Apache JMeter User's Manual

Click on the section name to go straight to the section. Click on the "+" to go to the relevant section of the detailed section list, where you can select individual subsections.

1. Getting Started

1.0 Overview

When using JMeter you will usually follow this process:

1.0.1 Test plan building

To do that, you will run JMeter in GUI Mode.
Then you can either choose to record the application from a browser, or native application. You can use for that the menu *File* → *Templates...* → *Recording*

Note you can also manually build your plan. Ensure you read this documentation to understand major concepts.

You will also debug it using one of these options:

- *Run* → *Start no pauses*
- *Run* → *Start*
- *Validate* on Thread Group

and View Results Tree renderers or Testers (CSS/JQUERY, JSON, Regexp, XPath).
Ensure you follow best-practices when building your Test Plan.

1.0.2 Load Test running

Once your Test Plan is ready, you can start your Load Test. The first step is to configure the injectors that will run JMeter, this as for any other Load Testing tool includes:

- Correct machine sizing in terms of CPU, memory and network
- OS Tuning
- Java setup: Ensure you install the latest version of Java supported by JMeter
- **Increase the Java Heap size**. By default JMeter runs with a heap of 1 GB, this might not be enough for your test and depends on your test plan and number of threads you want to run

Once everything is ready, you will use CLI mode (Command-line mode previously called Non-GUI mode) to run it for the Load Test.

Don't run load test using GUI mode !

Using CLI mode, you can generate a CSV (or XML) file containing results and have JMeter generate an HTML report at end of Load Test. JMeter will by default provide a summary of load test while it's running.
You can also have real-time results during your test using Backend Listener.

1.0.3 Load Test analysis

Once your Load Test is finished, you can use the HTML report to analyze your load test.

1.0.4 Let's start

The easiest way to begin using JMeter is to first download the latest production release and install it. The release contains all of the files you need to build and run most types of tests, e.g. Web (HTTP/HTTPS), FTP, JDBC, LDAP, Java, JUnit and more.

If you want to perform JDBC testing, then you will, of course, need the appropriate JDBC driver from your vendor. JMeter does not come with any JDBC drivers.

JMeter includes the JMS API jar, but does not include a JMS client implementation. If you want to run JMS tests, you will need to download the appropriate jars from the JMS provider.

See the JMeter Classpath section for details on installing additional jars.

Next, start JMeter and go through the Building a Test Plan section of the User Guide to familiarize yourself with JMeter basics (for example, adding and removing elements).

Finally, go through the appropriate section on how to build a specific type of Test Plan. For example, if you are interested in testing a Web application, then see the section Building a Web Test Plan. The other specific Test Plan sections are:

- Advanced Web Test Plan
- JDBC
- FTP
- JMS Point-to-Point
- JMS Topic
- LDAP
- LDAP Extended
- WebServices (SOAP)
- Building Test Plan programmatically

Once you are comfortable with building and running JMeter Test Plans, you can look into the various configuration elements (timers, listeners, assertions, and others) which give you more control over your Test Plans.

1.1 Requirements

JMeter requires that your computing environment meets some minimum requirements.

1.1.1 Java Version

JMeter is compatible with Java 8 or higher. We highly advise you to install latest minor version of your major version for security and performance reasons.

Because JMeter uses only standard Java APIs, please do not file bug reports if your JRE fails to run JMeter because of JRE implementation issues.

Although you can use a JRE, it is better to install a JDK as for recording of HTTPS, JMeter needs **keytool** utility from JDK.

1.1.2 Operating Systems

JMeter is a 100% Java application and should run correctly on any system that has a compliant Java implementation.

Operating systems tested with JMeter can be viewed on this page on JMeter wiki.

Even if your OS is not listed on the wiki page, JMeter should run on it provided that the JVM is compliant.

1.2 Optional

If you plan on doing JMeter development, then you will need one or more optional packages listed below.

1.2.1 Java Compiler

If you want to build the JMeter source or develop JMeter plugins, then you will need a fully compliant JDK 8 or higher.

1.2.2 SAX XML Parser

JMeter comes with Apache's Xerces XML parser. You have the option of telling JMeter to use a different XML parser. To do so, include the classes for the third-party parser in JMeter's classpath, and update the jmeter.properties file with the full classname of the parser implementation.

1.2.3 Email Support

JMeter has extensive Email capabilities. It can send email based on test results, and has a POP3(S)/IMAP(S) sampler. It also has an SMTP(S) sampler.

1.2.4 SSL Encryption

To test a web server using SSL encryption (HTTPS), JMeter requires that an implementation of SSL be provided, as is the case with Sun Java 1.4 and above. If your version of Java does not include SSL support, then it is possible to add an external implementation. Include the necessary encryption packages in JMeter's classpath. Also, update **system.properties** to register the SSL Provider.

JMeter HTTP defaults to protocol level TLS. This can be changed by editing the JMeter property **https.default.protocol** in **jmeter.properties** or **user.properties**.

The JMeter HTTP samplers are configured to accept all certificates, whether trusted or not, regardless of validity periods, etc. This is to allow the maximum flexibility in testing servers.

If the server requires a client certificate, this can be provided.

There is also the SSL Manager, for greater control of certificates.

The JMeter proxy server (see below) supports recording HTTPS (SSL)

The SMTP sampler can optionally use a local trust store or trust all certificates.

1.2.5 JDBC Driver

You will need to add your database vendor's JDBC driver to the classpath if you want to do JDBC testing. Make sure the file is a jar file, not a zip.

1.2.6 JMS client

JMeter now includes the JMS API from Apache Geronimo, so you just need to add the appropriate JMS Client implementation jar(s) from the JMS provider. Please refer to their documentation for details. There may also be some information on the JMeter Wiki.

1.2.7 Libraries for ActiveMQ JMS

You will need to add the jar **activemq-all-X.X.X.jar** to your classpath, e.g. by storing it in the **lib/** directory.

See ActiveMQ initial configuration page for details.

See the JMeter Classpath section for more details on installing additional jars.

1.3 Installation

We recommend that most users run the latest release.

To install a release build, simply unzip the zip/tar file into the directory where you want JMeter to be installed. Provided that you have a JRE/JDK correctly installed and the **JAVA_HOME** environment variable set, there is nothing more for you to do.

There can be problems (especially with client-server mode) if the directory path contains any spaces.

The installation directory structure should look something like this (where **X.Y** is version number):

apache-jmeter-X.Y

apache-jmeter-X.Y/bin

apache-jmeter-X.Y/docs

apache-jmeter-X.Y/extras

apache-jmeter-X.Y/lib/

apache-jmeter-X.Y/lib/ext

apache-jmeter-X.Y/lib/junit

apache-jmeter-X.Y/licenses

apache-jmeter-X.Y/printable_docs

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

1.4 Running JMeter

To run JMeter, run the **jmeter.bat** (for Windows) or **jmeter** (for Unix) file. These files are found in the **bin** directory. After a short time, the JMeter GUI should appear.

GUI mode should only be used for creating the test script, CLI mode (NON GUI) must be used for load testing

There are some additional scripts in the **bin** directory that you may find useful. Windows script files (the .CMD files require Win2K or later):

jmeter.bat

run JMeter (in GUI mode by default)

jmeterw.cmd

run JMeter without the windows shell console (in GUI mode by default)

jmeter-n.cmd

drop a JMX file on this to run a CLI mode test

jmeter-n-r.cmd

drop a JMX file on this to run a CLI mode test remotely

jmeter-t.cmd

drop a JMX file on this to load it in GUI mode

jmeter-server.bat

start JMeter in server mode

mirror-server.cmd

runs the JMeter Mirror Server in CLI mode

shutdown.cmd

Run the Shutdown client to stop a CLI mode instance gracefully

stoptest.cmd

Run the Shutdown client to stop a CLI mode instance abruptly

The special name **LAST** can be used with **jmeter-n.cmd**, **jmeter-t.cmd** and **jmeter-n-r.cmd** and means the last test plan that was run interactively.

