

Frequently Asked Questions

Table of contents

1 Questions.....	2
1.1 1. Common Grinder Questions.....	2
1.1.1 1.1. General.....	2
1.1.2 1.2. Running the tests.....	3
1.1.3 1.3. Statistics.....	4
1.1.4 1.4. HTTP.....	5
1.1.5 1.5. Problems.....	6
1.2 2. The Grinder 2.....	9
1.2.1 2.1. General.....	9
1.2.2 2.2. Scripts.....	10
1.3 3. The Grinder 3.....	11
1.3.1 3.1. General.....	11
1.3.2 3.2. Scripts.....	13
1.3.3 3.3. SSL.....	18
1.3.4 3.4. The TCPProxy.....	18
1.3.5 3.5. The Console.....	20

Questions

1. Common Grinder Questions

1.1. General

1.1.1. So its called "The Grinder" then?

Yes. "*The* Grinder" is preferred over just plain "Grinder".

1.1.2. Should I use The Grinder 2 or The Grinder 3?

The Grinder 3. The Grinder 3 has many [enhancements](http://grinder.sourceforge.net/g3/whats-new.html) (<http://grinder.sourceforge.net/g3/whats-new.html>) over The Grinder 2. The Grinder 3 is in active development.

If you continue to use The Grinder 2, be aware that it has defects (including [this one](#) and [this one](#)) that will not be fixed.

1.1.3. How does The Grinder stack up against a commercial tool like Mercury Interactive's LoadRunner™?

Here is an edited version of [Tom Braverman](mailto:tom.braverman@bea.com) (<mailto:tom.braverman@bea.com>) 's post to [grinder-use](mailto:grinder-use@lists.sourceforge.net) (<mailto:grinder-use@lists.sourceforge.net>) .

A few reasons:

- The Grinder is lightweight
Compared to setting up LoadRunner or some other full featured tools, The Grinder is trivial to install and get running.
- The Grinder is a *programmer's load tester*
Too often, programmers defer load testing to some other group (e.g., QA) and don't test their own components for scalability. The Grinder is designed for people who understand the code that they're hitting - it's not just a "black box" with a set of associated response times.

Since tests can be coded - and not simply scripted, programmers get to test interior tiers of their application and not just response time via the user interface.
- The Grinder is free
I'm a consulting professional and I have to come up with solutions to deadlocks and slow downs. Sometimes I only have hours to recreate a problem and then attempt to resolve it. I can't count on my client having a given load testing tool and many (most?) development teams don't have any such tool (they defer this type of testing to QA as mentioned above). I can bring The Grinder in and set it up and apply load quite quickly.

Summarizing: I don't try and persuade my clients that The Grinder is a replacement for LoadRunner, etc. I tell them that The Grinder is for use by the developers and that they'll still want the QA team to generate scalability metrics using LoadRunner or

some other tool approved for the purpose by management.

Story: One client was attempting to determine scalability based on LoadRunner. The LoadRunner team - with no understanding at all of the app - was telling them that some pages were giving response times of 30 seconds. The project manager knew this was patent BS since he could hit the enter key on his page and only count to 3 or 4 before the page displayed. The client spent many resource days attempting to understand LoadRunner numbers. Within a few hours, The Grinder was up and running and reporting numbers that appeared to track with the user experience. The Grinder became the gold standard that the client used to measure LoadRunner.

Its worth adding to this that many companies *are* using The Grinder for production load testing.

1.1.4. How do I subscribe/unsubscribe from The Grinder mailing lists?

You can subscribe and unsubscribe at [Sourceforge](http://www.sourceforge.net/mail/?group_id=18598) (http://www.sourceforge.net/mail/?group_id=18598) . [More details](http://grinder.sourceforge.net/support.html) (<http://grinder.sourceforge.net/support.html>) .

1.2. Running the tests

1.2.1. How do I control the number of simulated users?

Typically a worker thread is thought of as a client context which can simulate a concurrent user. You can control the number of worker processes per agent and the number of threads per process. You shouldn't need to run more than one agent process per machine. The number of simulated users is then

```
number of worker threads x number of worker processes x number test machines
```

To simulate additional users it is usually better to increase the number of worker threads rather than the number of processes except where:

- Your JVM threading implementation is particularly inefficient. This is not such a problem nowadays. Running several hundred threads per process is usually fine.
- The protocol you are using will behave differently when distributed across multiple processes. For example, the WebLogic Server t3 protocol is optimised so that a single TCP/IP connection is used between any two processes.

1.2.2. Can I run different test scripts against the same console?

Yes.

The console receives reports and updates the graph based on test number. Thus it is possible to have different agent processes running scripts with different test numbers (e.g. one process running tests 1 to 5 and another running tests 6 to 10)., reporting to the same console.

1.2.3. How do I run the tests for a specific period of time?

Use the console to control the test. If you want the tests to run for an hour and your sample interval is set to 5 seconds, set the console to collect 720 samples.

If you're not using the console, see [grinder.duration](http://grinder.sourceforge.net/g3/properties.html) (<http://grinder.sourceforge.net/g3/properties.html>) .

1.2.4. Is there any way to record CPU utilisation of the server?

No. The Grinder only measures the response of the test server as a black box. Look to other tools such as `sar` and `vmstat` to measure server CPU utilisation.

