

Script Gallery

Table of contents

1 Hello World.....	2
2 Simple HTTP example.....	2
3 Recording many HTTP interactions as one test.....	3
4 HTTP/J2EE form based authentication.....	3
5 HTTP digest authentication.....	4
6 HTTP cookies.....	4
7 HTTP multipart form submission.....	6
8 Enterprise Java Beans.....	6
9 Grinding a database with JDBC.....	7
10 Simple HTTP Web Service.....	8
11 JAX-RPC Web Service.....	9
12 XML-RPC Web Service.....	9
13 Hello World, with functions.....	10
14 The script life cycle.....	10
15 Accessing test statistics.....	11
16 Java Message Service - Queue Sender.....	12
17 Java Message Service - Queue Receiver.....	13
18 Using The Grinder with other test frameworks.....	15
19 Email.....	16
20 Run test scripts in sequence.....	17
21 Run test scripts in parallel.....	17
22 Thread ramp up.....	18

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.

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`.

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.output() method.
log = grinder.logger.output

# 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")

# Wrap the log() method with our Test and call the result logWrapper.
# Calls to logWrapper() will be recorded and forwarded on to the real
# log() method.
logWrapper = test1.wrap(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):
        logWrapper("Hello World")
```

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 = test1.wrap(HTTPRequest())

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
# using a FilenameFactory.
def writeToFile(text):
    filename = grinder.getFilenameFactory().createFilename(
        "page", "-%d.html" % grinder.runNumber)

    file = open(filename, "w")
    print >> file, text
    file.close()
```

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')

page1Test = Test(1, "First page").wrap(page1)

class TestRunner:
    def __call__(self):
        page1Test()
```

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")

class TestRunner:
    def __call__(self):
        request = protectedResourceTest.wrap(
            HTTPRequest(url="http://localhost:7001/console"))

        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.output("Challenged, authenticating")

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

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

        return request.POST(authenticationFormData)

```

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 = test1.wrap(HTTPRequest())

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')

```

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.output

# 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

CookieModule.setCookiePolicyHandler(MyCookiePolicyHandler())

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

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 will 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")

```

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 = test1.wrap(HTTPRequest(url="http://localhost:7001/"))

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)

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

```

8. Enterprise Java Beans

```

# Exercise a stateful session EJB from the BEA WebLogic Server 7.0
# 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
# 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")
homeTest = tests["home"].wrap(home)

random = Random()

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

        trader = homeTest.create()

        tradeTest = tests["trade"].wrap(trader)

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

        grinder.sleep(100)           # Idle a while

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

        queryTest = tests["query"].wrap(trader)
        balance = queryTest.getBalance()
        log("Balance is $%.2f" % balance)

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

    # Can obtain information about the thread context...
    if grinder.threadNumber == 0 and grinder.runNumber == 0:
        # ...and navigate from the proxy back to the test
        d = queryTest.__test__
        log("Query test is test %d, (%s)" % (d.number,
d.description))
```

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.

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
        statement = None

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

            testInsert = test1.wrap(statement)
            testInsert.execute("insert into grinder_fun values(%d, %d)"
                               %
                               (grinder.threadNumber,
grinder.runNumber))

            testQuery = test2.wrap(statement)
            testQuery.execute("select * from grinder_fun where
thread=%d" %
                               grinder.threadNumber)

        finally:
            ensureClosed(statement)
            ensureClosed(connection)

```

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.

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 jvdom
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 = bookDetailsTest.wrap(
            HTTPRequest(url="http://xml.amazon.com/onca/xml"))

        parameters = (
            NVPair("v", "1.0"),
            NVPair("f", "xml"),
            NVPair("t", "webservicex-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.output(str(result))
```

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.webservicex.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()
portTest = Test(1, "JAXP Port test").wrap(port)

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

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 Sebastian 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"

serverWrapper = test1.wrap(XmlRpcClient(server_url))

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

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

        grinder.logger.output("SUM %d" % sum)

```

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")
logTest = test1.wrap(grinder.logger.output)

def doRun():
    logTest("Hello World")

def TestRunner():
    return doRun

```

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
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.output("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.output(
            "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.output("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.output("Thread shutting down")
```

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 = Test(1, "Basic request").wrap(
    HTTPRequest(url = "http://localhost:7001"))

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

grinder.logger.output("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
# 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 = Test(2, "Request resource").wrap(
        HTTPRequest(url =
"http://localhost:7001"))

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

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

    grinder.logger.output("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 = Test(3, "Page").wrap(page)

instrumentedPage(self)

```

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()
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

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

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

        sender = session.createSender(queue)
        instrumentedSender = Test(1, "Send a message").wrap(sender)

        message = createBytesMessage(session, 100)

        log("Sending ten messages")

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

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

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
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)

recordTest = Test(1, "Receive messages").wrap(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.output

        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.
            recordTest(deliveryTime)

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

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

    # 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()

```

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 = [Test(i, name).wrap(HttpTest(name))
         for name, i in zip(testNames, range(len(testNames)))]

# 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()

```

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
# 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!")

        # Wrap transport object in a Grinder Jython Test Wrapper
        transport = emailSendTest1.wrap(session.getTransport("smtp"))

        transport = emailSendTest1.wrap(transport)
        transport.connect(smtpHost, "username", "password")
        transport.sendMessage(message,
message.getRecipients(Message.RecipientType.TO))
        transport.close()
```

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()]

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

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()

```

22. Thread ramp up

```

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

from net.grinder.script.Grinder import grinder
from net.grinder.common import Logger

def log(message):
    """Log to the console, the message will include the thread ID"""
    grinder.logger.output(message, Logger.TERMINAL)

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__()")

```