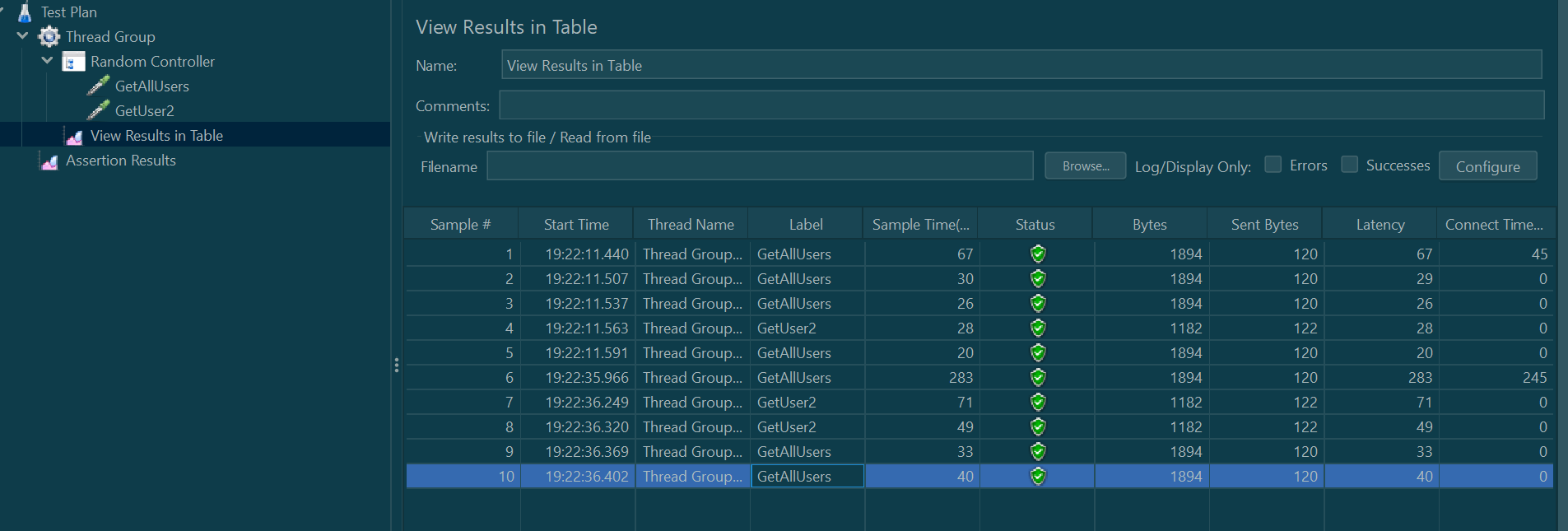
Select Controller    >   Right Click   >   Select “**Disable**” Option

