The Grinder 3

Table of contents

1 Project... 5
1.1 The Grinder, a Java Load Testing Framework... 5
1.1.1 What is The Grinder?... 5
1.1.2 Authors... 6
1.1.3 Credits... 6
1.2 The Grinder License... 6
1.2.1 The Grinder... 6
1.2.2 HTTPClient... 7
1.2.3 Jython... 7
1.2.4 jEdit Syntax... 7
1.2.5 Apache XMLBeans... 7
1.2.6 PicoContainer... 8
1.2.7 ASM... 8
1.2.8 JSR 166y... 8
1.2.9 SLF4J... 8

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

1.2.10 Logback........8
1.2.11 Clojure........8
1.2.12 Ring........8
1.2.13 Compojure........8
1.2.14 ring-middleware-format........8
1.2.15 Jetty........8
1.2.16 Clojure tools.logging........9
1.2.17 Supporting license text........9
1.3 Downloading The Grinder........10
1.3.1 Download........10
1.3.2 Downloading The Grinder using Maven........10
1.4 Support........11
1.4.1 Mailing lists........11
1.5 External references........11
1.5.1 Related Software Projects........11
1.5.2 Articles........15
1.5.3 Commercials........16
2 The Grinder 3........18
2.1 Getting started........18
2.1.1 The Grinder processes........18
2.1.2 Tests and test scripts........20
2.1.3 Network communication........21
2.1.4 Output........21
2.1.5 How do I start The Grinder?........22
2.2 Agents and Workers........24
2.2.1 Agents and Workers........24
2.2.2 The Grinder 3 Properties File........24
2.2.3 Logging........28
2.3 The Console........30
2.3.1 The Console User Interface........30
2.3.2 The Console Service........36
2.4 The TCPProxy........43
2.4.1 Starting the TCPProxy........44
2.4.2 Preparing the Browser........44
2.4.3 Using the EchoFilter........46
2.4.4 Using the HTTP TCPProxy filters........47

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

2.4.5 SSL and HTTPS support..51
2.4.6 Using the TCPProxy with other proxies..53
2.4.7 Using the TCPProxy as a port forwarder..53
2.4.8 Summary of TCPProxy options..54
2.5 Scripts..55
2.5.1 Scripts..55
2.5.2 Jython..59
2.5.3 Clojure..60
2.5.4 Script Instrumentation..61
2.5.5 Coordination..63
2.5.6 Script Gallery..64
2.6 Plug-ins..80
2.6.1 The HTTP Plug-in..80
2.7 Statistics..85
2.7.1 Standard statistics..85
2.7.2 Distribution of statistics..85
2.7.3 Querying and updating statistics..85
2.7.4 Registering new expressions..86
2.8 SSL Support..86
2.8.1 Before we begin..86
2.8.2 Controlling when new SSL sessions are created..87
2.8.3 Using client certificates..87
2.8.4 FAQ..88
2.8.5 Picking a certificate from a key store [Advanced]..88
2.8.6 Debugging..89
2.9 Advice..89
2.9.1 How should I set up a project structure for The Grinder?..89
2.9.2 A Step-By-Step Script Tutorial..91
2.9.3 Weighted Distribution Of Tests..94
2.9.4 Garbage Collection..98
2.10 Features of The Grinder 3..100
2.10.1 Capabilities of The Grinder..100
2.10.2 Open Source..100
2.10.3 Standards..100
2.10.4 The Grinder Architecture..101
2.10.5 Console..101

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

2.10.6 Statistics, Reports, Charts....................101
2.10.7 Script.................... 102
2.10.8 The Grinder Plug-ins....................102
2.10.9 HTTP Plug-in....................102
2.10.10 TCP Proxy....................103
2.10.11 Documentation....................103
2.10.12 Support.................... 103

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

1 Project

1.1 The Grinder, a Java Load Testing Framework

1.1.1 What is The Grinder?

The Grinder is a Java^{TM} load testing framework that makes it easy to run a distributed test using many load injector machines. It is freely available under a BSD-style open-source license (../license.html) .

The latest news, downloads, and mailing list archives can be found on SourceForge.net (https://www.sourceforge.net/projects/grinder) .

1.1.1.1 Key features

- **Generic Approach** Load test anything that has a Java API. This includes common cases such as HTTP web servers, SOAP and REST web services, and application servers (CORBA, RMI, JMS, EJBs), as well as custom protocols.
- **Flexible Scripting** Test scripts are written in the powerful Jython (http://www.jython.org/) and Clojure (http://clojure.org/) languages.
- **Distributed Framework** A graphical console allows multiple load injectors to be monitored and controlled, and provides centralised script editing and distribution.
- **Mature HTTP Support** Automatic management of client connections and cookies. SSL. Proxy aware. Connection throttling. Sophisticated record and replay of the interaction between a browser and a web site.

See the longer features list (../g3/features.html) for further details.

1.1.1.2 Dynamic Scripting

Test scripts are written using a dynamic scripting language, and specify the tests to run. The default script language is Jython (http://www.jython.org/) , a Java implementation of the popular Python language.

The script languages provide the following capabilities:

Test any Java code
The Grinder 3 allows any code (Java, Jython, or Clojure) code to be encapsulated as a test. Java libraries available for an enormous variety of systems and protocols, and they can all be exercised using The Grinder.

Dynamic test scripting
The Grinder 2 worker processes execute tests sequentially in a fixed order, and there is limited support in some of the The Grinder 2 plug-ins for checking test results. The Grinder 3 allows arbitrary branching and looping and makes test results directly available to the test script, allowing different test paths to be taken depending on the outcome of each test.

The Grinder 2 HTTP plug-in's string bean (../g2/http-plugin.html#string-bean) feature provides simple support for requests that contain dynamic data. The Grinder 3 can use the full power of Jython or Clojure to create dynamic requests of arbitrary complexity.

The powerful scripting removes the need to write custom plug-ins that extend The Grinder engine. Although plug-ins are no longer responsible for performing tests, they can still be useful to manage objects that the tests use. For example, the standard

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

HTTP plug-in manages a pool of connections for each worker thread, and provides an `HTTPRequest` object that makes use of these connections.

Kind of dry, huh? If you never seen any Python, take a look at the Script Gallery (../g3/script-gallery.html) in the user manual where you can sample the power of The Grinder 3.

1.1.1.3 History

The Grinder was originally developed for the book *Professional Java 2 Enterprise Edition with BEA WebLogic Server* by Paco Gómez and Peter Zadrozny. Philip Aston took ownership of the code, reworked it to create The Grinder 2, and shortly after began work on The Grinder 3. The Grinder 3 provides many new features, the most significant of which is dynamic test scripting. Philip continues to enhance and maintain The Grinder.

In 2003, Peter, Philip and Ted Osborne published the book J2EE Performance Testing (../links.html#book) which makes extensive use of The Grinder 2.

Support for Clojure (http://clojure.org/) as an alternative script language was introduced in 3.6.

1.1.2 Authors

Over the years, many individuals (../../mvn-site/team-list.html) have contributed features, bug fixes, and translations to The Grinder.

1.1.3 Credits

I thank Paco Gómez and Peter Zadrozny for the key ideas embodied in the original version of The Grinder.

I am grateful to SourceForge, Inc. (http://www.sourceforge.com/) for The Grinder's home on the Internet.

I thank Atlassian (http://www.atlassian.com/) for the free Clover (http://www.atlassian.com/clover) and FishEye (http://fisheye3.cenqua.com/browse/grinder/) licenses, and to Headway Software (http://www.headwaysoftware.com/) for the free Structure 101 (http://www.headwaysoftware.com/products/structure101) license.

This site is built with Apache Forrest (http://forrest.apache.org/) , and uses SyntaxHighlighter (http://alexgorbatchev.com/SyntaxHighlighter/) .

Philip Aston

1.2 The Grinder License

The Grinder is free software. It also repackages other free software. This section explains what you can and cannot do with The Grinder and the software included with it.

1.2.1 The Grinder

Copyright (c) 2000 Paco Gómez
Copyright (c) 2000-2012 Philip Aston
All rights reserved.

Additional contributions have been made by individuals listed in the AUTHORS file supplied with this distribution. Each individual's claim to copyright is asserted in the files to which they contributed.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the names of the copyright holders nor the names of the contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1.2.2 HTTPClient

The Grinder includes Ronald Tschalär's HTTPClient library (http://www.innovation.ch/java/HTTPClient/index.html (http://www.innovation.ch/java/HTTPClient/)). The HTTPClient library is distributed under the GNU Lesser Public License 2.1 (http://www.opensource.org/licenses/lgpl-2.1.php) . Under the term 6 of the GNU Lesser Public License, The Grinder is a "work that uses the Library".

1.2.3 Jython

The Grinder includes the software Jython, created by Jim Hugunin, Barry Warsaw and the Jython team (http://www.jython.org/). This is distributed under the terms of the Jython and JPython software licenses (http://www.jython.org/license.html) .

1.2.4 jEdit Syntax

The Grinder includes the jEdit Syntax highlighting package (http://syntax.jedit.org/). This is distributed according to the jEdit Syntax copyright and usage statement.

1.2.5 Apache XMLBeans

The Grinder includes Apache XMLBeans (http://xmlbeans.apache.org/), under the terms of the Apache Software License Version 2.0. See the XMLBeans NOTICE.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

1.2.6 PicoContainer

The Grinder includes PicoContainer (http://picocontainer.codehaus.org/ (http://picocontainer.org/)). This is distributed under the terms of the PicoContainer license.

1.2.7 ASM

The Grinder includes ASM (http://asm.objectweb.org/ (http://picocontainer.org/)). This is distributed under the terms of the ASM license (http://asm.ow2.org/license.html) .

1.2.8 JSR 166y

The Grinder includes components from the extra166y package. This package is in the public domain. See http://g.oswego.edu/dl/concurrency-interest/ (http://g.oswego.edu/dl/concurrency-interest/) .

1.2.9 SLF4J

The Grinder includes SLF4J (http://www.slf4j.org/), under the terms of the SLF4J license (http://www.slf4j.org/license.html) .

1.2.10 Logback

The Grinder includes Logback (http://logback.qos.ch/), under the terms of the Eclipse Public License, Version 1.0 (http://www.opensource.org/licenses/lgpl-2.1.php) .

1.2.11 Clojure

The Grinder includes Clojure (http://clojure.org/), under the terms of the Eclipse Public License, Version 1.0 (http://www.eclipse.org/legal/epl-v10.html) .

1.2.12 Ring

The Grinder includes Ring (https://github.com/mmcgrana/ring), under the terms of the Ring license (https://github.com/mmcgrana/ring/blob/master/LICENSE) .

1.2.13 Compojure

The Grinder includes Compojure (https://github.com/weavejester/compojure), under the terms of the Eclipse Public License, Version 1.0 (http://www.eclipse.org/legal/epl-v10.html) .

1.2.14 ring-middleware-format

The Grinder includes ring-middleware-format (https://github.com/ngrunwald/ring-middleware-format (https://github.com/ngrunwald/ring-middleware-format)), under the terms of the Eclipse Public License, Version 1.0 (http://www.eclipse.org/legal/epl-v10.html) .

1.2.15 Jetty

The Grinder includes Jetty (https://github.com/weavejester/compojure), under the terms of the Eclipse Public License, Version 1.0 (http://www.eclipse.org/legal/epl-v10.html) , with the exceptions explained in the NOTICE (http://dev.eclipse.org/svnroot/rt/org.eclipse.jetty/jetty/trunk/NOTICE.txt) file.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

1.2.16 Clojure tools.logging

The Grinder includes Clojure tools.logging (https://github.com/clojure/tools.logging), under the terms of the Eclipse Public License, Version 1.0 (http://www.eclipse.org/legal/epl-v10.html) .

1.2.17 Supporting license text

Most licenses have been referred to above by linking to external sites. A copy of the full text of each license can be found in The Grinder distribution.

1.2.17.1 jEdit Syntax copyright and usage statement

```
The jEdit 2.2.1 syntax highlighting package contains code that is
Copyright 1998-1999 Slava Pestov, Artur Biesiadowski, Clancy Malcolm,
Jonathan Revusky, Juha Lindfors and Mike Dillon.

You may use and modify this package for any purpose. Redistribution is
permitted, in both source and binary form, provided that this notice
remains intact in all source distributions of this package.

-- Slava Pestov
25 September 2000
<sp@gjt.org>
```

1.2.17.2 XMLBeans NOTICE

```
=========================================================================
==  NOTICE file corresponding to section 4(d) of the Apache License,   ==
==  Version 2.0, in this case for the Apache XmlBeans distribution.    ==
=========================================================================

This product includes software developed by The Apache Software
Foundation (http://www.apache.org/).

Portions of this software were originally based on the following:
 - software copyright (c) 2000-2003, BEA Systems,
   <http://www.bea.com/>.

Aside from contributions to the Apache XMLBeans project, this software
also includes:

 - one or more source files from the Apache Xerces-J and Apache Axis
   products, Copyright (c) 1999-2003 Apache Software Foundation

 - W3C XML Schema documents Copyright 2001-2003 (c) World Wide Web
   Consortium (Massachusetts Institute of Technology, European
   Research Consortium for Informatics and Mathematics, Keio
   University)

 - Piccolo XML Parser for Java from http://piccolo.sourceforge.net/,
   Copyright 2002 Yuval Oren under the terms of the Apache Software
   License 2.0

 - JSR-173 Streaming API for XML from
   http://sourceforge.net/projects/xmlpullparser/,
   Copyright 2005 BEA under the terms of the Apache Software
   License 2.0
```

1.2.17.3 PicoContainer License

```
Copyright (c) 2003-2005, PicoContainer Organization
All rights reserved.
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

  Redistributions of source code must retain the above copyright notice, this
  list of conditions and the following disclaimer.

  Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.

  Neither the name of the PicoContainer Organization nor the names of its
  contributors may be used to endorse or promote products derived from this
  software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
```

1.3 Downloading The Grinder

1.3.1 Download

The Grinder can be downloaded from SourceForge.net (https://www.sourceforge.net/projects/grinder) . New users are advised (../faq.html#g2vsg3) to start with The Grinder 3. The source code (../development/contributing.html#source) is also available.

The Grinder 3 is distributed as two zip files which you should expand using `unzip`, WinZip (http://www.winzip.com/) or similar. Everything required to run The Grinder is in the zip file labelled `grinder-`*version*`.zip`. The remaining files that are needed to build The Grinder are distributed in the zip file labelled `grinder-`*version*`-src.zip`; these are mainly of interest to developers wanting to extend The Grinder (../development/contributing.html) .

1.3.1.1 What else do I need?

To run The Grinder:

Java Standard Edition 6 (http://www.oracle.com/technetwork/java/javase/downloads/index.html) , equivalent, or later	For The Grinder 3.
Java 2 Standard Edition 1.3 (http://www.oracle.com/technetwork/java/javase/downloads/index.html) , equivalent, or later	For The Grinder 2.
JSSE (Java Secure Socket Extension) 1.0.2 (http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136007.html)	For SSL support with The Grinder 2. JSSE is a standard part of Java 2 Standard Edition 1.4.1 and later, so this extension is not requried for The Grinder 3.

1.3.2 Downloading The Grinder using Maven

Some users will find it preferable to use Maven to manage The Grinder. On release, the jar files are deployed to the Sonatype (http://oss.sonatype.org/) OSS Nexus repository,

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

and will be synchronised to Maven Central (http://search.maven.org) soon afterwards. You can choose either to depend on the the zip file, which should be identical to the Sourceforge download, or the individual jar files.

1.4 Support

1.4.1 Mailing lists

Requests for help should be sent to grinder-use@lists.sourceforge.net (mailto:grinder-use@lists.sourceforge.net) .

> Note:
> To reduce spam, **you must subscribe (https://www.sourceforge.net/p/grinder/mailman/) to a list before you can send email to it**. The email address you send mail from must be the one you used to subscribe.

grinder-announce@lists.sourceforge.net (mailto:grinder-announce@lists.sourceforge.net) is a low-volume mailing list which is used to announce new releases and other items of interest to users of The Grinder.

Please contribute bug fixes and enhancements (../development/contributing.html) to grinder-development@lists.sourceforge.net (mailto:grinder-development@lists.sourceforge.net) .

You can subscribe and unsubscribe (https://www.sourceforge.net/p/grinder/mailman/) to the lists, and search their archives, through SourceForge.net (https://www.sourceforge.net/projects/grinder) . Gmane (http://gmane.org/find.php?list=grinder) provides alternative searchable archives, together with NNTP feeds and an optional Blog-like interface.

When Philip Aston finds the time to respond to mail about The Grinder, **messages not copied to one of the above mail lists are likely to be ignored**. Philip freely copies responses to the lists; if there is a particular reason why you want to keep your communication private you must say so. Finally, if you can provide answers to questions sent to the lists, please don't be shy!

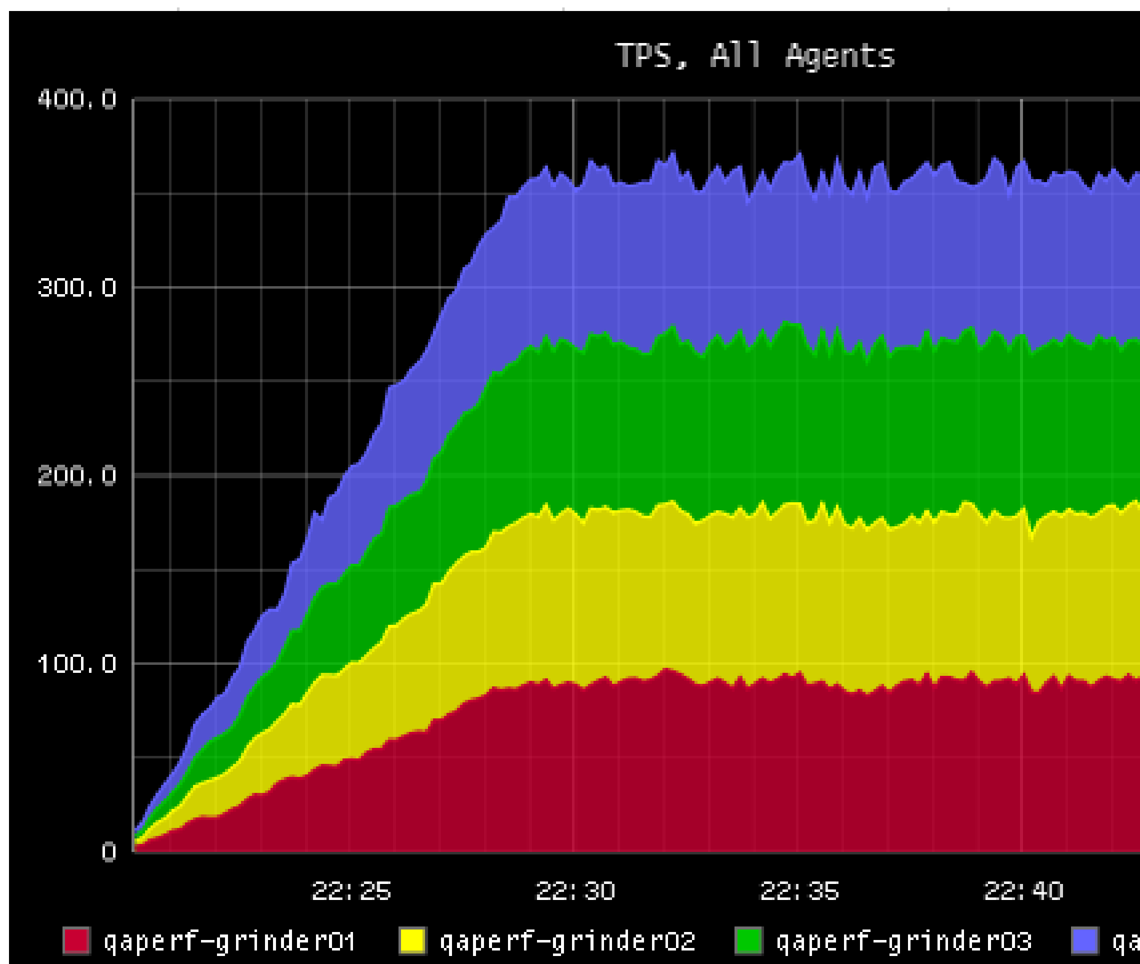
> Note:
> The Grinder is free software. No one is under any obligation to fix your problem. If all else fails, you have the source.

1.5 External references

1.5.1 Related Software Projects

Another Groovy script engine (http://www.cubrid.org/wiki_ngrinder/entry/groovy-script)		A second Groovy script engine from the nGrinder (http://www.nhnopensource.org/ngrinder) team.
Grinder to Graphite (http://grinder-to-graphite.readthedocs.org/en/latest/)		Grinder to Graphite (g2g) is a tool that analyzes the logs from your Grinder tests, and sends the data into Graphite where it can be visualized in a variety of ways.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

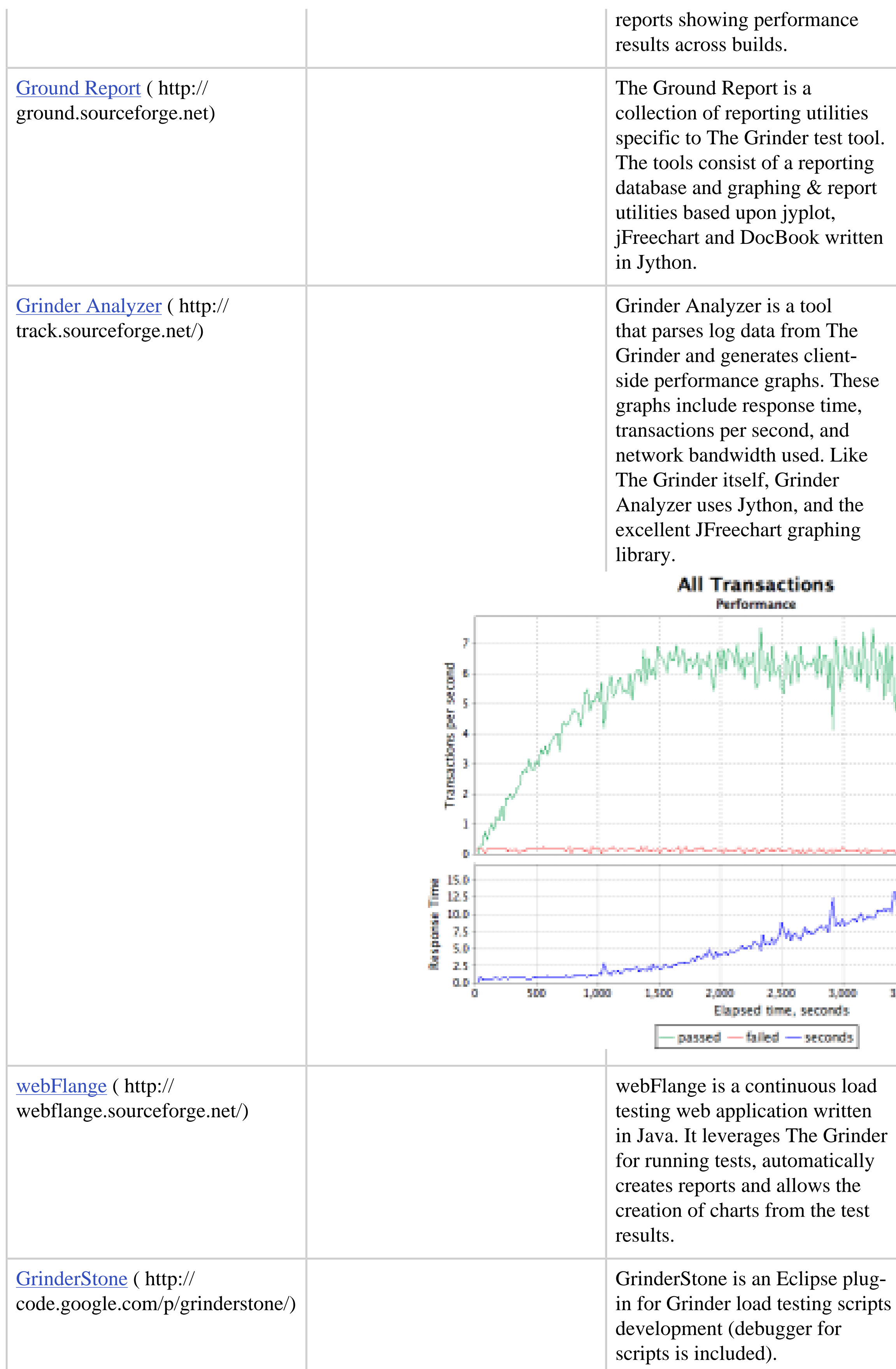


grinder-groovy (https:// github.com/DealerDotCom/ grinder-groovy)		Alternative script engine for The Grinder: write your test scripts in Groovy.
Grinder maven plugin (http:// code.google.com/p/grinder-maven-plugin)		A Maven Plugin for The Grinder, with Grinder Analyzer integration. The plugin allows you to run The Grinder from a Maven build, and analyse the results.
nGrinder (http:// www.nhnopensource.org/ ngrinder)		A web based testing framework built on top of The Grinder. The demo video is particularly slick.
Grinder Webtest (http:// www.automation-excellence.com/software/grinder-webtest)		This custom module allows execution of Visual Studio *webtest* files. It supports parameterization, capturing of variables in HTTP responses, and response validation using regular expressions. Test scripts may be logically grouped into test sets, allowing them to share variables and captured values. Test sets can be run sequentially, randomly, in a specific thread, or according to a percentage-

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

		based weighting. A correlating test runner is also provided, making it easier for you to find and capture values in your HTTP responses. We have successfully used this module to run load tests of more than 300 virtual users, with a scenario involving 21 different webtest scripts recorded in Fiddler.
HTTP Quality Assurance Toolkit (http://http-qat.sf.net)		HTTP functional and non-functional (load and performance) toolkit based on jython/grinder (http://grinder.sf.net) ...includes capabilities to support: SOA services, REST, json/xml encoding, AES and WS security ... and a stub to collect requests.
The Grinder Agent Installer (http://clinker.klicap.es/projects/demeter)		The Grinder Agent Installer is useful when you want to execute load tests in a heterogeneous context, and to simulate real users accessing the target application through a firewall, 3G, VPN, direct (router) and from different locales, where you haven't access to all this computers for run the agent. It provides an installer any user can install and execute with only click next and provisioning some data in a graphic environment. You only have to wait the connections in the console.
Grinder In The Cloud (http://developer.amazonwebservices.com connect/entry.jspa?externalID=2055&categoryID=101		Grinder in the Cloud leverages the well known Grinder load test framework by putting it in the cloud. It offers an easy to use load test framework with virtually unlimited firepower at a competitive price. This Windows based AMI starts the Grinder console. It starts Grinder agent AMIs to generate the load. The Agents automatically connect to the console. Built by Jörg Kalsbach.
Grinder Plugin for Hudson (http://hudson.gotdns.com/wiki/display/HUDSON/Grinder+Plugin)		This plug-in reads output result files from performance tests run with The Grinder, and will generate reports showing test results for every build and trend

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

		reports showing performance results across builds.
Ground Report (http:// ground.sourceforge.net)		The Ground Report is a collection of reporting utilities specific to The Grinder test tool. The tools consist of a reporting database and graphing & report utilities based upon jyplot, jFreechart and DocBook written in Jython.
Grinder Analyzer (http:// track.sourceforge.net/)		Grinder Analyzer is a tool that parses log data from The Grinder and generates client-side performance graphs. These graphs include response time, transactions per second, and network bandwidth used. Like The Grinder itself, Grinder Analyzer uses Jython, and the excellent JFreechart graphing library. 
webFlange (http:// webflange.sourceforge.net/)		webFlange is a continuous load testing web application written in Java. It leverages The Grinder for running tests, automatically creates reports and allows the creation of charts from the test results.
GrinderStone (http:// code.google.com/p/grinderstone/)		GrinderStone is an Eclipse plug-in for Grinder load testing scripts development (debugger for scripts is included).