1.3. Statistics

1.3.1. Where is the raw data stored?

Liam Morley asked:

```
I understand that all the information from the workers is
sent back to the client, but all I see thus far is rough
statistical data which seems to have been compiled from
the raw data. When the data is sent back to the console,
where is it stored?
```

For efficiency, the worker processes only send back aggregate information to the console, not the raw data. Each worker process sends the console a report every 500ms. The report contains information for each test that was invoked during the period which includes the number of invocations, the total time taken, the number of errors, and any other custom statistics.

Each worker process writes out the raw information to a process specific data file. These are the `data_...` files that are stored with the other log files.

Apart from the raw data, another advantage that these files give you is that the console doesn't have a record of the testing time line. Whilst the console graphs present this information graphically, the console only allows you to save snapshots of the current statistics.

1.3.2. How does the Grinder calculate the mean time and TPS?

Each worker process records the time taken by each thread for each test and regularly reports the mean time for each test to the console. The console calculates the mean time for each test across all worker processes. This is the *mean time*.

Tests per second (TPS) is calculated by the console. It is the sum of the number of tests performed by all worker processes within the sampling period, divided by the duration of the sampling period. Using longer sampling periods will give better averages, but the console display will be updated less frequently.

1.3.3. What is the relation between mean time and TPS?

In general there is not a linear relationship between the test time and the number of tests per second.

Most time in a typical server-side application is spent waiting on I/O. Elapsed time is not the sum the time spent by all of the threads. If I have a simple servlet or EJB that does the following:

```
void doit() { Thread.sleep(1000); // 1 second }
```

Say with this I get 100 TPS and a 1.2 second response time. If I now change the code to:

```
void doit() { Thread.sleep(10000); // 10 seconds }
```

the system isn't doing any more work and I'd expect to get 100 TPS and a 10.2 second response time.

Note:

This is only true assuming we have infinite server side threads. In practice, the size of a server side thread pool has a significant effect on the behaviour of a system. I hope that this discussion helps you see that there is no direct connection between TPS and test time.

1.4. HTTP

1.4.1. Why do POSTs take longer than GETs?

Dennis Linnell wondered this. This is what he found out:

A network protocol trace revealed that the differences in POST vs. GET times were attributable to the following:

1. The Grinder (HTTPClient) sends a GET request as a single TCP protocol data unit (PDU), which requires a single acknowledgement (ACK). This is normal and expected.
2. HTTPClient sends a POST as two PDUs: The first includes everything but the Name/Value pairs and requires an ACK; the second includes only the Name/Value pairs and also requires an ACK. This is a consequence of the conservative implementation of HTTP/1.1 pipelining in HTTPClient.
3. When the first PDU of the POST arrives at the Windows XP TCP/IP protocol stack on the server, the *delayed ACK* feature of TCP/IP RFC 1122 kicks in. If another request were to arrive, the stack would send an ACK; otherwise it waits up to 200 ms. and then sends an ACK. Since, in this case, no other request arrives, [the stack delays the ACK](http://support.microsoft.com/kb/328890/) (<http://support.microsoft.com/kb/328890/>) . Naturally, I made the changes outlined in the Microsoft knowledge base and it solved the problem for me. Case closed? Maybe not.
4. Still, the question remains, *Is the HTTPClient behavior reasonable?* I traced Microsoft Internet Explorer 6 (latest patch level) and it sends a POST in one PDU and thus does not invoke the dreaded *delayed ACK*. On the other hand, Firefox 1.0.4 (oddly) requires 3 pairs of PDUs to get the job done. So HTTPClient is somewhere in between.
5. As HTTPClient's behavior is not in violation of the HTTP/1.1 RFC, I have no reason to complain. And, looking at the HTTPClient code, I'm certainly not bold enough to touch it. Still, it doesn't quite model the behavior of the market-leading web browser accurately.

1.4.2. For a given web page, why does a Grinder script take a much longer time to complete than my browser takes to display the page?

How your web application behaves under non-trivial load and how long it takes to display a page are two distinct concerns.

The Grinder runs through test script elements sequentially while browsers, such as MS Internet Explorer, can request several elements at the same time using parallel threads. The number of parallel threads depends on the browser type, MS Internet Explorer typically uses 3. Therefore the browser should complete requesting the complete list of elements from a web page before the Grinder.

If you are concerned over page download times rather than application behaviour under load then [Peter Booth](mailto:pbooth@marketaxess.com) writes:

To understand how a typical browser will download and display *your* page you should use a tool like IBM's [Page Detailer](http://www.alphaworks.ibm.com/tech/pagedetailer) (<http://www.alphaworks.ibm.com/tech/pagedetailer>) which intercepts and instruments all browser activity and visualises this in an easy to interpret fashion. It can highlight tuning opportunities like coalescing JavaScript, reducing GIF count, that are more in the page design than dynamic application design arena.

From the IBM Page Detailer Web Site

IBM Page Detailer places a probe in the Windows Socket Stack to capture data about timing, size, and data flow, and then it presents the collected data in both graphical and tabular views, clearly showing how the page was delivered to the browser.