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



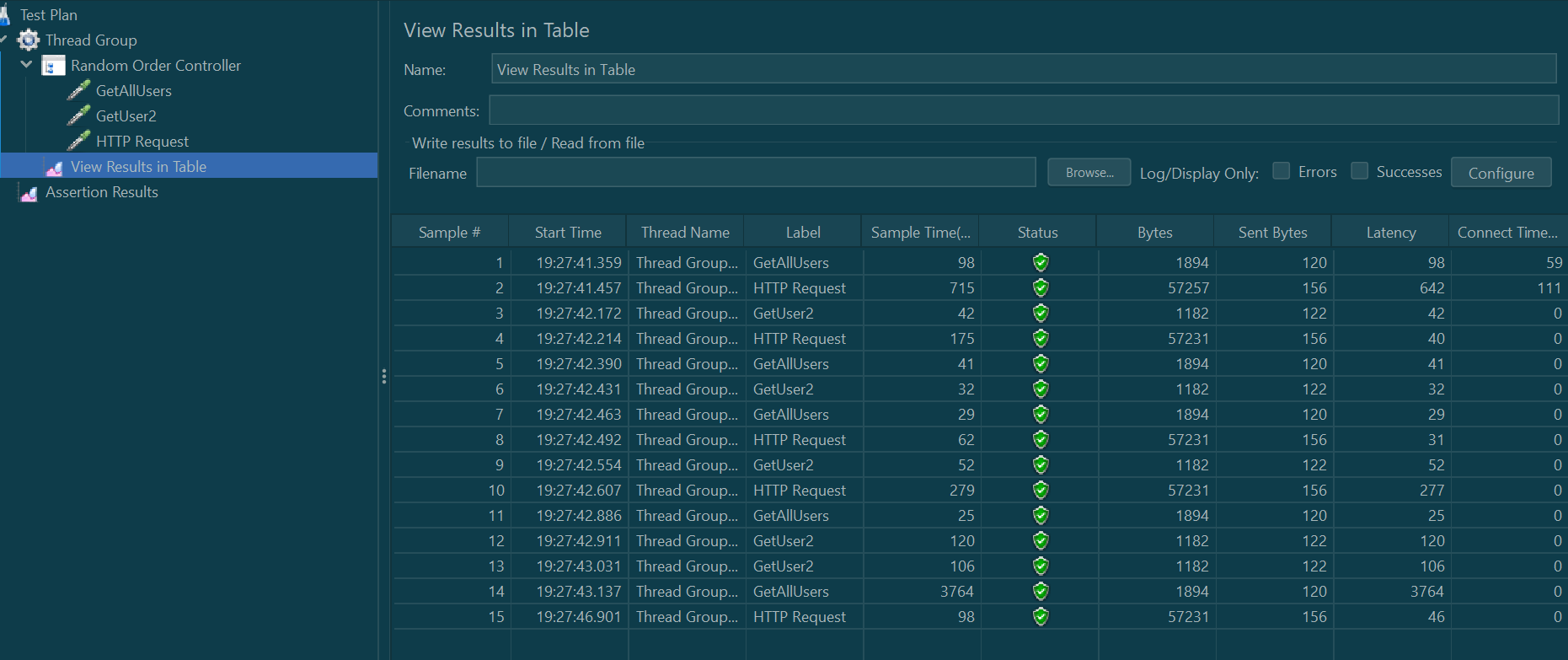
## Random Controller

Random Controller 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.



## Random order controller

Random Order controller is similar to “***Simple Controller***”, it will make all samplers run in random order in each thread loop. Each sampler is executed for a number of iterations randomly.



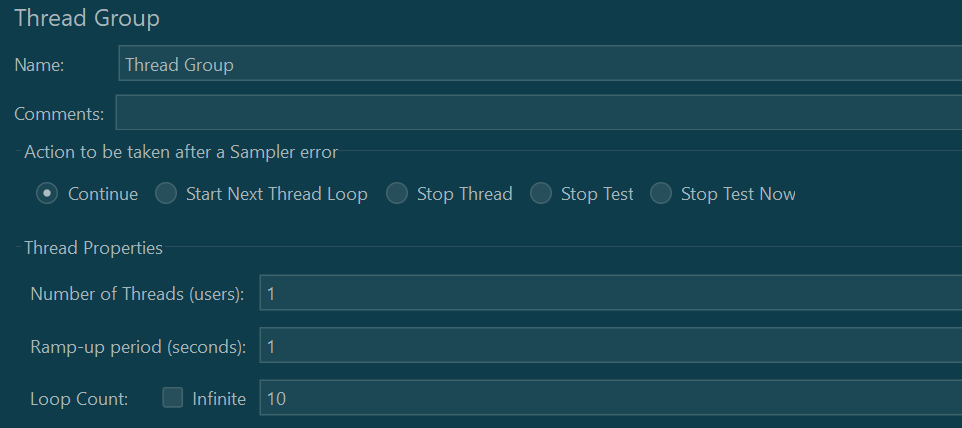
# Pre-processor and Post Processor

When building your JMeter test, [samplers](https://jmeter.apache.org/usermanual/component_reference.html#samplers) provide the core functionality. They are responsible for sending the actual HTTP requests and handling those responses. However, as your test becomes more complex and sophisticated, it may be desirable to add conditional logic to your requests. Furthermore, you may wish to dynamically process the inputs and outputs to these. JMeter provides this functionality though a host of [preprocessors](https://jmeter.apache.org/usermanual/component_reference.html#samplers) and [postprocessors](https://jmeter.apache.org/usermanual/component_reference.html#samplers).