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

1.5.2 Articles

The Agile Grind (http://agilegrind.blogspot.co.uk/)	Gary Mulder's blog on using The Grinder.
Black Anvil: Visualizing Grinder Data With Other External Metrics (http://blackanvil.blogspot.com/2011/12/integrating-grinder-performance-data.html)	Using Graphite to visualise the test results.
Rough Book (http://vivin.net/tag/the-grinder/)	A series of in-depth blog entries that introduce The Grinder and present a rich framework for scripts.
PerformanceEngineer.com: Introduction To The Grinder (http://www.performanceengineer.com/blog/introduction-to-the-grinder/)	An introductory blog entry showing how to set up The Grinder with GrinderStone (http://code.google.com/p/grinderstone/) .
PC Pro article (http://www.pcpro.co.uk/features/230550/technology-you-can-bet-on/page3.html)	"Technology you can bet on" - Paddy Power uses The Grinder.
InfoQ News (http://www.infoq.com/news/2008/02/the-grinder-3)	Alexander Olaru interviews Philip Aston for InfoQ.
Pure Danger Tech: The Grinder 3.0 Released (http://tech.puredanger.com/2008/01/25/the-grinder-30-released/)	Alex Miller says some very nice things about The Grinder. *"...I was really pleasantly surprised by everything that I found. The Grinder has a fairly clean aesthetic that is hard to quantify but makes getting started a pleasant experience. What I found the most enjoyable about it was the use of Jython to script the actual test activity. ...It is trivial to start up the console and your agents, then have very fast modify / run cycles as nothing needs to be restarted. You just modify the test in your editor and hit play on the console. This allows you to very rapidly whip your test into shape. Kind of reminds me of Rails..."*
The Black Anvil: Shootout: Load Runner vs The Grinder vs Apache JMeter (http://blackanvil.blogspot.com/2006/06/shootout-load-runner-vs-grinder-vs.html#links)	Detailed comparison of The Grinder, JMeter, and Load Runner from Travis Bear. *"...I recommended The Grinder as the tool to go forward with. It has a simple, clean UI that clearly shows what is going on without trying to do too much, and offers great power and simplicity with its unique Jython-based scripting approach. Jython allows complex scripts to be developed much more rapidly than in more formal languages like Java, yet it can access any Java library or class easily, allowing us to re-use elements of our existing work."* Travis has since assisted with the implementation of slow socket support for The Grinder.
Performance Testing using The Grinder (http://cdjdn.com/downloads/performancetesting-grinder.pdf)	A high-level overview of test methodology using The Grinder from Paul Evans/Blue Slate Solutions. Hosted by the Capital District Java Developers Network.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

Grinder Test Automation for the WebLogic Server (http://www.anser-e.com/testing/GrinderAutomationTutorial.html)	An custom automated test environment for WebLogic built on The Grinder.
Gash: Load Testing Java Applications (http://gashalot.com/writing/blog-grinder.php)	Replacing JMeter with The Grinder 3 *"I went from a freshly downloaded tarball to fully functional test environment in about 2.5 hours. That's powerful."*
WikiWikiWeb (http://c2.com/cgi/wiki/wiki?TheGrinder)	Entry on the Wiki of Wiki's.
Stress Testing with The Grinder and Cactus (http://www.abcseo.com/papers/grinder.htm)	Using The Grinder 2's JUnit plug-in with Cactus.
The Grinder: Load Testing for Everyone (http://dev2dev.bea.com/articles/aston.jsp)	An introductory article on The Grinder 2 from Phil Aston.
Anticlue (http://www.anticlue.net/archives/000395.htm)	Blog entry on The Grinder 3.
Load Testing Web Services with Grinder (http://www.oreillynet.com/pub/wlg/6743)	An article on testing Web Services with The Grinder 3.
Massive Propeller: The Grinder (http://www.massivepropeller.com/users/austin/blogs/whatsnew/archive/000043.html)	Blog entry on The Grinder 2.
Mr Worm's GonePage: The Grinder (http://82.133.140.67/MrWorm/35)	Blog entry on The Grinder.
Dan Moore!: The Grinder (http://www.mooreds.com/weblog/archives/000111.html)	Blog entry on The Grinder 3.

1.5.3 Commercials

This section contains links to commercial products and services related to The Grinder. You should not assume any relationship other than those documented below between the listed individuals and companies and The Grinder project. If you have a product or service related to The Grinder and would like to add information to this page, please email details to grinder-use (mailto:grinder-use@lists.sourceforge.net) .

1.5.3.1 Synoty

Performance has become a critical part of product development these days. The need for speed is here, users expect faster and responsive applications. At Synoty we realized a consolidated performance service which enables our customers to provide great applications to their users is needed.

Synoty is an application performance consulting service with a difference. Our service provides our customers with cloud based or inside firewall load and performance testing, user experience testing and performance engineering such as application code, database and operating system performance analysis and tuning.

We also included tools and software needed for performance analysis and an amazing performance portal to bring it all in one place. The Grinder is our load generator in the cloud.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

To learn how we can help you with performance please visit us at www.synoty.com (http://www.synoty.com) .

1.5.3.2 Perfmetrix

Perfmetrix is a global group of highly skilled and experienced system architects and performance experts ready to assist you with a comprehensive range of services to create or improve software applications that meet or exceed your business needs. We have presence in the United States, Europe, the Middle East, Africa and Latin America.

Perfmetrix is led by Peter Zadrozny, who was the Chief Technologist of BEA Systems for Europe, Middle East and Africa, a role he had since he started the operations of WebLogic in Europe (prior to the BEA acquisition).

Peter is the author of J2EE Performance Testing (../links.html#book) (Expert Press, 2002), coauthor of "Professional J2EE Programming with BEA WebLogic Server" (WroxPress, 2000) and "Beginning EJB 3 Application Development" (Apress 2006). He is the founding editor of the WebLogic Developer's Journal, and a frequent speaker on technology issues around the world. Peter was also part of the team that created The Grinder.

Peter Zadrozny, Perfmetrix (http://www.perfmetrix.com)

1.5.3.3 Anser Enterprise

One of my consulting services is helping performance analysts to set up company-internal blogs on their performance activities to help them communicate better with their developers and management. As part of my consulting service I can offer usage and customization tips on The Grinder and a separate data visualization tool to show Grinder test results on their company's intranet. Much of this is in the area of test automation and mining test results.

Here's a link (http://www.anser-e.com/run6/Run6a.html) which provides several example web pages on communicating WebLogic 8.1/Grinder testing results. It requires downloading the Java 1.5 plug-in for charting Grinder test results.

Todd Nichols, Anser Enterprise (http://www.anser-e.com/)

1.5.3.4 TestPros

TestPros provides load testing and performance tuning services using Grinder. We can provide our services in one or a combination of three ways - remotely via our Internet server farm, at our test labs, or at our customer's location.

For more information:

- 1-877-783-7855
- info@TestPros.com (mailto:info@TestPros.com)
- www.TestPros.com (http://www.TestPros.com/)

1.5.3.5 swtest-discuss

I run a mailing list for software testers called swtest-discuss (http://lists.topica.com/lists/swtest-discuss/) . There are a few people there (including me) who are interested in talking about how people do testing for open source projects. I haven't yet found a community of open source testers that cuts across multiple tools/applications.

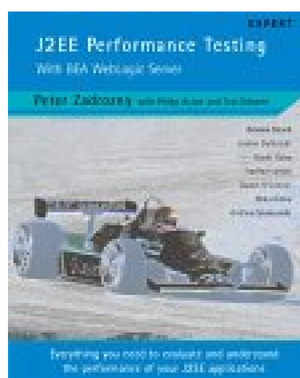
Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

If you're interested in sharing your experiences in testing open source software, please consider joining swtest-discuss, at least long enough to see if there's any interest in having an on-going forum on this topic. If you do subscribe, please either send me a private email or introduce yourself to the list so we know you're there.

Danny R. Faught, Tejas Software Consulting (http://tejasconsulting.com/)

1.5.3.6 J2EE Performance Testing

I'm pleased to announce the availability of *J2EE Performance Testing with BEA WebLogic Server (http://www.amazon.com/exec/obidos/tg/detail/-/159059181X/qid=1064753861/sr=1-1/ref=sr_1_1/002-5481898-2224815)* by Peter Zadrozny, Philip Aston and Ted Osborne, originally published by Expert Press and now by APress.



This book uses The Grinder 2 throughout, and indeed was responsible for driving the development of many of The Grinder's features. The book shows how to performance test complete J2EE applications and how to explore key performance issues surrounding the most popular J2EE APIs. The performance tests are carried out using BEA WebLogic Server™, but are generally applicable to any J2EE application server.

Most importantly, the book contains in-depth coverage of The Grinder 2 including a full user guide and case studies showing how to apply The Grinder to real world problems. The testing approach is equally applicable when using The Grinder 3.

Following several requests, I've made the source code for the book available from The Grinder SourceForge page (https://www.sourceforge.net/projects/grinder) . This source is supplied unsupported and with no warranty.

Philip Aston

2 The Grinder 3

2.1 Getting started

> **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) .

2.1.1 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-*

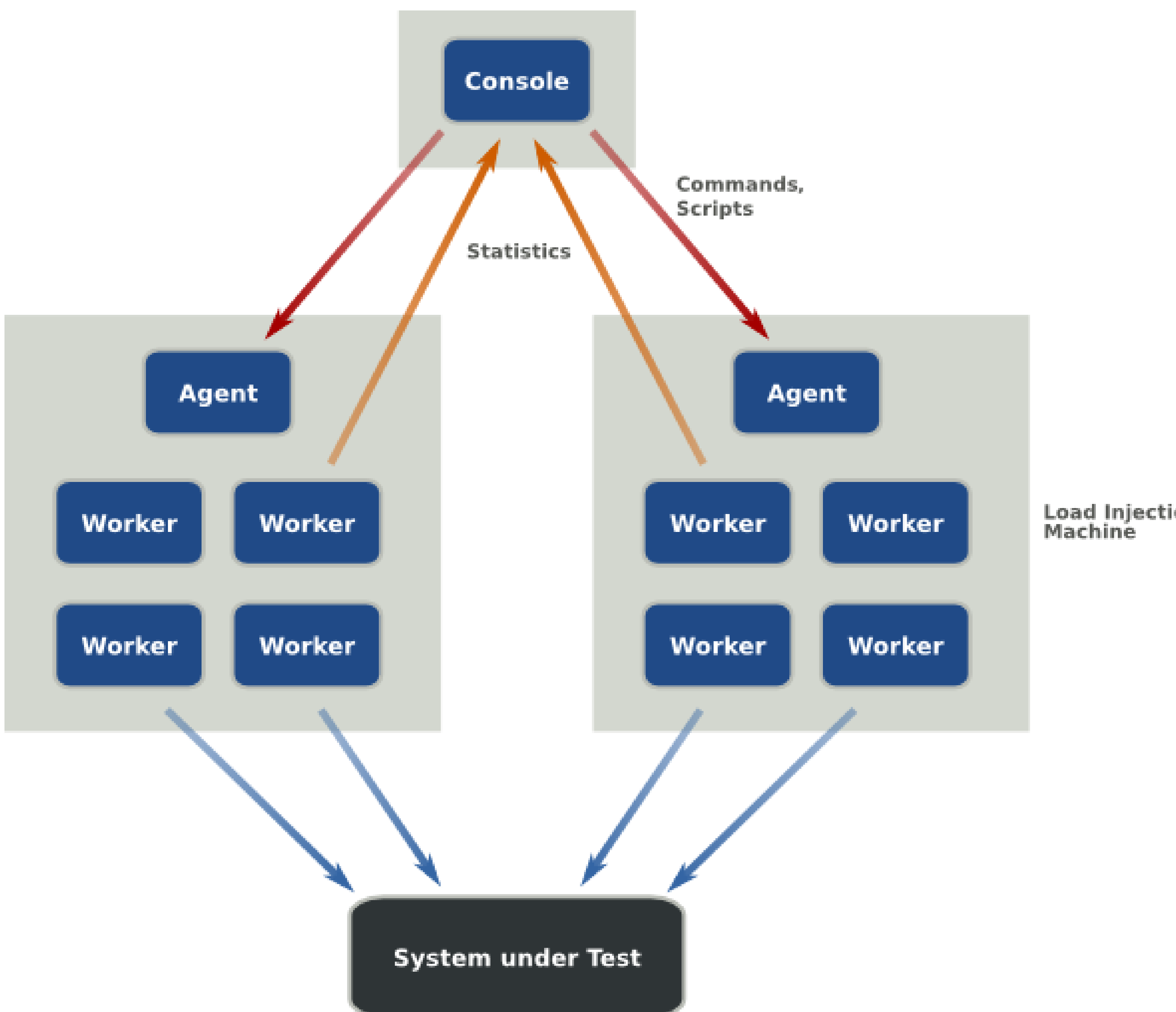
Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

and-workers.html#agent-processes) , and the console *(../g2/console.html)* . The responsibilities of each of the process types are:

- **Worker processes**
  - Interprets test scripts and performs the tests.
    Each worker process can run many tests in parallel using a number of *worker threads*.
- **Agent processes**
  - Long running process that starts and stops worker processes as required.
  - Maintains a local cache of test scripts distributed from the console.
- **The Console**
  - Coordinates the other processes.
  - Collates and displays statistics.
  - Provides script editing and distribution.

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

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.



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.

2.1.2 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 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.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

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.

2.1.3 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 should match those specified in the console options dialog.

2.1.4 Output

Each worker process writes logging information to a file called `host-n.log`, where `host` is the machine host name and `n` is the worker process number.

Data about individual test invocations is written into a file called `host-n-data.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 log file of each process) looks something like this:

```
Final statistics for this process:

              Successful
              Tests         Errors        Mean Test    Test Time
                                          Time (ms)    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
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
Test 7         25          0          73.20        12.83
Test 8         25          0          141.92       18.36
Test 9         25          0          104.68       19.86        "Logout page"

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.

2.1.5 How do I start The Grinder?

It's 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 test 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 (../g2/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 test script is usually stored alongside the properties file. If you like, you can 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.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

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

- startAgent.cmd

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

- startConsole.cmd

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

- startProxy.cmd

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

Unix

- setGrinderEnv.sh

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

- startAgent.sh

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

- startConsole.sh

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

- startProxy.sh

```
#!/usr/bin/ksh
. (path to setGrinderEnv.sh)/setGrinderEnv.sh
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
java -classpath $CLASSPATH net.grinder.TCPProxy -console -http > grinder.py
```

2.2 Agents and Workers

2.2.1 Agents and Workers

Refer to The Grinder processes (../g3/getting-started.html#The+Grinder+processes) for an overview of the various processes. This page provides some further details.

2.2.1.1 Agent processes

It is typical to run a single agent process on each load injector machine.

When an agent is started, it attempts to connect to the console (../g2/console.html) . If it can connect, it will wait for a signal from the console before starting worker processes. Otherwise, the agent process will start a number of worker processes as specified by its local grinder.properties (../g3/properties.html) file.

If the network connection between the agent and the console is terminated, or the console exits, the agent will exit. If you want the agent to keep running and try regularly to reconnect to the console, use the `-daemon` command line switch. This might prove useful if you register an agent as an operating system service.

Summary of agent process options

Most agent options are controlled by the grinder.properties (../g3/properties.html) file. You can set properties on the command line (../g3/properties.html#Specifying+properties +on+the+command+line) .

`-daemon [reconnect time]`	If this option is specified on the agent command line, and the connection to the console cannot be established or the connection is lost, the agent will sleep for a while and then attempt to connect to the console again. The default sleep time is 60 seconds, but this can be controlled by providing a *reconnect time* in seconds.

2.2.1.2 Worker processes

Worker processes are started by a controlling agent process. The agent process passes each worker a set of properties (../g3/properties.html) that control its behaviour.

2.2.2 The Grinder 3 Properties File

The Grinder worker and agent processes are controlled by setting properties in the `grinder.properties` file.

All properties have default values. If you start The Grinder agent process without a `grinder.properties` file it will communicate with the console using default addresses, use one worker process, one thread, and make one run through the test script found in the file `grinder.py`. This is not much use, so read on...

2.2.2.1 Table of properties

This table lists the properties understood by The Grinder engine.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

Property	Description	Default
`grinder.processes`	The number of worker processes the agent should start.	1
`grinder.threads`	The number of worker threads that each worker process spawns.	1
`grinder.runs`	The number of runs of the test script each thread performs. 0 means "run forever", and should be used when you are using the console to control your test runs.	1
`grinder.processIncrement`	If set, the agent will *ramp up* the number of worker processes, starting the number specified every `grinder.processesIncreme` milliseconds. The upper limit is set by `grinder.processes`.	Start all worker processes together.
`grinder.processIncrement`	Used in conjunction with `grinder.processIncrement` this property sets the interval in milliseconds at which the agent starts new worker processes.	60000 ms
`grinder.initialProcesses`	Used in conjunction with `grinder.processIncrement` this property sets the initial number of worker processes to start.	The value of `grinder.processIncrement`.
`grinder.duration`	The maximum length of time in milliseconds that each worker process should run for. `grinder.duration` can be specified in conjunction with `grinder.runs`, in which case the worker processes will terminate if either the duration time or the number of runs is exceeded.	Run forever.
`grinder.script`	The file name of the Jython script (../g3/scripts.html) to run.	grinder.py
`grinder.jvm`	Use an alternate JVM for worker processes. Defaults to `java` so you do not need to specify this if your `PATH` is sensible.	java
`grinder.jvm.classpath`	Use to adjust the classpath used for the worker process JVMs. Anything specified here will be prepended to the classpath used to start the Grinder processes. Relative paths are evaluated based on the worker process working directory (../g3/	

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

Property	Description	Default
	scripts.html#cwd) . Scripts distributed using the console can refer to libraries in the distribution directory by using relative paths in this property.	
`grinder.jvm.arguments`	Additional arguments to worker process JVMs.	
`grinder.logDirectory`	Directory to write log files to. Created if it doesn't already exist.	The local directory.
`grinder.hostID`	Override the "host" string used in log filenames and logs.	The host name.
`grinder.consoleHost`	The IP address or host name that the agent and worker processes use to contact the console.	All the network interfaces of the local machine.
`grinder.consolePort`	The IP port that the agent and worker processes use to contact the console.	6372
`grinder.useConsole`	Set to `false` to set the agent and worker processes not to use the console.	true
`grinder.reportToConsole`	For advanced use only. The period at which each process sends updates to the console.	500 ms
`grinder.initialSleepTime`	The maximum time in milliseconds that each thread waits before starting. Unlike the sleep times specified in scripts, this is varied according to a flat random distribution. The actual sleep time will be a random value between 0 and the specified value. Affected by `grinder.sleepTimeFactor`, but not `grinder.sleepTimeVariation`	0 ms
`grinder.sleepTimeFactor`	Apply a factor to all the sleep times you've specified, either through a property of in a script. Setting this to `0.1` would run the script ten times as fast.	1
`grinder.sleepTimeVariation`	The Grinder varies the sleep times specified in scripts according to a Normal distribution. This property specifies a fractional range within which nearly all (99.75%) of the times will lie. E.g., if the sleep time is specified as 1000 and the sleepTimeVariation is set to `0.1`, then 99.75%	0.2

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

Property	Description	Default
	of the actual sleep times will be between 900 and 1100 milliseconds.	
`grinder.reportTimesToCo`	Set to `false` to disable reporting of timing information (../../faq.html#timing) to the console; other statistics are still reported.	true
`grinder.debug.singlepro`	If set to `true`, the agent process spawns engines in threads rather than processes, using special class loaders to isolate the engines. This allows the engine to be easily run in a debugger. This is primarily a tool for debugging The Grinder engine, but it might also be useful to advanced users. GrinderStone (http://code.google.com/p/grinderstone/) uses this property to allow interactive debugging. If you want instrumentation to work, you must specify `-javaagent:path/grinder-dcr-agent-version.jar` on the command line. Here, *path* is the full path to the agent jar file that can be found in the `lib` directory, and *version* depends on the version of The Grinder.	false
`grinder.debug.singlepro`	For advanced use only. Specifies a comma separated list of names of classes that should be shared between the worker engines when `grinder.debug.singlepro` is `true`. Class names can end with a `*` wildcard. See bug 134 (https://www.sourceforge.net/p/grinder/bugs/134) for more details.	

2.2.2.2 Specifying properties on the command line

You can also specify these properties as Java system properties in the agent command line. For example, on UNIX systems the following command line can be used to generate log directories based on the current date.

```
java -Dgrinder.logDirectory="log/$(date +%y%m%d)" net.grinder.Grinder
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

Property values set as Java system properties override values set in the `grinder.properties` file. Only properties with names that start "`grinder.`" are considered.

2.2.3 Logging

2.2.3.1 Introduction

The Grinder 3.7 replaced a previous custom logging framework with Logback (http://logback.qos.ch/) . Scripts now use a standard logging API (SLF4J (http://www.slf4j.org/)), and Logback can be configured to alter the output format, manage archiving of log files, and direct log streams to alternative locations.

2.2.3.2 Changing the Logback configuration

The Grinder uses two Logback configuration files:

- `logback.xml` - Used by all processes. Logs to the terminal (`stdout`, `stderr`).
- `logback-worker.xml` - Used by worker processes. Configures logging to the log file and the data log file.

Both configuration files are located in the `grinder-core.jar` file. Refer to the Logback manual (http://logback.qos.ch/manual/index.html) for full details of the configuration file settings.

Let's change the archive settings for the output log to keep more than one archive file. First, extract the configuration file.

```
cd lib
jar xf grinder-core-3.7.jar logback-worker.xml
```

Open the `logback-worker.xml` file in a text editor and locate the `log-file` appender. To keep five archive files, simply change the `maxIndex` setting to`5` so it matches the following:

```
<appender name="log-file"
  class="ch.qos.logback.core.rolling.RollingFileAppender">
  <file>${PREFIX}.log</file>

  <encoder>
    <pattern>%d %-5level %logger{0} %marker: %message%n</pattern>
  </encoder>

  <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
    <fileNamePattern>${PREFIX}.log%i</fileNamePattern>
    <minIndex>1</minIndex>
    <maxIndex>5</maxIndex>
  </rollingPolicy>

  <triggeringPolicy class="net.grinder.util.logback.RollOnStartUp" />

</appender>
```

Save the file under the same name (`logback-worker.xml`). To use the modified configuration, add the file's directory to the `CLASSPATH` used to start The Grinder. We extracted the file into the `lib` directory, so we could start the agent process with something like the following:

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
cd $GRINDER_HOME
java -classpath lib:lib/grinder.jar net.grinder.Grinder
```

2.2.3.3 Logging data to a database

The `logback-worker.xml` file configures two Logback loggers: `worker` for the main log file, and `data` for the data log file. Let's change the configuration to send test data to a database. To do this, we'll configure a new appender and add it to the data logger. Logback's database appender supports several databases; here's a suitable configuration for an Oracle database.

```
<appender name="data-db" class="ch.qos.logback.classic.db.DBAppender">
  <connectionSource class="ch.qos.logback.core.db.DriverManagerConnectionSource">
    <driverClass>oracle.jdbc.OracleDriver</driverClass>
    <url>jdbc:oracle:thin:@localhost:1521:XE</url>
    <user>grinder</user>
    <password>grinder</password>
  </connectionSource>
</appender>

<logger name="data" additivity="false">
  <appender-ref ref="data-file" />
  <appender-ref ref="data-db" />
</logger>
```

To expose any problems with the configuration, we'll also enable the Logback debug output by changing the first line of the configuration.