Tools like IBM Page Detailer are a useful addition to your arsenal when looking at the complete picture. The [tamperdata](https://addons.mozilla.org/firefox/966/) (<https://addons.mozilla.org/firefox/966/>) add-on for Firefox is also worth a look.

1.5. Problems

1.5.1. I've seen The Grinder report negative test times?!

Are you running Linux kernel 2.2.18? Mikael Suokas reports:

In some load situations, on some hardware, older Linux kernels can generate system times that jump backwards. I have experienced this problem on one machine (Intel Pentium/150, 128M RAM, Adaptec AHA-2940, SCSI disks) running the stock Red Hat Linux 7.0 kernel (2.2.16).

The Grinder occasionally reported negative response times in the -900 to -700 ms range whenever the load became high. Interestingly, the same OS + kernel version on several other machines never showed these negative response times. The problem was not Grinder or Java specific: a C program polling `gettimeofday()` also showed system time jumping backwards.

Upgrading to kernel 2.2.19 fixed this issue for me. Since the release notes for Linux 2.2.18 mention time keeping locking fixes, I don't think that was a coincidence.

Of course, you should upgrade any 2.2.x kernel to 2.2.19 anyway, because because of the many security fixes.

1.5.2. What can I do about address in use exceptions on Microsoft Windows machines?

Answer courtesy of Venkat:

Recently, there have been a couple posts from people facing the *address in use* exception when running The Grinder. I also faced the same issue and here is what I found and how I resolved it.

Windows OS TCP-IP system has a parameter that controls the maximum port number used when an application requests any available user port from the system. By default, ephemeral (that is, short-lived) ports are allocated between the values of 1024 and 5000 inclusive. During a Grinder test, if we exceed this limit, we get the *address in use* exception. To make it more complicated, there is another parameter that says once the application closes a TCP connection, how long the OS will wait before reclaiming the port for the connection. The default value for this is 4 minutes.

Due to the above, often the default Windows configuration isn't sufficient to run load tests. To change this, modify the following Registry values under the key `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters`

```
MaxUserPort          = dword:00004e20 (20,000 decimal)
TcpTimedWaitDelay    = dword:0000001e (30 decimal)
```

The above was sufficient for my testing. Adjust the `MaxUserPort` to a larger number if you still run into the exception. Of course, the OS needs to be rebooted once the setting is changed. MS documentation says the `MaxUserPort` setting is available since Windows NT 3.51 Service Pack 5. I am not sure which version introduced the `TcpTimedWaitDelayed` setting. You can confirm whether your OS version supports this by looking into the Registry location above.

1.5.3. What can I do about address in use exceptions on Linux machines?

Prabhat Jha writes:

1. Increase the ephemeral port range:

```
echo "32768 65535" > /proc/sys/net/ipv4/ip_local_port_range
```

2. Enable Port Recycling

```
echo "1" > /proc/sys/net/ipv4/tcp_tw_recycle
```

3. Reduce the timeout (default is 60 seconds)

```
echo "10" > /proc/sys/net/ipv4/tcp_fin_timeout
```

1.5.4. How do I specify the ephemeral port range?

When a client connects to a TCP server, the client side port uses a temporary *ephemeral* port. Occasionally users want to set the ephemeral ports that are used for:

- Connections from the agent and worker processes to the console.
- Connections created by the TCPProxy.
- Connections created by the HTTP Plugin.

The Grinder does not specify the ephemeral port numbers, and instead uses values assigned by the operating system's TCP/IP stack. If you wish to control the ephemeral port numbers, please see [NCFTP](http://www.ncftp.com/ncftpd/doc/misc/ephemeral_ports.html) (http://www.ncftp.com/ncftpd/doc/misc/ephemeral_ports.html) for a good HOWTO.

1.5.5. Why can't I create more than ten connections per second on Windows XP?

If you're using Windows XP SP2, [this article](http://www.speedguide.net/read_articles.php?id=1497) (http://www.speedguide.net/read_articles.php?id=1497) may provide the answer.

1.5.6. What are the limits to accurate timing?

The Grinder records test times for each successful test. By default, this is done by a section of code that looks like:

```
context.startTimer(); // Critical section starts

try {
    // do test
}
finally {
    context.stopTimer(); // Critical section ends
}
```

This is repeated for each test.

If there are many threads within the worker process, (and the sleep time is small or the test takes a long time or the test performs I/O), it is *highly* likely that the JVM will swap the thread out in the critical section causing an erroneously large test time to be reported.

Similarly, if the load injector machine that you are running The Grinder on is also running other active processes (such as other worker processes), it is *highly* likely that the operating system will swap the process out in the critical section, again causing an erroneously large test time to be reported. If The Grinder is co-hosted with the target server, and the plug-in uses a synchronous protocol, (e.g. the HTTP plug-in), such swapping is a *certainty*.

Further, as the CPU utilisation rises, the contention on the critical section rises non-linearly and in a way that is difficult to quantify. The recorded time becomes more a measure of how efficiently the OS and JVM can swap between multiple threads and less a measure of server performance.

This is a generic problem with all test harnesses and is not limited to The Grinder or Java. Within the scope of a single machine there is little that can be done about this whilst realistically using multiple threads and processes.