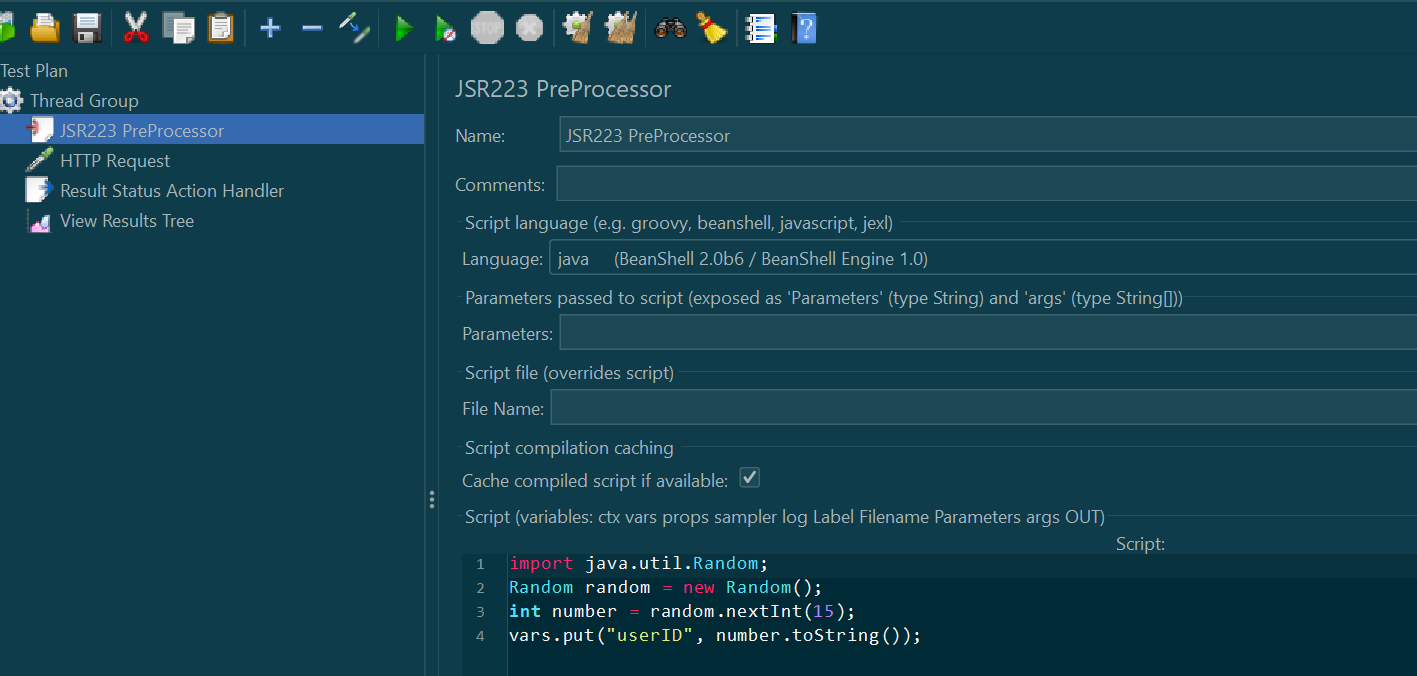
The key difference between a preprocessor and a postprocessor is *when* your custom logic is executed. As the name suggests, a preprocessor runs before any requests existing at the same level in your test plan.

Lets say, we want to write a preprocessor which will generate a random number and pass as variable to our https request. We will also add a post processor, which will stop the iterations if any request fails.

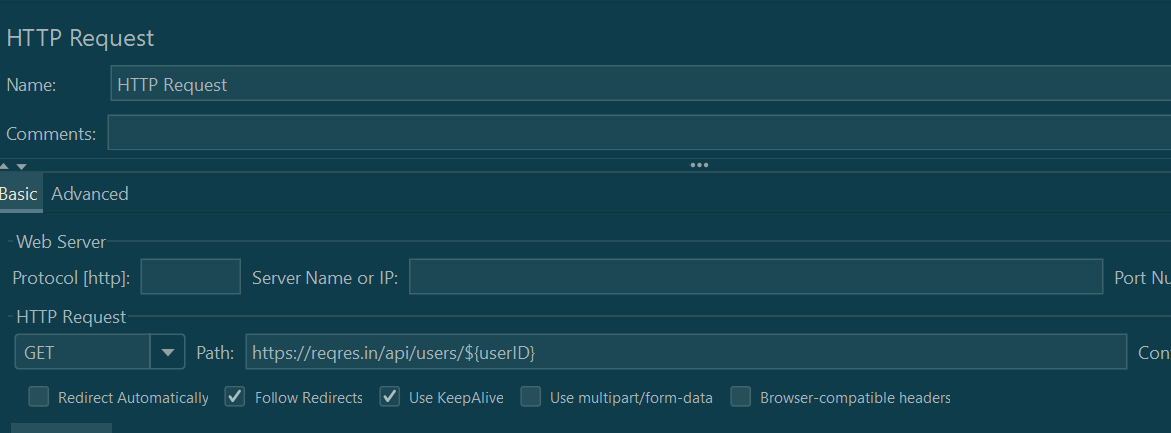
* 1. We will add a thread group with below config