```
<configuration debug="true">
```

Before we can use the database appender, we need to set up the appropriate database tables. To do this, create a suitable database account (the configuration above uses an account called *grinder*), download the Logback distribution, and locate and execute the appropriate DDL (`logback-classic/src/main/java/ch/qos/logback/classic/db/dialect/oracle.sql` for Oracle).

To run the configuration, add the directory containing `logback-worker.xml` to the `CLASSPATH`, along with the appropriate database driver. For example:

```
java -classpath /usr/lib/oracle/xe/app/oracle/product/10.2.0/server/jdbc/lib/
ojdbc14.jar:lib:lib/grinder.jar  net.grinder.Grinder
```

2.2.3.4 Writing a custom appender for data logs

If you tried out the database configuration in the previous section you may have noted the following drawbacks.

- It's not particularly fast. Maximum logging throughput is of the order of a hundred log events per second, and this severely constrains the rate at which a worker process can perform tests.
- The log data is written as a string to a single `formatted_message` column. This is not amenable to further processing.

To address these problems, you will have to write a custom database appender, perhaps by modifying the standard Logback-supplied appender. If you write such an appender,

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

please consider making it generic and contributing it back to The Grinder. The following sections contain some implementation ideas.

Improving database logging performance

The most beneficial change from a performance perspective would be to buffer the log events, and write many events to the database at once. JDBC batching would further improve performance. I suspect that this change alone would allow tens of thousands of events to be logged per second.

The standard appender includes caller data (filename, class, method, line) that is expensive to obtain and is of little use for The Grinder data log. It also logs exception and property information. These features can be removed.

To support the secondary exception and property tables, the standard appender needs to obtain the primary key of the newly logged event. Unfortunately this uses an appender level lock (unnecessarily, it could have synchronised on the database connection instead), and becomes a bottleneck when many worker threads are using the appender. Since the exception and property tables are unnecessary, this complexity can also be removed.

Customising data log output

The Grinder data logger generates `ILoggingEvents` with the formatted string set to a comma-separated string (formatted as in the standard data log). It also supplies an instance of `net.grinder.engine.process.DataLogArguments` as the first and only argument. This can be accessed using `ILoggingEvent.getArgumentArray()[0]`.

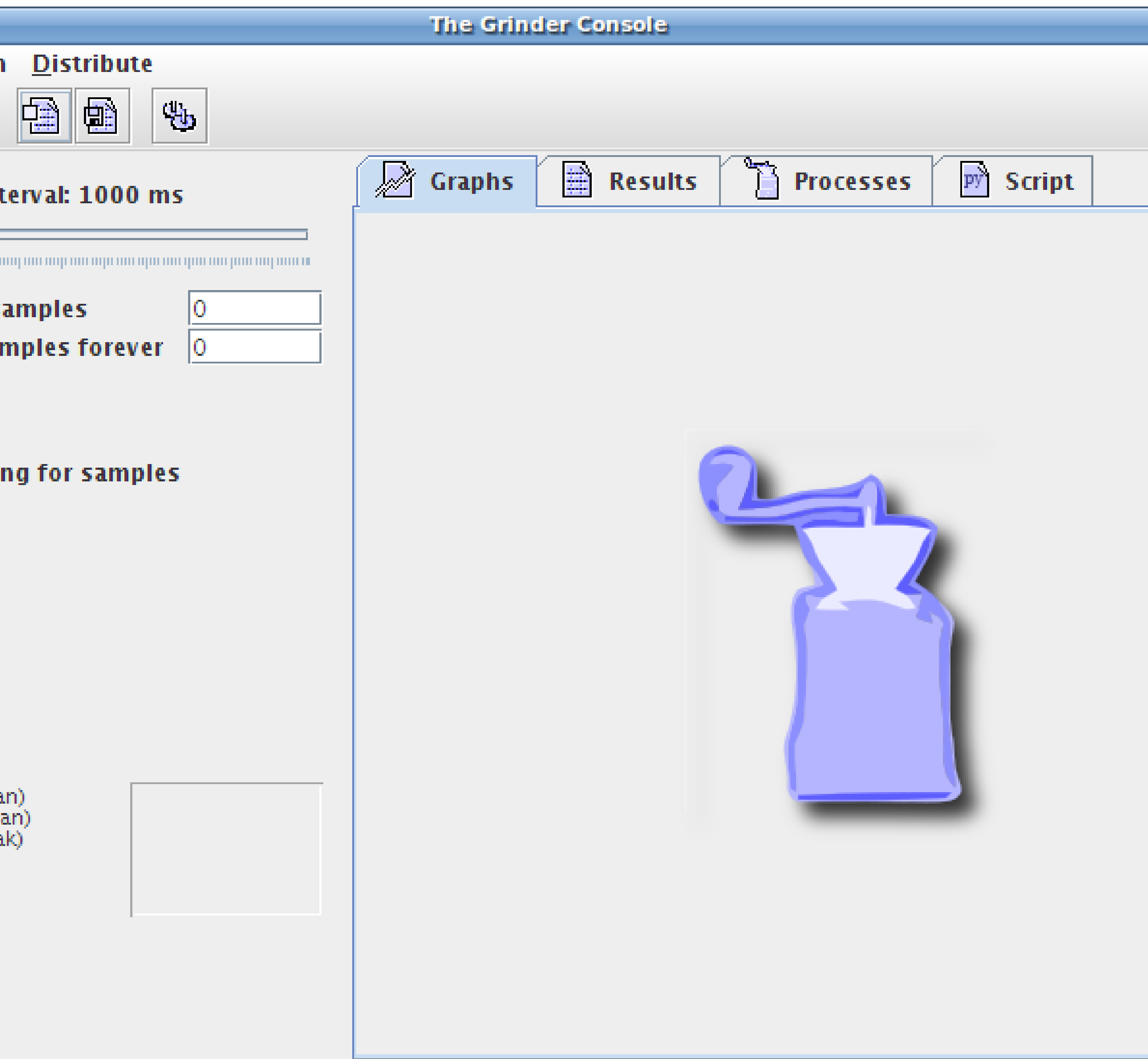
The `DataLogArguments` object provides all the information you might need about a particular data log event, including the thread and run numbers, the `Test`, and the raw statistics. Refer to the `net.grinder.engine.processs.ThreadDataLogger` source code for an example of how to extract the appropriate statistics values from the raw statistics.

2.3 The Console

2.3.1 The Console User Interface

Follow these instructions (../g3/getting-started.html#howtostart) to start the console.

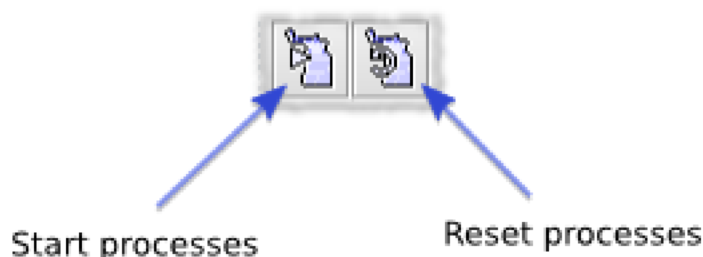
Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.



2.3.1.1 Process controls

The *Start processes*, *Reset processes*, and *Stop processes* menu items send signals to Grinder processes that are listening. (See the properties (../g3/properties.html) `grinder.useConsole`, `grinder.consoleHost` and `consolePort`.) *Start processes* and *Reset processes* are also tool bar buttons.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.



These controls will be disabled if no agents are connected to the console. You can check whether any agents are connected on the Processes tab.

Worker processes that are controlled by the console can be in one of three states:

1. Initiated (waiting for a console signal)
2. Running (performing tests, reporting to console)
3. Finished (waiting for a console signal)

The *Start processes* control signals to worker processes that they should move into the running state. Processes that are already running will ignore this signal. Processes that are in the finished state exit; the agent process will then reread the properties file, and launch new worker processes in the running state.

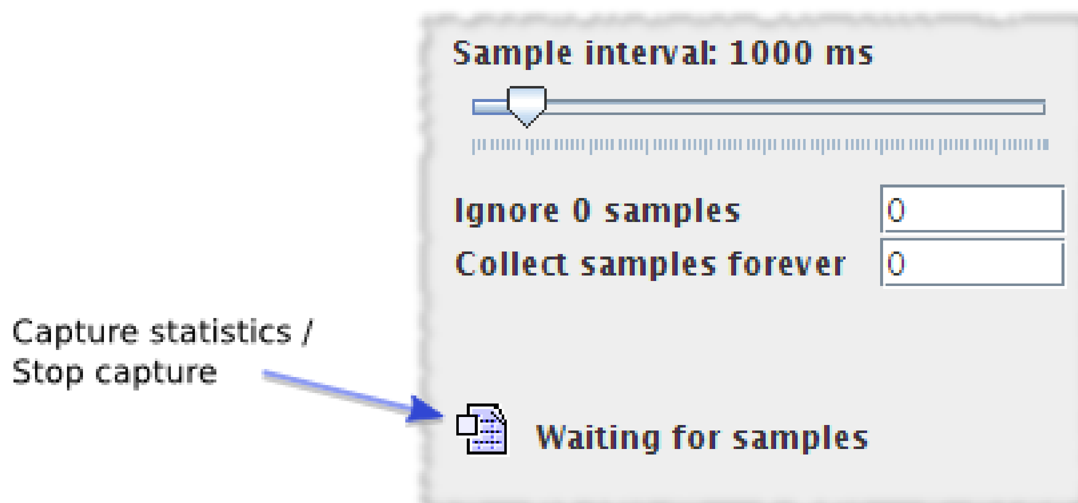
The *Reset processes* control signals all the worker processes to exit. The agent process will then reread the properties file and launch new worker processes.

The *Stop processes* control signals all processes, including the agent processes, to exit. This is infrequently used, you usually want to use *Reset processes* instead.

> **Note:**
> Each time the worker processes run, they generate a new set of logs. Logs from previous runs are "archived" by renaming them. The number of logs that are kept from previous runs can be controlled with `grinder.numberOfOldLogs`.

2.3.1.2 Sample controls

The sample controls determine how the console captures reports from the worker processes. It is important to understand that these only control the console behaviour. For example, they do not adjust the frequency at which the worker processes send reports (see `grinder.reportToConsole.interval` (../g3/properties.html) for that). Additionally, the sample controls do not interact in any way with the process controls.



Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

The slider controls the period at which the console will take a *sample*. This involves adding up all the reports received over that sample interval and calculating the TPS as (number of tests that occurred)/(interval length). It is also the period at which the console graphs and statistics are updated.

By default, the console starts updating the display and calculating totals from the first non-zero sample period. A non-zero sample period is one in which an update from a worker process was received. You can adjust how many non-zero sample periods the console ignores before starting capture with the *ignore samples* text field.

The third control allows you to adjust how many samples the console will collect before stopping capture.

You can also manually start and stop the sampling with the *Capture statistics*/*Stop capture* control. Use the *Save statistics* control to save the current set of statistics to a file.

2.3.1.3 The Graphs and Results tabs

On the console there are two tabs which display information about The Grinder and its tests. These are detailed below:

Graphs

Each graph displays the 25 most recent Tests Per Second (TPS) values for a particular test. A new value is added every console sample period. The y-axis is scaled so that the full height represents the peak TPS value received for the test since the display was last reset.

The colours are based on the relative response time. Long response times are more red, short response times are more yellow. This acts as an eye-catcher, allowing expensive tests to be easily spotted.

Results

The Results tab shows the results from The Grinder instrumentation.

Test	The test number as specified in the test script, eg. tests[14000] will display as Test 14000.
Description	The test description as specified in the test script.
Successful Tests	The total number of iterations of the test that were successfully executed by The Grinder during the test run.
Errors	The total number of iterations of the test that failed to be fully executed by The Grinder during the test run.
Mean Time	The mean time taken to execute the test and receive the full response from the target server/application, in milliseconds.
Mean Time Standard Deviation	The mean standard deviation of the time taken to execute the test and receive the full response from the target server/application, in milliseconds.
TPS	Transactions per second. The average number of iterations of the test that successfully ran in a one second interval.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

Peak TPS	Peak Transactions per second. The maximum number of iterations of the test that successfully ran in a one second interval.

There is additional instrumentation provided by the HTTPPlugin.

Mean Response Length	The mean size of HTTP response from the target server/application in response to the executed test, in bytes.
Response Bytes per Second	The mean number of bytes per second received from the target server/application, in bytes per second. This gives an indication of the amount of bandwidth being consumed by the test. This does not take into account the amount of traffic being sent to the target server/application.
Response Errors	The total number of HTTP Response Error Codes (eg, 404, 500 etc) received during the test run.
Mean Time to Resolve Host	The mean time taken to resolve the ip address of the target server from the Fully Qualified Domain Name, via hosts file or DNS, in milliseconds. This is the time relative to the start of the test iteration.
Mean Time to Establish Connection	The mean time taken to establish a tcp connection to the target server/application, in milliseconds. This is the time relative to the start of the test iteration.
Mean Time to First Byte	The mean time taken to receive the first byte of response from the target server/application, in milliseconds. This is the time relative to the start of the test iteration.

2.3.1.4 Processes tab

This tab displays information about the Agents, their worker processes and associated threads.

Process	The name of the process. A parent process will take the hostname of the box on which it is running Its child processes take the name of the parent process and add a suffix of "-x" where x is an integer, eg. myserver-0.
Type	The type of process, eg. Agent or Worker.
State	Information about the state of the process, eg. "Connected" for an agent process and "Running" and "Finished" for a Worker process.

2.3.1.5 Script tab

This tab contains the console support for script editing and distribution. The distribution controls are also accessible through the **Distribute** menu.

> **Note:**
> Script editing and distribution is optional. You don't have to use it, but then you must copy property files and scripts to each machine that runs an agent, or use a shared drive.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

To use the script distribution, follow these steps:

1. Set the directory for the script distribution
2. Create a script and a property file
3. Select the properties file to use
4. Distribute the changed files to the agents
5. Start the Worker processes

Set the directory for the script distribution

The file tree on the left hand side of Script tab is shows the a view of local files on the console machine. Use the **Distribute/Set directory...** menu option or the tool bar button to set the distribution directory to the place where you want to store your scripts. All of the files below the directory will be distributed to the worker processes, so don't set it to */home* or *C:*.

If you are using The Grinder for the first time, you might like to set the distribution directory to the `examples` directory in The Grinder installation.

Create a script and a property file

You can use the console to create, view, and edit script files in the distribution directory. The editor is rudimentary, but good enough for basic editing.

If your script relies on other files (including Jython modules), copy them below the distribution directory.

You can also edit files in the distribution directory with a text editor of your choice. For convenience, you can define an external editor in the console options (**File/Options.../Script Editor**), and launch it by right-clicking on a file in the file tree and selecting **Open with external editor**.

Once you have your script ready, create a properties (../g3/properties.html) file. The file name extension should be `properties`, and unless you have many different properties files in the directory, the file is usually called `grinder.properties`. If your script is not called `grinder.py`, add a `grinder.script` property to your properties file:

```
grinder.script = myscript.py
```

The properties sent from the console are combined with any set in a `grinder.properties` file in the agent's working directory or set on the agent command line (../g3/properties.html#Specifying+properties+on+the+command+line) . If a property is specified in several places, the order of precedence is

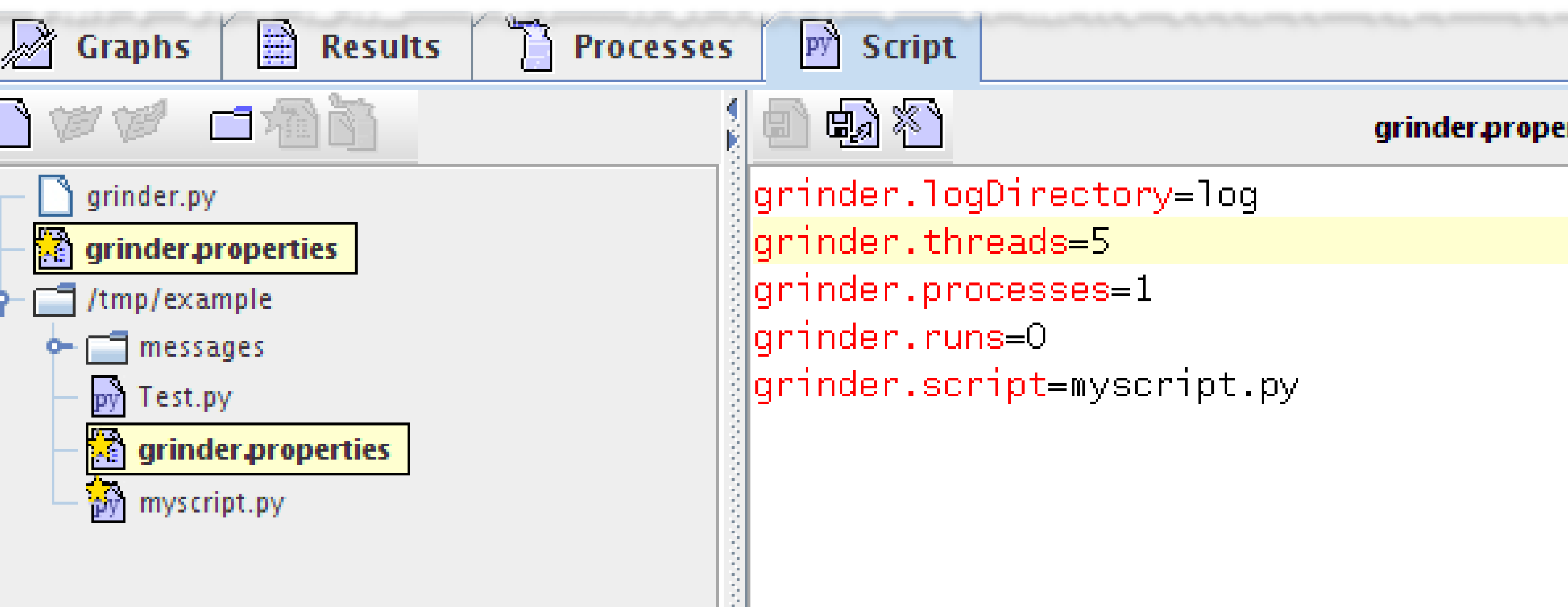
- Properties sent by the console*[most important]*
- Properties set on the agent command line
- Properties in the agent's local `grinder.properties` file *[least important]*

> Note:
> If your agents are running remotely to the console, you will need to set the `grinder.consoleHost` property (and `grinder.consolePort` if the console isn't using the default port) in the agent's command line or local `grinder.properties` so it can make the initial connection to the console.

Select the properties file to use

Right-click on the properties file and chose `Select properties`.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.



The properties file and the script to which it refers will be indicated with a star.

Distribute the changed files to the agents

Select the **Distribute/Distribute files** menu item, or click on the toolbar button.

Each agent maintains its own local cache of the files below the distribution directory. When you select **Distribute files**, any files that have changed will be sent to the agents. The distribution controls will only be enabled if one or more agents is connected to the console, and one or more files has been edited.

Start the Worker processes

Select **Start processes** as described above.

2.3.1.6 Internationalisation help wanted

If you are bilingual you might fancy translating the console (../development/contributing.html#translating) into a language of your choice.

2.3.2 The Console Service

2.3.2.1 Overview

The console service provides an interface for automating The Grinder. It allows The Grinder to be controlled by a scheduler or a Continuous Integration framework such as Hudson/Jenkins; remote monitoring using a web browser; and creative possibilities such monitoring and influencing the test execution from a test script, perhaps by starting additional worker processes.

You can use the console service to start and stop worker processes; change console options; distribute script files; start and stop recordings; and obtain aggregated test results.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

The first version of the console service was released as part of The Grinder 3.10, and provides REST web services. Future releases will provide other flavours of interface, such as a browser-based user interface, and event-driven publication of data.

2.3.2.2 Configuration

The console hosts an HTTP server that runs the console service. When the console is started, the server listens for HTTP requests on port 6373. For most users, the console service should work out of the box with no further configuration.

If port 6373 is unavailable, an error message will be presented. This usually occurs because another program has claimed the port. Perhaps there two copies of the console have been started. You can change the HTTP port using the console options, and also set the HTTP host to your publicly accessible host name or IP address. In fact, unless you change the host name, the HTTP server will listen on localhost, and you'll only be able to connect to the console from local processes.

You can check that the console service has started correctly by using your browser to access http://localhost:6373/version. If the service is running, the browser will display the version of The Grinder.

Running without a GUI

If you don't use the graphical user interface (../g3/console.html) , you can start the console in in a terminal mode by passing a `-headless` option as follows.

```
java -classpath lib/grinder.jar net.grinder.Console -headless
```

Setting the HTTP address and port on the command line

You can also specify the console service address and port on the command line, overriding the console options:

```
java -classpath lib/grinder.jar -Dgrinder.console.httpHost=myhost -
Dgrinder.console.httpPort=8080 net.grinder.Console
```

Here `myhost` should resolve to a local IP address.

2.3.2.3 The REST interface

The REST interface accepts HTTP `GET`, `POST`, and `PUT` requests. The request's `Accept` header is used to select the formatting of the response.

Accept header	Response body format
`application/clojure`	Clojure data structure
`application/json`	JSON
`application/x-yaml`	YAML
`text/html`	YAML wrapped in HTML
No accept header	JSON
Other values	*406 Not Acceptable*

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

The YAML in HTML support allows simple access to some of the services (those that use `GET`) from a web browser.

Some of the `POST` and `PUT` requests require additional data to be supplied in the body of the request. The request's `Content-Type` header is used to determine whether the request body should be parsed as JSON, YAML, or a Clojure data structure.

Content-Type header	Request body format
`application/clojure` `application/x-clojure`	Clojure map
`application/json` `application/x-json`	JSON object

`application/yaml`
`application/x-yaml`
`text/yaml`
`text/x-yaml`
	YAML map

Other values	*Ignored*

Available services

The following services are available.

Method	URL	Description
`POST`	`/agents/start-workers`	Send a start signal to the agents to start worker processes. Equivalent to the start processes (../g3/console.html#process-controls) button.
`GET`	`/agents/status`	Returns the status of the agent and worker processes.
`POST`	`/agents/stop`	Terminates all agents and their worker processes. You will usually want `/agents/stop-workers` instead.
`POST`	`/agents/stop-workers`	Send a stop signal to connected worker processes. Equivalent to the reset processes (../g3/console.html#process-controls) button.
`POST`	`/files/distribute`	Start the distribution of files to agents that have an out of date cache. Distribution may take some time, so the service will return immediately and the files will be distributed in proceeds in the background. The service returns a map with an `:id` entry that can be used to identify the particular distribution request.
`GET`	`/files/status`	Returns whether the agent caches are stale (i.e. they are out of date with respect to the console's

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

Method	URL	Description
		central copy of the files), and the status of the last file distribution.
GET	`/properties`	Return the current values of the console options.
PUT	`/properties`	Set console options. The body of the request should be a map of keys to new values; you can provide some or all of the properties. A map of the keys and their new values will be returned. You can find out the names of the keys by issuing a GET to `/properties`.
POST	`/properties/save`	Save the current console options in the preferences file. The preferences file is called `.grinder_console` and is stored in the home directory of the user account that is used to run the console.
GET	`/recording/data`	Return the current recorded data. Equivalent to the data in the results tab (../g3/console.html#Results) .
GET	`/recording/data-latest`	Return the latest sample of recorded data. Equivalent to the data in the lower pane of the results tab (../g3/console.html#Results) .
POST	`/recording/start`	Start capturing data. An initial number of samples may be ignored, depending on the configured console options.
POST	`/recording/stop`	Stop the data capture.
GET	`/recording/status`	Return the current recording status.
POST	`/recording/reset`	Discard all recorded data. After a reset, the model loses all knowledge of Tests; this can be useful when swapping between scripts. It makes sense to reset with the worker processes stopped.
POST	`/recording/zero`	Reset the recorded data values to zero.
GET	`/version`	Returns the version of The Grinder.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

2.3.2.4 Example session

Let's have a look at an example terminal session that exercises the REST interface. We'll use curl (http://curl.haxx.se/) as a client, but other HTTP clients will work will as well.

> Note:
> A web cast of a similar example session is available on YouTube (http://www.youtube.com/watch?v=OzB3bvQnS7U) .

Starting up

First, we start the console, specifying `-headless` because we're not going to be using the GUI.

```
% java -classpath lib/grinder.jar net.grinder.Console -headless

2012-05-30 18:33:30,472 INFO  console: The Grinder 3.10-SNAPSHOT
2012-05-30 18:33:30,505 INFO  org.eclipse.jetty.server.Server: jetty-7.6.1.v20120215
2012-05-30 18:33:30,538 INFO  org.eclipse.jetty.server.AbstractConnector:
 Started SelectChannelConnector@:6373
```

You can see the console service is listening on port 6373, as expected. Now open another terminal window, and check the lights are on.

```
% curl http://localhost:6373/version

The Grinder 3.10-SNAPSHOT
```

The console service has responded with the appropriate version string, as expected.

Next let's ask for the current console options.

```
% curl http://localhost:6373/properties

{"httpPort":6373,"significantFigures":3,"collectSampleCount":0,
"externalEditorCommand":"","consolePort":6372,"startWithUnsavedBuffersAsk":true,
"scanDistributionFilesPeriod":6000,"resetConsoleWithProcesses":false
"sampleInterval":3100,"resetConsoleWithProcessesAsk":true,
"frameBounds":[373,168,1068,711],"httpHost":"","externalEditorArguments":"",
"ignoreSampleCount":0,"consoleHost":"","distributeOnStartAsk":false,
"propertiesNotSetAsk":true,"distributionDirectory":"/tmp/grinder-3.9.1/foo",
"propertiesFile":"/tmp/grinder-3.9.1/foo/grinder.properties",
"distributionFileFilterExpression":
"^CVS/$|^\\.svn/$|^.*~$|^(out_|error_|data_)\\w+-\\d+\\.log\\d*$",
"saveTotalsWithResults":false,"stopProcessesAsk":true,"lookAndFeel":null}
```

The console options are returned in the response body as a JSON object containing key/value pairs. This format is easily to parse with a scripting language, or JavaScript in a browser.

Setting the properties

Some of the console options are only relevant to the GUI, but others also affect the console service. The following command changes the distribution directory to the examples directory in our distribution, and selects the `grinder.properties` file.

```
% curl -H "Content-Type: application/json" -X PUT http://localhost:6373/properties
    -d '{"distributionDirectory":"examples", "propertiesFile":"grinder.properties"}'

{"propertiesFile":"grinder.properties","distributionDirectory":"examples"}
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

The properties that were changed are returned in the response body.

Connecting an agent

In a third terminal window, let's start an agent. We'll be distributing files to the agent which it will cache in its working directory, so we'll do so in a temporary directory.

```
% cd /tmp
% java -classpath ${GRINDER_HOME}/lib/grinder.jar net.grinder.Grinder

2012-05-30 18:54:30,674 INFO  agent: The Grinder 3.10-SNAPSHOT
2012-05-30 18:54:30,737 INFO  agent: connected to console at localhost/127.0.0.1:6372
2012-05-30 18:54:30,737 INFO  agent: waiting for console signal
```

The agent has connected to the console. We could start up other agents, perhaps on other machines; we'd just need to add `-Dgrinder.console.Host=console-machine` before `net.grinder.Grinder`.

We can confirm that the console knows about the agent.

```
% curl http://localhost:6373/agents/status

[{"id":"paston02:968414967|1338400470671|425013298:0","name":"paston02","number":-1,
"state":"RUNNING","workers":[]}]
```

The agent is running, and it has not yet started any worker processes. Now we'll distribute the scripts to the agent.

```
% curl -X POST http://localhost:6373/files/distribute

{"id":1,"state":"started","files":[]}
```

File distribution is asynchronous - the result indicates that the distribution request has been queued, and allocated id 1. We can find out where it's got to by querying the status.

```
% curl http://localhost:6373/files/status

{"stale":false,"last-distribution":{"per-cent-complete":100,"id":1,"state":"finished",
"files":
["cookies.py","digestauthentication.py","ejb.py","jdbc.py","httpg2.py","console.py",
"slowClient.py","httpunit.py","sequence.py","jmssender.py","grinder.properties","sync.py",
"amazon.py","helloworldfunctions.py","form.py","xml-rpc.py","parallel.py","jaxrpc.py",
"scenario.py","threadrampup.py","statistics.py","jmsreceiver.py","helloworld.py",
"helloworld.clj","proportion.py","fba.py","scriptlifecycle.py","email.py","http.py"]}}
```

This tells us that the agent caches are no longer stale, and the distribution 1 completed, sending the list of files to the agents.

Starting the workers

We're going to have The Grinder start some worker processes and run the helloworld.py (../g3/script-gallery.html#helloworld.py) script, which is one of the files we've just sent.

We previously set the console option *propertiesFile* to a properties file in the distributed files (we chose `grinder.properties`). Setting this option causes the agent to first look for any script file in its distribution cache, falling back to its working directory if the

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

file isn't found. We can override the values in the distributed `grinder.properties` file in properties sent with the start command.

> **Note:**
> Distributing the files to the agents is optional. If you do so, then be sure to set *propertiesFile* to a valid properties file in the distribution. Otherwise, the agent will resolve the script file name relative to its working directory, ignoring the files in the distribution cache. If you don't distribute the files you'll have to make sure the agent can find the script through some other means, such as a file system share.
> Properties supplied with the start command override those specified with *propertiesFile*, which in turn override those specified as system properties on the agent or worker process command lines, which in turn override those found in a `grinder.properties` file in the agent's working directory.