There are a few environment variables, that can be used to customize the JVM settings for JMeter. An easy way to set those is by creating a file named **setenv.bat** in the **bin** directory. Such a file could look like:

rem This is the content of bin\setenv.bat,

rem it will be called by bin\jmeter.bat

set JVM_ARGS=-Xms1024m -Xmx1024m -Dpropname=value

The **JVM_ARGS** can be used to override JVM settings in the **jmeter.bat** script and will get set when starting JMeter, e.g.:

jmeter -t test.jmx ...

The following environment variables can be defined:

DDRAW

JVM options to influence usage of direct draw, e.g. **-Dsun.java2d.ddscale=true**. Default is empty.

GC_ALGO

JVM garbage collector options. Defaults to **-XX:+UseG1GC -XX:MaxGCPauseMillis=250 -XX:G1ReservePercent=20**

HEAP

JVM memory settings used when starting JMeter. Defaults to **-Xms1g -Xmx1g -XX:MaxMetaspaceSize=256m**

JMETER_BIN

JMeter bin directory (must end in ****). Value will have been guessed, when **setenv.bat** is called.

JMETER_COMPLETE_ARGS

If set indicates, that **JVM_ARGS** and **JMETER_OPTS** are to be used, only. All other options like **HEAP** and **GC_ALGO** will be ignored. Default is empty.

JMETER_HOME

installation directory. Will be guessed from location of **jmeter.bat**

JMETER_LANGUAGE

Java runtime options to specify used language. Defaults to: **-Duser.language="en" -Duser.region="EN"**

JM_LAUNCH

Name of the java executable, like **java.exe** (default) or **javaw.exe**

JVM_ARGS

Java options to be used when starting JMeter. These will be added last to the java command. Default is empty

Un*x script files; should work on most Linux/Unix systems:

jmeter

run JMeter (in GUI mode by default). Defines some JVM settings which may not work for all JVMs.

jmeter-server

start JMeter in server mode (calls jmeter script with appropriate parameters)

jmeter.sh

very basic JMeter script (You may need to adapt JVM options like memory settings).

mirror-server.sh

runs the JMeter Mirror Server in CLI mode

shutdown.sh

Run the Shutdown client to stop a CLI mode instance gracefully

stoptest.sh

Run the Shutdown client to stop a CLI mode instance abruptly

It may be necessary to set a few environment variables to configure the JVM used by JMeter. Those variables can be either set directly in the shell starting the **jmeter** script. For example setting the variable **JVM_ARGS** will override most pre-defined settings, for example

JVM_ARGS="-Xms1024m -Xmx1024m" jmeter -t test.jmx [etc.]

will override the HEAP settings in the script.

To set those variables permanently, you can place them in a file called **setenv.sh** in the **bin** directory. This file will be sourced when running JMeter by calling the **jmeter** script. An example for **bin/setenv.sh** could look like:

This is the file bin/setenv.sh,

it will be sourced in by bin/jmeter

Use a bigger heap, but a smaller metaspace, than the default

export HEAP="-Xms1G -Xmx1G -XX:MaxMetaspaceSize=192m"

Try to guess the locale from the OS. The space as value is on purpose!

export JMETER_LANGUAGE=" "

The following environment variables can be defined:

GC_ALGO

Java runtime options to specify JVM garbage collection algorithm. Defaults to **-XX:+UseG1GC -XX:MaxGCPauseMillis=250 -XX:G1ReservePercent=20**

HEAP

Java runtime options for memory management used when JMeter is started. Defaults to **-Xms1g -Xmx1g -X:MaxMetaspaceSize=256m**

JAVA_HOME

Must point at your Java Development Kit installation. Required to run the with the "**debug**" argument. On some OSes it JMeter will try its best to guess the location of the JVM.

JMETER_COMPLETE_ARGS

If set indicates, that **JVM_ARGS** and **JMETER_OPTS** are to be used, only. All other options like **HEAP** and **GC_ALGO** will be ignored. Default is empty.

JMETER_HOME

May point to your JMeter install dir. If empty it will be set relative to the **jmeter** script.

JMETER_LANGUAGE

Java runtime options to specify used language. Defaults to **-Duser.language=en -Duser.region=EN**

JMETER_OPTS

Java runtime options used when JMeter is started. Special options for operating systems might be added by JMeter.

JRE_HOME

Must point at your Java Runtime installation. Defaults to **JAVA_HOME** if empty.
If **JRE_HOME** and **JAVA_HOME** are both empty, JMeter will try to guess **JAVA_HOME**.
If **JRE_HOME** and **JAVA_HOME** are both set, **JAVA_HOME** is used.

JVM_ARGS

Java options to be used when starting JMeter. These will be added before **JMETER_OPTS** and after the other JVM options. Default is empty

1.4.1 JMeter's Classpath

JMeter automatically finds classes from jars in the following directories:

JMETER_HOME/lib

used for utility jars

JMETER_HOME/lib/ext

used for JMeter components and plugins

If you have developed new JMeter components, then you should jar them and copy the jar into JMeter's **lib/ext** directory. JMeter will automatically find JMeter components in any jars found here. Do not use **lib/ext** for utility jars or dependency jars used by the plugins; it is only intended for JMeter components and plugins.

If you don't want to put JMeter plugin jars in the **lib/ext** directory, then define the property **search_paths** in **jmeter.properties**.

Utility and dependency jars (libraries etc) can be placed in the **lib** directory.

If you don't want to put such jars in the **lib** directory, then define the property **user.classpath** or **plugin_dependency_paths** in **jmeter.properties**. See below for an explanation of the differences.

Other jars (such as JDBC, JMS implementations and any other support libraries needed by the JMeter code) should be placed in the **lib** directory - not the **lib/ext** directory, or added to **user.classpath**.

JMeter will only find **.jar** files, not **.zip**.

You can also install utility Jar files in **$JAVA_HOME/jre/lib/ext**, or you can set the property **user.classpath** in **jmeter.properties**

Note that setting the **CLASSPATH** environment variable will have no effect. This is because JMeter is started with "**java -jar**", and the java command silently ignores the **CLASSPATH** variable, and the **-classpath/-cp** options when **-jar** is used.

This occurs with all Java programs, not just JMeter.

Example:

jmeter -E https -H my.proxy.server -P 8000 -u username -a password -N localhost

You can also use **--proxyScheme**, **--proxyHost**, **--proxyPort**, **--username**, and **--password** as parameter names

Parameters provided on a command-line may be visible to other users on the system.

If the proxy scheme is provided, then JMeter sets the following System properties:

- **http.proxyScheme**

If the proxy host and port are provided, then JMeter sets the following System properties:

- **http.proxyHost**
- **http.proxyPort**
- **https.proxyHost**
- **https.proxyPort**

The user and password used for a proxy can be given by the System properties **http.proxyUser** and **http.proxyUser**. They will get overridden by the above arguments or values set in the HTTP Samplers.

If a nonproxy host list is provided, then JMeter sets the following System properties:

- **http.nonProxyHosts**
- **https.nonProxyHosts**

So if you don't wish to set both http and https proxies, you can define the relevant properties in **system.properties** instead of using the command-line parameters.

Proxy Settings can also be defined in a Test Plan, using either the HTTP Request Defaults configuration or the HTTP Request sampler elements.

JMeter also has its own in-built Proxy Server, the HTTP(S) Test Script Recorder. This is only used for recording HTTP or HTTPS browser sessions. This is not to be confused with the proxy settings described above, which are used when JMeter makes HTTP or HTTPS requests itself.

1.4.4 CLI Mode (Command Line mode was called NON GUI mode)

For load testing, you must run JMeter in this mode (Without the GUI) to get the optimal results from it. To do so, use the following command options:

-n

This specifies JMeter is to run in cli mode

-t

[name of JMX file that contains the Test Plan].

-l

[name of JTL file to log sample results to].

-j

[name of JMeter run log file].