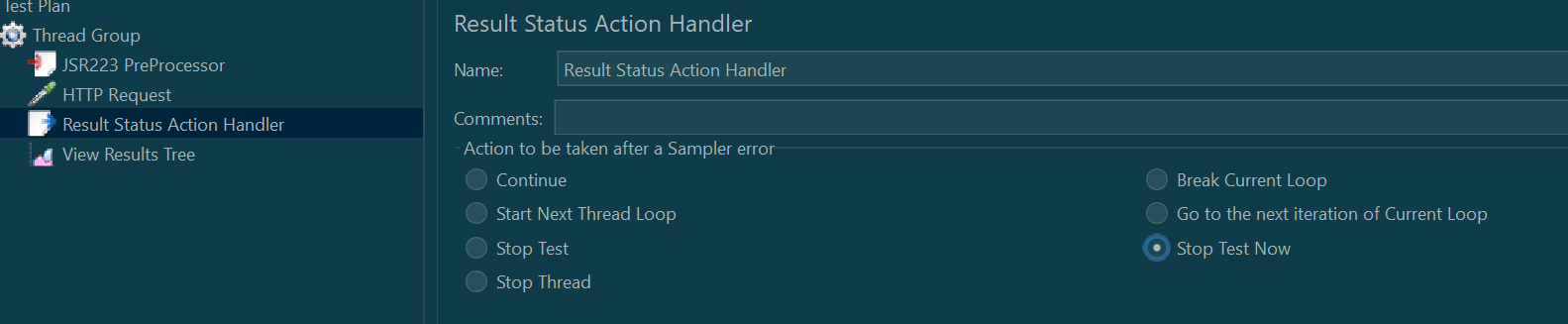
* 1. We will add a JSR223 pre processor to generate a random number and store in a variable



* 1. We will add a http request sampler, which will take the userID as input parameter in request URL.



* 1. We will add a post processor which will stop the iteration if any request fails



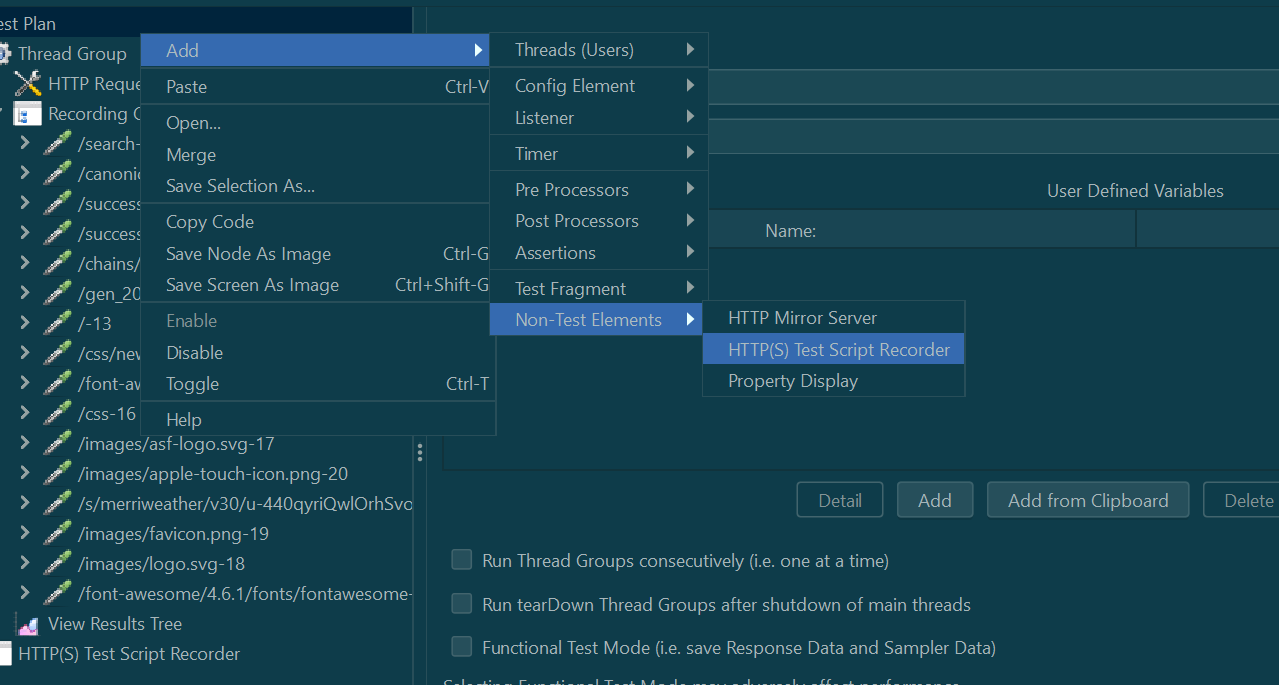
# HTTP Proxy Server in JMeter: Record Example Script