The following starts two worker processes, to perform three runs of *helloworld.py*, using five worker threads each.

```
% curl -H "Content-Type: application/json" -X POST http://localhost:6373/agents/start-
workers -d '{"grinder.processes" : "2", "grinder.threads" : "5", "grinder.runs" : "3",
 "grinder.script" : "helloworld.py" }'

success
```

Obtaining the results

Let's stop the recording. Until we do this, the TPS will be calculated over an increasing duration, and steadily fall. When doing real tests, it's more common to set `grinder.runs` to `0` so that the workers don't stop until instructed to do so, and to record a period of data before they are stopped.

```
% curl -X POST http://localhost:6373/recording/stop

{"state":"Stopped","description":"Collection stopped"}
```

We can now retrieve the recording data.

```
% curl http://localhost:6373/recording/data

{"status":{"state":"Stopped","description":"Collection stopped"},
"columns":["Tests","Errors","Mean Test Time (ms)","Test Time Standard Deviation
 (ms)","TPS","Peak TPS"],
"tests":[{"test":1,"description":"Log method","statistics":
[30,0,0.2,0.4,9.674298613350532,
9.67741935483871]}],
"totals":[30,0,0.2,0.4,9.674298613350532,9.67741935483871]}
```

There were 30 executions of Test 1 as expected (2 worker processes x 5 worker threads x 3 runs), with an average execution time of 0.2 ms.

```
% curl http://localhost:6373/recording/data-latest

{"status":{"state":"Stopped","description":"Collection stopped"},
"columns":["Tests","Errors","Mean Test Time (ms)","Test Time Standard Deviation
 (ms)","TPS","Peak TPS"],
"tests":[{"test":1,"description":"Log method","statistics":
[30,0,0.2,0.4,9.674298613350532,
9.67741935483871]}],
"totals":[30,0,0.2,0.4,9.674298613350532,9.67741935483871]}
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

Adding the -latest will retrieve the latest sample data available. This is most useful to get near real time data a currently executing test.
Again, there were 30 executions of Test 1 as expected (2 worker processes x 5 worker threads x 3 runs), with an average execution time of 0.2 ms.

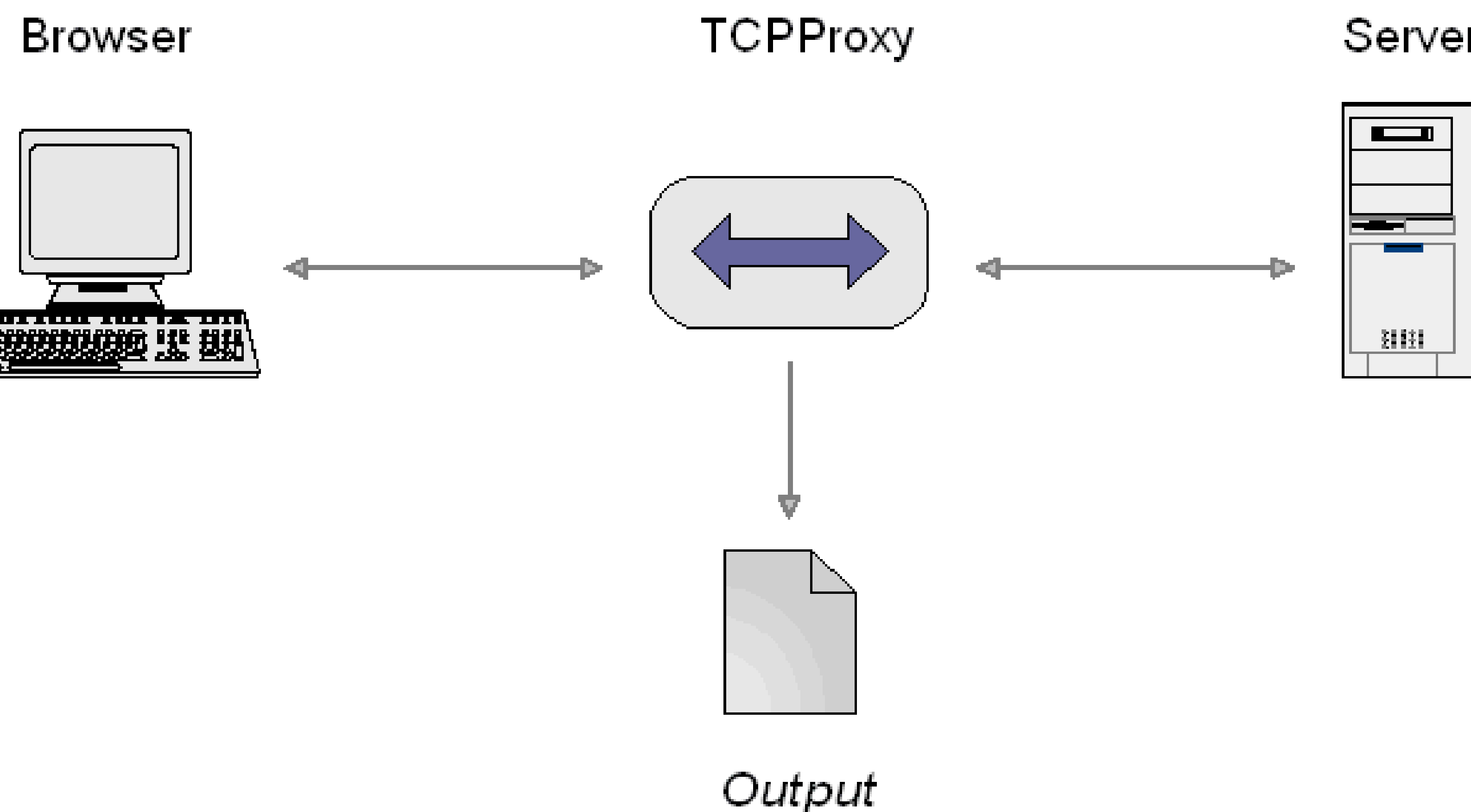
Conclusion

I hope you've enjoyed this quick tour of the console service. Start the console and an agent yourself, and have a play.

> Note:
>
> **Tips**
> If a call to a service results in *Resource not found*, check you've used the appropriate HTTP method (GET, PUT, or POST).
> You might find it simpler to run the console GUI (don't add `-headless` to the command line). This will allow you to see the current console status at a glance.

2.4 The TCPProxy

The TCPProxy is a proxy process that you can place in a TCP stream, such as the HTTP connection between your browser and a server. It filters the request and response streams, sending the results to the terminal window (`stdout`). You can control its behaviour by specifying different filters.



The TCPProxy's main purpose is to automatically generate HTTP test scripts that can be replayed with The Grinder's HTTP plugin. Because the TCPProxy lets you see what's going on at a network level it is also very useful as a debugging tool in its own right.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

2.4.1 Starting the TCPProxy

You start the TCPProxy with something like:

```
CLASSPATH=/opt/grinder/lib/grinder.jar
export CLASSPATH

java net.grinder.TCPProxy
```

Say `java net.grinder.TCPProxy -?` to get a full list of the command line options.

With no additional options, the TCPProxy will start and display the following information:

```
Initialising as an HTTP/HTTPS proxy with the parameters:
   Request filters:  EchoFilter
   Response filters: EchoFilter
   Local address:    localhost:8001
Engine initialised, listening on port 8001
```

This indicates that the TCPProxy is listening as an HTTP proxy on port `8001` (the default, but you can change it with `-localPort`).

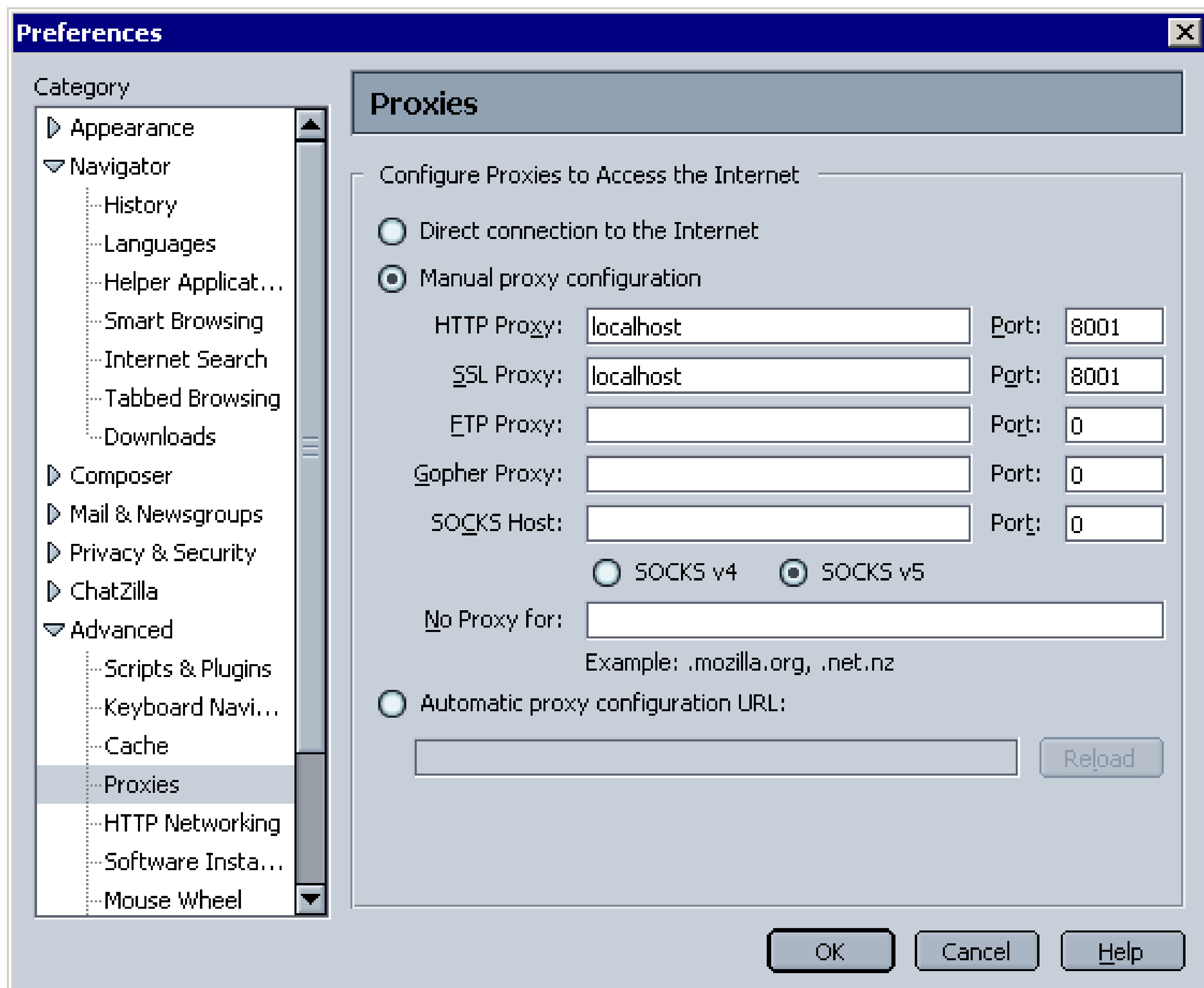
The TCPProxy appears to your browser just like any other HTTP proxy server, and you can use your browser as normal. If you type `http://grinder.sourceforge.net` into your browser it will display The Grinder home page and the TCPProxy will output all of the HTTP interactions between the browser and the SourceForge site.

The TCPProxy will proxy both HTTP and HTTPS. See below for details on customising the SSL configuration.

2.4.2 Preparing the Browser

You should now set your browser connection settings to specify the TCPProxy as the HTTP proxy. In the browser options dialog, set the proxy host to be the host on which the TCPProxy is running and proxy port to be `8001`).

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.



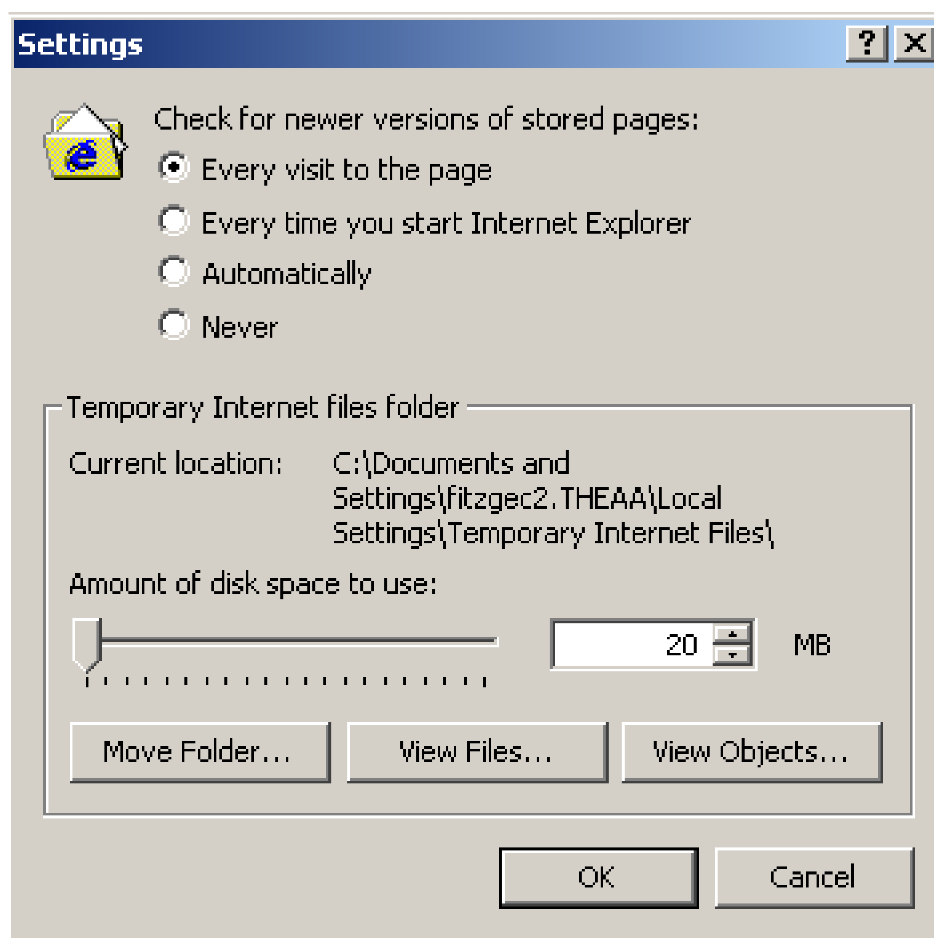
The relevant options dialog can be accessed by the following steps:

MSIE: Tools -> Internet Options -> Connections -> Local Area Network Settings.
Mozilla/Netscape: Edit -> Preferences -> Advanced - Proxies.
Mozilla/Firefox: Tools -> Options -> General -> Connection Settings.
Opera: Tools -> Preferences -> Advanced -> Network -> Proxy Servers.

It is important to remember to remove any "bypass proxy server" or "No Proxy for" settings that you might have so that all the traffic flows through the TCPProxy and can be captured.

It might also be a good idea to clear out any cache/temporary Internet files that might be on your workstation. On the other hand, you might decide not to do this if you want to record a script representing a frequent user to your site who has images are resources in their browser cache (../faq.html#http-caching) . Also for IE users, changing the temporary Internet files settings to check for a newer version on every visit to a page can be useful.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.



2.4.3 Using the EchoFilter

The EchoFilter is the default filter used by the TCPProxy if no options are specified in the startup command. The EchoFilter outputs the stream activity to the terminal. It can be very useful for debugging as described in this FAQ (../faq.html#use-the-tcpproxy) .

Bytes that do not have a printable ASCII representation are displayed in hexadecimal between square brackets. Here's some example output:

```
------ 127.0.0.1:2263->ads.osdn.com:80 ------
GET /?ad_id=5839&alloc_id=12703&site_id=2&request_id=8320720&1102173982760 HTTP/1.1
Host: ads.osdn.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.7.5) Gecko/20041107
 Firefox/1.0
Accept: image/png,*/*;q=0.5
Accept-Language: en-gb,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://sourceforge.net/projects/grinder


--- ads.osdn.com:80->127.0.0.1:2263 opened --
------ ads.osdn.com:80->127.0.0.1:2273 ------
HTTP/1.1 200 OK
Date: Sat, 04 Dec 2004 15:26:27 GMT
Server: Apache/1.3.29 (Unix) mod_gzip/1.3.26.1a mod_perl/1.29
Pragma: no-cache
Cache-control: private
Connection: close
Transfer-Encoding: chunked
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
Content-Type: image/gif


------ ads.osdn.com:80->127.0.0.1:2273 ------
80B
GIF89ae[00])[00D50000C3C3C3FEFDFD]hhhVVVyyy[F5CCD2D4D4D4CBCBCBD7]'F
```

Information lines are displayed to indicate the end point addresses and direction of the information flow and also whether a connection has just been opened or closed.

2.4.4 Using the HTTP TCPProxy filters

You can use the TCPProxy to generate an HTTP script suitable for use with The Grinder. The Grinder provides a pair of HTTP filters for this purpose. These filters are enabled by the `-http` command line option.

The first step is to start the TCPProxy with an HTTP filter:

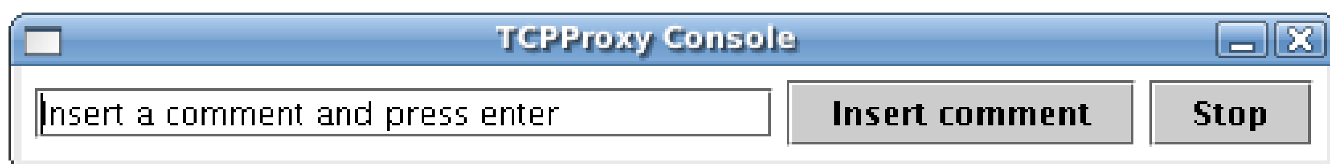
```
java net.grinder.TCPProxy -console -http > grinder.py
```

The `> grinder.py` part of the line sends the script to a file called `grinder.py`.

The terminal output of the TCPProxy looks like:

```
14/03/06 17:04:25 (tcpproxy): Initialising as an HTTP/HTTPS proxy with the
parameters:
   Request filters:    HTTPRequestFilter
   Response filters:   HTTPResponseFilter
   Local address:      localhost:8001
14/03/06 17:04:27 (tcpproxy): Engine initialised, listening on port 8001
```

The console (initiated by *-console*) displays a simple control window that allows the TCPProxy to be shut down cleanly. This is needed because some terminal shells, e.g. Cygwin bash, do not allow Java processes to be interrupted cleanly, so filters cannot rely on standard shut down hooks. The console also allows a user to add ad-hoc commentary to the script during the recording. The console looks like this:



The TCPProxy console will be incorporated into the main console (../g2/console.html) in a future release.

Set your browser to use the TCPProxy as the HTTP proxy as described earlier), and run through your test scenario on your website.

Having finished your run through, press "Stop" on the TCPProxy console and the generated script will be written to `grinder.py`.

The `grinder.py` file contains headers, requests and a logical grouping of requests into pages, of the recorded tests.

For example, the headers section:

```
# The Grinder 3.11-SNAPSHOT
# HTTP script recorded by TCPProxy at 05-Jul-2012 09:20:55
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
from net.grinder.script import Test
from net.grinder.script.Grinder import grinder
from net.grinder.plugin.http import HTTPPluginControl, HTTPRequest
from HTTPClient import NVPair
connectionDefaults = HTTPPluginControl.getConnectionDefaults()
httpUtilities = HTTPPluginControl.getHTTPUtilities()

# To use a proxy server, uncomment the next line and set the host and port.
# connectionDefaults.setProxyServer("localhost", 8001)

def createRequest(test, url, headers=None):
    """Create an instrumented HTTPRequest."""
    request = HTTPRequest(url=url)
    if headers: request.headers=headers
    test.record(request, HTTPRequest.getHttpMethodFilter())
    return request

# These definitions at the top level of the file are evaluated once,
# when the worker process is started.

connectionDefaults.defaultHeaders = \
  [ NVPair('Accept-Encoding', 'gzip, deflate'),
    NVPair('Accept-Language', 'en-gb,en;q=0.5'),
    NVPair('Cache-Control', 'no-cache'),
    NVPair('User-Agent', 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:13.0)
 Gecko/20100101 Firefox/13.0.1'), ]

headers0= \
  [ NVPair('Accept', 'text/css,*/*;q=0.1'),
    NVPair('Referer', 'http://grinder.sourceforge.net/'), ]

headers1= \
  [ NVPair('Accept', '*/*'),
    NVPair('Referer', 'http://grinder.sourceforge.net/'), ]

headers2= \
  [ NVPair('Accept', 'image/png,image/*;q=0.8,*/*;q=0.5'),
    NVPair('Referer', 'http://grinder.sourceforge.net/'), ]

headers3= \
  [ NVPair('Accept', 'image/png,image/*;q=0.8,*/*;q=0.5'),
    NVPair('Referer', 'http://grinder.sourceforge.net/skin/screen.css'), ]

headers4= \
  [ NVPair('Accept', 'image/png,image/*;q=0.8,*/*;q=0.5'),
    NVPair('Referer', 'http://grinder.sourceforge.net/skin/profile.css'), ]

#....
```

In the requests section, a request object for each unique URL is created:

```
url0 = 'http://grinder.sourceforge.net:80'
url1 = 'http://www.ohloh.net:80'
url2 = 'http://sourceforge.net:80'

request101 = createRequest(Test(101, 'GET /'), url0)

request102 = createRequest(Test(102, 'GET profile.css'), url0, headers0)

request103 = createRequest(Test(103, 'GET screen.css'), url0, headers0)

request104 = createRequest(Test(104, 'GET print.css'), url0, headers0)

# ...
```

Note the use of the `createRequest` helper function, which was defined earlier. This function creates a `HTTPRequest` object and instruments its `GET`, `POST`, ..., methods to report call statistics against the supplied `Test`.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

Finally the TestRunner class. This section groups the requests into pages and defines each page as a method, sets the sleep interval between requests, and provides an instrumented method for the return of data from the tests:

```
  # A method for each recorded page.
  def page1(self):
    """GET / (requests 101-131)."""
    result = request101.GET('/', None,
      ( NVPair('Accept', 'text/html,application/xhtml+xml,application/xml;q=0.9,*/
*;q=0.8'), ))
    self.token_subject = \
      httpUtilities.valueFromBodyURI('subject') # 'Feedback on The Grinder web
 site index.h...'
    self.token_sitesearch = \
      httpUtilities.valueFromHiddenInput('sitesearch') # 'grinder.sourceforge.net'

    grinder.sleep(176)
    request102.GET('/skin/profile.css')

    request103.GET('/skin/screen.css')

    request104.GET('/skin/print.css')

    request105.GET('/skin/basic.css')

#.....

    return result

  def page2(self):

#.....

  def __call__(self):
    """Called for every run performed by the worker thread."""
    self.page1()      # GET / (requests 101-131)

    grinder.sleep(39)
    self.page2()      # GET project_users.js (requests 201-202)
    self.page3()      # GET pdfdoc.gif (requests 301-305)
    self.page4()      # GET sflogo.php (request 401)
    self.page5()      # GET external-link.gif (request 501)


# Instrument page methods.
Test(100, 'Page 1').record(TestRunner.page1)
Test(200, 'Page 2').record(TestRunner.page2)

#.....
```

Once you've recorded your script you have two methods that you can use to replay your script:

1. You can create a simple `grinder.properties` (../g3/properties.html) file and you can replay the recorded scenario with The Grinder. Your properties file should at least set `grinder.script` to `grinder.py`.
2. Alternately you can use the console to distribute your script to an agent and set it as the script to run (../g3/console.html#Script+tab) . Each agent will still need a simple `grinder.properties` (../g3/properties.html) file containing the console address, though you will not need to set the `grinder.script` property.

The recorded script grinder.py can be edited by hand to suit your needs.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

2.4.4.1 Generating a Clojure script

You can generate a Clojure script using `-http clojure` on the command line. For example:

```
java net.grinder.TCPProxy -http clojure -console
```

2.4.4.2 Altering the output with custom stylesheet

The TCPProxy HTTP filters installed with `-http`, `-http jython`, and `-http clojure`, each create their output by transforming an XML model of the HTTP request/response stream using an XLST stylesheet.

These standard stylesheets can be found in `etc`. You can use a stylesheet of of your own making to customise the output of the filter. You should pass the file name of your custom stylesheet as a command line argument directly after `-http`.

If you want to see the intermediate XML model you can use:

```
java net.grinder.TCPProxy -http etc/httpToXML.xsl -console
```

The model confirms to the XML schema `etc/tcpproxy-http.xsd`.

2.4.4.3 How to offset test numbers

If sometimes useful to offset test numbers for a test script when running several different scripts together, perhaps using the sequence (../g3/script-gallery.html#sequence.py) , or parallel (../g3/script-gallery.html#parallel.py) examples from the script gallery. This gives the tests contributed by each script a distinct range of test numbers, which is important because the test number uniquely identifies the test in the console and the data logs.

The HTTP TCPProxy filter allows the recording of a test script with off-setting test numbers. This is done using the `HTTPPlugin.initialTest` property, which can either be set directly on the command line, or in a file using the `-properties` option. Here's an example that will start the test numbers at 1000:

```
java -DHTTPPlugin.initialTest=1000  net.grinder.TCPProxy  -http
```

Its also simple to offset test values by modifying the script.

Edit the recorded script to replace:

```
from net.grinder.script import Test
```

with:

```
from net.grinder.script import Test as StandardTest

def Test(number, description):
    # Adjust the 1000 to the appropriate offset.
    return StandardTest(number + 1000, description)
```

Neither technique allows different test scripts to be merged together into one because you also have to alter the identifiers used for headers, URLs, pages, tokens, and so on. If you want to do this, you might consider a custom stylesheet.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

2.4.4.4 How to record additional headers

By default, the following HTTP headers are recorded from the HTTP stream.

- Accept
- Accept-Charset
- Accept-Encoding
- Accept-Language
- Cache-Control
- Referer
- User-Agent
- Content-Type
- If-Modified-Since
- If-None-Match

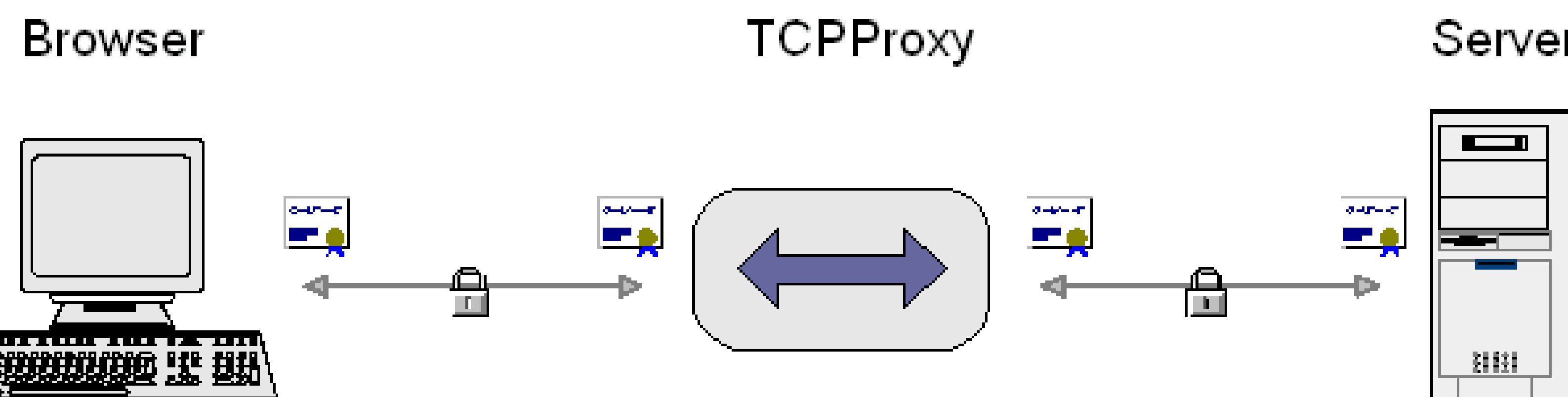
Additional headers can be specified with the `HTTPPlugin.additionalHeaders` system property. The value is a comma-separated list of header names. For example:

```
java net.grinder.TCPProxy -DHTTPPlugin.additionalHeaders=MyHeader,AnotherHeaderName -
http
```

2.4.5 SSL and HTTPS support

The TCPProxy has SSL support based on Java's JSSE (http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136007.html) framework.

SSL relationships are necessarily point to point. When you interpose the TCPProxy in SSL communications between a browser and a server you end up with two SSL connections. Each SSL connection has its own set of client and server certificates (both of which are optional).



The TCPProxy will negotiate appropriate certificates for both connections using built-in certificates or those from a user-specified Java key store. In particular, the TCPProxy needs a self-signed server certificate for the connection from the browser. By default, the TCPProxy will use a built-in certificate.

When first establishing a connection, your browser will present a warning and confirmation dialog. This is because the built-in certificate isn't authorised by any of the certificate authorities that the browser trusts. Additionally, the built-in certificate

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

authorises `localhost` so if your server doesn't listen at this address the browser will complain. Choose the "accept this certificate for this session" option.

> **Warning:**
> The Grinder deliberately accelerates SSL initialisation by using a random number generator that is seeded with a fixed number. This does not hinder SSL communication, but theoretically makes it less secure. No guarantee is made as to the cryptographic strength of any SSL communication using The Grinder.

2.4.5.1 Custom certificates

With more complicated pages, a browser may not give you the option to accept the test certificate. In this case, you can specify your own server certificate for the connection from the browser, or add client certificates for the connection to the server, using the `-keystore`, `-keystorepassword`, and `-keystoretype` options. See the J2SE/ JSSE documentation for how to set up a key store.

If you fail to provide a key store with a valid server certificate , you may get a *No available certificate corresponds to the SSL cipher suites which are enabled* exception, and your browser may report that it cannot communicate as it has no common encryption algorithms. Internet Explorer likes to be different. If start the TCPProxy without a valid server certificate and then connect through it using Internet Explorer, the TCPProxy will report "SSL peer shut down incorrectly. The browser will just spin away until it times out. The easiest way to provide a server certificate is to copy the *testkeys* file from the JSSE samples distribution (http://www.oracle.com/technetwork/java/jsse-136410.html) and start the proxy using:

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

Alfin Haji provided the following helpful write-up explaining how he solved a problem using a custom keystore:

The site we were testing had an embedded iframe that was making a call out to an HTTPS endpoint using an AJAX call via javascript. This endpoint was further making a call out to another HTTPS endpoint. The self-signed cert that Grinder was issuing was causing the following error to be thrown in developer tools of Chrome: net::ERR_INSECURE_RESPONSE. As a result, all the content in that iframe was blank and not being rendered (IE was throwing a content blocked error). IE developer tools was also throwing an error in developer tools that indicated the content was in mixed security format (HTTP and HTTPS) - SEC7111 "HTTPS security is compromised by [name of resource]".

Now since all traffic needs to go through a local proxy (TCPProxy), and since some of that traffic was secured, TCPProxy had to do a MITM in order to decrypt the secure traffic. However, since TCPProxy had an untrusted cert with hostnames not matching those endpoints that our app was calling out to, the browser generated an error.

Resolution: We created a self-signed cert using keytool.exe and we added the sites/ endpoints we was testing in the Subject Alternative Name section of the certificate. We then added the new certificate to the browser’s trust store:

1. Create certificate using keytool.exe and add the sites/endpoints you are testing that are blocking content from being shown in browser. Example below:

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

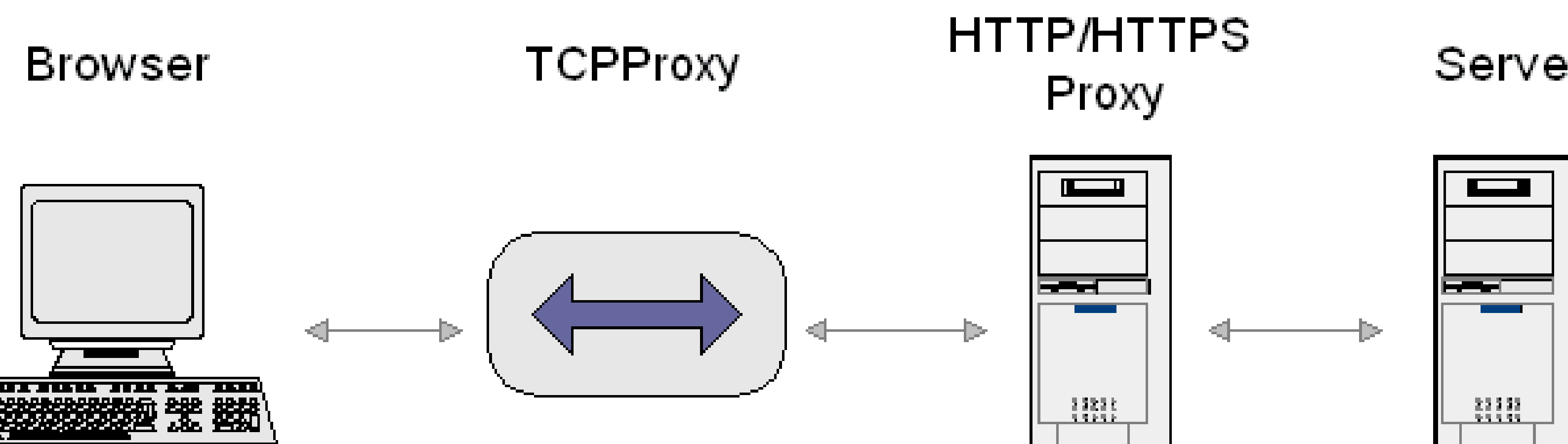
```
keytool -genkeypair -keystore keystore -dname "CN=test, OU=Unknown, O=Unknown,
 L=Unknown, ST=Unknown, C=Unknown" -storepass password -keyalg RSA -alias self-
signed-cert -ext SAN=dns:domain1,dns:domain2
```

You can add as many SANs as you want. Delimit them with ":" and if you are adding a DNS name, start with `dns:`

2. Launch tcpproxy with the keystore generated above: `java -classpath %CLASSPATH% net.grinder.TCPProxy -keyStore` *path to above keystore* `-keyStorePassword password -console -http > script.py`
3. Point your browser to the proxy, you will get a certificate error. IE for some reason didn't allow us to export the certificate, so we used Chrome. Export in base64 format.
4. Then in IE, imported the certificate to the trust store: *Internet Options* > *Content* > *Certificates* > *Trusted Root Certification Authorities* > *Import*. Browse to the exported certificate from step 3 above and import.
5. Restart the browser and navigate to the app. Your certificate should now be valid and content that was blocked should now be visible since the domains that were blocking the content are valid for the certificate provided (from step 1).

2.4.6 Using the TCPProxy with other proxies

The TCPProxy can be used with other HTTP/HTTPS proxies.



Use the `-httpproxy` option to specify the host name and port of the proxy. Use the `-httpsproxy` option only if your HTTPS proxy requires separate settings.

2.4.7 Using the TCPProxy as a port forwarder

It is normally most useful to use the TCPProxy in its HTTP Proxy mode as described above.

When using the TCPProxy as a debugging tool it occasionally is useful to use it in *port forwarding* mode. This mode is enabled when one or more of `-remotehost` and `-remoteport` are specified. In port forwarding mode, the TCPProxy simply listens on `localhost:localport` and forwards to `remotehost:remoteport`.

To understand why HTTP Proxy mode is usually better than port forwarding mode when using a browser, consider what happens if the remote server returns a page with an absolute URL link back to itself. If you click on the link, the browser will contact the server directly, bypassing the TCPProxy. Another disadvantage is that you can't use the TCPProxy with more than one remote sever.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

2.4.8 Summary of TCPProxy options

Option	Description
Commonly used options	
`-console`	Display a simple console that has a control button that allows The TCPProxy to be shut down cleanly. This can help in certain situations where a hard kill of the TCPProxy process would lose output that is still buffered in memory.
`-http [stylesheet]`	Adds a standard request filter and response filter to produce a Jython script for The Grinder suitable for use with the HTTP plugin. The default filter generates a Jython script and is equivalent to `-http jython`. Alternatively, use `clojure` to produce a Clojure script, or the output can be customised completely by providing the file name of an XSLT style sheet.
`-requestfilter filter`	Add a request filter. `filter` can be the name of a class that implements `net.grinder.tools.tcpproxy.TCPProxyFilter` or one of `NONE`, `ECHO`. The option can be specified multiple times, in which case the filters are invoked one after another. If the not specified, the default `ECHO` filter is used.
`-responsefilter filter`	Add a response filter. `filter` can be the name of a class that implements `net.grinder.tools.tcpproxy.TCPProxyFilter` or one of `NONE`, `ECHO`. The option can be specified multiple times, in which case the filters are invoked one after another. If the not specified, the default `ECHO` filter is used.
`-localhost host`	Set the host name or IP address to listen on. This must correspond to an interface of the machine the TCPProxy is started on. The default is `localhost`.
`-localport port`	Set the port to listen on. The default is `8001`.
`-keystore file`	Specify a custom key store. Usually the built-in keystore is good enough so `-keystore` does not need to be specified.
`-keystorepassword password`	Set the key store password. Only used if `-keystore` is set. Optional for some key store types.
`-keystoretype type`	Set the key store type. Only used if `-keystore` is set. If not specified, the default value depends on JSSE configuration but is usually `jks`.
Less frequently used options	
`-properties file`	Specify a file containing properties that are passed on to the filters.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

Option	Description
`-remotehost host`	Set the host name or port the TCPProxy should connect to in port forwarding mode. The TCPProxy starts in port forwarding mode if either –remotehost or -remoteport is set. The default is localhost.
`-remoteport port`	Set the port the TCPProxy should connect to in port forwarding mode. The TCPProxy starts in port forwarding mode if either -remotehost or -remoteport is set. The default is 7001.
`-timeout seconds`	Set an idle timeout. This is how long the TCPProxy will wait for a request before timing out and freeing the local port. The TCPProxy will not time out if there are active connections.
`-httpproxy host port`	Specify that output should be directed through another HTTP/HTTPS proxy. This may help you reach the Internet. This option is not supported in port forwarding mode.
`-httpsproxy host port`	Specify that output should be directed through a HTTPS proxy. Overrides any -httpproxy setting. This option is not supported in port forwarding mode.
`-ssl`	Use SSL in port forwarding mode. This will make both the TCPProxy's local socket and the connections to the target server use SSL. The default *HTTP Proxy mode* ignores this option and always listens as an HTTP proxy and an HTTPS proxy.
`-colour`	Specify that a simple colour scheme should be used to distinguish request streams from response schemes. This uses terminal control codes that only work on ANSI compliant terminals.
`-component class`	Register a component class with the filter PicoContainer.
`-debug`	Make PicoContainer chatty.

2.5 Scripts

2.5.1 Scripts

This section describes The Grinder 3 scripting API. If you've used The Grinder 2 for HTTP testing and you're not a programmer, you might be a bit daunted. Don't worry, its just as easy to record and replay HTTP scripts with The Grinder 3.

2.5.1.1 Jython and Python

The default scripting engine is Jython - the Java implementation of Python. Python is powerful, popular and easy on the eye. If you've not seen any Python before, take a look at the script gallery (../g3/script-gallery.html) and Richard Perks' tutorial (../g3/tutorial-

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

perks.html) to get a taste of what its like. There are plenty of resources on the web, here are a few of them to get you started:

- The Jython home page (http://www.jython.org/)
- The Python language web site (http://www.python.org/)
- Ten Python pitfalls (http://zephyrfalcon.org/labs/python_pitfalls.html)

I recommend the Jython Essentials (http://www.amazon.com/exec/obidos/tg/detail/-/0596002475/qid%3D1044795121/103-7145719-3118225) book; you can read the introductory chapter (http://www.oreilly.com/catalog/jythoness/chapter/ch01.html) for free.

Alternative languages

The Grinder 3.6 and later support test scripts written in Clojure (../g3/tcpproxy.html#clojure-script) .

Ryan Gardner has written an add-on script engine for Groovy (http://code.google.com/p/grinder-maven-plugin) .

2.5.1.2 Jython scripting

Script structure

Jython scripts must conform to a few conventions in order to work with The Grinder framework. I'll lay the rules out in fairly dry terms before proceeding with an example. Don't worry if this makes no sense to you at first, the examples are much easier to comprehend.

1. **Scripts must define a class called `TestRunner`**

   When a worker process starts up it runs the test script once. The test script must define a class called `TestRunner`. The Grinder engine then creates an instance of `TestRunner` for each worker thread. A thread's TestRunner instance can be used to store information specific to that thread.

   > Note:
   >
   > Although recommended, strictly `TestRunner` doesn't need to be a class. See the Hello World with Functions (../g3/script-gallery.html#helloworldfunctions.py) example.

2. **The `TestRunner` instance must be callable**

   A Python object is callable if it defines a `__call__` method. Each worker thread performs a number of *runs* of the test script, as configured by the property `grinder.runs`. For each run, the worker thread calls its `TestRunner`; thus the `__call__` method can be thought of as the definition of a run.
3. **The test script can access services through the `grinder` object**

   The engine makes an object called `grinder` available for the script to import. It can also be imported by any modules that the script calls. This is an instance of the Grinder.ScriptContext (../../g3/script-javadoc/net/grinder/script/Grinder.ScriptContext.html) class and provides access to context information (such as the worker thread ID) and services (such as logging and statistics).
4. **The script file name must end in `.py`**

   The file name suffix is used to identify Jython scripts.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

Canonical test script structure

This is an example of a script that conforms to the rules above. It doesn't do very much - every run will log *Hello World* to the worker process log.

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

# An instance of this class is created for every thread.
class TestRunner:
    # This method is called for every run.
    def __call__(self):
        # Per thread scripting goes here.
        grinder.logger.info("Hello World")
```

Automatically generating scripts

If you are creating a script for a website or web application, you can use the TCPProxy (../g3/tcpproxy.html#HTTPPluginTCPProxyFilter) to generate an HTTPPlugin script suitable for use with The Grinder.

2.5.1.3 Tests

Although our simple test script can be used with The Grinder framework and can easily be started in many times in many worker processes on many machines, it doesn't report any statistics. For this we need to create some tests. A Test (../../g3/script-javadoc/net/grinder/script/Test.html) has a unique test number and description. If you are using the console (../g2/console.html) , it will update automatically to display new `Tests` as they are created.

Let's add a `Test` to our script.

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

# Create a Test with a test number and a description.
test1 = Test(1, "Log method")

class TestRunner:
    def __call__(self):
        grinder.logger.info("Hello World")
```

Here we have created a single `Test` with the test number *1* and the description *Log method*. Note how we import the `grinder` object and the `Test` class in a similar manner to Java.

Now the console knows about our `Test`, but we're still not using it to record anything. Let's record how long our `grinder.logger.info` method takes to execute. `Test.record` adds the appropriate instrumentation code to the byte code of method. The time taken and the number of calls will be recorded and reported to the console.

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

test1 = Test(1, "Log method")

# Instrument the info() method with our Test.
test1.record(grinder.logger.info)

class TestRunner:
    def __call__(self):
        grinder.logger.info("Hello World")
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

This is a complete test script that works within The Grinder framework and reports results to the console.

You're not restricted to instrument method calls. In fact, it's more common to instrument objects. Here's an example using The Grinder's HTTP plug-in (../g3/http-plugin.html) .

```
# A simple example using the HTTP plugin that shows the retrieval of a
# single page via HTTP.

from net.grinder.script import Test
from net.grinder.script.Grinder import grinder
from net.grinder.plugin.http import HTTPRequest

test1 = Test(1, "Request resource")
request1 = HTTPRequest()
test1.record(request1)

class TestRunner:
    def __call__(self):
        result = request1.GET("http://localhost:7001/")
```

2.5.1.4 The Grinder script API

With what you've seen already you have the full power of Jython at your finger tips. You can use practically *any* Java or Python code in your test scripts.

The Grinder script API can be used to access services from The Grinder. The Javadoc (../../g3/script-javadoc/index.html) contains full information on all the packages, classes and interfaces that make up the core API, as well as additional packages added by the shipped plug-ins. This section provides overview information on various areas of the API. See also the HTTP plugin documentation (../g3/http-plugin.html) .

The net.grinder.script (../../g3/script-javadoc/net/grinder/script/package-summary.html) package

An instance of Grinder.ScriptContext (../../g3/script-javadoc/net/grinder/script/Grinder.ScriptContext.html) called `grinder` is automatically available to all scripts. This object provides access to context information and acts a starting point for accessing other services. The instance can be explicitly imported from other Python modules as `net.grinder.script.Grinder.grinder`.

We have described the use of the Test (../../g3/script-javadoc/net/grinder/script/Test.html) class above.

The Statistics (../../g3/script-javadoc/net/grinder/script/Statistics.html) interface allows scripts to query and modify statistics (../g3/statistics.html) , provide custom statistics, and register additional views of standard and custom statistics.

The net.grinder.common (../../g3/script-javadoc/net/grinder/common/package-summary.html) package

This package contains common interfaces and utility classes that are used throughout The Grinder and that are also useful to scripts.

2.5.1.5 Working directory

When the script has been distributed using the console, the working directory (CWD) of the worker process will be the local agent's cache of the distributed files. This allows the script to conveniently refer to other distributed files using relative paths.

Otherwise, the working directory of the worker process will be that of the agent process that started it.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

Distributing Java code

You can add Java `jar` or `.class` files to your console distribution directory and use the file distribution mechanism to push the code to the agent's cache. Use relative paths and the `grinder.jvm.classpath` property to add the files to the worker process `CLASSPATH`.

For example, you might distribute the following files

```
grinder.properties
myscript.py
lib/myfile.jar
```

where `grinder.properties` contains:

```
grinder.script=myscript.py
grinder.jvm.classpath=lib/myfile.jar
```

2.5.2 Jython

2.5.2.1 Scripts

The core requirements for Jython scripts can be found in the introduction (../g3/scripts.html#jython-scripts) .

Importing modules

Scripts can use code packaged in Jython modules (http://docs.python.org/tutorial/modules.html) . The Grinder adds both the directory containing the script and the working directory (../g3/scripts.html#cwd) of the worker process (which may be the same) to the Python path, allowing modules to be imported from these locations.

If you want to load modules from other locations, you should adjust the Python path. One way to do this is to set the JYTHONPATH (http://www.jython.org/docs/using/cmdline.html#environment-variables) environment variable.

2.5.2.2 The Jython distribution and installation

The Grinder 3.11 includes Jython 2.5.3 and the Jython implementation of the standard Python library.

Setting the Jython cache directory

A Jython bug prevents the correct calculation of a default cache directory. If you don't have a Jython cache directory, wild card imports of Java packages (e.g. `from java.util import *`) may not work, The Grinder will take a little longer to start, and ugly error messages will be displayed:

```
28/09/08 17:57:11 (agent): worker paston01-0 started
*sys-package-mgr*: can't create package cache dir, '/home/performance/lib/jython.jar/
cachedir/packages'
```

You can specify the cache directory either by setting the `python.home` as described below (in which case the directory will that specified in the Python registry), or by setting the Java property `python.cachedir` in your properties (../g3/properties.html) file:

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

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

or on the command line:

```
java -Dgrinder.jvm.arguments = -Dpython.cachedir=/tmp/mycache net.grinder.Grinder
```

You can only set `grinder.jvm.arguments` once, so if you want to set both the cache directory and `python.home` either use the registry, or do this:

```
grinder.jvm.arguments = -Dpython.home=/opt/jython/jython2.5.3 -Dpython.cachedir=/tmp/
mycache
```

Using an alternative Jython version.

If you want use a different version of Jython, you should place it at the start of the classpath used to start the agent process.

If you don't use its standalone option, the Jython installer will create a new directory containing the Jython jar file, the library modules, examples, and documentation. To use the standard library modules, you need to tell The Grinder the location of this directory. You can do this either by adding the following to your properties (../g3/properties.html) file:

```
grinder.jvm.arguments = -Dpython.home=/opt/jython/jython2.5.3
```

or on the agent command line:

```
java -Dgrinder.jvm.arguments=-Dpython.home=/opt/jython/jython2.5.3 net.grinder.Grinder
```

In both cases, change `/opt/jython/jython2.5.3` to the directory in which you installed Jython. You must install Jython on all of the agent machines. If the version of Jython is different to that included with The Grinder (2.5.3), you should also add the installation's `jython.jar` to the start of the `CLASSPATH` used to launch the agent.

Jython picks up user and site preferences from several sources (see http://www.jython.org/docs/registry.html). A side effect of setting `python.home` is that the installed registry file will be used.

2.5.3 Clojure

The Grinder 3.6 and later optionally support Clojure (http://clojure.org/) as an alternative language for test scripts.

2.5.3.1 How to use Clojure

Install Clojure and add the path to the installation's `clojure-x.x.x.jar` file to the start of the `CLASSPATH` you use for The Grinder agent processes.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

2.5.3.2 Clojure scripting

Script structure

Clojure scripts must conform to a few conventions in order to work with The Grinder framework.

1. **Scripts must return a function that creates test runner functions**

   When a worker process starts, it runs the test script once. The test script should return a factory function that creates and returns a *test runner function*.

   Each worker thread calls the factory function to create a test runner function. Worker threads perform a number of *runs* of the test script, as configured by the property `grinder.runs`. For each run, the worker thread calls its test runner function; thus the test runner function can be thought of as the definition of a run.
2. **The test script can access services through the `grinder` object**

   The engine makes an object called `grinder` available for the script to import. It can also be imported by any modules that the script calls. This is an instance of the Grinder.ScriptContext (../../g3/script-javadoc/net/grinder/script/Grinder.ScriptContext.html) class and provides access to context information (such as the worker thread ID) and services (such as logging and statistics).
3. **The script file name must end in `.clj`**

   The file name suffix is used to identify Clojure scripts.

Canonical test script structure

This is an example of a script that conforms to the rules above. It doesn't do very much - every run will log *Hello World* to the output log.

```
;; helloworld.clj
(let [grinder net.grinder.script.Grinder/grinder]

  ;; The script returns a factory function, called once by each worker
  ;; thread.
  (fn []

    ;; The factory function returns test runner function.
    (fn []
      (do
        (.. grinder (getLogger) (info "Hello World"))))))
```

Recording an HTTP script

You can use the TCPProxy to record a Clojure script (../g3/tcpproxy.html#clojure-script) from a browser session.

2.5.4 Script Instrumentation

2.5.4.1 About Instrumentation

The Grinder allows a script to mark the parts of the script code that should be recorded. This is called *instrumentation*.

Code is instrumented for a Test (../g3/scripts.html#tests) . When instrumented code is called, the test's statistics are updated. The standard statistics record the time taken, number of calls, and number of errors. Advanced scripts can add additional custom statistics (../g3/statistics.html) .

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

We've seen an example of using instrumentation in the introduction (../g3/ scripts.html#tests) . To recap, you instrument an object by using a `Test` to modify the Java byte code of the object. Here's the example code again.

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

test1 = Test(1, "Log method")

# Instrument the info() method with our Test.
test1.record(grinder.logger.info)

class TestRunner:
    def __call__(self):
        log("Hello World")
```

Each time "Hello World" is written to the log file, the time taken will be recorded by The Grinder.

Instrumentation can be *nested*. For example, you might instrument a method with Test 1, and the method code might call two `HTTPRequests` that are instrumented with Test 2 and Test 3. The code instrumented by Tests 2 and 3 is nested within the Test 1 code. The time recorded against the Test 1 will be greater than the total time recorded for Tests 2 and 3. It will also include any time spent in the function itself, for example calls to `grinder.sleep()`.

2.5.4.2 Supported targets

A wider range of target objects can be instrumented.

Java instance	Each call to a non-static method is recorded, including calls to super classes methods. Instances of arrays and primitive types cannot be instrumented.
Java class	Each call made to a constructor or a static method declared by the class is recorded. Calls of non-static methods or static methods defined by super classes are not recorded.
Jython instance	Each call to an instance method is recorded.
Jython function or method	Each call of the function or method is recorded.
Jython class	Each call made to the Jython class (i.e. constructor calls) is recorded.
Clojure function	Each call of the function is recorded.

JVM classes loaded in the bootstrap classloader, and classes from The Grinder's `net.grinder.engine.process` implementation package cannot be instrumented.

2.5.4.3 Selective instrumentation

The Grinder 3.7 adds an overloaded version of record (../../g3/script-javadoc/net/grinder/ script/Test.html#record(java.lang.Object, net.grinder.script.Test.InstrumentationFilter)) that allows the target object to be instrumented selectively.

Selective instrumentation is useful for instrumenting instances of the HTTPRequest (../../g3/script-javadoc/net/grinder/plugin/http/HTTPRequest.html) class, which has

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

ancillary methods that typically need to be called without affecting test statistics. Here's an example of how to use selective instrumentation.

```
from net.grinder.script import Test
from net.grinder.plugin.http import HTTPRequest

test = Test(1, "my test")

class GetAndPostFilter(Test.InstrumentationFilter):
  def matches(self, method):
    return method.name in ["GET", "POST"]

request = HTTPRequest(url="http://grinder.sourceforge.net")
test.record(request, GetAndPostFilter())

class TestRunner:
    def __call__(self):
        # GET() is instrumented, so call statistics are reported.
        request.GET()

        # getUrl() is not instrumented, no call statistics are reported.
        print "Called %s" % request.url
```

2.5.4.4 Troubleshooting Instrumentation

The instrumentation relises on Dynamic Code Redefinition, a Java 6 feature.

When you start an agent process, you will normally see a line like this in the worker process log file (../g3/getting-started.html#Output) .

```
    16/11/09 08:02:18 (process paston01-0): instrumentation agents:
    byte code transforming instrumenter for Jython 2.1/2.2; byte code transforming
 instrumenter for Java
```

If you see the following line, you should check you are using a Java 6 JVM.

```
      16/11/09 07:59:42 (process paston01-0): instrumentation agents: NO INSTRUMENTER
 COULD BE LOADED
```

2.5.5 Coordination

Most scripts are written so that their worker threads operate independently of each other. For web load generation, a worker thread corresponds to the actions of a single, independent user. Worker threads can generate unique data using methods such as getProcessNumber() (../../g3/script-javadoc/net/grinder/script/Grinder.ScriptContext.html#getProcessNumber()) and getThreadNumber() (../../g3/script-javadoc/net/grinder/script/Grinder.ScriptContext.html#getThreadNumber()) . Coordination of activity within a worker process can use standard Java or Jython synchronisation APIs.