Fiddling with Thread scheduling - a partial fix

From The Grinder 2.6.1, the call to `startTimer` makes a `Thread.yield()` call before recording the start time which means that a thread is more likely to be swapped out/in just before the critical sections. It dramatically reduced the response times I measured (e.g. 30 ms to 3 ms). I consider this only an approximate fix to the problem - it does not prevent the OS from swapping the process out.

The recorded response time should always be considered an upper bound on the actual response time. Doing the `yield()` makes that bound more accurate.

An argument against doing this is that it slightly alters the statistical distribution of the client invocations. I'd counter that without the `yield()` the distribution is not even; its down to the OS and JVM scheduling so threads/processes are far likely to be swapped at some points (e.g. waiting on I/O) than others. Because of this I decided there is little point

in making the `yield()` optional.

The *Timer Client* model - a solution?

One solution to this problem is to dedicate a single machine to the measuring of response times. I call this the "timer client" model.

As of The Grinder 2.6.1 you can set a property ([grinder.recordTime](http://grinder.sourceforge.net/g2/properties.html) (http://grinder.sourceforge.net/g2/properties.html) for The Grinder 2, renamed to [grinder.reportTimesToConsole](http://grinder.sourceforge.net/g3/properties.html) (http://grinder.sourceforge.net/g3/properties.html) for The Grinder 3) to be `false` which will cause the worker processes that use that `grinder.properties` file not report the test times to the console. You should run all but one of your worker processes with this property set to `false`. These are the *load clients*.

You should copy the `grinder.properties` file to a dedicated *timer client* machine, change the `grinder.recordTime/grinder.reportTimesToConsole` property to be `true`, and set [grinder.processes](http://grinder.sourceforge.net/g2/properties.html) (http://grinder.sourceforge.net/g2/properties.html) and [grinder.threads](http://grinder.sourceforge.net/g2/properties.html) (http://grinder.sourceforge.net/g2/properties.html) to 1. The single worker process will run on the timing client, record all timing information and (optionally) report it to the console. The less other stuff you run on the timing client, the better.

The disadvantage of this method is that the statistical sample of the test times is much smaller.

An update after some experimentation

Testing has shown that the difference the timer client model makes is only measurable when those clients are co-hosted with the server. When you have separate server and client machines its better to not use the timer client model because it decreases the sample size of test times.

I recommend trying both models. If you discover something interesting, please report it to [grinder-use](mailto:grinder-use@lists.sourceforge.net) (mailto:grinder-use@lists.sourceforge.net) .

2. The Grinder 2

2.1. General

2.1.1. What do I need to do to set up multicast?

You must set up multicast if you want to use the console with The Grinder 2. It is used to send signals from the Console to the Grinder processes (start, stop). Multicast is no longer necessary for The Grinder 3.

Multicast addresses lie in the range 224.0.0.0 to 239.255.255.255. Ports lie in the range 0 to 65535. You should ensure that the address and ports you chose does not clash with other applications running on your LAN. The example files uses the address 228.1.1.1:1234:

```
grinder.receiveConsoleSignals=true
grinder.grinderAddress=228.1.1.1
grinder.grinderPort=1234
```

For most modern TCP stacks, e.g Windows 95/98/NT, Linux, multicast works out of the box.

Under Linux, you may also need to set up the routing table. Try:

```
route add -net 224.0.0.0 netmask 240.0.0.0 dev eth0
```

Some Windows VPN clients (e.g Bay Networks Extranet) interfere with multicast. You may need to disable them.

2.1.2. What do I need to do to set up multicast under Windows 2000?

With a stand alone Windows 2000 machine, you might experience similar grief to myself. I found that I could only get multicast to work if my LAN NIC had a carrier and the MS loop-back adapter is not installed (or disabled). The following links contain experiences that tally with mine:

- <http://www.web3d.org/WorkingGroups/vrtp/dis-java-vrml/hypermail/2001/0021.html>
(<http://web.archive.org/web/20030802192353/www.web3d.org/WorkingGroups/vrtp/dis-java-vrml/hypermail/2001/0021.html>)
- <http://www.web3d.org/WorkingGroups/vrtp/dis-java-vrml/hypermail/2001/0023.html>
(<http://web.archive.org/web/20030802192353/www.web3d.org/WorkingGroups/vrtp/dis-java-vrml/hypermail/2001/0023.html>)
- <http://www.web3d.org/WorkingGroups/vrtp/dis-java-vrml/hypermail/2001/0025.html>
(<http://web.archive.org/web/20030802192353/www.web3d.org/WorkingGroups/vrtp/dis-java-vrml/hypermail/2001/0025.html>)

The last of these says:

I have been looking for a solution for this problem a long time without success :- (I found several dummy loop-back IP stacks but none that supports multicast. The solution I have been using is to bring along a tiny hub and connect my laptop to that hub when doing demos. Windows is a bit stupid in the way that it only checks if it has a carrier.

I now use a slightly cheaper/lighter solution; namely I have cropped a network cable short and twisted my own physical loop-back adapter. Needless to say, if anyone figures out how to get multicast working with a stand alone W2K machine, I'm more than interested.

Note:

Multicast is no longer required by The Grinder 3.