Record Testing help tester to record & run their activity against test target. It is a type of automated testing but for multiple users.

The Proxy Server allows JMeter to watch and record user activity while they are browsing web application with a normal browser.

Steps:

* 1. Add a thread group
  2. Add config elements http request header
  3. Add recording controller
  4. Add Https script recorder



* 1. Change browser proxy settings
  2. Start recording
  3. Run the script(recording will be saved in recording controller)

# JMeter functions

JMeter functions can be referred as special values that can populate fields of any Sampler or other element in a test tree.

Syntax:

${\_\_functionName(var1,var2,var3)}

Here "\_\_functionName" matches the name of a function and Parentheses surround the parameters sent to the function.

**Example:**

Log function:

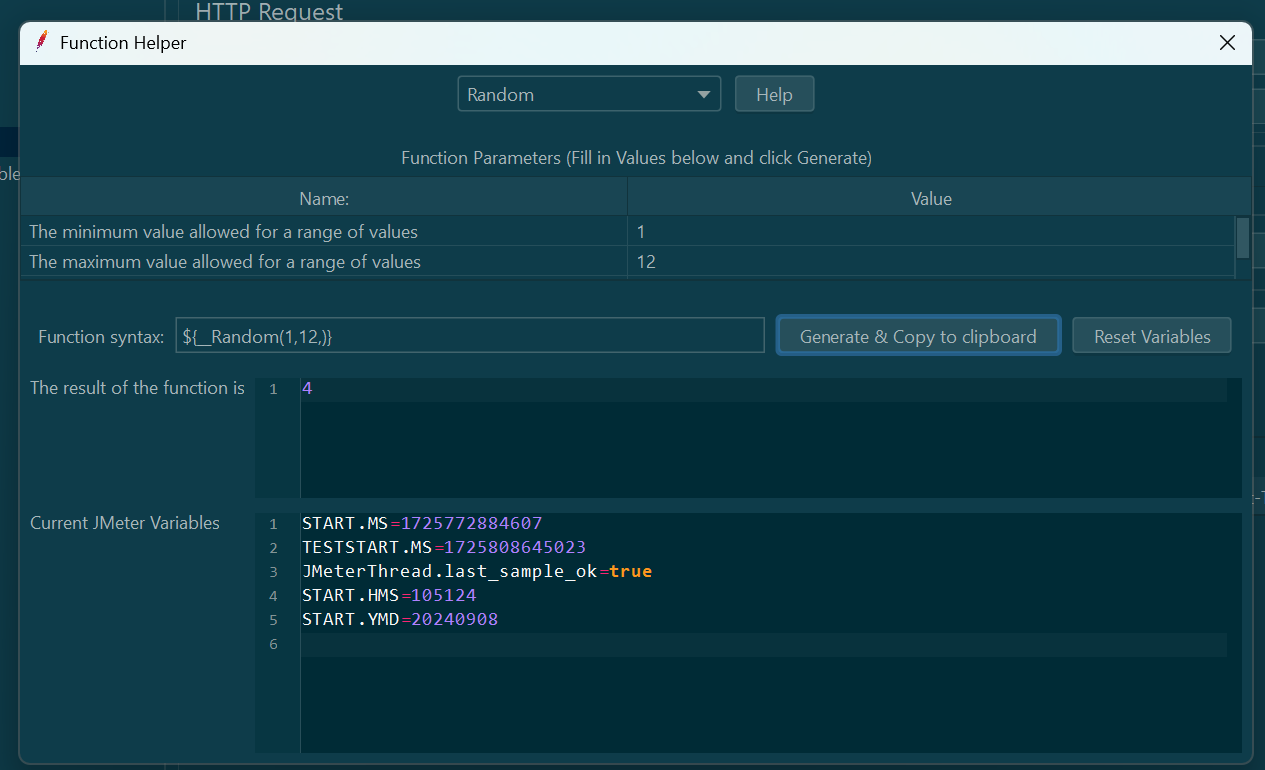
${\_\_log("Users")} 🡪 Give this as sampler name