-r

Run the test in the servers specified by the JMeter property "**remote_hosts**"

-R

[list of remote servers] Run the test in the specified remote servers

-g

[path to CSV file] generate report dashboard only

-e

generate report dashboard after load test

-o

output folder where to generate the report dashboard after load test. Folder must not exist or be empty

The script also lets you specify the optional firewall/proxy server information:

-H

[proxy server hostname or ip address]

-P

[proxy server port]

Example

jmeter -n -t my_test.jmx -l log.jtl -H my.proxy.server -P 8000

If the property **jmeterengine.stopfail.system.exit** is set to **true** (default is **false**), then JMeter will invoke **System.exit(1)** if it cannot stop all threads. Normally this is not necessary.

1.4.5 Server Mode

For distributed testing, run JMeter in server mode on the remote node(s), and then control the server(s) from the GUI. You can also use CLI mode to run remote tests. To start the server(s), run **jmeter-server[.bat]** on each server host.

The script also lets you specify the optional firewall/proxy server information:

-H

[proxy server hostname or ip address]

-P

[proxy server port]

Example:

jmeter-server -H my.proxy.server -P 8000

If you want the server to exit after a single test has been run, then define the JMeter property **server.exitaftertest=true**.

To run the test from the client in CLI mode, use the following command:

jmeter -n -t testplan.jmx -r [-Gprop=val] [-Gglobal.properties] [-X]

where:

-G

is used to define JMeter properties to be set in the servers

-X

means exit the servers at the end of the test

-Rserver1,server2

can be used instead of **-r** to provide a list of servers to start. Overrides **remote_hosts**, but does not define the property.

If the property **jmeterengine.remote.system.exit** is set to **true** (default is **false**), then JMeter will invoke **System.exit(0)** after stopping RMI at the end of a test. Normally this is not necessary.

1.4.6 Overriding Properties Via The Command Line

Java system properties and JMeter properties can be overridden directly on the command lin (instead of modifying **jmeter.properties**). To do so, use the following options:

-D[prop_name]=[value]

defines a java system property value.

-J[prop_name]=[value]

defines a local JMeter property.

-G[prop_name]=[value]

defines a JMeter property to be sent to all remote servers.

-G[propertyfile]

defines a file containing JMeter properties to be sent to all remote servers.

-L[category]=[priority]

overrides a logging setting, setting a particular category to the given priority level.

The **-L** flag can also be used without the category name to set the root logging level.

Examples:

```
jmeter -Duser.dir=/home/mstover/jmeter_stuff \
  -Jremote_hosts=127.0.0.1 -Ljmeter.engine=DEBUG
jmeter -LDEBUG
```

The command line properties are processed early in startup, but after the logging system has been set up.

1.4.7 Logging and error messages

Since 3.2, JMeter logging is not configured through properties file(s) such as **jmeter.properties** any more, but it is configured through a Apache Log4j 2 configuration file (**log4j2.xml** in the directory from which JMeter was launched, by default) instead. Also, every code including JMeter and plugins MUST use SLF4J library to leave logs since 3.2.

Here is an example **log4j2.xml** file which defines two log appenders and loggers for each category.

```
<Configuration status="WARN" packages="org.apache.jmeter.gui.logging">

 <Appenders>

  <!-- The main log file appender to jmeter.log in the directory from which JMeter was launched, by default. -->
  <File name="jmeter-log" fileName="${sys:jmeter.logfile:-jmeter.log}" append="false">
   <PatternLayout>
    <pattern>%d %p %c{1.}: %m%n</pattern>
   </PatternLayout>
  </File>

  <!-- Log appender for GUI Log Viewer. See below. -->
  <GuiLogEvent name="gui-log-event">
   <PatternLayout>
```

```
      <pattern>%d %p %c{1.}: %m%n</pattern>
    </PatternLayout>
  </GuiLogEvent>

 </Appenders>

 <Loggers>

  <!-- Root logger -->
  <Root level="info">
    <AppenderRef ref="jmeter-log" />
    <AppenderRef ref="gui-log-event" />
  </Root>

  <!-- SNIP -->

  <!--
    # Apache HttpClient logging examples
  -->
  <!-- # Enable header wire + context logging - Best for Debugging -->
  <!--
  <Logger name="org.apache.http" level="debug" />
  <Logger name="org.apache.http.wire" level="error" />
  -->

  <!-- SNIP -->

 </Loggers>

</Configuration>
```

So, if you want to change the log level for **org.apache.http** category to debug level for instance, you can simply add (or uncomment) the following logger element in **log4j2.xml** file before launching JMeter.

```
<Loggers>
  <!-- SNIP -->
  <Logger name="org.apache.http" level="debug" />
  <!-- SNIP -->
</Loggers>
```

For more detail on how to configure **log4j2.xml** file, please see Apache Log4j 2 Configuration page.

Log level for specific categories or root logger can be overridden directly on the command line (instead of modifying **log4j2.xml**) as well. To do so, use the following options:

-L[category]=[priority]

Overrides a logging setting, setting a particular category to the given priority level. Since 3.2, it is recommended to use a full category name (e.g, **org.apache.jmeter** or **com.example.foo**), but if the category name starts with either **jmeter** or **jorphan**, **org.apache.** will be prepended internally to the category name input to construct a full category name (i.e, **org.apache.jmeter** or **org.apache.jorphan**) for backward compatibility.

Examples:

jmeter -Ljmeter.engine=DEBUG

jmeter -Lorg.apache.jmeter.engine=DEBUG

jmeter -Lcom.example.foo=DEBUG

jmeter -LDEBUG

1.4.8 Full list of command-line options

Invoking JMeter as "**jmeter -?**" will print a list of all the command-line options. These are shown below.

```
  --?
     print command line options and exit
  -h, --help
     print usage information and exit
  -v, --version
     print the version information and exit
  -p, --propfile <argument>
     the jmeter property file to use
```

-q, --addprop <argument>

additional JMeter property file(s)

-t, --testfile <argument>

the jmeter test(.jmx) file to run

-l, --logfile <argument>

the file to log samples to

-i, --jmeterlogconf <argument>

jmeter logging configuration file (log4j2.xml)

-j, --jmeterlogfile <argument>

jmeter run log file (jmeter.log)

-n, --nongui

run JMeter in nongui mode

-s, --server

run the JMeter server

-H, --proxyHost <argument>

Set a proxy server for JMeter to use

-P, --proxyPort <argument>

Set proxy server port for JMeter to use

-N, --nonProxyHosts <argument>

Set nonproxy host list (e.g. *.apache.org|localhost)

-u, --username <argument>

Set username for proxy server that JMeter is to use

-a, --password <argument>

Set password for proxy server that JMeter is to use

-J, --jmeterproperty <argument>=<value>

Define additional JMeter properties

-G, --globalproperty <argument>=<value>

Define Global properties (sent to servers)

e.g. -Gport=123

or -Gglobal.properties

-D, --systemproperty <argument>=<value>

Define additional system properties

-S, --systemPropertyFile <argument>

additional system property file(s)

-f, --forceDeleteResultFile

force delete existing results files and web report folder if present before starting the test

-L, --loglevel <argument>=<value>

[category=]level e.g. jorphan=INFO, jmeter.util=DEBUG or com.example.foo=WARN

-r, --runremote

Start remote servers (as defined in remote_hosts)

-R, --remotestart <argument>

Start these remote servers (overrides remote_hosts)

-d, --homedir <argument>

the jmeter home directory to use

-X, --remoteexit

Exit the remote servers at end of test (CLI mode)

-g, --reportonly <argument>

generate report dashboard only, from a test results file

-e, --reportatendofloadtests

generate report dashboard after load test

-o, --reportoutputfolder <argument>

output folder for report dashboard

Note: the JMeter log file name is formatted as a SimpleDateFormat (applied to the current date) if it contains paired single-quotes, .e.g. '**jmeter_'yyyyMMddHHmmss'.log**'

If the special name **LAST** is used for the **-t**, **-j** or **-l** flags, then JMeter takes that to mean the last test plan that was run in interactive mode.