2.1.3. Why do I get HTTPClient.RetryException: Premature EOF encountered errors when I increase HTTP load?

This is a known issue with The Grinder 2 that is fixed in The Grinder 3.0b14. I have no plans to backport the fix to The Grinder 2, but see [this e-mail](mailto:philip@grinder.sourceforge.net) (<http://article.gmane.org/gmane.comp.java.grinder.user/893/match=premature+eof>) for details of how to do this yourself.

2.2. Scripts

2.2.1. How do I simulate different users with The Grinder 2?

You probably want to use a [String Bean](http://grinder.sourceforge.net/g2/http-plugin.html#string-bean) (<http://grinder.sourceforge.net/g2/http-plugin.html#string-bean>) .

2.2.2. How do I generate POST data with a String Bean?

Sean Kroah writes:

```
grinder.test0.parameter.post=<getPostData> doesn't work.
```

I didn't really see this in the docs anywhere so if anyone is interested. I needed dynamic post data and the HTTP plugin post parameter only took a file name. The key was that the file can optionally contain a reference to a [String Bean](http://grinder.sourceforge.net/g2/http-plugin.html#string-bean) (<http://grinder.sourceforge.net/g2/http-plugin.html#string-bean>) method. I tried it on a fluke and it worked.

```
grinder.test0.parameter.url=<getLoginUrl>
grinder.test0.parameter.post=getPostData.dat
```

getPostData.dat has one line in it:

```
<getPostData>
```

This tells The Grinder to call my String Bean method which returns my dynamic POST content based on the test description. I'm pretty pleased with that.

2.2.3. Why do my POSTs have an extra new line?

An anonymous user reports:

The Grinder 2 appends a new line character to the end of file which contains post information for a test. This appears to happen only under Windows.

This is a known issue with The Grinder 2 that is fixed in The Grinder 3.0. I have no plans to backport the fix to The Grinder 2, but see the [bug report](http://www.sourceforge.net/tracker/index.php?func=detail&aid=1081136&group_id=18598&atid=1185) (http://www.sourceforge.net/tracker/index.php?func=detail&aid=1081136&group_id=18598&atid=1185) for how to do this yourself.

3. The Grinder 3

3.1. General

3.1.1. Why does The Grinder take a while to start up?

The Grinder 3 worker process start up time is marginally slower than The Grinder 2. However, it is much slower when you add new Java libraries to your CLASSPATH.

At start up, the Jython engine processes all new Java libraries it finds. When it does this you'll see output similar to the following in the window you used to start The Grinder:

```
*sys-package-mgr*: processing new jar,
'E:\src\grinder3\lib\jakarta-oro-2.0.6.jar'
*sys-package-mgr*: processing new jar, 'E:\src\grinder3\lib\jython.jar'
```

Jython caches the result of this processing, so subsequent start up times are much reduced. You can control the location of the cache directory, see the information on [Jython installation](http://grinder.sourceforge.net/g3/scripts.html#jython-installation) (<http://grinder.sourceforge.net/g3/scripts.html#jython-installation>).

3.1.2. What happened to the JUnit plugin?

The Grinder 2 had a JUnit plugin which allowed JUnit test cases to be called. There is no

JUnit plugin in The Grinder 3 but you can call arbitrary Java code so you should be able to invoke your JUnit test cases directly.

If you have a lot of JUnit test cases, it would be appropriate to wrap up the steps necessary to invoke a test case in a Jython library. Contributions to the [script gallery](http://grinder.sourceforge.net/g3/script-gallery.html) (<http://grinder.sourceforge.net/g3/script-gallery.html>) are welcome.

See also [this article](#)

(http://sourceforge.net/mailarchive/forum.php?thread_id=1687434&forum_id=2650) on *grinder-use*.

3.1.3. The Grinder pauses before either threads start sending data, or the TCPProxy carries out a request, when interacting with web servers. Why is this?

[Karol Muszynski](mailto:karol.muszynski@hp.com) (<mailto:karol.muszynski@hp.com>) writes:

Solution:

We found that it was a problem with DNS configuration. Solution was to add the web servers host names and ip addresses to the windows hosts file.

Verification:

Simply on the command line write "nslookup", then hit enter, and on the following line type your web server "hostname" and hit enter. If this takes a long time to get an ip address, it can mean that this is your problem.

Common locations for hosts files:

Windows: "C:\winnt\system32\drivers\etc\hosts"

Unix: "/etc/hosts"

3.1.4. How do I debug worker processes?

Worker processes are separate child processes of agent processes, so they are sometimes a bit fiddly to debug. It is possible to run an agent and its workers in a single process. To do this, please start the agent with:

```
java -Dgrinder.debug.singleprocess=true net.grinder.Grinder
```

This puts the worker and agent code in a single process, making it easier to take a thread dump of the worker code.

Use the minimum number of threads that cause the problem. When the hang is observed, take a [thread dump](#) and post the output to the [grinder-use](mailto:grinder-use@lists.sourceforge.net) (<mailto:grinder-use@lists.sourceforge.net>) mail list.

3.1.5. How do I take a thread dump?

A thread dump is a diagnostic report of the state of a Java process. If you report an unresponsive grinder process to one of the mailing lists, you most likely will be asked to take a thread dump. This answer explains how to do that.