Occasionally a script needs to coordinate worker threads across multiple worker processes. The Grinder supports this requirement through a distributed synchronisation feature, *barriers*.

2.5.5.1 Barriers

A barrier (../../g3/script-javadoc/net/grinder/script/Barrier.html) is a pre-arranged synchronisation point at which worker threads will wait for each other. There can be many synchronisation points; each uses a unique barrier name.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

Each worker thread that wants to participate in a synchronisation point should create a barrier with the given name using the ScriptContext (../../g3/script-javadoc/net/grinder/script/Grinder.ScriptContext.html#barrier(java.lang.String)) . The worker thread can wait for all other threads that have created barriers with a particular name by calling await (../../g3/script-javadoc/net/grinder/script/Barrier.html#await()) .

Barriers are usually created in the `TestRunner.__init__` constructor to ensure every worker thread has created its barriers before any of the threads try to wait for the barrier.

Sample script

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

class TestRunner:
  def __init__(self):
    # Each worker thread joins the barrier.
    self.phase1CompleteBarrier = grinder.barrier("Phase 1")

  def __call__(self):

    # ... Phase 1 actions.

    # Wait for all worker threads to reach this point before proceeding.
    self.phase1CompleteBarrier.await()

    # ... Further actions.
```

Barrier scope

Distributed barriers that allow coordination across worker processes require that the worker processes are started with the console.

Barriers are not shared across worker processes that are not started using the console, even if they are started by the same agent. In this case, each barrier will only provide coordination locally, between the worker threads of a worker process.

Barrier life cycle

A worker thread can reuse a barrier by calling await (../../g3/script-javadoc/net/grinder/script/Barrier.html#await()) again. The call will block until the other workers using barriers with the same name all call `await`.

A worker thread can wait for a limited time by using one of the versions of `await` that allow a timeout to be specified. If the timeout elapses, the barrier instance will be cancelled and become invalid. Other worker threads will no longer wait for the cancelled barrier. A new barrier can be created if required.

Worker threads can remove themselves from a synchronisation point by cancelling (../../g3/script-javadoc/net/grinder/script/Barrier.html#cancel()) a barrier directly.

2.5.6 Script Gallery

This page contains examples of Jython scripts and script snippets that can be used with The Grinder 3. The scripts can also be found in the `examples` directory of the distribution. To use one of these scripts, you'll need to set up a `grinder.properties` file. Please also make sure you are using the latest version of The Grinder 3.

If you're new to Python, it might help to know that that blocks are delimited by lexical indentation.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

The scripts make use of The Grinder script API. The `grinder` object in the scripts is an instance of `ScriptContext` through which the script can obtain contextual information (such as the worker process ID) and services (such as logging).

If you have a script that you would like to like to see to this page, please send it to grinder-use.

2.5.6.1 Hello World

```
# A minimal script that tests The Grinder logging facility.
#
# This script shows the recommended style for scripts, with a
# TestRunner class. The script is executed just once by each worker
# process and defines the TestRunner class. The Grinder creates an
# instance of TestRunner for each worker thread, and repeatedly calls
# the instance for each run of that thread.

from net.grinder.script.Grinder import grinder
from net.grinder.script import Test

# A shorter alias for the grinder.logger.info() method.
log = grinder.logger.info

# Create a Test with a test number and a description. The test will be
# automatically registered with The Grinder console if you are using
# it.
test1 = Test(1, "Log method")

# Instrument the info() method with our Test.
test1.record(log)

# A TestRunner instance is created for each thread. It can be used to
# store thread-specific data.
class TestRunner:

    # This method is called for every run.
    def __call__(self):
        log("Hello World")
```

2.5.6.2 Simple HTTP example

```
# A simple example using the HTTP plugin that shows the retrieval of a
# single page via HTTP. The resulting page is written to a file.
#
# More complex HTTP scripts are best created with the TCPProxy.

from net.grinder.script.Grinder import grinder
from net.grinder.script import Test
from net.grinder.plugin.http import HTTPRequest

test1 = Test(1, "Request resource")
request1 = HTTPRequest()
test1.record(request1)

class TestRunner:
    def __call__(self):
        result = request1.GET("http://localhost:7001/")

        # result is a HTTPClient.HTTPResult. We get the message body
        # using the getText() method.
        writeToFile(result.text)

# Utility method that writes the given string to a uniquely named file.
def writeToFile(text):
    filename = "%s-page-%d.html" % (grinder.processName, grinder.runNumber)

    file = open(filename, "w")
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
    print >> file, text
    file.close()
```

2.5.6.3 Recording many HTTP interactions as one test

```
# This example shows how many HTTP interactions can be grouped as a
# single test by wrapping them in a function.

from net.grinder.script.Grinder import grinder
from net.grinder.script import Test
from net.grinder.plugin.http import HTTPRequest
from HTTPClient import NVPair

# We declare a default URL for the HTTPRequest.
request = HTTPRequest(url = "http://localhost:7001")

def page1():
    request.GET('/console')
    request.GET('/console/login/LoginForm.jsp')
    request.GET('/console/login/bea_logo.gif')

Test(1, "First page").record(page1)

class TestRunner:
    def __call__(self):
        page1()
```

2.5.6.4 HTTP/J2EE form based authentication

```
# A more complex HTTP example based on an authentication conversation
# with the server. This script demonstrates how to follow different
# paths based on a response returned by the server and how to post
# HTTP form data to a server.
#
# The J2EE Servlet specification defines a common model for form based
# authentication. When unauthenticated users try to access a protected
# resource, they are challenged with a logon page. The logon page
# contains a form that POSTs username and password fields to a special
# j_security_check page.

from net.grinder.script.Grinder import grinder
from net.grinder.script import Test
from net.grinder.plugin.http import HTTPRequest
from HTTPClient import NVPair

protectedResourceTest = Test(1, "Request resource")
authenticationTest = Test(2, "POST to j_security_check")

request = HTTPRequest(url="http://localhost:7001/console")
protectedResourceTest.record(request)

class TestRunner:
    def __call__(self):
        result = request.GET()
        result = maybeAuthenticate(result)

        result = request.GET()

# Function that checks the passed HTTPResult to see whether
# authentication is necessary. If it is, perform the authentication
# and record performance information against Test 2.
def maybeAuthenticate(lastResult):
    if lastResult.statusCode == 401 \
    or lastResult.text.find("j_security_check") != -1:

        grinder.logger.info("Challenged, authenticating")

        authenticationFormData = ( NVPair("j_username", "weblogic"),
                                   NVPair("j_password", "weblogic"),)
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
        request = HTTPRequest(url="%s/j_security_check" % lastResult.originalURI)
        authenticationTest.record(request)

        return request.POST(authenticationFormData)
```

2.5.6.5 HTTP digest authentication

```
# Basically delegates to HTTPClient's support for digest
# authentication.
#
# Copyright (C) 2008 Matt Moran
# Copyright (C) 2008 Philip Aston
# Distributed under the terms of The Grinder license.

from net.grinder.plugin.http import HTTPPluginControl
from HTTPClient import AuthorizationInfo

# Enable HTTPClient's authorisation module.
HTTPPluginControl.getConnectionDefaults().useAuthorizationModule = 1

test1 = Test(1, "Request resource")
request1 = HTTPRequest()
test1.record(request1)

class TestRunner:
    def __call__(self):
        threadContextObject = HTTPPluginControl.getThreadHTTPClientContext()

        # Set the authorisation details for this worker thread.
        AuthorizationInfo.addDigestAuthorization(
            "www.my.com", 80, "myrealm", "myuserid", "mypw", threadContextObject)

        result = request1.GET('http://www.my.com/resource')
```

2.5.6.6 HTTP cookies

```
# HTTP example which shows how to access HTTP cookies.
#
# The HTTPClient library handles cookie interaction and removes the
# cookie headers from responses. If you want to access these cookies,
# one way is to define your own CookiePolicyHandler. This script defines
# a CookiePolicyHandler that simply logs all cookies that are sent or
# received.
#
# The script also demonstrates how to query what cookies are cached for
# the current thread, and how add and remove cookies from the cache.
#
# If you really want direct control over the cookie headers, you
# can disable the automatic cookie handling with:
#    HTTPPluginControl.getConnectionDefaults().useCookies = 0

from net.grinder.script.Grinder import grinder
from net.grinder.script import Test
from net.grinder.plugin.http import HTTPRequest, HTTPPluginControl
from HTTPClient import Cookie, CookieModule, CookiePolicyHandler
from java.util import Date

log = grinder.logger.info

# Set up a cookie handler to log all cookies that are sent and received.
class MyCookiePolicyHandler(CookiePolicyHandler):
    def acceptCookie(self, cookie, request, response):
        log("accept cookie: %s" % cookie)
        return 1

    def sendCookie(self, cookie, request):
        log("send cookie: %s" % cookie)
        return 1
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
CookieModule.setCookiePolicyHandler(MyCookiePolicyHandler())

test1 = Test(1, "Request resource")
request1 = HTTPRequest()
test1.record(request1)


class TestRunner:
    def __call__(self):
        # The cache of cookies for each  worker thread will be reset at
        # the start of each run.

        result = request1.GET("http://localhost:7001/console/?request1")

        # If the first response set any cookies for the domain,
        # they willl be sent back with this request.
        result2 = request1.GET("http://localhost:7001/console/?request2")

        # Now let's add a new cookie.
        threadContext = HTTPPluginControl.getThreadHTTPClientContext()

        expiryDate = Date()
        expiryDate.year += 10

        cookie = Cookie("key", "value","localhost", "/", expiryDate, 0)

        CookieModule.addCookie(cookie, threadContext)

        result = request1.GET("http://localhost:7001/console/?request3")

        # Get all cookies for the current thread and write them to the log
        cookies = CookieModule.listAllCookies(threadContext)
        for c in cookies: log("retrieved cookie: %s" % c)

        # Remove any cookie that isn't ours.
        for c in cookies:
            if c != cookie: CookieModule.removeCookie(c, threadContext)

        result = request1.GET("http://localhost:7001/console/?request4")
```

2.5.6.7 HTTP multipart form submission

```
# This script uses the HTTPClient.Codecs class to post itself to the
# server as a multi-part form. Thanks to Marc Gemis.

from net.grinder.script.Grinder import grinder
from net.grinder.script import Test
from net.grinder.plugin.http import HTTPRequest
from HTTPClient import Codecs, NVPair
from jarray import zeros

test1 = Test(1, "Upload Image")
request1 = HTTPRequest(url="http://localhost:7001/")
test1.record(request1)

class TestRunner:
    def __call__(self):

        files = ( NVPair("self", "form.py"), )
        parameters = ( NVPair("run number", str(grinder.runNumber)), )

        # This is the Jython way of creating an NVPair[] Java array
        # with one element.
        headers = zeros(1, NVPair)

        # Create a multi-part form encoded byte array.
        data = Codecs.mpFormDataEncode(parameters, files, headers)
        grinder.logger.output("Content type set to %s" % headers[0].value)
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
        # Call the version of POST that takes a byte array.
        result = request1.POST("/upload", data, headers)
```

2.5.6.8 Enterprise Java Beans

```
# Exercise a stateful session EJB from the Oracle WebLogic Server
# examples. Additionally this script demonstrates the use of the
# ScriptContext sleep(), getThreadId() and getRunNumber() methods.
#
# Before running this example you will need to add the EJB client and
# the WebLogic classes to your CLASSPATH.

from java.lang import String
from java.util import Properties,Random
from javax.naming import Context,InitialContext
from net.grinder.script.Grinder import grinder
from net.grinder.script import Test
from weblogic.jndi import WLInitialContextFactory

tests = {
    "home" : Test(1, "TraderHome"),
    "trade" : Test(2, "Trader buy/sell"),
    "query" : Test(3, "Trader getBalance"),
    }

# Initial context lookup for EJB home.
p = Properties()
p[Context.INITIAL_CONTEXT_FACTORY] = WLInitialContextFactory.name

home = InitialContext(p).lookup("ejb20-statefulSession-TraderHome")
tests["home"].record(home)

random = Random()

class TestRunner:
    def __call__(self):
        log = grinder.logger.info

        trader = home.create()
        tests["trade"].record(trader.sell)
        tests["trade"].record(trader.buy)
        tests["query"].record(trader.getBalance)

        stocksToSell = { "BEAS" : 100, "MSFT" : 999 }
        for stock, amount in stocksToSell.items():
            tradeResult = trader.sell("John", stock, amount)
            log("Result of trader.sell(): %s" % tradeResult)

        grinder.sleep(100)              # Idle a while

        stocksToBuy = { "BEAS" : abs(random.nextInt()) % 1000 }
        for stock, amount in stocksToBuy.items():
            tradeResult = trader.buy("Phil", stock, amount)
            log("Result of trader.buy(): %s" % tradeResult)

        balance = trader.getBalance()
        log("Balance is $%.2f" % balance)

        trader.remove()                 # We don't record the remove() as a test
```

2.5.6.9 Grinding a database with JDBC

```
# Some simple database playing with JDBC.
#
# To run this, set the Oracle login details appropriately and add the
# Oracle thin driver classes to your CLASSPATH.
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
from java.sql import DriverManager
from net.grinder.script.Grinder import grinder
from net.grinder.script import Test
from oracle.jdbc import OracleDriver

test1 = Test(1, "Database insert")
test2 = Test(2, "Database query")

# Load the Oracle JDBC driver.
DriverManager.registerDriver(OracleDriver())

def getConnection():
    return DriverManager.getConnection(
        "jdbc:oracle:thin:@127.0.0.1:1521:mysid", "wls", "wls")

def ensureClosed(object):
    try: object.close()
    except: pass

# One time initialisation that cleans out old data.
connection = getConnection()
statement = connection.createStatement()

try: statement.execute("drop table grinder_fun")
except: pass

statement.execute("create table grinder_fun(thread number, run number)")

ensureClosed(statement)
ensureClosed(connection)

class TestRunner:
    def __call__(self):
        connection = None
        insertStatement = None
        queryStatement = None

        try:
            connection = getConnection()
            insertStatement = connection.createStatement()

            test1.record(insertStatement)
            insertStatement.execute("insert into grinder_fun values(%d, %d)" %
                                    (grinder.threadNumber, grinder.runNumber))

            test2.record(queryStatement)
            queryStatement.execute("select * from grinder_fun where thread=%d" %
                                   grinder.threadNumber)

        finally:
            ensureClosed(insertStatement)
            ensureClosed(queryStatement)
            ensureClosed(connection)
```

2.5.6.10 Simple HTTP Web Service

```
# Calls an Amazon.com web service to obtain information about a book.
#
# To run this script you must install the standard Python xml module.
# Here's one way to do that:
#
#   1. Download and install Jython 2.1
#   2. Add the following line to grinder.properties (changing the path appropriately):
#          grinder.jvm.arguments=-Dpython.home=c:/jython-2.1
#   3. Add Jakarta Xerces (or one of the other parsers supported by
#      the xml module) to your CLASSPATH.
#
# You may also need to obtain your own Amazon.com web service license
# and replace the script text <insert license key here> with the
# license key, although currently that doesn't appear to be necessary.
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
from net.grinder.script.Grinder import grinder
from net.grinder.script import Test
from net.grinder.plugin.http import HTTPRequest
from HTTPClient import NVPair
from xml.dom import javadom
from org.xml.sax import InputSource

bookDetailsTest = Test(1, "Get book details from Amazon")
parser = javadom.XercesDomImplementation()

class TestRunner:
    def __call__(self):
        if grinder.runNumber > 0 or grinder.threadNumber > 0:
            raise RuntimeError("Use limited to one thread, one run; "
                               "see Amazon Web Services terms and conditions")

        request = HTTPRequest(url="http://xml.amazon.com/onca/xml")
        bookDetailsTest.record(request)

        parameters = (
            NVPair("v", "1.0"),
            NVPair("f", "xml"),
            NVPair("t", "webservices-20"),
            NVPair("dev-t", "<insert license key here>"),
            NVPair("type", "heavy"),
            NVPair("AsinSearch", "1904284000"),
            )

        bytes = request.POST(parameters).inputStream

        # Parse results
        document = parser.buildDocumentUrl(InputSource(bytes))

        result = {}

        for details in document.getElementsByTagName("Details"):
            for detailName in ("ProductName", "SalesRank", "ListPrice"):
                result[detailName] = details.getElementsByTagName(
                    detailName)[0].firstChild.nodeValue

        grinder.logger.info(str(result))
```

2.5.6.11 JAX-RPC Web Service

```
# Exercise a basic Web Service from the BEA WebLogic Server 7.0
# examples.
#
# Before running this example you will need to add the generated
# JAX-RPC client classes and webserviceclient.jar to your CLASSPATH.

from net.grinder.script.Grinder import grinder
from net.grinder.script import Test
from examples.webservices.basic.javaclass import HelloWorld_Impl
from java.lang import System

System.setProperty( "javax.xml.rpc.ServiceFactory",
 "weblogic.webservice.core.rpc.ServiceFactoryImpl")

webService = HelloWorld_Impl("http://localhost:7001/basic_javaclass/HelloWorld?WSDL")

port  = webService.getHelloWorldPort()
Test(1, "JAXP Port test").record(port)

class TestRunner:
    def __call__(self):
        result = port.sayHello(grinder.threadNumber, grinder.grinderID)
        grinder.logger.info("Got '%s'" % result)
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

2.5.6.12 XML-RPC Web Service

```
# A server should be running on the localhost. This script uses the
# example from
# http://xmlrpc-c.sourceforge.net/xmlrpc-howto/xmlrpc-howto-java-server.html
#
# Copyright (C) 2004 Sebastiï¿½n Fontana
# Distributed under the terms of The Grinder license.

from java.util import Vector
from java.lang import Integer
from net.grinder.script.Grinder import grinder
from net.grinder.script import Test

from org.apache.xmlrpc import XmlRpcClient

test1 = Test(1, "XML-RPC example test")
server_url = "http://localhost:8080/RPC2"

client = XmlRpcClient(server_url)
test1.record(client)

class TestRunner:
    def __call__(self):
        params = Vector()
        params.addElement(Integer(6))
        params.addElement(Integer(3))

        result = client.execute("sample.sumAndDifference", params)
        sum = result.get("sum")

        grinder.logger.info("SUM %d" % sum)
```

2.5.6.13 Hello World, with functions

```
# The Hello World example re-written using functions.
#
# In previous examples we've defined TestRunner as a class; calling
# the class creates an instance and calling that instance invokes its
# __call__ method. This script is for the Luddites amongst you and
# shows how The Grinder engine is quite happy as long as the script
# creates a callable thing called TestRunner that can be called to
# create another callable thing.

from net.grinder.script.Grinder import grinder
from net.grinder.script import Test

test1 = Test(1, "Log method")
test1.record(grinder.logger.info)

def doRun():
    grinder.logger.info("Hello World")

def TestRunner():
    return doRun
```

2.5.6.14 The script life cycle

```
# A script that demonstrates how the various parts of a script and
# their effects on worker threads.

# The "top level" of the script is called once for each worker
# process. Perform any one-off initialisation here. For example,
# import all the modules, set up shared data structures, and declare
# all the Test objects you will use.

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

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
from java.lang import System

# The totalNumberOfRuns variable is shared by all worker threads.
totalNumberOfRuns = 0

# An instance of the TestRunner class is created for each worker thread.
class TestRunner:

    # There's a runsForThread variable for each worker thread. This
    # statement specifies a class-wide initial value.
    runsForThread = 0

    # The __init__ method is called once for each thread.
    def __init__(self):
        # There's an initialisationTime variable for each worker thread.
        self.initialisationTime = System.currentTimeMillis()

        grinder.logger.info("New thread started at time %s" %
                            self.initialisationTime)

    # The __call__ method is called once for each test run performed by
    # a worker thread.
    def __call__(self):

        # We really should synchronise this access to the shared
        # totalNumberOfRuns variable. See JMS receiver example for how
        # to use the Python Condition class.
        global totalNumberOfRuns
        totalNumberOfRuns += 1

        self.runsForThread += 1

        grinder.logger.info(
            "runsForThread=%d, totalNumberOfRuns=%d, initialisationTime=%d" %
            (self.runsForThread, totalNumberOfRuns, self.initialisationTime))

        # You can also vary behaviour based on thread ID.
        if grinder.threadNumber % 2 == 0:
            grinder.logger.info("I have an even thread ID.")

    # Scripts can optionally define a __del__ method. The Grinder
    # guarantees this will be called at shutdown once for each thread
    # It is useful for closing resources (e.g. database connections)
    # that were created in __init__.
    def __del__(self):
        grinder.logger.info("Thread shutting down")
```

2.5.6.15 Accessing test statistics

```
# Examples of using The Grinder statistics API with standard
# statistics.

from net.grinder.script.Grinder import grinder
from net.grinder.script import Test
from net.grinder.plugin.http import HTTPRequest


class TestRunner:
    def __call__(self):
        request = HTTPRequest(url = "http://localhost:7001")
        Test(1, "Basic request").record(request)

        # Example 1. You can get the time of the last test as follows.
        result = request.GET("index.html")

        grinder.logger.info("The last test took %d milliseconds" %
                            grinder.statistics.forLastTest.time)


        # Example 2. Normally test results are reported automatically
        # when the test returns. If you want to alter the statistics
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
# after a test has completed, you must set delayReports = 1 to
# delay the reporting before performing the test. This only
# affects the current worker thread.
grinder.statistics.delayReports = 1

result = request.GET("index.html")

if grinder.statistics.forLastTest.time > 5:
    # We set success = 0 to mark the test as a failure. The test
    # time will be reported to the data log, but not included
    # in the aggregate statistics sent to the console or the
    # summary table.
    grinder.statistics.forLastTest.success = 0

# With delayReports = 1 you can call report() to explicitly.
grinder.statistics.report()

# You can also turn the automatic reporting back on.
grinder.statistics.delayReports = 0


# Example 3.
# getForCurrentTest() accesses statistics for the current test.
# getForLastTest() accesses statistics for the last completed test.

def page(self):
    resourceRequest =HTTPRequest(url = "http://localhost:7001")
    Test(2, "Request resource").record(resourceRequest)

    resourceRequest.GET("index.html");
    resourceRequest.GET("foo.css");

    grinder.logger.info("GET foo.css returned a %d byte body" %
                        grinder.statistics.forLastTest.getLong(
                              "httpplugin.responseLength"))

    grinder.logger.info("Page has taken %d ms so far" %
                        grinder.statistics.forCurrentTest.time)

    if grinder.statistics.forLastTest.time > 10:
        grinder.statistics.forCurrentTest.success = 0

    resourceRequest.GET("image.gif");

instrumentedPage = page
Test(3, "Page").record(instrumentedPage)

instrumentedPage(self)
```

2.5.6.16 Java Message Service - Queue Sender

```
# JMS objects are looked up and messages are created once during
# initialisation. This default JNDI names are for the WebLogic Server
# 7.0 examples domain - change accordingly.
#
# Each worker thread:
#  - Creates a queue session
#  - Sends ten messages
#  - Closes the queue session

from net.grinder.script.Grinder import grinder
from net.grinder.script import Test
from jarray import zeros
from java.util import Properties, Random
from javax.jms import Session
from javax.naming import Context, InitialContext
from weblogic.jndi import WLInitialContextFactory

# Look up connection factory and queue in JNDI.
properties = Properties()
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
properties[Context.PROVIDER_URL] = "t3://localhost:7001"
properties[Context.INITIAL_CONTEXT_FACTORY] = WLInitialContextFactory.name

initialContext = InitialContext(properties)

connectionFactory =
 initialContext.lookup("weblogic.examples.jms.QueueConnectionFactory")
queue = initialContext.lookup("weblogic.examples.jms.exampleQueue")
initialContext.close()

# Create a connection.
connection = connectionFactory.createQueueConnection()
connection.start()

random = Random()

def createBytesMessage(session, size):
    bytes = zeros(size, 'b')
    random.nextBytes(bytes)
    message = session.createBytesMessage()
    message.writeBytes(bytes)
    return message

test1 =  Test(1, "Send a message")

class TestRunner:
    def __call__(self):
        log = grinder.logger.info

        log("Creating queue session")
        session = connection.createQueueSession(0, Session.AUTO_ACKNOWLEDGE)

        sender = session.createSender(queue)
        test1.record(sender)

        message = createBytesMessage(session, 100)

        log("Sending ten messages")

        for i in range(0, 10):
            sender.send(message)
            grinder.sleep(100)

        log("Closing queue session")
        session.close()
```

2.5.6.17 Java Message Service - Queue Receiver

```
# JMS objects are looked up and messages are created once during
# initialisation. This default JNDI names are for the WebLogic Server
# 7.0 examples domain - change accordingly.
#
# Each worker thread:
#  - Creates a queue session
#  - Receives ten messages
#  - Closes the queue session
#
# This script demonstrates the use of The Grinder statistics API to
# record a "delivery time" custom statistic.
#
# Copyright (C) 2003, 2004, 2005, 2006 Philip Aston
# Copyright (C) 2005 Dietrich Bollmann
# Distributed under the terms of The Grinder license.

from java.lang import System
from java.util import Properties
from javax.jms import MessageListener, Session
from javax.naming import Context, InitialContext
from net.grinder.script.Grinder import grinder
from net.grinder.script import Test
from threading import Condition
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
from weblogic.jndi import WLInitialContextFactory

# Look up connection factory and queue in JNDI.
properties = Properties()
properties[Context.PROVIDER_URL] = "t3://localhost:7001"
properties[Context.INITIAL_CONTEXT_FACTORY] = WLInitialContextFactory.name

initialContext = InitialContext(properties)

connectionFactory =
 initialContext.lookup("weblogic.examples.jms.QueueConnectionFactory")
queue = initialContext.lookup("weblogic.examples.jms.exampleQueue")
initialContext.close()

# Create a connection.
connection = connectionFactory.createQueueConnection()
connection.start()

# Add two statistics expressions:
# 1. Delivery time:- the mean time taken between the server sending
#    the message and the receiver receiving the message.
# 2. Mean delivery time:- the delivery time averaged over all tests.
# We use the userLong0 statistic to represent the "delivery time".

grinder.statistics.registerDataLogExpression("Delivery time", "userLong0")
grinder.statistics.registerSummaryExpression(
              "Mean delivery time",
                        "(/ userLong0(+ timedTests untimedTests))")

# We record each message receipt against a single test. The
# test time is meaningless.
def recordDeliveryTime(deliveryTime):
    grinder.statistics.forCurrentTest.setValue("userLong0", deliveryTime)

Test(1, "Receive messages").record(recordDeliveryTime)

class TestRunner(MessageListener):

    def __init__(self):
        self.messageQueue = []          # Queue of received messages not yet recorded.
        self.cv = Condition()           # Used to synchronise thread activity.

    def __call__(self):
        log = grinder.logger.info

        log("Creating queue session and a receiver")
        session = connection.createQueueSession(0, Session.AUTO_ACKNOWLEDGE)

        receiver = session.createReceiver(queue)
        receiver.messageListener = self

        # Read 10 messages from the queue.
        for i in range(0, 10):

            # Wait until we have received a message.
            self.cv.acquire()
            while not self.messageQueue: self.cv.wait()
            # Pop delivery time from first message in message queue
            deliveryTime = self.messageQueue.pop(0)
            self.cv.release()

            log("Received message")

            # We record the test a here rather than in onMessage
            # because we must do so from a worker thread.
            recordDeliveryTime(deliveryTime)

        log("Closing queue session")
        session.close()

        # Rather than over complicate things with explict message
        # acknowledgement, we simply discard any additional messages
        # we may have read.
        log("Received %d additional messages" % len(self.messageQueue))
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
    # Called asynchronously by JMS when a message arrives.
    def onMessage(self, message):
        self.cv.acquire()

        # In WebLogic Server JMS, the JMS timestamp is set by the
        # sender session. All we need to do is ensure our clocks are
        # synchronised...
        deliveryTime = System.currentTimeMillis() - message.getJMSTimestamp()

        self.messageQueue.append(deliveryTime)

        self.cv.notifyAll()
        self.cv.release()
```