1.4.9 CLI mode shutdown

Prior to version 2.5.1, JMeter invoked **System.exit()** when a CLI mode test completed. This caused problems for applications that invoke JMeter directly, so JMeter no longer invokes **System.exit()** for a normal test completion. [Some fatal errors may still invoke **System.exit()**] JMeter will exit all the non-daemon threads it starts, but it is possible that some non-daemon threads may still remain; these will prevent the JVM from exiting. To detect this situation, JMeter starts a new daemon thread just before it exits. This daemon thread waits a short while; if it returns from the wait, then clearly the JVM has not been able to exit, and the thread prints a message to say why.

The property **jmeter.exit.check.pause** can be used to configure the delay before printing non-daemon threads. If set to **0** (default value), then JMeter does not print non-terminated threads at the end of the test.

1.5 Configuring JMeter

If you wish to modify the properties with which JMeter runs you need to either modify the **user.properties** in the **/bin** directory or create your own copy of the **jmeter.properties** and specify it in the command line.

Note: You can define additional JMeter properties in the file defined by the JMeter property **user.properties** which has the default value **user.properties**. The file will be automatically loaded if it is found in the current directory or if it is found in the JMeter bin directory. Similarly, **system.properties** is used to update system properties.

Parameters

Attribute

Description

Required

ssl.provider

You can specify the class for your SSL implementation if you don't want to use the built-in Java implementation.

No

xml.parser

You can specify an implementation as your XML parser. The default value is: **org.apache.xerces.parsers.SAXParser**

No

remote_hosts

Comma-delimited list of remote JMeter hosts (or **host:port** if required). If you are running JMeter in a distributed environment, list the machines where you have JMeter remote servers running. This will allow you to control those servers from this machine's GUI

No

not_in_menu

A list of components you do not want to see in JMeter's menus. As JMeter has more and more components added, you may wish to customize your JMeter to show only those components you are interested in. You may list their classname or their class label (the string that appears in JMeter's UI) here, and they will no longer appear in the menus.

No

search_paths

List of paths (separated by **;**) that JMeter will search for JMeter plugin classes, for example additional samplers. A path item can either be a jar file or a directory. Any jar file in such a directory will be automatically included in **search_paths**, jar files in sub directories are ignored. The given value is in addition to any jars found in the **lib/ext** directory.

No

user.classpath

List of paths that JMeter will search for utility and plugin dependency classes. Use your platform path separator to separate multiple paths. A path item can either be a jar file or a directory. Any jar file in such a directory will be automatically included in **user.classpath**, jar files in sub directories are ignored. The given value is in addition to any jars found in the lib directory. All entries will be added to the class path of the system class loader and also to the path of the JMeter internal loader.

No

plugin_dependency_paths

List of paths (separated by **;**) that JMeter will search for utility and plugin dependency classes. A path item can either be a jar file or a directory. Any jar file in such a directory will be automatically included in **plugin_dependency_paths**, jar files in sub directories are ignored. The given value is in addition to any jars found in the **lib** directory or given by the **user.classpath** property. All entries will be added to the path of the JMeter internal loader only. For plugin dependencies using **plugin_dependency_paths** should be preferred over **user.classpath**.

No

user.properties

Name of file containing additional JMeter properties. These are added after the initial property file, but before the **-q** and **-J** options are processed.

No

system.properties

Name of file containing additional system properties. These are added before the **-S** and **-D** options are processed.

No

The command line options and properties files are processed in the following order:

1. **-p propfile**
2. **jmeter.properties** (or the file from the **-p** option) is then loaded
3. **-j logfile**
4. Logging is initialised
5. **user.properties** is loaded
6. **system.properties** is loaded
7. all other command-line options are processed

See also the comments in the jmeter.properties, user.properties and system.properties files for further information on other settings you can change.

2. Building a Test Plan

A test plan describes a series of steps JMeter will execute when run. A complete test plan will consist of one or more Thread Groups, logic controllers, sample generating controllers, listeners, timers, assertions, and configuration elements.

2.1 Adding and Removing Elements

Adding elements to a test plan can be done by right-clicking on an element in the tree, and choosing a new element from the "**add**" list. Alternatively, elements can be loaded from file and added by choosing the "**merge**" or "**open**" option.

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

2.2 Loading and Saving Elements

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

To save tree elements, right click on an element and choose the "**Save Selection As ...**" option. JMeter will save the element selected, plus all child elements beneath it. In this way, you can save test tree fragments and individual elements for later use.

2.3 Configuring Tree Elements

Any element in the test tree will present controls in JMeter's right-hand frame. These controls allow you to configure the behavior of that particular test element. What can be configured for an element depends on what type of element it is.

The Test Tree itself can be manipulated by dragging and dropping components around the test tree.

2.4 Saving the Test Plan

Although it is not required, we recommend that you save the Test Plan to a file before running it. To save the Test Plan, select "**Save**" or "**Save Test Plan As ...**" from the File menu (with the latest release, it is no longer necessary to select the Test Plan element first).

JMeter allows you to save the entire Test Plan tree or only a portion of it. To save only the elements located in a particular "branch" of the Test Plan tree, select the Test Plan element in the tree from which to start the "branch", and then click your right mouse button to access the "**Save Selection As ...**" menu item. Alternatively, select the appropriate Test Plan element and then select "**Save Selection As ...**" from the Edit menu.

2.5 Running a Test Plan

To run your test plan, choose "**Start**" (**Control** + **r**) from the "**Run**" menu item. When JMeter is running, it shows a small green box at the right hand end of the section just under the menu bar. You can also check the "**Run**" menu. If "**Start**" is disabled, and "**Stop**" is enabled, then JMeter is running your test plan (or, at least, it thinks it is).

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

Using GUI mode as described here should only be used when debugging your Test Plan. To run the real load test, use CLI mode.

2.6 Stopping a Test

There are two types of stop command available from the menu:

- **Stop** (**Control** + **.**) - stops the threads immediately if possible. Many samplers are Interruptible which means that active samples can be terminated early. The stop command will check that all threads have stopped within the default timeout, which is

5000 ms = 5 seconds. [This can be changed using the JMeter property **jmeterengine.threadstop.wait**] If the threads have not stopped, then a message is displayed. The Stop command can be retried, but if it fails, then it is necessary to exit JMeter to clean up.

- **Shutdown** (**Control** + **,**) - requests the threads to stop at the end of any current work. Will not interrupt any active samples. The modal shutdown dialog box will remain active until all threads have stopped.

If Shutdown is taking too long. Close the Shutdown dialog box and select **Run**/**Stop**, or just press **Control** + **.**.

When running JMeter in CLI mode, there is no Menu, and JMeter does not react to keystrokes such as **Control** + **.**. So JMeter CLI mode will listen for commands on a specific port (default **4445**, see the JMeter property **jmeterengine.nongui.port**). JMeter supports automatic choice of an alternate port if the default port is being used (for example by another JMeter instance). In this case, JMeter will try the next higher port, continuing until it reaches the JMeter property **jmeterengine.nongui.maxport**) which defaults to **4455**. If **maxport** is less than or equal to **port**, port scanning will not take place.
The chosen port is displayed in the console window.
The commands currently supported are:

- **Shutdown** - graceful shutdown
- **StopTestNow** - immediate shutdown

These commands can be sent by using the **shutdown[.cmd|.sh]** or **stoptest[.cmd|.sh]** script respectively. The scripts are to be found in the JMeter **bin** directory. The commands will only be accepted if the script is run from the same host.

2.7 Error reporting

JMeter reports warnings and errors to the **jmeter.log** file, as well as some information on the test run itself. JMeter shows the number of warnings/errors found in **jmeter.log** file next to the warning icon (triangle) at the right hand end of its window. Click on the warning icon to show the **jmeter.log** file at the bottom of JMeter's window. Just occasionally there may be some errors that JMeter is unable to trap and log; these will appear on the command console. If a test is not behaving as you expect, please check the log file in case any errors have been reported (e.g. perhaps a syntax error in a function call).

Sampling errors (e.g. HTTP 404 - file not found) are not normally reported in the log file. Instead these are stored as attributes of the sample result. The status of a sample result can be seen in the various different Listeners.

3. Elements of a Test Plan

This section describes the different parts of a test plan.

A minimal test will consist of the Test Plan, a Thread Group and one or more Samplers.