First identify the relevant [process](#)

(<http://grinder.sourceforge.net/g3/getting-started.html#The+Grinder+processes>), and start it from a command line terminal. The output will go to the command line, so either set the terminal up so that it has plenty of lines of scroll-back or redirect the output of the process

to a file. For example, you might start an agent process with:

```
C:\> java net.grinder.Grinder > output.txt
```

If you are using a UNIX-like operating system, taking the thread dump is simple. Just use `ps` to identify the process id, then issue a `kill -3 <pid>` to send the process a SIGQUIT signal. You will find the thread dump in the terminal window (or the file to which you redirected output).

On Windows, use one of the following:

- If it is an *agent* or *console* process you are interested in (not a *worker* process), type `Ctrl-Break` in the command line terminal.
- If it is a worker process, things are a little more involved. This is because worker processes are children of agent processes, and Java does not pass on `Ctrl-Break` signals to child processes. You can work around this problem by running the [workers and the agent in the same process](#), then using `Ctrl-Break` as above.

Alternatively, you can run a recent version of the Sun JVM (J2SE 5 for Linux or Solaris, J2SE 6 for Windows) and use the `jstack` command line. The alternative to `jstack` for the BEA JRockit JVM is the powerful `jrcmd` command.

3.1.6. Should I be concerned about the "can't create package cache dir" message?

At start up, the agent may log a message about not being able to create a package cache directory:

```
2/14/08 2:20:00 PM (agent): worker deep-0 started *sys-package-mgr*:  
can't create package cache dir  
'D:\grinder-3.0.1\lib\jython.jar\cachedir\packages'
```

This is due to a [Jython 2.2.1 bug](#)

(http://www.sourceforge.net/tracker/index.php?func=detail&aid=1894900&group_id=12867&atid=1128)

.

You can ignore the warning if you don't rely on Jython [package scanning](#) (<http://wiki.python.org/jython/PackageScanning>) (this is likely).

Alternatively, you can specify a cache directory by adding the following to `grinder.properties` (adjusting `/tmp` accordingly):

```
grinder.jvm.arguments: -Dpython.cachedir=/tmp
```

3.2. Scripts

3.2.1. How do I record an HTTP script?

Using the Grinder 3 it is possible to automatically generate a script of your HTTP journey. To do this you need to make use of the TCPProxy and its associated HTTP TCPProxy filters. The recommended filter to use is "-http" as the other filters are now deprecated. See this [link](#) (<http://grinder.sourceforge.net/g3/tcpproxy.html#HTTPPluginTCPProxyFilter>) for how to use the TCPProxy and the recommended filter.

3.2.2. How do I use standard Python modules in my scripts?

The Python standard library contains a number of very useful modules. For example the regular expression module `re`, the threading module, and the `random` module. To use these modules you must [install Jython](http://grinder.sourceforge.net/g3/scripts.html#jython-installation) (<http://grinder.sourceforge.net/g3/scripts.html#jython-installation>).

3.2.3. How do I simulate different users with The Grinder 3?

[Calum Fitzgerald](mailto:calum.fitzgerald@environment-agency.gov.uk) (mailto:calum.fitzgerald@environment-agency.gov.uk) writes:

This is a method we use to generate random users for tests on web applications.

We store the usernames and passwords in a text file (eg.users.txt) with the format:

```
user,password
```

We then read in the file and grab the components. The reason for using a file to store the usernames is that if you are using hundreds or thousands of usernames then Jython/Java has a limitation with arrays over 64KB.

```
#
# testRandomise.py
#
import random
import string

class TestRandomise:
    def __init__(self, filename):
        self._users = []
        infile = open(filename, "r")

        for line in infile.readlines():
            self._users.append(string.split((line), ','))
        infile.close()

    def getUserInfo(self):
        "Pick a random (user, password) from the list."
        return random.choice(self._users)

#
# Test script. Originally recorded by the TCPProxy.
#
from testRandomise import TestRandomise
tre = TestRandomise("users.txt")

class TestRunner:
    def __call__(self):
        # Get user for this run.
        (user, passwd) = tre.getUserInfo()

        # ...

        # Use the user details to log in.

        tests[2002].POST('https://host:443/securityservlet',
            ( NVPair('functionname', 'Login'),
              NVPair('pagename', 'Login'),
              NVPair('ms_emailAddress', user),
              NVPair('ms_password', passwd), ))
```

3.2.4. Is there a way to make my requests depend on previous responses?

[Carlos Franco](mailto:carlos.franco@xeridia.com) (mailto:carlos.franco@xeridia.com) asks:

```
Is there a way (with Grinder 3) to make that my requests
depends on the response?
```

Absolutely. This is one of the compelling reasons to use The Grinder 3 over The Grinder 2.

```
I will make an example. Suppose that when I make a request
the server gives me one list of, for example, cars, and I
want to edit the information of one, but I don't know its
ID, so I have to look into the response and look for it.
Can I do that?
```

I'm assuming HTTP and that the cars come back in a table such as

```
<td>Jaguar</td><td>1234</td>
<td>Bentley</td><td>0012</td>
<td>Rolls Royce</td><td>8323</td>
<td>Rover</td><td>9876</td>
```

You can get the text of the HTTP response body with something like:

```
response = myrequest.GET("/carindex.html")
text = response.text
```

See the [script API](http://g3/script-javadoc/index.html) (g3/script-javadoc/index.html) for other things you can do with the HTTPResponse object.

