The Flight Loadtester

David Holroyd

Daniel Strawson

The Flight Loadtester
by David Holroyd and Daniel Strawson

Copyright © 2000 by David Holroyd

The copyright holders make no representation about the suitability of this software for any purpose. It is provided “as is” without expressed or implied warranty.

Table of Contents

1. Introduction **5**
The Software 5
Flight Simulator 5
11Features 5
Flight Recorder 5
12Features 5
2. Flight Simulator **7**
Introduction 7
Installing (TODO) 7
Invocation 7
Reports 8
3. Flight Recorder **11**
Introduction 11
Installing Muffin (TODO) 11
Installing The FlightRecorder filter 11
Configuring your browser (TODO) 13
A. Configuration Script Syntax **14**
Scripting Guide 14
API Reference 15
Object `Agent` 15
A1Object Properties 16
Object `Form` 17
A2Object Properties 17
B. Future Plans / Bugs / Todo **18**
Bugs 18
B1 18
Ideas for the Future 18
B2 18

List of Examples

2-1. `load.log` 8
A-1. A simple script 14
A-2. Setting the referring page 14
A-3. Setting a persistant header 15
A-4. Posting a form 15

Chapter 1. Introduction

The Flight Load Tester is a system for stressing a website or 'web application' to see how it might perform once launched, with lots of people are trying to use it at once. By allowing you to define the testing procedure using JavaScript, quite complex tests can be defined; however the basic functionalty can be used without needing very much JavaScript experience.

The Software

Flight Simulator

The main software component of Flight Load Tester is called the Flight Simulator. This does the actual load testing work by requesting webpages as if it were a user of the site (or as if it where hundreds or thousands of users of the site).

Features

Cookie support

Cookies are supported, allowing the system to work with sites that use *sessions*.

(No) Authentication support

Support is soon to be added.

Scriptability

You can produce complex behaviours by using JavaScript

Flight Recorder

This program allows you to record the steps you make as you visit a site. The result of doing this is a file which you can 'replay' with Flight Simulator.

Features

Cookie suport

Cookies set by the server are ignored (Flight Simulator takes care of them), but cookies which

appear to have been set at the client (e.g. by browser scripting) will be recorded.

POST data

Unfortunately, posted form data is not currently captured

Conditional Requests

Conditional requests by the browser for objects Flight Recorder has seen before will be recorded *relatively*. i.e. Flight Simulator will be able to make the condition an object's *actual* last modification time, rather than the value when the script was recorded.

Chapter 2. Flight Simulator

Introduction

Flight Simulator is a command-line program that simulates requests to a website. You need to provide it with a configuration script which it will will run though, requesting the pages the script defines. It can be made to run though the script several times, one after the other, and to do multiple 'runs' in parallel (to simulate several people using the site at once).

Installing (TODO)

Invocation

Synopsis of command-line options:

`java uk.co.badgersinfoil.flight.simulator.Main -start-threads=`*`n`*` -thread-inc=`*`n`*` -loops=`*`n`*` -rounds=`*`n`*` -timeout=`*`n`* *`script`*

Where...

`script`

is the name of the file containing the configuration script.

`-start-threads=`*`n`*

Gives the number of concurrent threads that should be started initialy. The defualt is one thread.

`-thread-inc=`*`n`*

Specifies how many additional threads should be added in each round (when `-rounds` specifies there should be more than one). The default is one thread per round.

`-loops=`*`n`*

Specifies that the given script should be executed *`n`* times by every thread, every round. The default is to only execute the script once.

`-rounds=`*`n`*

Specifies that *`n`* rounds should be run. Running several rounds lets you see how the server performs when the number of cocurrent users increases. The default is one round – only use the initial number of threads.

`-timeout=`*`n`*

Attempts to connect to the server should fail after *`n`* seconds. The default behaviour is to wait forever for the server to respond.

All parameters apart from *`script`* are optional. Therefore, if only *`script`* is specified, one thread will execute the file once and then the program will exit; as there is only one round.

Reports

The output of a Flight Simulator session will be a file called `load.log`.

Example 2-1. `load.log`

```
threads requests time transfer-rate
1 90 4236 131691
2 180 4016 127438
3 540 4616 116062
4 810 6219 106140
5 1350 8923 88011
6 1440 12197 85094
7 2520 14291 65093
8 3240 13339 59263
9 4050 15843 53958
10 4950 19258 48998
```

File format

threads

The number of concurrent threads making requests to the server.

requests

The number of requests made to the server.