3.0 Test Plan

The Test Plan object has a checkbox called "**Functional Testing**". If selected, it will cause JMeter to record the data returned from the server for each sample. If you have selected a file in your test listeners, this data will be written to file. This can be useful if you are doing a small run to ensure that JMeter is configured correctly, and that your server is returning the expected results. The consequence is that the file will grow huge quickly, and JMeter's performance will suffer. This option should be off if you are doing stress-testing (it is off by default).

If you are not recording the data to file, this option makes no difference.

You can also use the **Configuration** button on a listener to decide what fields to save.

3.1 Thread Group

Thread group elements are the beginning points of any test plan. All controllers and samplers must be under a thread group. Other elements, e.g. Listeners, may be placed directly under the test plan, in which case they will apply to all the thread groups. As the name implies, the thread group element controls the number of threads JMeter will use to execute your test. The controls for a thread group allow you to:

- Set the number of threads
- Set the ramp-up period
- Set the number of times to execute the test

Each thread will execute the test plan in its entirety and completely independently of other test threads. Multiple threads are used to simulate concurrent connections to your server application.

The ramp-up period tells JMeter how long to take to "ramp-up" to the full number of threads chosen. If 10 threads are used, and the ramp-up period is 100 seconds, then JMeter will take 100 seconds to get all 10 threads up and running. Each thread will start 10 (100/10) seconds after the previous thread was begun. If there are 30 threads and a ramp-up period of 120 seconds, then each successive thread will be delayed by 4 seconds.

Ramp-up needs to be long enough to avoid too large a work-load at the start of a test, and short enough that the last threads start running before the first ones finish (unless one wants that to happen).

Start with Ramp-up = number of threads and adjust up or down as needed.

By default, the thread group is configured to loop once through its elements.

Thread Group also allows to specify **Thread lifetime**. Click the checkbox at the bottom of the Thread Group panel to enable/disable extra fields in which you can enter the duration of test and the startup delay You can configure **Duration (seconds)** and **Startup Delay (seconds)** to control the duration of each thread group and the after how much seconds it starts. When the test is started, JMeter will wait **Startup Delay (seconds)** before starting the Threads of the Thread Group and run for the configured **Duration (seconds)** time.

3.2 Controllers

JMeter has two types of Controllers: Samplers and Logical Controllers. These drive the processing of a test.

Samplers tell JMeter to send requests to a server. For example, add an HTTP Request Sampler if you want JMeter to send an HTTP request. You can also customize a request by adding one or more Configuration Elements to a Sampler. For more information, see Samplers.

Logical Controllers let you customize the logic that JMeter uses to decide when to send requests. For example, you can add an Interleave Logic Controller to alternate between two HTTP Request Samplers. For more information, see Logical Controllers.

3.2.1 Samplers

Samplers tell JMeter to send requests to a server and wait for a response. They are processed in the order they appear in the tree. Controllers can be used to modify the number of repetitions of a sampler.

JMeter samplers include:

- FTP Request
- HTTP Request (can be used for SOAP or REST Webservice also)
- JDBC Request
- Java object request
- JMS request
- JUnit Test request
- LDAP Request
- Mail request
- OS Process request
- TCP request

Each sampler has several properties you can set. You can further customize a sampler by adding one or more Configuration Elements to the Test Plan.

If you are going to send multiple requests of the same type (for example, HTTP Request) to the same server, consider using a Defaults Configuration Element. Each controller has one or more Defaults elements (see below).

Remember to add a Listener to your test plan to view and/or store the results of your requests to disk.

If you are interested in having JMeter perform basic validation on the response of your request, add an Assertion to the sampler. For example, in stress testing a web application, the server may return a successful "HTTP Response" code, but the page may have errors on it or may be missing sections. You could add assertions to check for certain HTML tags, common error strings, and so on. JMeter lets you create these assertions using regular expressions.

JMeter's built-in samplers

3.2.2 Logic Controllers

Logic Controllers let you customize the logic that JMeter uses to decide when to send requests. Logic Controllers can change the order of requests coming from their child elements. They can modify the requests themselves, cause JMeter to repeat requests, etc.

To understand the effect of Logic Controllers on a test plan, consider the following test tree:

- Test Plan
    - Thread Group
        - Once Only Controller
            - Login Request (an HTTP Request)
        - Load Search Page (HTTP Sampler)
        - Interleave Controller
            - Search "A" (HTTP Sampler)
            - Search "B" (HTTP Sampler)
            - HTTP default request (Configuration Element)
        - HTTP default request (Configuration Element)
        - Cookie Manager (Configuration Element)

The first thing about this test is that the login request will be executed only the first time through. Subsequent iterations will skip it. This is due to the effects of the Once Only Controller.

After the login, the next Sampler loads the search page (imagine a web application where the user logs in, and then goes to a search page to do a search). This is just a simple request, not filtered through any Logic Controller.

After loading the search page, we want to do a search. Actually, we want to do two different searches. However, we want to re-load the search page itself between each search. We could do this by having 4 simple HTTP request elements (load search, search "A", load search, search "B"). Instead, we use the Interleave Controller which passes on one child request each time through the test. It keeps the ordering (i.e. it doesn't pass one on at random, but "remembers" its place) of its child elements. Interleaving 2 child requests may be overkill, but there could easily have been 8, or 20 child requests.

Note the HTTP Request Defaults that belongs to the Interleave Controller. Imagine that "Search A" and "Search B" share the same PATH info (an HTTP request specification includes domain, port, method, protocol, path, and arguments, plus other optional items). This makes sense - both are search requests, hitting the same back-end search engine (a servlet or cgi-script, let's say). Rather than configure both HTTP Samplers with the same information in their PATH field, we can abstract that information out to a single Configuration Element. When the Interleave Controller "passes on" requests from "Search A" or "Search B", it will fill in the blanks with values from the HTTP default request Configuration Element. So, we leave the PATH field blank for those requests, and put that information into the Configuration Element. In this case, this is a minor benefit at best, but it demonstrates the feature.

The next element in the tree is another HTTP default request, this time added to the Thread Group itself. The Thread Group has a built-in Logic Controller, and thus, it uses this Configuration Element exactly as described above. It fills in the blanks of any Request that passes through. It is extremely useful in web testing to leave the DOMAIN field blank in all your HTTP Sampler elements, and instead, put that information into an HTTP default request element, added to the Thread Group. By doing so, you can test your application on a different server simply by changing one field in your Test Plan. Otherwise, you'd have to edit each and every Sampler.

The last element is a HTTP Cookie Manager. A Cookie Manager should be added to all web tests - otherwise JMeter will ignore cookies. By adding it at the Thread Group level, we ensure that all HTTP requests will share the same cookies.

Logic Controllers can be combined to achieve various results. See the list of built-in Logic Controllers.

3.2.3 Test Fragments

The Test Fragment element is a special type of controller that exists on the Test Plan tree at the same level as the Thread Group element. It is distinguished from a Thread Group in that it is not executed unless it is referenced by either a Module Controller or an Include_Controller.

This element is purely for code re-use within Test Plans

3.3 Listeners

Listeners provide access to the information JMeter gathers about the test cases while JMeter runs. The Graph Results listener plots the response times on a graph. The "View Results Tree" Listener shows details of sampler requests and responses, and can display basic HTML and XML representations of the response. Other listeners provide summary or aggregation information.

Additionally, listeners can direct the data to a file for later use. Every listener in JMeter provides a field to indicate the file to store data to. There is also a Configuration button which can be used to choose which fields to save, and whether to use CSV or XML format.

Note that all Listeners save the same data; the only difference is in the way the data is presented on the screen.

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.

There are several listeners that come with JMeter.

3.4 Timers

By default, a JMeter thread executes samplers in sequence without pausing. We recommend that you specify a delay by adding one of the available timers to your Thread Group. If you do not add a delay, JMeter could overwhelm your server by making too many requests in a very short amount of time.

A timer will cause JMeter to delay a certain amount of time **before** each sampler which is in its scope.

