

Getting started

Table of contents

1 The Grinder processes.....	2
2 Tests and test scripts.....	3
3 Network communication.....	4
4 Output.....	4
5 How do I start The Grinder?.....	5

Note:

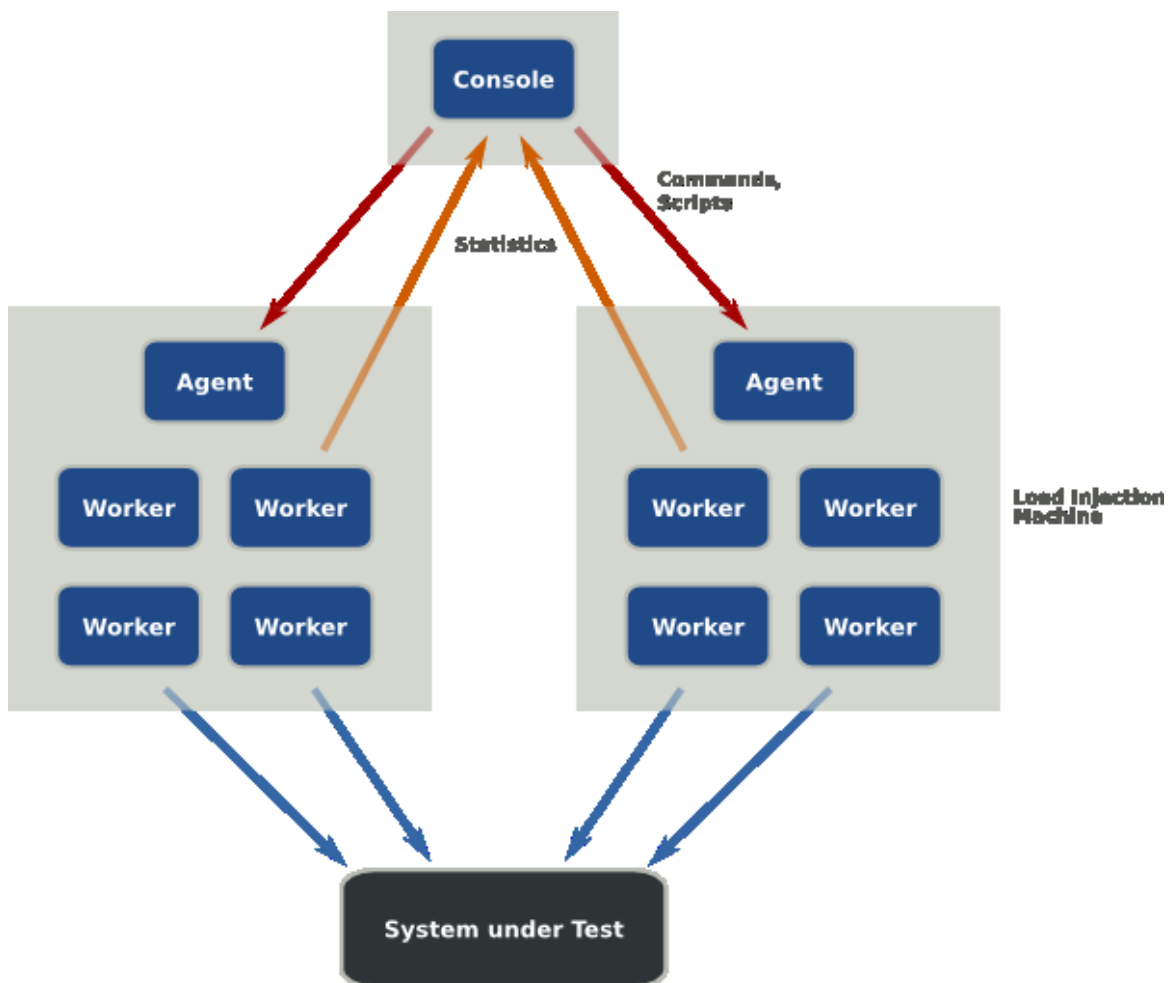
This section takes a top down approach to The Grinder. If you are happy figuring things out for yourself and want to get your hands dirty, you might like to read [How do I start The Grinder?](#) and then jump to the [Script Gallery](#) ([../g3/script-gallery.html](#)).