2.5.6.18 Using The Grinder with other test frameworks

```
# Example showing how The Grinder can be used with HTTPUnit.
#
# Copyright (C) 2003, 2004 Tony Lodge
# Copyright (C) 2004 Philip Aston
# Distributed under the terms of The Grinder license.

from net.grinder.script.Grinder import grinder
from net.grinder.script import Test

from com.zaplet.test.frontend.http import HttpTest

# These correspond to method names on the test class.
testNames = [ "testRedirect",
              "testRefresh",
              "testNegativeLogin",
              "testLogin",
              "testPortal",
              "testHeader",
              "testAuthoringLink",
              "testTemplateDesign",
              "testSearch",
              "testPreferences",
              "testAboutZaplet",
              "testHelp",
              "testLogoutLink",
              "testNavigationFrame",
              "testBlankFrame",
              "testContentFrame",
              "testLogout", ]

tests=[]

for name, i in zip(testNames, range(len(testNames))):
  t = HttpTest(name)
  Test(i, name).record(t)
  tests.append(t)

# A TestRunner instance is created for each thread. It can be used to
# store thread-specific data.
class TestRunner:
    def __call__(self):
        for t in tests:
            result = t.run()
```

2.5.6.19 Email

```
# Send email using Java Mail (http://java.sun.com/products/javamail/)
#
# This Grinder Jython script should only be used for legal email test
# traffic generation within a lab testbed environment. Anyone using
# this script to generate SPAM or other unwanted email traffic is
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
# violating the law and should be exiled to a very bad place for a
# very long time.
#
# Copyright (C) 2004 Tom Pittard
# Copyright (C) 2004-2008 Philip Aston
# Distributed under the terms of The Grinder license.

from net.grinder.script.Grinder import grinder
from net.grinder.script import Test

from java.lang import System
from javax.mail import Message, Session
from javax.mail.internet import InternetAddress, MimeMessage


emailSendTest1 = Test(1, "Email Send Engine")

class TestRunner:
    def __call__(self):
        smtpHost = "mailhost"

        properties = System.getProperties()
        properties["mail.smtp.host"] = smtpHost
        session = Session.getInstance(System.getProperties())
        session.debug = 1

        message = MimeMessage(session)
        message.setFrom(InternetAddress("TheGrinder@yourtestdomain.net"))
        message.addRecipient(Message.RecipientType.TO,
                             InternetAddress("you@yourtestdomain.net"))
        message.subject = "Test email %s from thread %s" % (grinder.runNumber,
                                                              grinder.threadNumber)

        # One could vary this by pointing to various files for content
        message.setText("SMTPTransport Email works from The Grinder!")

        transport = session.getTransport("smtp")

        # Instrument transport object.
        emailSendTest1.record(transport)

        transport.connect(smtpHost, "username", "password")
        transport.sendMessage(message,
                              message.getRecipients(Message.RecipientType.TO))
        transport.close()
```

2.5.6.20 Run test scripts in sequence

```
# Scripts are defined in Python modules (helloworld.py, goodbye.py)
# specified in grinder.properties:
#
#   script1=helloworld
#   script2=goodbye

from net.grinder.script.Grinder import grinder

from java.util import TreeMap

# TreeMap is the simplest way to sort a Java map.
scripts = TreeMap(grinder.properties.getPropertySubset("script"))

# Ensure modules are initialised in the process thread.
for module in scripts.values(): exec("import %s" % module)

def createTestRunner(module):
    exec("x = %s.TestRunner()" % module)
    return x

class TestRunner:
    def __init__(self):
        self.testRunners = [createTestRunner(m) for m in scripts.values()]
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
    # This method is called for every run.
    def __call__(self):
        for testRunner in self.testRunners: testRunner()
```

2.5.6.21 Run test scripts in parallel

```
# Run TestScript1 in 50% of threads, TestScript2 in 25% of threads,
# and TestScript3 in 25% of threads.

from net.grinder.script.Grinder import grinder

scripts = ["TestScript1", "TestScript2", "TestScript3"]

# Ensure modules are initialised in the process thread.
for script in scripts: exec("import %s" % script)

def createTestRunner(script):
    exec("x = %s.TestRunner()" % script)
    return x

class TestRunner:
    def __init__(self):
        tid = grinder.threadNumber

        if tid % 4 == 2:
            self.testRunner = createTestRunner(scripts[1])
        elif tid % 4 == 3:
            self.testRunner = createTestRunner(scripts[2])
        else:
            self.testRunner = createTestRunner(scripts[0])

    # This method is called for every run.
    def __call__(self):
        self.testRunner()
```

2.5.6.22 Thread ramp up

```
# A simple way to start threads at different times.
#

from net.grinder.script.Grinder import grinder

def log(message):
    grinder.logger.info(message)

class TestRunner:
    def __init__(self):
        log("initialising")

    def initialSleep( self):
        sleepTime = grinder.threadNumber * 5000  # 5 seconds per thread
        grinder.sleep(sleepTime, 0)
        log("initial sleep complete, slept for around %d ms" % sleepTime)

    def __call__( self ):
        if grinder.runNumber == 0: self.initialSleep()

        grinder.sleep(500)
        log("in __call__()")
```

2.5.6.23 Hello World in Clojure

```
;; A simple Clojure script.
(let [grinder net.grinder.script.Grinder/grinder]
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
; The script returns a factory function, called once by each worker
; thread.
(fn []

  ; The factory function returns test runner function.
  (fn []
    (do
      (.. grinder (getLogger) (info "Hello World"))))))
```

2.6 Plug-ins

2.6.1 The HTTP Plug-in

2.6.1.1 What's it for?

The HTTPPlugin is a mature plug-in for testing HTTP services. It has a number of utilities useful for HTTP scripts as well as a tool, the TCPProxy (../g3/tcpproxy.html#HTTPPluginTCPProxyFilter) , which allows HTTP scripts to be automatically recorded. Recorded scripts are often customised, for example to simulate multiple users. This requires you to know a little about writing scripts (../g3/scripts.html) .

The HTTPPlugin is built into The Grinder and is automatically initialised whenever a script imports one of its classes. For example:

```
from net.grinder.plugin.http import HTTPRequest
```

The key class provided by the plug-in is HTTPRequest (../../g3/script-javadoc/net/grinder/plugin/http/HTTPRequest.html) . The best way to see how to use this class is to record a script with the TCPProxy.

The plug-in wires itself into The Grinder script life cycle. It maintains a cache of connections and cookies for each worker thread which it resets at the beginning of each run. Each run performed by a worker thread simulates a browser session carried out by a user. Resetting a thread's cookies at the beginning of a run will cause server applications that use cookie-based tracking to create a new session.

If your server application uses some other mechanism for session tracking (e.g. URL rewriting or hidden parameters), the script will have to capture and resend the appropriate token. The TCPProxy goes to some lengths to identify and record these tokens.

If an `HTTPRequest` is instrumented with a `Test`, the plug-in contributes additional statistics, including the HTTP status code, the response body length, and additional connection timing information. These statistics appear in the console and are recorded to the process data log. If several `HTTPRequests` are instrumented within the same `Test` (e.g. they are called within an instrumented function), the status code of the last response is recorded.

2.6.1.2 Controlling the HTTPPlugin

The behaviour of the plug-in can be controlled from within scripts run by The Grinder through the use of the HTTPPluginControl (../../g3/script-javadoc/net/grinder/plugin/http/HTTPPluginControl.html) facade.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

Levels of Control

There are three levels of control of the behaviour of the HTTPPlugin that the HTTPPluginControl facade gives you access to:

1. **Default Connection Behaviour**
    - Method: `getConnectionDefaults`
    - Returns a `HTTPPluginConnection` that can be used to set the default behaviour of new connections.
2. **Thread Connection Behaviour**
    - Method: `getThreadConnection`
    - Returns a `HTTPPluginConnection` for a particular URL.
    - The resulting `HTTPPluginConnection` is valid for the current thread and the current run. It can be used to set specific authentication details, default headers, cookies, proxy servers, and so on for the current thread/run on a per-URL basis.
    - This method will throw a `GrinderException` if not called from a worker thread.
3. **Thread HTTPClient Context Object Behaviour**
    - Method: `getThreadHTTPClientContext`
    - Returns the HTTPClient context object for the calling worker thread. This is useful when calling HTTPClient methods directly, e.g. `CookieModule.listAllCookies(Object)`.
    - This method will throw a `GrinderException` if not called from a worker thread.

Importing the HTTPPluginControl

Place the following line at the top of your grinder script along with your other import statements

```
from net.grinder.plugin.http import HTTPPluginControl
```

Setting HTTPClient Authorization Module

The HTTPClient Authorization module is no longer enabled by default because it prevents raw authentication headers being sent through. The module also slows things down as HTTPClient must parse responses for challenges.

Users who still wish to use the HTTPClient Authorization module can enable it using:

```
control = HTTPPluginControl.getConnectionDefaults()
control.setUseAuthorizationModule(1)
```

The authentication details can be set using the AuthorizationInfo (../../g3/script-javadoc/HTTPClient/AuthirizationInfo.html) API. HTTPClient maintains authentication information separately in each context, so the API must be called by each worker thread. See the Digest Authentication sample (../g3/script-gallery.html#digestauthentication.py) in the script gallery, as well as the example in the next section.

Setting an HTTP proxy

Should you need to specify an HTTP proxy to route requests through the following code can be used to specify the default proxy.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
control = HTTPPluginControl.getConnectionDefaults()
control.setProxyServer("localhost", 8001)
```

HTTP proxies can also be specified at the thread connection level. This is useful to set proxies on a per URL basis.

```
proxyURL1 = HTTPPluginControl.getThreadConnection("http://url1")
proxyURL2 = HTTPPluginControl.getThreadConnection("http://url2")
proxyURL1.setProxyServer("localhost", 8001)
proxyURL2.setProxyServer("localhost", 8002)
```

If the HTTP proxy requires authentication, enable the HTTPClient Authorization Module, as described in the previous section. Having so, each worker thread can set up the appropriate authentication details using the AuthorizationInfo (../../g3/script-javadoc/HTTPClient/AuthirizationInfo.html) API. For example:

```
from net.grinder.plugin.http import HTTPRequest, HTTPPluginControl
from HTTPClient import AuthorizationInfo

defaults = HTTPPluginControl.getConnectionDefaults()
defaults.useAuthorizationModule = 1
defaults.setProxyServer("localhost", 3128)

class TestRunner:
  def __init__(self):
    AuthorizationInfo.addBasicAuthorization("localhost",
                                            8001,
                                            "My Proxy Realm",
                                            "joeuser",
                                            "pazzword",

 HTTPPluginControl.getThreadHTTPClientContext())

  def __call__(self):
    # ...
```

Setting HTTP Headers

The HTTPPlugin allows you to set the HTTP Headers sent with requests. The method takes the settings as header-name/value pairs

```
control = HTTPPluginControl.getConnectionDefaults()
control.setDefaultHeaders(NVPair("header-name", "value"),))
```

Typical headers you might want to set here are `Accept` and its `Accept-*` relatives, `Connection`, `From`, `User-Agent`, etc.

For example to disable persistent connections:

```
control = HTTPPluginControl.getConnectionDefaults()
control.setDefaultHeaders(NVPair("Connection", "close"),))
```

Setting Encoding

Encoding for Content or for Transfer can be switched on and off using boolean flags

```
control = HTTPPluginControl.getConnectionDefaults()
control.setUseContentEncoding(0)
control.setUseTransferEncoding(1)
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

Setting Redirect Behaviour

Setting the HTTPPlugin behaviour with regards to following redirects can be switched on and off using boolean flags

```
control = HTTPPluginControl.getConnectionDefaults()
control.setFollowRedirects(0)
```

Setting Local Address

Should you be conducting your tests on a server with multiple network interfaces you can set the local IP address used by the HTTPPlugin for outbound connections.

```
control = HTTPPluginControl.getConnectionDefaults()
control.setLocalAddress("192.168.1.77")
```

Setting Timeout Value

The timeout value for used for creating connections and reading responses can be controlled via the HTTPPlugin. The time is specified in milliseconds.

The following example sets a default timeout value of 30 seconds for all connections.

```
control = HTTPPluginControl.getConnectionDefaults()
control.setTimeout(30000)
```

Setting Cookie Behaviour

Setting the HTTPPlugin behaviour with regards to whether cookies are used or not can be switched on and off using boolean flags

```
control = HTTPPluginControl.getConnectionDefaults()
control.setUseCookies(0)
```

Automatic decompression of gzipped responses

For load testing, its often not practical to uncompress the response. It's simply too expensive in CPU terms to do all that decompression in the client worker process. This doesn't mean you can't test a server that compresses its responses, just that you can't parse the responses in the script.

On the other hand, there are times you may want to do this. The Grinder supports decompression which it inherits from the `HTTPClient` library, you just need to enable it. If your server encrypts the content and sets a `Content-Encoding` header that starts with one of { `gzip`, `deflate`, `compress`, `identity` }, you can automatically decrypt the responses by adding the following lines to the beginning of your script:

```
from net.grinder.plugin.http import HTTPPluginControl
connectionDefaults = HTTPPluginControl.getConnectionDefaults()
connectionDefaults.useContentEncoding = 1
```

Similarly, if your server sets a `Transfer-Encoding` header that starts with one of { `gzip`, `deflate`, `compress`, `chunked`, `identity` }, you can enable the HTTPClient Transfer Encoding Module with `connectionDefaults.useTransferEncoding = 1`.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

There is no support for automatically decrypting things based on their `Content-Type` (as opposed to `Content-Encoding`, `Transfer-Encoding`). Your browser doesn't do this, so neither should The Grinder. If you really want to do this, you can use Java or Jython decompression libraries from your script.

Streaming requests and response

The HTTPRequest (../../g3/script-javadoc/net/grinder/plugin/http/HTTPRequest.html) class has support for sending request data from a stream. This allows an arbitrarily large amount of data to be sent without requiring a corresponding amount of memory. To do this, use these versions of the POST (../../g3/script-javadoc/net/grinder/plugin/http/HTTPRequest.html#POST(java.lang.String, java.io.InputStream)) , PUT (../../g3/script-javadoc/net/grinder/plugin/http/HTTPRequest.html#POST(java.lang.String, java.io.InputStream)) , OPTIONS (../../g3/script-javadoc/net/grinder/plugin/http/HTTPRequest.html#POST(java.lang.String, java.io.InputStream)) , methods.

`HTTPRequest` allows the response body to be handled as a stream. Refer to the Javadoc for the setReadResponseBody (../../g3/script-javadoc/net/grinder/plugin/http/HTTPRequest.html#setReadResponseBody(boolean)) method for more details.

2.6.1.3 Using HTTPUtilities

The HTTPPlugin provides an `HTTPUtilties` class:

```
net.grinder.plugin.http.HTTPUtilities
```

This class has several methods which are useful for HTTP scripts.

Setting Basic Authorization

The HTTPUtilities class can create an NVPair for an HTTP Basic Authorization header using the following method:

```
httpUtilities = HTTPPluginControl.getHTTPUtilities()
httpUtilities.basicAuthorizationHeader('username', 'password')
```

Include the header with each HTTPRequest that requires the authentication.

```
request101.GET('/', (),
      ( httpUtilities.basicAuthorizationHeader('prelive', 'g3tout'), ))
```

Getting the Last Response

The HTTPUtilities class can return the response for the last request made by the calling worker thread using the following method:

```
httpUtilities = HTTPPluginControl.getHTTPUtilities()
httpUtilities.getLastResponse()
```

This returns the response, or `null` if the calling thread has not made any requests.

This must be called from a worker thread, if not it throws a GrinderException.

Getting a Token Value from a Location URI

The HTTPUtilities class can return the value for a path parameter or query string name-value token with the given `tokenName` in a Location header from the last response. If

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

there are multiple matches, the first value is returned. This utility can be invoked using the following method:

```
httpUtilities = HTTPPluginControl.getHTTPUtilities()
httpUtilities.valueFromLocationURI(tokenName)
```

If there is no match, an empty string is returned rather than `null`. This makes scripts more robust (as they don't need to check the value before using it), but they lose the ability to distinguish between a missing token and an empty value.

This must be called from a worker thread, if not it throws a GrinderException.

Getting a Token Value from a URI in the Body of the Response

The HTTPUtilities class can return the value for a path parameter or query string name-value token with the given `tokenName` in a URI in the body of the last response. If there are multiple matches, the first value is returned. This utility can be invoked using the following method:

```
httpUtilities = HTTPPluginControl.getHTTPUtilities()
httpUtilities.valueFromBodyURI(tokenName)
```

This returns the first value if one is found, or `null`.

This must be called from a worker thread, if not it throws a GrinderException.

2.7 Statistics

2.7.1 Standard statistics

Details of the statistics provided by The Grinder can be found in the documentation of the Statistics (../../g3/script-javadoc/net/grinder/script/Statistics.html) interface. Scripts can use this interface to:

- Query whether a test was successful
- Obtain statistic values, such as the test time of the last test
- Modify or set a test's statistics before they are sent to the log and the console
- Report custom statistics
- Register additional views of standard and custom statistics

2.7.2 Distribution of statistics

All the statistics displayed in the console are aggregates (totals or averages) of a number of tests received in the appropriate period. The reason for this is efficiency. The Grinder would not perform or scale if every data point was transferred back to the console.

The only place per-test statistics are available is in the process `data_*` files.

2.7.3 Querying and updating statistics

A script can query the statistics about the last completed test using grinder.statistics.forLastTest (../../g3/script-javadoc/net/grinder/script/Statistics.html#getForLastTest()) . Script code instrumented by a test can access information about the statistics for the test (which may be incomplete) using grinder.statistics.forCurrentTest (../../g3/script-javadoc/net/grinder/script/Statistics.html#getForCurrentTest()) . For details of the query and update

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

methods, see StatisticsForTest (../../g3/script-javadoc/net/grinder/script/Statistics.StatisticsForTest.html) . Refer to the documentation of the Statistics (../../g3/script-javadoc/net/grinder/script/Statistics.html) interface for other details.

An example script (../g3/script-gallery.html#statistics.py) demonstrating these APIs can be found in the Script Gallery.

2.7.4 Registering new expressions

Custom statistic expressions can be added to console views and the worker process summary tables (found in the `out_*` log files) using the registerSummaryExpression (../../g3/script-javadoc/net/grinder/script/Statistics.html#registerSummaryExpression(java.lang.String, java.lang.String)) method.

Custom expressions can be added to worker process `data_*` using the registerDataLogExpression (../../g3/script-javadoc/net/grinder/script/Statistics.html#registerDataLogExpression(java.lang.String, java.lang.String)) method.

Both methods take a *displayName* and an *expression* as parameters.

The *displayName* is the label used for the expression. For expressions displayed in the console, this string is converted to a key for an internationalised resource bundle look up by prefixing the string with `statistic.` and replacing any whitespace with underscores; if no value for the key exists, the raw display name string is used.

Expressions are composed of statistic names (see Statistics (../../g3/script-javadoc/net/grinder/script/Statistics.html)) in a simple post-fix format using the symbols +, -, / and *, which have their usual meanings, in conjunction with simple statistic names or sub-expressions. Precedence is controlled by grouping expressions in parentheses. For example, the error rate is `(* (/ errors period) 1000)` errors per second. The symbol `sqrt` can be used to calculate the square root of an expression.

Sample statistics, such as `timedTests`, must be introduced with one of `sum`, `count`, or `variance`, depending on the attribute of interest. For example, the statistic `expression (/ (sum timedTests) (count timedTests))` gives the mean test time in milliseconds.

2.8 SSL Support

The Grinder 3 supports the use of SSL by scripts. The Grinder 3 implements SSL using the Java Secure Socket Extension (JSSE) included in the Java run time. When used with the HTTP Plug-in, this is as simple as using `https` instead of `http` in URIs. Scripts can obtain a suitable `SSLContext` and hence a `SSLSocketFactory` for non-HTTP use cases, and can control the allocation of SSL sessions to worker threads.

2.8.1 Before we begin

2.8.1.1 Performance

Simulating multiple SSL sessions on a single test machine may or may not be realistic. A typical browser running on a desktop PC has the benefit of a powerful CPU to run the SSL cryptography. Be careful that your results aren't constrained due to inadequate test client CPU power.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

2.8.1.2 The Grinder's SSL implementation is not secure

To reduce the client side performance overhead, The Grinder deliberately accelerates SSL initialisation by using a random number generator that is seeded with a fixed number. Further, no validation of server certificates is performed. Neither of these hinder SSL communication, but they do make it less secure.

> **Warning:**
> No guarantee is made as to the cryptographic strength of any SSL communication using The Grinder.

This acceleration affects initialisation time only and should not affect timing information obtained using The Grinder.

2.8.2 Controlling when new SSL sessions are created

By default The Grinder creates a new SSL session for each run carried out by each worker thread. This is in line with the usual convention of simulating a user session with a worker thread executing the part of the script defined by `TestRunner.__call__()`.

Alternatively, scripts may wish to have an SSL session per worker thread, i.e. for each thread to reuse SSL sessions on subsequent executions of `TestRunner.__call__()`. This can be done with the `SSLControl.setShareContextBetweenRuns()` method:

```
from net.grinder.script.Grinder import grinder
grinder.SSLControl.shareContextBetweenRuns = 1
```

This will cause each worker thread to reuse SSL sessions between runs. SSL sessions will still not be shared between worker threads. Calling `setShareContextBetweenRuns()` affects all of the worker threads.

2.8.3 Using client certificates

If a server requests or requires a client certificate, The Grinder must have some way of providing one - this involves specifying a key store.

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

class TestRunner:
    def __call__(self):
        grinder.SSLControl.setKeyStoreFile("mykeystore.jks", "passphrase")
```

It is only valid to use `setKeyStoreFile` from a worker thread, and it only affects that worker thread.

There is also a method called `setKeyStore` which takes a `java.io.InputStream` which may be useful if your key store doesn't live on the local file system. Both methods have an overloaded version that allows the key store type to be specified, otherwise the default type is used (normally `jks`).

Whenever `setKeyStoreFile`, `setKeyStore`, or `setKeyManagers` (see below) is called, the current SSL session for the thread is discarded. Consequently, you usually want to call these methods at the beginning of your `__call__()` method or from the `TestRunner.__init__()` constructor. Setting the thread's key

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

store in `TestRunner.__init__()` is especially recommended if you calling `setShareContextBetweenRuns(true)` to share SSL sessions between runs.

2.8.4 FAQ

The astute reader who is familiar with key stores may have a few questions. Here's a mini FAQ:

1. *If I have several suitable certificates in my key store, 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`.
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`.
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.
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.

2.8.5 Picking a certificate from a key store [Advanced]

Here's an example script that provides its own `X509KeyManager` implementation which controls which client certificate to use. The example is hard coded to always use the certificate with the alias `myalias`.

```
from com.sun.net.ssl import KeyManagerFactory,X509KeyManager
from java.io import FileInputStream
from java.security import KeyStore
from jarray import array

class MyManager(X509KeyManager):
    def __init__(self, keyStoreFile, keyStorePassword):
        keyStore = KeyStore.getInstance("jks")
        keyStore.load(FileInputStream(keyStoreFile), keyStorePassword)

        keyManagerFactory = \
         KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm())
        keyManagerFactory.init(keyStore, keyStorePassword)

        # Assume we have one key manager.
        self._delegate = keyManagerFactory.keyManagers[0]

    def __getattr__(self, a):
        """Some Python magic to pass on all invocations of methods we
        don't define on to our delegate."""

        if self.__dict__.has_key(a): return self.__dict__[a]
        else: return getattr(self._delegate, a)
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
    def chooseClientAlias(self, keyTypes, issuers):
        return "myalias"

myManager = MyManager("keystore.jks", "password")
myManagerArray = array((myManager,), X509KeyManager)

class TestRunner:
    def __call__(self):
        grinder.SSLControl.setKeyManagers(myManagerArray)
        # ...
```

2.8.6 Debugging

When debugging SSL interactions, you may find it useful to set the following in `grinder.properties`.

```
grinder.jvm.arguments=-Djavax.net.debug=ssl
# or -Djavax.net.debug=all
```

2.9 Advice

2.9.1 How should I set up a project structure for The Grinder?

Well the short answer is however works best for you. Many people will already know how they want to set up their directory structure and will have no issue implementing The Grinder as one of their many tools. For those looking for a little guidance it is worth asking yourself questions like:

- How many projects will I be working on?
- Will I need to revisit projects from time to time?
- Do I need repeatability?
- Is this a shared implementation?
- ...etc.

Below is given an example of a directory structure for setting up The Grinder.