If you choose to add more than one timer to a Thread Group, JMeter takes the sum of the timers and pauses for that amount of time before executing the samplers to which the timers apply. Timers can be added as children of samplers or controllers in order to restrict the samplers to which they are applied.

To provide a pause at a single place in a test plan, one can use the Flow Control Action Sampler.

3.5 Assertions

Assertions allow you to assert facts about responses received from the server being tested. Using an assertion, you can essentially "test" that your application is returning the results you expect it to.

For instance, you can assert that the response to a query will contain some particular text. The text you specify can be a Perl-style regular expression, and you can indicate that the response is to contain the text, or that it should match the whole response.

You can add an assertion to any Sampler. For example, you can add an assertion to a HTTP Request that checks for the text, "**</HTML>**". JMeter will then check that the text is present in the HTTP response. If JMeter cannot find the text, then it will mark this as a failed request.

Note that assertions apply to all samplers which are in their scope. To restrict an assertion to a single sampler, add the assertion as a child of the sampler.

To view assertion results, add an Assertion Listener to the Thread Group. Failed Assertions will also show up in the Tree View and Table Listeners, and will count towards the error %age for example in the Aggregate and Summary reports.

3.6 Configuration Elements

A configuration element works closely with a Sampler. Although it does not send requests (except for HTTP(S) Test Script Recorder), it can add to or modify requests.

A configuration element is accessible from only inside the tree branch where you place the element. For example, if you place an HTTP Cookie Manager inside a Simple Logic Controller, the Cookie Manager will only be accessible to HTTP Request Controllers you place inside the Simple Logic Controller (see figure 1). The Cookie Manager is accessible to the HTTP requests "Web Page 1" and "Web Page 2", but not "Web Page 3".

Also, a configuration element inside a tree branch has higher precedence than the same element in a "parent" branch. For example, we defined two HTTP Request Defaults elements, "Web Defaults 1" and "Web Defaults 2". Since we placed "Web Defaults 1" inside a Loop Controller, only "Web Page 2" can access it. The other HTTP requests will use "Web Defaults 2", since we placed it in the Thread Group (the "parent" of all other branches).

Figure 1 - Test Plan Showing Accessibility of Configuration Elements

The User Defined Variables Configuration element is different. It is processed at the start of a test, no matter where it is placed. For simplicity, it is suggested that the element is placed only at the start of a Thread Group.

3.7 Pre-Processor Elements

A Pre-Processor executes some action prior to a Sampler Request being made. If a Pre-Processor is attached to a Sampler element, then it will execute just prior to that sampler element running. A Pre-Processor is most often used to modify the settings of a Sample Request just before it runs, or to update variables that aren't extracted from response text. See the **scoping rules** for more details on when Pre-Processors are executed.

3.8 Post-Processor Elements

A Post-Processor executes some action after a Sampler Request has been made. If a Post-Processor is attached to a Sampler element, then it will execute just after that sampler element runs. A Post-Processor is most often used to process the response data, often to extract values from it. See the **scoping rules** for more details on when Post-Processors are executed.

3.9 Execution order

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

Please note that Timers, Assertions, Pre- and Post-Processors are only processed if there is a sampler to which they apply. Logic Controllers and Samplers are processed in the order in which they appear in the tree. Other test elements are processed according to the scope in which they are found, and the type of test element. [Within a type, elements are processed in the order in which they appear in the tree].

For example, in the following test plan:

- Controller
    - Post-Processor 1
    - Sampler 1
    - Sampler 2
    - Timer 1
    - Assertion 1
    - Pre-Processor 1
    - Timer 2
    - Post-Processor 2

The order of execution would be:

Pre-Processor 1

Timer 1

Timer 2

Sampler 1

Post-Processor 1

Post-Processor 2

Assertion 1

Pre-Processor 1

Timer 1

Timer 2

Sampler 2

Post-Processor 1

Post-Processor 2

Assertion 1

3.10 Scoping Rules

The JMeter test tree contains elements that are both hierarchical and ordered. Some elements in the test trees are strictly hierarchical (Listeners, Config Elements, Post-Processors, Pre-Processors, Assertions, Timers), and some are primarily ordered (controllers, samplers). When you create your test plan, you will create an ordered list of sample request (via Samplers) that represent a set of steps to be executed. These requests are often organized within controllers that are also ordered. Given the following test tree:

Example test tree

The order of requests will be, One, Two, Three, Four.

Some controllers affect the order of their subelements, and you can read about these specific controllers in the component reference.

Other elements are hierarchical. An Assertion, for instance, is hierarchical in the test tree. If its parent is a request, then it is applied to that request. If its parent is a Controller, then it affects all requests that are descendants of that Controller. In the following test tree:

Hierarchy example

Assertion #1 is applied only to Request One, while Assertion #2 is applied to Requests Two and Three.

Another example, this time using Timers:

complex example

In this example, the requests are named to reflect the order in which they will be executed. Timer #1 will apply to Requests Two, Three, and Four (notice how order is irrelevant for hierarchical elements). Assertion #1 will apply only to Request Three. Timer #2 will affect all the requests.

Hopefully these examples make it clear how configuration (hierarchical) elements are applied. If you imagine each Request being passed up the tree branches, to its parent, then to its parent's parent, etc., and each time collecting all the configuration elements of that parent, then you will see how it works.

The Configuration elements Header Manager, Cookie Manager and Authorization manager are treated differently from the Configuration Default elements. The settings from the Configuration Default elements are merged into a set of values that the Sampler has access to. However, the settings from the Managers are not merged. If more than one Manager is in the scope of a Sampler, only one Manager is used, but there is currently no way to specify *which* is used.

3.11 Properties and Variables

JMeter *properties* are defined in **jmeter.properties** (see Getting Started - Configuring JMeter for more details).
Properties are global to jmeter, and are mostly used to define some of the defaults JMeter uses. For example the property **remote_hosts** defines the servers that JMeter will try to run remotely. Properties can be referenced in test plans - see Functions - read a property - but cannot be used for thread-specific values.

JMeter *variables* are local to each thread. The values may be the same for each thread, or they may be different.
If a variable is updated by a thread, only the thread copy of the variable is changed. For example the Regular Expression Extractor Post-Processor will set its variables according to the sample that its thread has read, and these can be used later by the same thread. For details of how to reference variables and functions, see Functions and Variables

Note that the values defined by the Test Plan and the User Defined Variables configuration element are made available to the whole test plan at startup. If the same variable is defined by multiple UDV elements, then the last one takes effect. Once a thread has started, the initial set of variables is copied to each thread. Other elements such as the User Parameters Pre-Processor or Regular Expression Extractor Post-Processor may be used to redefine the same variables (or create new ones). These redefinitions only apply to the current thread.

The setProperty function can be used to define a JMeter property. These are global to the test plan, so can be used to pass information between threads - should that be needed.

Both variables and properties are case-sensitive.

3.12 Using Variables to parameterise tests

Variables don't have to vary - they can be defined once, and if left alone, will not change value. So you can use them as short-hand for expressions that appear frequently in a test plan. Or for items which are constant during a run, but which may vary between runs. For example, the name of a host, or the number of threads in a thread group.

When deciding how to structure a Test Plan, make a note of which items are constant for the run, but which may change between runs. Decide on some variable names for these - perhaps

use a naming convention such as prefixing them with **C_** or **K_** or using uppercase only to distinguish them from variables that need to change during the test. Also consider which items need to be local to a thread - for example counters or values extracted with the Regular Expression Post-Processor. You may wish to use a different naming convention for these.

For example, you might define the following on the Test Plan:

HOST www.example.com

THREADS 10

LOOPS 20

You can refer to these in the test plan as **${HOST} ${THREADS}** etc. If you later want to change the host, just change the value of the **HOST** variable. This works fine for small numbers of tests, but becomes tedious when testing lots of different combinations. One solution is to use a property to define the value of the variables, for example:

HOST ${__P(host,www.example.com)}

THREADS ${__P(threads,10)}

LOOPS ${__P(loops,20)}

You can then change some or all of the values on the command-line as follows:

jmeter ... -Jhost=www3.example.org -Jloops=13

4. Building a Web Test Plan

In this section, you will learn how to create a basic Test Plan to test a Web site. You will create five users that send requests to two pages on the JMeter Web site. Also, you will tell the users to run their tests twice. So, the total number of requests is (5 users) x (2 requests) x (repeat 2 times) = 20 HTTP requests. To construct the Test Plan, you will use the following elements: Thread Group, HTTP Request, HTTP Request Defaults, and Graph Results.