1. The Grinder processes

The Grinder is a framework for running test scripts across a number of machines. The framework is comprised of three types of *process* (or program): [worker processes](#) ([../g3/agents-and-workers.html#worker-processes](#)), [agent processes](#) ([../g3/agents-and-workers.html#agent-processes](#)), and the [console](#) ([../g3/console.html](#)). The responsibilities of each of the process types are:

- **Worker processes**
 - Interpret Jython test scripts and perform tests using a number of *worker threads*
- **Agent processes**
 - Manage worker processes
- **The console**
 - Coordinate the other processes
 - Collate and display statistics
 - Script editing and distribution

As The Grinder is written in Java, each of these processes is a Java Virtual Machine (JVM).



For heavy duty testing, you start an agent process on each of several load injector machines. The worker processes they launch can be controlled and monitored using the console. There is little reason to run more than one agent on each load injector, but you can if you wish.

2. Tests and test scripts

A *test* is a unit of work against which statistics are recorded. Tests are uniquely defined by a *test number* and also have a *description*. Users specify which tests to run using a Jython [test script](#) (`../g3/scripts.html`). If you wish your scripts can report many different actions (e.g. different web page requests) against the same test, The Grinder will aggregate the results.

The script is executed many times in a typical testing scenario. Each worker process has a number of worker threads, and each worker thread calls the script a number of times. A single execution of a test script is called a *run*.

You can write scripts for use with the Grinder by hand. There are a number of examples of how to do this in the [Script Gallery](#) (`../g3/script-gallery.html`). See the [Scripts](#) (`../g3/scripts.html`) section for more details on how to create scripts.

If you are creating a script to test a web site or web application, you can use the [TCPProxy](#) (`../g3/tcpproxy.html#HTTPPluginTCPProxyFilter`) to record a browser session

as a script.

3. Network communication

Each worker process sets up a network connection to the console to report statistics. Each agent process sets up a connection to the console to receive commands, which it passes on to its worker processes. The console listens for both types of connection on a particular address and port. By default, the console listens on port 6372 on all local network interfaces of the machine running the console.

If an agent process fails to connect to the console, or the `grinder.useConsole` property is `false`, the agent will continue independently without the console and automatically will start its worker processes. The worker processes will run to completion and not report to the console. This can be useful when you want to quickly try out a test script without bothering to start the console.

Note:

To change the console addresses, set the `grinder.consoleHost` and `grinder.consolePort` properties in the [grinder.properties](#) (`../g3/properties.html`) file before starting The Grinder agents. The values must match those specified in the console options dialog.

4. Output

Each worker process writes logging information to a file called `out-host-n.log`, where `host` is the machine host name and `n` is the worker process number. Errors are written to `error-host-n.log`. If no errors occur, an error file will not be created.

Data about individual test invocations is written into a file called `data-host-n.log` that can be imported into a spreadsheet tool such as Microsoft Excel™ for further analysis. The data file is the only place where information about individual tests is recorded; the console displays only aggregate information.

The final statistics summary (in the `out-*` files of each process) looks something like this:

Final statistics for this process:

	Successful Tests	Errors	Mean Test Time (ms)	Test Time Standard Deviation (ms)
Test 0	25	0	255.52	22.52
Test 1	25	0	213.40	25.15
Test 2	25	0	156.80	20.81
"Image"				
Test 3	25	0	90.48	14.41
Test 4	25	0	228.68	23.97
"Login page"				
Test 5	25	0	86.12	12.53
"Security check"				
Test 6	25	0	216.20	8.89
Test 7	25	0	73.20	12.83
Test 8	25	0	141.92	18.36

Test 9 "Logout page"	25	0	104.68	19.86
Totals	250	0	156.70	23.32

The console has a dynamic display of similar information collected from all the worker processes. Plug-ins and advanced test scripts can provide additional statistics; for example, the HTTP plug-in adds a statistic for the content length of the response body.