**Random Function**

[https://reqres.in/api/users/${\_\_Random(1,12,)}](https://reqres.in/api/users/$%7b__Random(1,12,)%7d)

## Function Helper

The Function Helper dialog is available from JMeter's Tools menu.



Using the Function Helper, you can select a function from the pull down, and assign values for its arguments. The left column in the table provides a brief description of the argument, and the right column is where you write in the value for that argument. Different functions take different arguments.

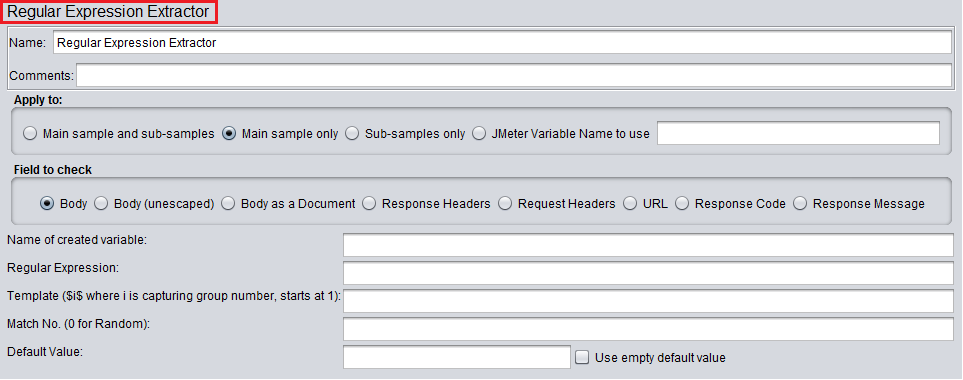
Once you have done this, click the "generate" button, and the appropriate string is generated for you to copy-paste into your test plan wherever you like.

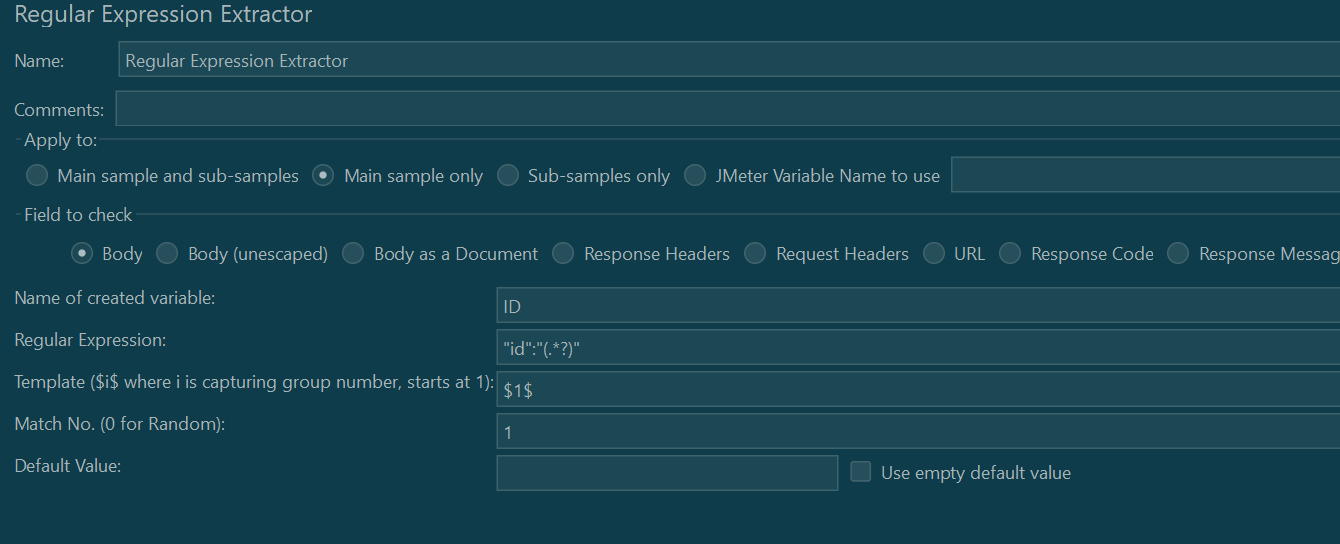
# Regular Expression Extractor in JMeter