```
.
`- Grinder
    |
    |-- bin
    |   |-- setGrinderEnv.sh/cmd
    |   |-- startAgent.sh/cmd
    |   |-- startConsole.sh/cmd
    |   `-- startProxy.sh/cmd
    |
    |-- engine
    |   |-- grinder-3.0-beta32
    |   |-- grinder-3.0
    |   `-- ...
    |
    |-- etc
    |   |-- grinder.properties
    |   `-- ...
    |
    |-- jvm
    |   |-- jdk1.3
    |   |-- jdk1.4.02
    |   `-- ...
    |
    |-- lib
    |   |-- jython2.1
    |   |-- jdom-1.0
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
|    |-- xerces_2_6_0
|    |-- xerces-2_6_2
|    |-- oracle
|    `-- ...
|
|-- logs
|    `-- ...
|
`-- projects
     |-- website_project
     |    |-- httpscript.py
     |    |-- httpscript_tests.py
     |    `-- ...
     |
     |-- db_project
     |    |-- jdbc.py
     |    `-- ...
     |
     `-- ...
```

First off the **bin** directory has been created for storing executable files for the implementation. The sample start scripts from "How do I start The Grinder?" (../ g3/getting-started.html#howtostart) have been included in this directory. The **engine** directory has been created for storing the versions of The Grinder that may be used. Strictly speaking the versions of The Grinder could be stored under the **lib** directory but for this example The Grinder has been given its own directory. The **etc** directory has been created to store the configuration files for the implementation such as the grinder.properties file. The **jvm** directory has been created to store the various jdks and their versions that could be used in testing. The **lib** directory has been created to store the various third party libraries and their respective versions that projects may require. For example if you wanted to use the full set of libraries (../g3/jython.html#jython-installation) which come with jython then this is the directory into which you would install. Remember to update your CLASSPATH with the libraries you require. The **logs** has been created to store the various logs that the grinder generates during its runs.The **projects** directory has been created to store the scripts to be run by The Grinder and organise them by project/body of work.

The above example would be useful as a simple implementation for one person who works on one project at a time. As the number of projects grows, more people share the implementation, or projects need to be revisited with repeatability ensured, then it makes sense, in this example, to modularize the implementation around the projects. To do this simply create the **bin**, **etc** and **logs** directories under the respective projects like so:

```
|
`-- projects
     |-- website_project
     |    |-- bin
     |    |    |-- setGrinderEnv.sh/cmd
     |    |    |-- startAgent.sh/cmd
     |    |    |-- startConsole.sh/cmd
     |    |    `-- startProxy.sh/cmd
     |    |-- etc
     |    |    |-- grinder.properties
     |    |    `-- ...
     |    |-- httpscript.py
     |    |-- httpscript_tests.py
     |    |-- logs
     |    |    `-- ...
     |    `-- ...
     |
     |-- db_project
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

Once this has been done the environment can be set to use the engine, JVM and libraries required by a particular project, rather than setting the environment for all the projects (as would happen in the simple implementation). This allows you, for example, to retain projects which were run using legacy versions of libraries and/or engine and re-run them at a later date with the same setup. Also different projects may require different versions of the same library which would have caused issues when using an implementation-wide CLASSPATH. The grinder.properties file can also be customised on a per project basis.

Modularizing the implementation like this gives greater flexibility and repeatability and opens up the prospect of multiple people using the implementation concurrently.

2.9.2 A Step-By-Step Script Tutorial

2.9.2.1 Introduction

The is a step-by-step tutorial of how to write a number of dynamic HTTP tests using various aspects of The Grinder and Jython APIs. The test script contains a number of tests that are requests to the same URL. For each request, a different XML parameter is specified. The resulting HTML data is checked on return and if the test was not successful, the statistics API is used to mark that test as failed.

Richard Perks

2.9.2.2 Script Imports

```
import string
import random
from java.lang import String
from java.net import URLEncoder
from net.grinder.script import Test
from net.grinder.plugin.http import HTTPRequest
from net.grinder.common import GrinderException
```

Firstly when writing a script come the import statements. These include imports of standard Python modules such as `string` and `random`, and other Java imports including some language and network classes. Finally there are imports for Grinder specific methods. A powerful feature of the Jython scripts that are used with The Grinder is the ability to take a mix and match approach to script programming. In some cases using a Python API is quicker and easier than always using the corresponding Java API calls, so feel free to use whichever API makes most sense.

2.9.2.3 Test Definition

```
tests = {
    "News01"      : Test(1, "News 1 posting"),
    "Sport01"     : Test(2, "Sport 1 posting"),
    "Sport02"     : Test(3, "Sport 2 posting"),
    "Trading01"   : Test(4, "Trading 1 query"),
    "LifeStyle01" : Test(5, "LifeStyle 1 posting"),
}
```

To keep the script code easy to read, we next define all the tests we are going to be running within this script. These are created as a Python dictionary and are name-value pairs. The name is the name of the test and the value is a `Test` object with a test numeric identifier and description.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

2.9.2.4 Bread crumbs

```
log = grinder.logger.info

# Server Properties
SERVER     = "http://serverhost:7001"
URI        = "/myServlet"
```

We next define some variables such as Grinder helper methods and server properties. The `log` variable is used to hold a reference to The Grinder logging mechanism and is used throughout the script.

2.9.2.5 The Test Interface

```
class TestRunner:
    def __call__(self):
```

Here is the definition of our test class and the method called by The Grinder by each test thread. All scripts must define this class and method. Whilst we are discussing classes and methods, an important point to remember when new to Jython script development is that Jython/Python code is scoped by indentation, rather than using braces like in a language like C or Java. The colon is used to delimit the start scope such as an `if` or method definition.

2.9.2.6 Using the Dictionary and Random Python Modules

```
for idx in range(len(tests)):
       testId = random.choice(tests.keys())
       log("Reading XML file %s " % testId)
```

As discussed earlier, the use of Python modules is encouraged during Grinder script development and I have used a few examples above when performing the test run. Within the test run, each of the tests defined in the test dictionary is looped round so that each Grinder thread executes five separate tests. Within the loop, a test is chosen randomly from one of the five tests. This prevents all threads of executing all the tests in the same order and helps simulate a more random load on the server.

Within the dictionary defined as `tests`, there are a number of useful methods such as are `keys()`, `items()` and `sort()`. We use the keys returned from the tests dictionary as the parameter to the `choice()` method in the random module. This randomly selects one of the tests keys as the current test identifier.

2.9.2.7 Forget the Java IO Package when Handling Files

```
file = open("./CAAssets/"+testId+".xml", 'r')
fileStr = URLEncoder.encode(String(file.read()))
file.close()

requestString = "%s%s%s%s" % (SERVER, URI, "?xmldata=", fileStr)
```

When having to retrieve the contents of files using Jython script, the use of the file operations blitz's Java IO for pure script development speed. In the code above, we need to open an XML document that has the name of a test, for example `News01.xml`. This will be used as a request parameter for the News01 test. The file is opened for reading and encoded using the Java URLEncoder.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

We next construct the request string to the server by concatenating the server, URI and XML documents together. *Tip*: if you need to remove spaces from within a string, you can use a method like the following:

```
requestString = string.join(requestString.split(), "")
```

2.9.2.8 Sending the Request and the Statistics API

```
grinder.statistics.delayReports = 1
request = HTTPRequest()
tests[testId].record(request)

log("Sending request %s " % requestString)
result = request.GET(requestString)
```

As part of the test execution, we want the ability to check the result of the HTTP request. If the response back from the server is not one that we except, we want to mark the test as unsuccessful and not include the statistics in the test times. To do this, the `delayReports` variable can be set to 1. Doing so will delay the reporting back of the statistics until after the test has completed and we have had chance to check its operation. The default is to report back when the test returns control back to the script, i.e. immediately after a test has executed.

Next we instrument the HTTPRequest with the test being executed. This enables any calls through HTTPRequest to be monitored by the Grinder. Any other time spent within the script will not be recorded by The Grinder. Be careful not to include extra script processing within a test; doing so will not give the correct statistics. Only test what is required.

The test itself is next executed which is a HTTP GET to the server using our previously constructed test string. Remember - these tests execute in a loop for the number of tests we have defined, using a random test each time.

```
if string.find(result.getText(), "SUCCESS") < 1:
    grinder.statistics.forLastTest.setSuccess(0)
    writeToFile(result.getText(), testId)
```

On return from the HTTP GET, we check the result for the string "SUCCESS". If the test has failed, this value will not be returned and the statistics object can be marked as unsuccessful. In the case of an unsuccessful test, we write the HTML output to a file for later analysis:

```
def writeToFile(text, testId):
    filename = "%s-%d-page-%d.html" % (grinder.processName,
                                       testId,
                                       grinder.runNumber)

    file = open(filename, "w")
    print >> file, text
    file.close()
```

2.9.2.9 Full Script Listing

```
# Send an HTTP request to the server with XML request values

import string
```

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
import random
from java.lang import String
from java.net import URLEncoder

from net.grinder.script import Test
from net.grinder.plugin.http import HTTPRequest
from net.grinder.common import GrinderException

tests = {
    "News01"      : Test(1, "News 1 posting"),
    "Sport01"     : Test(2, "Sport 1 posting"),
    "Sport02"     : Test(3, "Sport 2 posting"),
    "Trading01"   : Test(4, "Trading 1 query"),
    "LifeStyle01" : Test(5, "LifeStyle 1 posting"),
}

log = grinder.logger.info
out = grinder.logger.TERMINAL

# Server Properties

SERVER     = "http://serverhost:7001"
URI        = "/myServlet"

class TestRunner:
    def __call__(self):

        for idx in range(len(tests)):

           testId = random.choice(tests.keys())

           log("Reading XML file %s " % testId)

           file = open("./CAAssets/"+testId+".xml", 'r')
           fileStr = URLEncoder.encode(String(file.read()))
           file.close()

           # Send the request to the server
           requestString = "%s%s%s%s" % (SERVER, URI, "?xmldata=", fileStr)
           requestString = string.join(requestString.split(), "")

           grinder.statistics.delayReports = 1
           request = HTTPRequest()
           tests[testId].record(request)

           log("Sending request %s " % requestString)
           result = request.GET(requestString)

           if string.find(result.getText(), "SUCCESS") < 1:
              grinder.statistics.forLastTest.setSuccess(0)
              writeToFile(result.getText(), testId)


# Write the response
def writeToFile(text, testId):
    filename = "%s-%d-page-%d.html" % (grinder.processName,
                                       testId,
                                       grinder.runNumber)
    file = open(filename, "w")
    print >> file, text
    file.close()
```

2.9.3 Weighted Distribution Of Tests

2.9.3.1 Introduction

This is a step-by-step tutorial on how to schedule tests according to any "weight distribution" you desire. This is an exercise in data structures and random numbers, and as such it does not use any facilities of The Grinder (such as HTTPClient) except for its

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

core TestRunner functionality. Therefore, it is immediately applicable to almost any test scenario.

Walt Tuvell

2.9.3.2 Statement Of The Problem

Let's assume you have a collection of four kinds of tests you want to run, say CREATE, READ, UPDATE, DELETE. These might be operations on a Web Server, or a Database Server, for example.

Suppose further you want to run your tests using many threads (grinder.threads property in the grinder.properties file), and you want to schedule these threads amongst the tests according to a specified "weighted distribution". As an example, we'll assume you want to run: 20% CREATEs, 40% READs, 30% UPDATEs, 10% DELETEs.

How can you do this?

2.9.3.3 Test Cases

Note that the problem statement is independent of the actual tests themselves. So for illustrative purposes, we will choose dummy tests that simply print a message to stdout. (Your tests will likely make use of deeper facilities of The Grinder, such as HTTPClient, etc.)

```
def doCREATEtest():
    print 'Doing CREATE test ...'
def doREADtest():
    print 'Doing READ   test ...'
def doUPDATEtest():
    print 'Doing UPDATE test ...'
def doDELETEtest():
    print 'Doing DELETE test ...'
```

2.9.3.4 Weight Distribution Definition

The most flexible way to define test distribution is by means of "relative weights", that is, numbers which specify the number of times each test is to be run relative to one another (as opposed to an absolute number of runs - for that, see the grinder.runs property in the grinder.properties file).

For our example, we begin by defining our desired weight distribution in a table (Jython dictionary structure) like the following:

```
g_Weights = {
    'CREATE': 2,
    'READ'  : 4,
    'UPDATE': 3,
    'DELETE': 1,
}
```

Since the weights in this table are relative, we could multiply all their values by a constant and arrive at the same weight distribution. (The same goes for division, provided we end up with integers.) If the sum of weights adds up to 100, the weights can be interpreted directly as "percentages". For example, in our example, if we multiplied our weights by 10, we'd end up with exactly the percentage values in the original statement of our example.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

Note that string-names in the weight table are arbitrary tags (they will be mapped to the tests in TestRunner.__call__()). As a matter of style, the weight table should be placed near the top of your script, so its settings can be modified easily from run to run, according to the test scenarios you want to model.

2.9.3.5 Accumulator Function

All the magic of choosing which test to run according to your specified weight distribution is accomplished by the following "accumulator" function:

```
def weightAccumulator(i_dict):
    keyList = i_dict.keys()
    keyList.sort()  # sorting is optional - order coming-in doesn't matter, but
determinism is kinda cool
    listAcc = []
    weightAcc = 0
    for key in keyList:
        weightAcc += i_dict[key]
        listAcc.append((key, weightAcc))
    return (listAcc, weightAcc)  # order going-out does matter - hence "listAcc"
instead of "dictAcc"

g_WeightsAcc, g_WeightsAccMax = weightAccumulator(g_Weights)
g_WeightsAccLen, g_WeightsAccMax_1 = len(g_WeightsAcc), g_WeightsAccMax-1
```

This accumulator function takes a weight dictionary as input, and transforms it into an accumulated weight list, suitable for random indexing, as we will do below.

As shown above, the accumulator function is called with g_Weights as input, and its output is captured in two convenience variables. Two more convenience variables are also defined, for use below.

2.9.3.6 Random Numbers

Next, we prepare a random number generator, which we will use to index into our accumulated weight list. There are many choices available (including Jython), but for our purposes here we'll just use the Java standard generator:

```
g_rng = java.util.Random(java.lang.System.currentTimeMillis())

def randNum(i_min, i_max):
    assert i_min <= i_max
    range = i_max - i_min + 1  # re-purposing "range" is legal in Python
    assert range <= 0x7fffffff  # because we're using java.util.Random
    randnum = i_min + g_rng.nextInt(range)
    assert i_min <= randnum <= i_max
    return randnum
```

Here, we've constructed a random number generator, and seeded it with the time-of-day. (For test/simulation purposes, it is counterproductive to use secure random number generators, such as java.security.SecureRandom, or a secure seed source.)

Further, we've defined a randNum() function that takes minimum and maximum values as input, and returns a random number between them (inclusive of both endpoints).

Note: One advantage of using java.util.Random is that it's thread-safe, so we need construct only a single global generator. But that safety comes at the expense of some performance loss, especially if you are using the generator extensively, such as generating

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

massive random file/object content. In that case, you may want to use faster, non-thread-safe generators, constructing one for each thread's private use.

2.9.3.7 Test Runner Class

We are now ready to define our TestRunner class:

```
class TestRunner:
    def __call__(self):
        opNum = randNum(0, g_WeightsAccMax_1)
        opType = None  # flag for assertion below
        for i in range(g_WeightsAccLen):
            if opNum < g_WeightsAcc[i][1]:
                opType = g_WeightsAcc[i][0]
                break
        assert opType in g_Weights.keys()

        if   opType=='CREATE': doCREATEtest()
        elif opType=='READ'  : doREADtest()
        elif opType=='UPDATE': doUPDATEtest()
        elif opType=='DELETE': doDELETEtest()
        else                 : assert False
```

According to The Grinder framework, every worker thread calls TestRunner.__call__() in an infinite loop (until it terminates). In our case, for each run, each thread first chooses a random number, opNum, and then uses that random number to index into the accumulated weight list. (Well, it's not exactly "indexing" in the array or database access sense, but the idea is the same.) This results in the tag of an operation type, opType, to be called. The thread then maps the operation type tag to a test, and calls it.

2.9.3.8 Putting It All Together

Our example script is now complete, so we can run it.

Let's say we want to do 10,000 runs. In your grinder.properties file, set grinder.threads=20, grinder.runs=500. Then invoke startAgent.sh. You'll see 10,000 lines printed, each saying "Doing XXX test ...", where XXX is one of CREATE, READ, UPDATE, DELETE.

But did you get the weighted distribution of test cases you wanted? For that, you need to count various lines printed out by the test. In a Linux environment, you can do this conveniently by rerunning the test in a pipeline command as follows:

```
startAgent.sh |  \
    awk '/^Doing /{count[$0]+=1} END{for (test in count) print test, count[test]}'
```

A typical run of this command will produce results similar to the following:

```
Doing CREATE test ... 2006
Doing READ   test ... 4045
Doing UPDATE test ... 2993
Doing DELETE test ... 956
```

Inspection of these numbers shows you are indeed running the distribution you desired.

2.9.3.9 Full Script Listing

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
import java.lang.System, java.util.Random


g_Weights = {
    'CREATE': 2,
    'READ'  : 4,
    'UPDATE': 3,
    'DELETE': 1,
}


def doCREATEtest():
    print 'Doing CREATE test ...'
def doREADtest():
    print 'Doing READ   test ...'
def doUPDATEtest():
    print 'Doing UPDATE test ...'
def doDELETEtest():
    print 'Doing DELETE test ...'


def weightAccumulator(i_dict):
    keyList = i_dict.keys()
    keyList.sort()  # sorting is optional - order coming-in doesn't matter, but
 determinism is kinda cool
    listAcc = []
    weightAcc = 0
    for key in keyList:
        weightAcc += i_dict[key]
        listAcc.append((key, weightAcc))
    return (listAcc, weightAcc)  # order going-out does matter - hence "listAcc"
 instead of "dictAcc"

g_WeightsAcc, g_WeightsAccMax = weightAccumulator(g_Weights)
g_WeightsAccLen, g_WeightsAccMax_1 = len(g_WeightsAcc), g_WeightsAccMax-1


g_rng = java.util.Random(java.lang.System.currentTimeMillis())

def randNum(i_min, i_max):
    assert i_min <= i_max
    range = i_max - i_min + 1  # re-purposing "range" is legal in Python
    assert range <= 0x7fffffff  # because we're using java.util.Random
    randnum = i_min + g_rng.nextInt(range)
    assert i_min <= randnum <= i_max
    return randnum


class TestRunner:
    def __call__(self):
        opNum = randNum(0, g_WeightsAccMax_1)
        opType = None  # flag for assertion below
        for i in range(g_WeightsAccLen):
            if opNum < g_WeightsAcc[i][1]:
                opType = g_WeightsAcc[i][0]
                break
        assert opType in g_Weights.keys()

        if   opType=='CREATE': doCREATEtest()
        elif opType=='READ'  : doREADtest()
        elif opType=='UPDATE': doUPDATEtest()
        elif opType=='DELETE': doDELETEtest()
        else                 : assert False
```

2.9.4 Garbage Collection

2.9.4.1 Introduction

For high transactional workloads, a significant component of The Grinder's response time can include the performance of the Java Garbage Collector (GC), which is a necessary component of the Java Virtual Machine (JVM) that The Grinder workers run on.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

This page documents the improvements obtained by tuning garbage collection for a particular test configuration.

Gary Mulder

2.9.4.2 Testing

Comparison tests were performed for an identical complex test suite (5 million requests over 4 hours) with The Grinder deployed firstly on a single two quad core dual socket (i.e. 8 core) server with 12GB of RAM, and secondly on four dual core PCs with 4GB of RAM each (i.e. the same number of CPUs, but twice the sockets and so twice the memory bandwidth). All test variables were attempted to be controlled for, and the only significant change was The Grinder hardware used.

On the latter 4 PC configuration response times reported were 25% lower on average, and more significantly standard deviations of response times were 25% lower as well. The key changes between test scenarios were the change in JVM heap sizes from 1*8GB to 4*3GB, and the fact that four GCs were running simultaneously (i.e. one GC per JVM per PC). GC is very sensitive to memory bandwidth, so with four sockets (4 x 2 core) rather than two sockets (2 x 4 core) it is likely memory bandwidth for The Grinder was about doubled, which in turn reduced GC pause durations. Furthermore, with four GCs running simultaneously the times when The Grinder is subject to GC has been smoothed, which was directly reflected in the reduced response time standard deviations.

2.9.4.3 Conclusions

Conclusions are as follows. Your mileage may vary:

1. Use the Sun Hotspot JVM with CMS GC (not the Java 7 G1 GC which was observed to behave badly in some tests) with settings similar to the following (for 4GB dedicated PCs):

   ```
   grinder.jvm.arguments = -Xms3g -Xmx3g -XX:NewSize=2g -XX:MaxNewSize=2g
    -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseConcMarkSweepGC
    -XX:+UseParNewGC -XX:+ExplicitGCInvokesConcurrent -XX:+CMSConcurrentMTEnabled
    -XX:+AlwaysPreTouch
   ```

   `NewSize` is set proportionally large (67% of the heap size) as Jython seems to create a lot of short lived objects. CMS is used as it attempts to minimise GC pause times at the cost of transactional throughput. `Pretouch` is used to ensure the JVM is less likely to be paused waiting for memory pages from the Linux kernel.
2. Scale your Grinder clients horizontally (i.e. lots of cheap PCs) rather than vertically (i.e. big expensive multi-socket servers).
3. Keep a very close eye on the GC times reported by each Grinder's GC log. If The Grinder starts timing a request, pauses for GC, and then ends timing a request, some unknown amount of GC time will be added to the response time reported. GC times of 200ms are not uncommon, and GC pauses of 5 seconds can be produced by poorly tuned GCs. Under Linux, to redirect GC logs from stdout invoke the Java worker as follows:

   ```
   java net.grinder.Grinder $GRINDERPROPERTIES >> worker_out.log 2>&1
   ```

   The `2>&1` also redirects (http://tldp.org/LDP/abs/html/io-redirection.html) any errors to `worker_out.log`.
   To directly specify a GC log add the following JVM argument

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

```
-Xloggc:/tmp/gc_log
```

Make sure the JVM can write to the log file specified.

2.10 Features of The Grinder 3

Thanks to **Edwin DeSouza** for his help in compiling this feature list.
Last updated: 4 October 2011

2.10.1 Capabilities of The Grinder

Load Testing	Load Testing determines if an application can support a specified load (for example, 500 concurrent users) with specified response times. Load Testing is used to create benchmarks.
Capacity Testing	Capacity Testing determines the maximum load that an application can sustain before system failure.
Functional Testing	Functional Testing proves the correct behaviour of an application.
Stress Testing	Stress Testing is load testing over an extended period of time. Stress Testing determines if an application can meet specified goals for stability and reliability, under a specified load, for a specified time period.

2.10.2 Open Source

BSD style license	The Grinder is distributed under a BSD style license.
Dependencies	The Grinder depends on a number of other open source products including • Jython (http://www.jython.org/) • HTTPClient (http://www.innovation.ch/java/HTTPClient/) • JEdit Syntax (http://syntax.jedit.org/) • Apache XMLBeans (http://xmlbeans.apache.org/) • PicoContainer (http://picocontainer.org/) • Clojure (http://clojure.org/)

2.10.3 Standards

100% Pure Java	The Grinder works on any hardware platform and any operating system that supports J2SE 1.4 and above.
Web Browsers	The Grinder can simulate web browsers and other devices that use HTTP, and HTTPS.
Web Services	The Grinder can be used to test Web Service interfaces using protocols such as SOAP and XML-RPC.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

Database	The Grinder can be used to test databases using JDBC.
Middleware	The Grinder can be used to test RPC and MOM based systems using protocols such as IIOP, RMI/IIOP, RMI/JRMP, and JMS.
Other Internet protocols	The Grinder can be used to test systems that utilise other protocols such as POP3, SMTP, FTP, and LDAP.

2.10.4 The Grinder Architecture

Goal	Minimize system resource requirements while maximizing the number of test contexts ("virtual users").
Multi-threaded, multi-process	Each test context runs in its own thread. The threads can be split over many processes depending on the requirements of the test and the capabilities of the load injection machine.
Distributed	The Grinder makes it easy to coordinate and monitor the activity of processes across a network of many load injection machines from a central console.
Scalable	The Grinder typically can support several hundred HTTP test contexts per load injection machine. (The number varies depending on the type of test client). More load injection machines can be added to generate bigger loads.

2.10.5 Console

Graphical Interface	100% Java Swing user interface.
Process coordination	Worker processes can be started, stopped and reset from one central console.
Process monitoring	Dynamic display of current worker processes and threads.
Internationalised and Localised	English, French, Spanish, and German translations are supplied. Users can add their own translations.
Script editing	Central editing and management of test scripts.

2.10.6 Statistics, Reports, Charts

Test monitoring	Pre-defined charts for response time, test throughput. Display the number of invocations, test result (pass/fail), average, minimum and maximum values for response time and tests per second for each test.
Data collation	Collates data from worker processes. Data can be saved for import into a spreadsheet or other analysis tool.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

Instrument anything	The Grinder records statistics about the number of times each test has been called and the response times achieved. Any part of the test script can be marked as a test.
Statistics engine	Scripts can declare their own statistics and report against them. The values will appear in the console and the data logs. Composite statistics can be specified as expressions involving other statistics.

2.10.7 Script

Record real users	Scripts can be created by recording actions of a real user using the TCP Proxy. The script can then be customised by hand.
Powerful scripting in Python	Simple to use but powerful, fully object-oriented scripting.
Multiple scenarios	Arbitrary looping and branching allows the simulation of multiple scenarios. Simple scenarios can be composed into more complex scenarios. For example, you might allocate 10% of test contexts to a login scenario, 70% to searching, 10% to browsing, and 10% to buying; or you might have different workloads for specific times of a day.
Access to any Java API	Test scripts can directly access any Java API.
Parameterization of input data	Input data (e.g. URL parameters, form fields) can be dynamically generated. The source of the data can be anything including flat files, random generation, a database, or previously captured output.
Content Verification	Scripts have full access to test results. In the future, The Grinder will include support for enhanced parsing of common results such as HTML pages.

2.10.8 The Grinder Plug-ins

HTTP	The Grinder has special support for HTTP that automatically handles cookie and connection management for test contexts.
Custom	Users can write their own plug-ins to a documented interface; although this is rarely necessary due to the powerful scripting facilities.

2.10.9 HTTP Plug-in

HTTP 1.0, HTTP 1.1	Support for both HTTP 1.0 and HTTP 1.1 is provided.
HTTPS	The Grinder supports HTTP over SSL.
Cookies	Full support for Cookies is provided.
Multi-part forms	The Grinder supports multi-part forms.

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.

Connection throttling	Low bandwidth client connections can be simulated.

2.10.10 TCP Proxy

TCP proxy	A TCP proxy utility is supplied that can be used to intercept system interaction at the protocol level. It is useful for recording scripts and as a debugging tool.
HTTP Proxy	The TCP proxy can be configured as an HTTP/ HTTPS proxy for easy integration with web browsers.
SSL Support	The TCP proxy can simulate SSL sessions.
Filter-based architecture	The TCP proxy has a pluggable filter architecture. Users can write their own filters.

2.10.11 Documentation

User Guide	http://grinder.sourceforge.net/g3/getting-started.html (../g3/getting-started.html)
FAQs	http://grinder.sourceforge.net/faq.html (../faq.html)
Tutorial	http://grinder.sourceforge.net/g3/tutorial-perks.html (../g3/tutorial-perks.html)
Script Gallery	http://grinder.sourceforge.net/g3/script-gallery.html (../g3/script-gallery.html)
Articles	http://grinder.sourceforge.net/links.html (../ links.html)
Commercial books	*Professional Java 2 Enterprise Edition with BEA WebLogic Server* *J2EE Performance Testing (../links.html#book)*

2.10.12 Support

Mailing Lists	grinder-use@lists.sourceforge.net (mailto:grinder-use@lists.sourceforge.net) grinder-development@lists.sourceforge.net (mailto:grinder-development@lists.sourceforge.net) grinder-announce@lists.sourceforge.net (mailto:grinder-announce@lists.sourceforge.net)

Copyright © 2013 Philip Aston, Calum Fitzgerald All rights reserved.