Each test has one of two possible outcomes:

1. Success. The number of *Successful Tests* for that test is incremented. The time taken to perform the test is added to the *Total*.
2. Error. The execution of a test raised an exception. The number of *Errors* for the test is incremented. The time taken is discarded.

The *Total*, *Mean*, and *Standard Deviation* figures are calculated based only on successful tests.

5. How do I start The Grinder?

Its easy:

1. Create a [grinder.properties](#) (./g3/properties.html) file. This file specifies general control information (how the worker processes should contact the console, how many worker processes to use, ..), as well as the name of the Jython script that will be used to run the tests.
2. Set your CLASSPATH to include the `grinder.jar` file which can be found in the `lib` directory.
3. Start the [console](#) (./g3/console.html) on one of the test machines:

```
java net.grinder.Console
```

4. For each test machine, do steps 1. and 2. and start an agent process:

```
java net.grinder.Grinder
```

The agent will look for the `grinder.properties` file in the local directory. The Jython script is usually stored alongside the properties file. If you like, you can also specify an explicit properties file as the first argument. For example:

```
java net.grinder.Grinder myproperties
```

The console does not read the `grinder.properties` file. It has its own options dialog (choose the *File/Options* menu option) which you should use to set the communication addresses and ports to match those in the `grinder.properties` files. The console [process controls](#) (./g3/console.html#process-controls) can be used to trigger The Grinder test scenario. Each agent process then creates child worker processes to do the work.

Note:

When you know a little more about the console, you can use it to [edit and distribute properties files and scripts](#) (./g3/console.html#Script+tab) instead of copying them to each agent machine.

As the worker processes execute, they dynamically inform the console of the tests in the test script. If you start the console after the agent process, you should press the *Reset processes* button. This will cause the existing worker processes to exit and the agent

process to start fresh worker processes which will update the console with the new test information.

Included below are some sample scripts, for both Unix/Linux and Windows, for starting grinder agents, the console, and the [TCPProxy](#) (`../g3/tcpproxy.html`) for recording HTTP scripts.

Windows

- `setGrinderEnv.cmd`:

```
set GRINDERPATH=(full path to grinder install directory)
set GRINDERPROPERTIES=(full path to
grinder.properties)\grinder.properties
set CLASSPATH=%GRINDERPATH%\lib\grinder.jar;%CLASSPATH%
set JAVA_HOME=(full path to java install directory)
PATH=%JAVA_HOME%\bin;%PATH%
```

- `startAgent.cmd`:

```
call (path to setGrinderEnv.cmd)\setGrinderEnv.cmd
echo %CLASSPATH%
java -cp %CLASSPATH% net.grinder.Grinder %GRINDERPROPERTIES%
```

- `startConsole.cmd`:

```
call (path to setGrinderEnv.cmd)\setGrinderEnv.cmd
java -cp %CLASSPATH% net.grinder.Console
```

- `startProxy.cmd`:

```
call (path to setGrinderEnv.cmd)\setGrinderEnv.cmd
java -cp %CLASSPATH% net.grinder.TCPProxy -console -http > grinder.py
```

Unix

- `setGrinderEnv.sh`:

```
#!/usr/bin/ksh
GRINDERPATH=(full path to grinder install directory)
GRINDERPROPERTIES=(full path to
grinder.properties)/grinder.properties
CLASSPATH=$GRINDERPATH/lib/grinder.jar:$CLASSPATH
JAVA_HOME=(full path to java install directory)
PATH=$JAVA_HOME/bin:$PATH
export CLASSPATH PATH GRINDERPROPERTIES
```

- `startAgent.sh`:

```
#!/usr/bin/ksh
. (path to setGrinderEnv.sh)/setGrinderEnv.sh
java -cp $CLASSPATH net.grinder.Grinder $GRINDERPROPERTIES
```

- `startConsole.sh`:

```
#!/usr/bin/ksh
. (path to setGrinderEnv.sh)/setGrinderEnv.sh
java -cp $CLASSPATH net.grinder.Console
```

- `startProxy.sh`:

```
#!/usr/bin/ksh
. (path to setGrinderEnv.sh)/setGrinderEnv.sh
java -cp $CLASSPATH net.grinder.TCPProxy -console -http > grinder.py
```