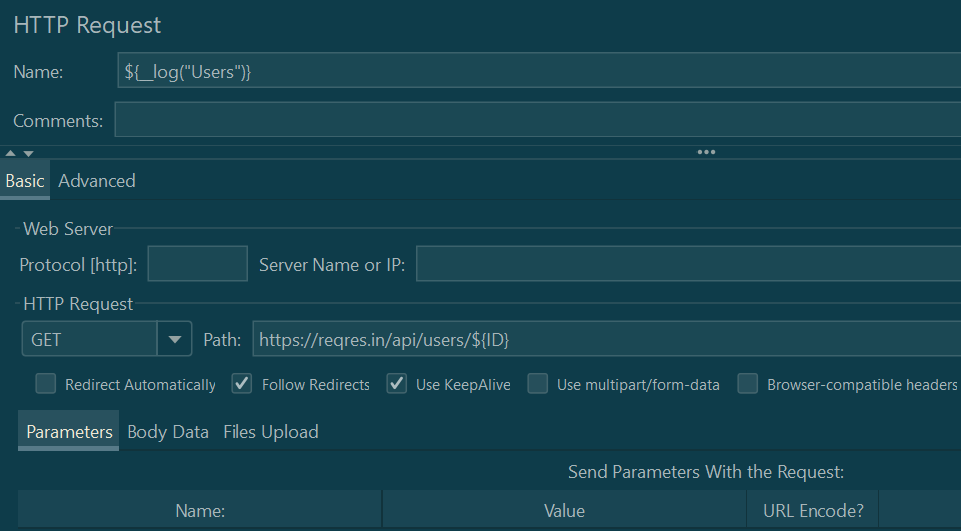
Regular Expression Extractor uses regular expression string to fetch some values from the response of one request and pass it on to another request. The value(s) that are extracted can be stored in any variable and can be used for reference in any further request in the test plan. Values can be extracted from Response Body, Response/Request Headers, URL, Response Code, and Response Message of the request when you add these Post Processors to it.

Using of Regular Expression Extractor in a test plan will save us a lot of time in creating or enhancing a test plan.

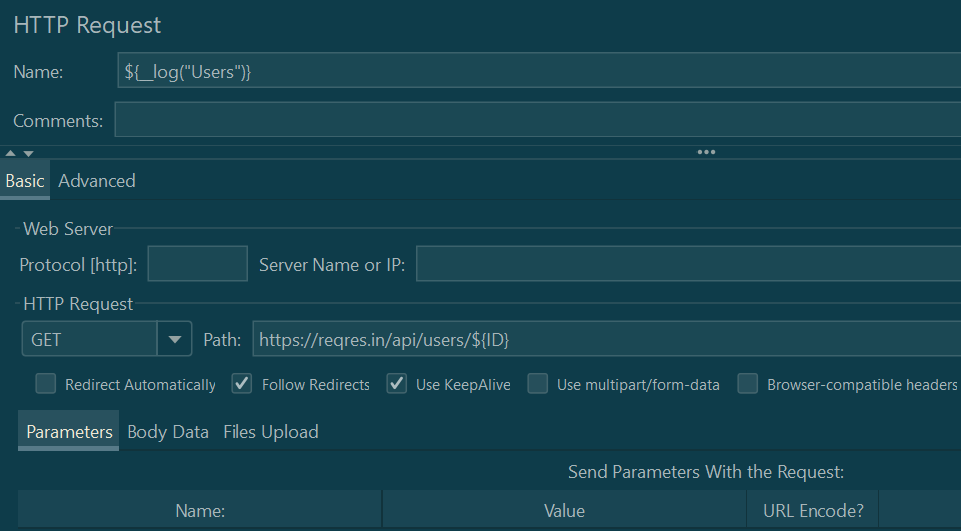
Adding Regular Expression Extractor in a test plan:  
Choose a request in a Test Plan from which you want to extract the value and Right click the request and select Add >>Post Processors >>Regular Expression Extractor.







We can check the regular expression in View Result Tree.



Variables in Response assertions