Now you have the choice of using Java or Python to extract the appropriate ID. Lets do it in Python. Rather than write low level string parsing, we'll use a regular expression for capturing interesting information. To use the standard `re` module, you should install Jython as described in [this FAQ](#).

Regular expressions are faster if compiled, so we'll do this up front:

```
# Add to top of script
import re
expression = re.compile(r"<td>([\w\s]*)</td>\s*<td>(\d*)</td>")
```

For eyes unused to magic of regular expressions, this one matches pairs of table cells, the first cell containing alphanumeric and white space, the second containing a number. `\w` matches an alphanumeric ("word") character, `\s` matches a white space character, `()` captures its contents in a group.

We can then use the regular expression to parse the response:

```
response = myrequest.GET("/carindex.html")

for car, id in expression.findall(response.text):
    print car, id
```

Of course you want to use this to select an ID and use it in a subsequent page. Lets pick a random one and use it for a new request:

```
response = myrequest.GET("/carindex.html")

# Import random for this to work.
car, id = random.choice(expression.findall(response.text))

request = myrequest.GET("/viewcar?id=%d" % id)
```

3.2.5. How do I replay a scenario, caching image files but not text files?

The Grinder itself does not cache files. It merely simulates browser behaviour. The `Not-Modified-Since` headers are recorded by the `TCPProxy` filter, so what you record with the `TCPProxy` is the caching behaviour of the browser you are using.

If this is not what you want you could add a function to your test script which takes an `HTTPRequest`, decides whether it should be considered as "cached" or not, and adds a `Not-Modified-Since` header as appropriate.

3.2.6. When using the script distribution feature, how can my scripts refer to other distributed files?

The directory of the distributed script is prepended to the Jython system path. You can do something like:

```
import sys
f = open("%s/myresources.properties" % sys.path[0])
# ...
```

3.2.7. Why does the Jython re module raise a ValueError when I compile a regular expression?

Each thread that you use to compile regular expressions must import the `re` module. If you fail to do this you'll get a `ValueError` like:

```
ValueError: ('unsupported operand type', 'in')
  File "D:\opt\jython\jython-2.1\Lib\sre_compile.py", line 141, in
  _compile
  File "D:\opt\jython\jython-2.1\Lib\sre_compile.py", line 61, in
  _compile
  File "D:\opt\jython\jython-2.1\Lib\sre_compile.py", line 352, in _code
  File "D:\opt\jython\jython-2.1\Lib\sre_compile.py", line 368, in
  compile
  File "D:\opt\jython\jython-2.1\Lib\sre.py", line 134, in _compile
  File "D:\opt\jython\jython-2.1\Lib\sre.py", line 90, in compile
  File "helloworld.py", line 24, in matchAll
  File "helloworld.py", line 39, in __call__
```

The best (most efficient) way to use regular expressions is to compile them in the main part of your script. For example:

```
import re

# Compile once in initialisation thread
p = re.compile('a.b', re.IGNORECASE)

class TestRunner:
    def __call__(self):
        # use p
```

Occasionally, the regular expression must be dynamically generated by a worker thread, so this technique doesn't work. In this case, you can import `re` in every worker thread:

```
class TestRunner:
    def __init__(self):
        import re
```

```
def __call__(self):
    # __call__ is executed by worker threads
    p = re.compile('a.b', re.IGNORECASE)
    # use p
```

Or equivalently:

```
from net.grinder.script.Grinder import grinder

def getRE():
    import re
    return re.compile('a.b', re.IGNORECASE)

class TestRunner:
    def __call__(self):
        # __call__ is executed by worker threads
        p = getRE()
        # use p
```

3.2.8. Why do I get syntax errors when editing my Jython scripts, even when the code looks correct?

Python and Jython use indentation to delimit blocks. You need to indent your script consistently.

Check out these links for more information:

http://diveintopython.org/getting_to_know_python/indenting_code.html
<http://www.python.org/doc/current/ref/indentation.html>

3.2.9. Whenever I try to run scripts I get "the script file 'grinder.py' does not exist or is not readable"?

You need to define which script should be used. If you do not define it the default script name is "grinder.py".

If your script is not called "grinder.py", either add a line to grinder.properties that specifies its name, e.g.:

```
grinder.script: http.py
```

or specify it on the command line, e.g.:

```
java -Dgrinder.script=http.py net.grinder.Grinder
```

or set it in the console. Go to the script tab, navigate to the location of your script. Select it and then click on the "Set script to run" button.

Finally, it could simply be that the script file is unreadable. Check the file permissions of the script and/or attempt to open it in an editor to check if its readable.

3.2.10. What does "java.lang.ClassFormatError: Invalid method Code length" mean?

The short answer is that a function in your script is too long and is tripping over a Java limitation on the length of a method inherited by Jython. Java methods are limited to 65535 characters or less.