For a more advanced Test Plan, see Building an Advanced Web Test Plan.

4.1 Adding Users

The first step you want to do with every JMeter Test Plan is to add a Thread Group element. The Thread Group tells JMeter the number of users you want to simulate, how often the users should send requests, and how many requests they should send.

Go ahead and add the ThreadGroup element by first selecting the Test Plan, clicking your right mouse button to get the Add menu, and then select Add → ThreadGroup.

You should now see the Thread Group element under Test Plan. If you do not see the element, then "expand" the Test Plan tree by clicking on the Test Plan element.

Next, you need to modify the default properties. Select the Thread Group element in the tree, if you have not already selected it. You should now see the Thread Group Control Panel in the right section of the JMeter window (see Figure 4.1 below)

Start by providing a more descriptive name for our Thread Group. In the name field, enter JMeter Users.

Next, increase the number of users (called threads) to 5.

In the next field, the Ramp-Up Period, leave the default value of 1 seconds. This property tells JMeter how long to delay between starting each user. For example, if you enter a Ramp-Up Period of 5 seconds, JMeter will finish starting all of your users by the end of the 5 seconds. So, if we have 5 users and a 5 second Ramp-Up Period, then the delay between starting users would be 1 second (5 users / 5 seconds = 1 user per second). If you set the value to 0, then JMeter will immediately start all of your users.

Finally enter a value of 2 in the Loop Count field. This property tells JMeter how many times to repeat your test. If you enter a loop count value of 1, then JMeter will run your test only once. To have JMeter repeatedly run your Test Plan, select the Forever checkbox.

In most applications, you have to manually accept changes you make in a Control Panel. However, in JMeter, the Control Panel automatically accepts your changes as you make them. If you change the name of an element, the tree will be updated with the new text after you leave the Control Panel (for example, when selecting another tree element).

Like most JMeter elements, the HTTP Request Defaults Control Panel has a name field that you can modify. In this example, leave this field with the default value.

Skip to the next field, which is the Web Server's Server Name/IP. For the Test Plan that you are building, all HTTP requests will be sent to the same Web server, jmeter.apache.org. Enter this domain name into the field. This is the only field that we will specify a default, so leave the remaining fields with their default values.

The HTTP Request Defaults element does not tell JMeter to send an HTTP request. It simply defines the default values that the HTTP Request elements use.

4.3 Adding Cookie Support

Nearly all web testing should use cookie support, unless your application specifically doesn't use cookies. To add cookie support, simply add an HTTP Cookie Manager to each Thread Group in your test plan. This will ensure that each thread gets its own cookies, but shared across all HTTP Request objects.

To add the HTTP Cookie Manager, simply select the Thread Group, and choose Add → Config Element → HTTP Cookie Manager, either from the Edit Menu, or from the right-click pop-up menu.

4.4 Adding HTTP Requests

In our Test Plan, we need to make two HTTP requests. The first one is for the JMeter home page (http://jmeter.apache.org/), and the second one is for the Changes page (http://jmeter.apache.org/changes.html).

JMeter sends requests in the order that they appear in the tree.

Start by adding the first HTTP Request to the JMeter Users element (Add → Sampler → HTTP Request). Then, select the HTTP Request element in the tree and edit the following properties (see Figure 4.6):

1. Change the Name field to "Home Page".
2. Set the Path field to "/". Remember that you do not have to set the Server Name field because you already specified this value in the HTTP Request Defaults element.

Next, add the second HTTP Request and edit the following properties (see Figure 4.7:

1. Change the Name field to "Changes".
2. Set the Path field to "/changes.html".

4.5 Adding a Listener to View Store the Test Results

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

Select the JMeter Users element and add a Graph Results listener (Add → Listener → Backend Listener).

4.6 Logging in to a web-site

It's not the case here, but some web-sites require you to login before permitting you to perform certain actions. In a web-browser, the login will be shown as a form for the user name and password, and a button to submit the form. The button generates a POST request, passing the values of the form items as parameters.

To do this in JMeter, add an HTTP Request, and set the method to POST. You'll need to know the names of the fields used by the form, and the target page. These can be found out by inspecting the code of the login page. [If this is difficult to do, you can use the JMeter Proxy Recorder to record the login sequence.] Set the path to the target of the submit button. Click the Add button twice and enter the username and password details. Sometimes the login form contains additional hidden fields. These will need to be added as well.

4.7 choose the same user or different users

When creating a Test Plan, on each Thread Group iteration, we can choose to simulate the same user running multiple iterations, or different users running one iteration. You can configure this behaviour on Thread Group element, and have HTTP Cache Manager, HTTP Cookie Manager, HTTP Authorization Manager controlled by this setting.

You can choose to clear the cookies/cache content/authorization in the CookieManager/CacheManager/Authorization Manager, or choose to be controlled by the Thread Group.

Figure 4.10. Use Thread Group to control CookieManagerFigure 4.11. Use Thread Group to control CacheManagerFigure 4.12. Use Thread Group to control Authorization Manager

5. Building an Advanced Web Test Plan

In this section, you will learn how to create advanced Test Plans to test a Web site.

For an example of a basic Test Plan, see Building a Web Test Plan.

5.1 Handling User Sessions With URL Rewriting

If your web application uses URL rewriting rather than cookies to save session information, then you'll need to do a bit of extra work to test your site.

To respond correctly to URL rewriting, JMeter needs to parse the HTML received from the server and retrieve the unique session ID. Use the appropriate HTTP URL Re-writing Modifier to accomplish this. Simply enter the name of your session ID parameter into the modifier, and it will find it and add it to each request. If the request already has a value, it will be replaced. If "Cache Session Id?" is checked, then the last found session id will be saved, and will be used if the previous HTTP sample does not contain a session id.

URL Rewriting Example

Download this example. In Figure 1 is shown a test plan using URL rewriting. Note that the URL Re-writing modifier is added to the SimpleController, thus assuring that it will only affect requests under that SimpleController.

In Figure 2, we see the URL Re-writing modifier GUI, which just has a field for the user to specify the name of the session ID parameter. There is also a checkbox for indicating that the session ID should be part of the path (separated by a ";"), rather than a request parameter

5.2 Using a Header Manager

The HTTP Header Manager lets you customize what information JMeter sends in the HTTP request header. This header includes properties like "User-Agent", "Pragma", "Referer", etc.

The HTTP Header Manager, like the HTTP Cookie Manager, should probably be added at the Thread Group level, unless for some reason you wish to specify different headers for the different HTTP Request objects in your test.

6. Building a Database Test Plan

In this section, you will learn how to create a basic Test Plan to test a database server. You will create fifty users that send 2 SQL requests to the database server. Also, you will tell the users to run their tests 100 times. So, the total number of requests is (50 users) x (2 requests) x (repeat 100 times) = 10'000 JDBC requests. To construct the Test Plan, you will use the following elements: Thread Group, JDBC Request, Summary Report.

This example uses the MySQL database driver. To use this driver, its containing **.jar** file (ex. **mysql-connector-java-X.X.X-bin.jar**) must be copied to the JMeter **./lib** directory (see JMeter's Classpath for more details).

6.1 Adding Users

The first step you want to do with every JMeter Test Plan is to add a Thread Group element. The Thread Group tells JMeter the number of users you want to simulate, how often the users should send requests, and how many requests they should send.

Go ahead and add the ThreadGroup element by first selecting the Test Plan, clicking your right mouse button to get the **Add** menu, and then select *Add → ThreadGroup*.

You should now see the Thread Group element under Test Plan. If you do not see the element, then *expand* the Test Plan tree by clicking on the Test Plan element.

Next, you need to modify the default properties. Select the Thread Group element in the tree, if you have not already selected it. You should now see the Thread Group Control Panel in the right section of the JMeter window (see Figure 6.1 below)

Start by providing a more descriptive name for our Thread Group. In the name field, enter **JDBC Users**.

You will need a valid database, database table, and user-level access to that table. In the example shown here, the database is '**cloud**' and the table name is '**vm_instance**'.

Next, increase the number of users to **50**.

In the next field, the Ramp-Up Period, leave the value of **10** seconds. This property tells JMeter how long to delay between starting each user. For example, if you enter a Ramp-Up Period of 10 seconds, JMeter will finish starting all of your users by the end of the 10 seconds. So, if we have 50 users and a 10 second Ramp-Up Period, then the delay between starting users would be 200 milliseconds (10 seconds / 50 users = 0.2 second per user). If you set the value to 0, then JMeter will immediately start all of your users.

Finally, enter a value of **100** in the Loop Count field. This property tells JMeter how many times to repeat your test. To have JMeter repeatedly run your Test Plan, select the Forever checkbox.

In most applications, you have to manually accept changes you make in a Control Panel. However, in JMeter, the Control Panel automatically accepts your changes as you make them. If you change the name of an element, the tree will be updated with the new text after you leave the Control Panel (for example, when selecting another tree element).

See Figure 6.2 for the completed JDBC Users Thread Group.

6.2 Adding JDBC Requests

Now that we have defined our users, it is time to define the tasks that they will be performing. In this section, you will specify the JDBC requests to perform.

Begin by selecting the **JDBC Users** element. Click your right mouse button to get the **Add** menu, and then select *Add → Config Element → JDBC Connection Configuration*. Then, select this new element to view its Control Panel (see Figure 6.3).

Set up the following fields (these assume we will be using a MySQL database called '**cloud**'):

- Variable name (here: **myDatabase**) bound to pool. This needs to uniquely identify the configuration. It is used by the JDBC Sampler to identify the configuration to be used.
- Database URL: **jdbc:mysql://ipOfTheServer:3306/cloud**
- JDBC Driver class: **com.mysql.jdbc.Driver**
- Username: *the username of database*
- Password: *password for the username*

The other fields on the screen can be left as the defaults.

JMeter creates a database connection pool with the configuration settings as specified in the Control Panel. The pool is referred to in JDBC Requests in the '**Variable Name**' field. Several different JDBC Configuration elements can be used, but they must have unique names. Every JDBC Request must refer to a JDBC Configuration pool. More than one JDBC Request can refer to the same pool.

Selecting the JDBC Users element again. Click your right mouse button to get the **Add** menu, and then select *Add → Sampler → JDBC Request*. Then, select this new element to view its Control Panel (see Figure 6.4).

In our Test Plan, we will make two JDBC requests. The first one is for select all 'Running' VM instances, and the second is to select 'Expunging' VM instance (obviously you should change these to examples appropriate for your particular database). These are illustrated below.

JMeter sends requests in the order that you add them to the tree.

Start by editing the following properties (see Figure 6.5):

- Change the Name to '**VM Running**'.
- Enter the Pool Name: '**myDatabase**' (same as in the configuration element)
- Enter the SQL Query String field.
- Enter the Parameter values field with '**Running**' value.
- Enter the Parameter types with '**VARCHAR**'.

Next, add the second JDBC Request and edit the following properties (see Figure 6.6):

- Change the Name to '**VM Expunging**'.
- Change the value of Parameter values to '**Expunging**'.

6.3 Adding a Listener to View/Store the Test Results

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

Select the *JDBC Users* element and add a Summary Report listener (*Add → Listener → Summary Report*).

Save the test plan, and run the test with the menu *Run → Start* or **Ctrl** + **R**

The listener shows the results.

7. Building an FTP Test Plan

In this section, you will learn how to create a basic Test Plan to test an FTP site. You will create four users that send requests for two files on a FTP site. Also, you will tell the users to run their tests twice. So, the total number of requests is (4 users) x (2 requests) x (repeat 2 times) = 16 FTP requests.

To construct the Test Plan, you will use the following elements: Thread Group, FTP Request, FTP Request Defaults, and View Results in Table.

7.1 Adding Users

The first step you want to do with every JMeter Test Plan is to add a Thread Group element. The Thread Group tells JMeter the number of users you want to simulate, how often the users should send requests, and the how many requests they should send.

Go ahead and add the Thread Group element by first selecting the Test Plan, clicking your right mouse button to get the Add menu, and then select **Add** → **ThreadGroup.**

You should now see the **Thread Group** element under **Test Plan.** If you do not see the element, then "expand" the Test Plan tree by clicking on the **Test Plan** element.

Next, you need to modify the default properties. Select the **Thread Group** element in the tree, if you have not already selected it. You should now see the Thread Group Control Panel in the right section of the JMeter window (see Figure 7.1 below)

Start by providing a more descriptive name for our **Thread Group.** In the name field, enter 'FTP Users'.

Next, increase the number of users to 4.

In the next field, the *Ramp-Up* Period, leave the default value of 0 seconds. This property tells JMeter how long to delay between starting each user. For example, if you enter a *Ramp-Up* Period of 5 seconds, JMeter will finish starting all of your users by the end of the 5 seconds. So, if we have 5 users and a 5 second *Ramp-Up* Period, then the delay between starting users would be 1 second (5 users / 5 seconds = 1 user per second). If you set the value to 0, then JMeter will immediately start all of your users.

Finally, enter a value of 2 in the *Loop Count* field. This property tells JMeter how many times to repeat your test. To have JMeter repeatedly run your **Test Plan,** select the *Forever* checkbox.

In most applications, you have to manually accept changes you make in a Control Panel. However, in JMeter, the Control Panel automatically accepts your changes as you make them. If you change the name of an element, the tree will be updated with the new text after you leave the Control Panel (for example, when selecting another tree element).

See Figure 7.2 for the completed FTP Users Thread Group.

7.2 Adding Default FTP Request Properties

Now that we have defined our users, it is time define the tasks that they will be performing. In this section, you will specify the default settings for your FTP requests. And then, in section 7.3, you will add **FTP Request** elements which use some of the default settings you specified here.

Begin by selecting the FTP Users element. Click your right mouse button to get the Add menu, and then select **Add** → **Config Element** → **FTP Request Defaults.** Then, select this new element to view its Control Panel (see Figure 7.3).

Like most JMeter elements, the FTP Request Defaults Control Panel has a name field that you can modify. In this example, leave this field with the default value.

Skip to the next field, which is the FTP Server's Server Name/IP. For the Test Plan that you are building, all FTP requests will be sent to the same FTP server, ftp.domain.com in this case. Enter

this domain name into the field. This is the only field that we will specify a default, so leave the remaining fields with their default values.

The FTP Request Defaults element does not tell JMeter to send an FTP request. It simply defines the default values that the FTP Request elements use.

See Figure 7.4 for the completed FTP Request Defaults element

7.3 Adding FTP Requests

In our **Test Plan**, we need to make two **FTP requests**.

JMeter sends requests in the order that they appear in the tree.

Start by adding the first FTP Request to the FTP Users element (**Add** → **Sampler** → **FTP Request**). Then, select the **FTP Request** element in the tree and edit the following properties (see Figure 7.5):

1. Change the *Name* to "File1".
2. Change the *Remote File* field to "/directory/file1.txt".
3. Change the *Username* field to "anonymous".
4. Change the *Password* field to "anonymous@test.com".

You do not have to set the *Server Name* field because you already specified this value in the **FTP Request Defaults** element.

Next, add the second **FTP Request** and edit the following properties (see Figure 7.6:

1. Change the *Name* to "File2".
2. Change the *Remote File* field to "/directory/file2.txt".
3. Change the *Username* field to "anonymous".
4. Change the *Password* field to "anonymous@test.com".

7.4 Adding a Listener to View/Store the Test Results

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

Select the FTP Users element and add a View Results in Table listener (**Add** → **Listener** → **View Results in Table**).

Run your test and view the results.