You'll need to break the script up into smaller pieces. E.g. if you have a script that looks

like

```
#...
class TestRunner:
    def __call__(self):
        # Lots of code - more than 65535 characters
```

you might break it up into smaller methods (functions belonging to TestRunner) as follows:

```
#...
class TestRunner:
    def part1(self):
        # some of the code - less than 65535 characters

    def part2(self):
        # more of the code

    def part3(self):
        # the rest of the code

    def __call__(self):
        self.part1()
        self.part2()
        self.part3()
```

3.3. SSL

3.3.1. If I have several suitable certificates in my keystore, how does The Grinder chose between them?

The Grinder relies on the JVM's default `KeyManager` implementations. This picks a certificate from the store based on SSL negotiation with the server. If there are several suitable certificates, the only way to control which is used is to [provide your own KeyManager](http://grinder.sourceforge.net/g3/ssl-support.html#own-key-manager) (<http://grinder.sourceforge.net/g3/ssl-support.html#own-key-manager>).

3.3.2. setKeyStoreFile has a parameter for the key store password. What about the pass phrase that protects the private key in the key store?

The pass phrases for keys must be the same as the key store password. This is a restriction of the default `KeyManagers`. If you don't like this, you can [provide your own KeyManager](http://grinder.sourceforge.net/g3/ssl-support.html#own-key-manager) (<http://grinder.sourceforge.net/g3/ssl-support.html#own-key-manager>).

3.3.3. Shouldn't I need to specify a set of certificates for trusted Certificate Authorities?

No. The Grinder does not validate certificates received from the server, so does not need a set of CA certificates.

3.3.4. Can I use the properties `javax.net.ssl.keyStore`, `javax.net.ssl.keyStoreType`, and `javax.net.ssl.keyStorePassword` to specify a global keystore?

No. The Grinder does not use these properties, primarily because the JSSE does not provide a way to access its default `SSLContext`.

3.4. The TCPProxy

3.4.1. What happened to the TCPSniffer?

Starting with The Grinder 3, the TCPSniffer was renamed to the TCPProxy to more correctly reflect its nature. The TCPProxy has also been significantly enhanced during the development of The Grinder 3.

3.4.2. My server works fine with a browser but has errors when I use the HTTP plugin?

This is probably down to one of the following:

- Bugs in your server code. Particularly, those due to the way it handles multiple concurrent requests. Don't be disheartened, you did good; finding bugs like this is a key reason to use The Grinder.
- Differences between the HTTP requests that the browser uses and those that The Grinder sends.

The HTTP plugin sends requests that will have minor differences between those that a browser send, but which rarely affect server behaviour. Its worth knowing how to examine the differences with the TCPProxy. First [set the TCPProxy as a browser proxy](http://grinder.sourceforge.net/g3/tcpproxy.html) (<http://grinder.sourceforge.net/g3/tcpproxy.html>) and record the output to a file. Secondly alter your tests so that all requests go via the TCPProxy (the best way is to set the test script connections to direct requests via the TCPProxy acting an HTTP proxy); again record the output to a file. Now grab a coffee and compare.

3.4.3. I'm getting java.security.InvalidKeyException: Illegal key size or default parameters

This is due to a mismatch of the SSL configuration of the server and that of the JDK you use to run The Grinder; most likely a clash between domestic and export strength key lengths. You should probably install the [Unlimited Strength Jurisdiction Policy Files](http://java.sun.com/javase/downloads/index.jsp) (<http://java.sun.com/javase/downloads/index.jsp>) for your Java version from Sun.

3.4.4. What are all those funny ^[[31 characters in the TCPProxy output?

Did you use the `-colour` switch? If so, the TCPProxy generates escape codes which work on ANSI compliant terminals and look very pretty.

For those of you on Windows platforms, this *doesn't work* with the `CMD . EXE` window unless you're using [Cygwin](http://cygwin.com/) (<http://cygwin.com/>) . (Of course, if you're using Cygwin you'd use `rxvt` in preference :-)).

3.4.5. How do I record a script for a server that uses SSL?

By default, the TCPProxy will use a built-in certificate to handle SSL traffic so you don't need to do anything extra, just record a script as normal.

Should you encounter problems with this, especially when using Internet Explorer to connect through the TCPProxy, you will need to provide a server certificate for The Grinder. The easiest way to provide a server certificate is to copy the `testkeys` file from the JSSE samples distribution and start the proxy using:

```
java net.grinder.TCPProxy -keyStore testkeys -keyStorePassword  
passphrase
```

You can specify your own keystore containing relevant client and root certificates. Tell the TCPProxy to use the keystore and ssl by using the following parameters when invoking the TCPProxy:

```
-keystore (keystore) -keystorepassword (password) -keystoretype (type)
```

Should you encounter further problems with your SSL connection you can turn on debug by using the following switch:

```
-Djavax.net.debug=ssl
```

Note:

This will generate verbose output.

3.5. The Console

3.5.1. Why is the test mean time for a page greater than the total mean?

The totals line only includes data from basic tests, i.e. tests that do not include other tests. It does not include the data from composite tests, i.e. tests that include other tests, such as those those wrapping pages in HTTP scripts generated by the TCPProxy. The time taken by these page tests is typically quite long since it also includes sleep time and can easily exceed the average time for basic tests.

Composite test lines in the Results tab have a grey background to emphasise that they are different.

This is done so that the totals line is meaningful, useful, and understandable. If data from composite tests were included in the totals line, some of the values (e.g. average test time) would be meaningless since time would be accounted for twice, once by the basic request tests, and once by the page test that wrapped the requests.

Data from composite tests is also not included in the totals line in the worker process output log files.