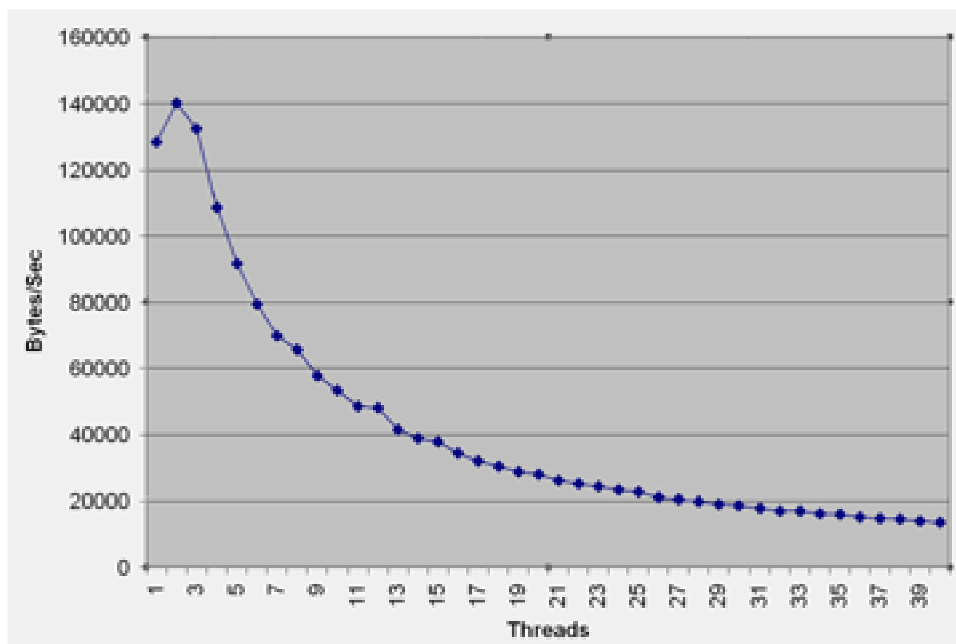
time

The total time to complete all the requests.

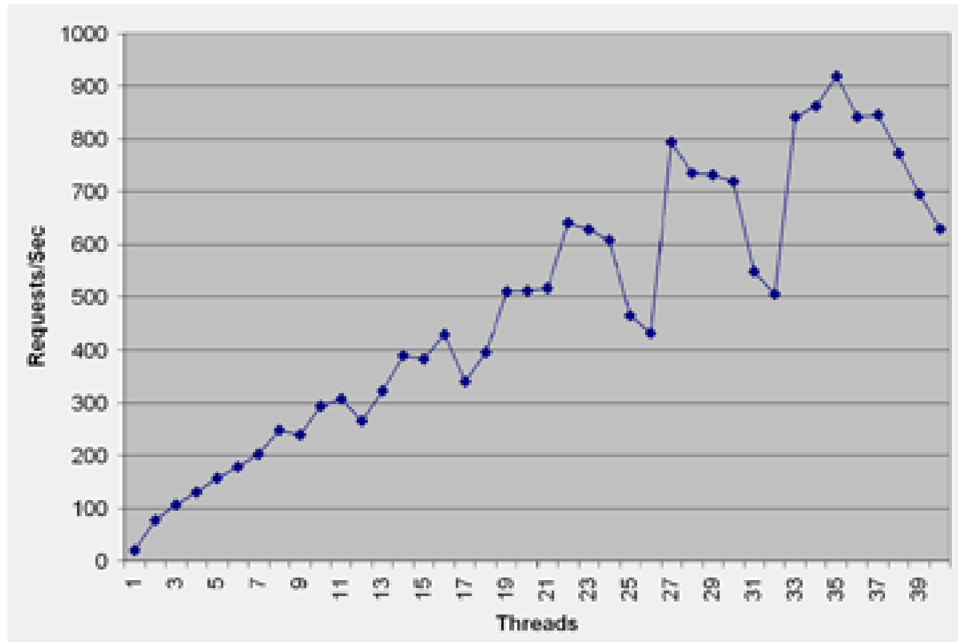
transfer-rate

The average transfer rate in bytes per second (for responses that returned data).

You can produce reports based on this data using applications such as gnuplot or Microsoft Excel. The following graphs where generated with Microsoft Excel from a data set with 40 rounds.



Graph of Bytes per Second against Number of Threads



Graph of Requests per Second against Number of Threads

Chapter 3. Flight Recorder

Introduction

Flight Recorder allows you to generate a configuration script for Flight Simulator by simply browsing the pages you want tested. You can configure your browser so that Flight Recorder will watch you interact with the website and generate a script that, when run with Flight Simulator will copy your actions exactly (well, nearly).

Flight Recorder works by acting as a Proxy Server. Proxy Servers are a normal part of the infrastructure of the web, normaly implemented to provide services such as ’caching’ – which allows some requests to be satisfied sooner. Flight Recorder is actualy much simpler than these systems; it just likes to watch, trying not to change anything that would’ve happend anyway.

Flight Recorder is implemented as a ’plugin’ to the Muffin filtering proxy server. Muffin has many, many features implemented as other plugins, but for our purposes none of these are needed, just Muffin’s general proxy framework is used.

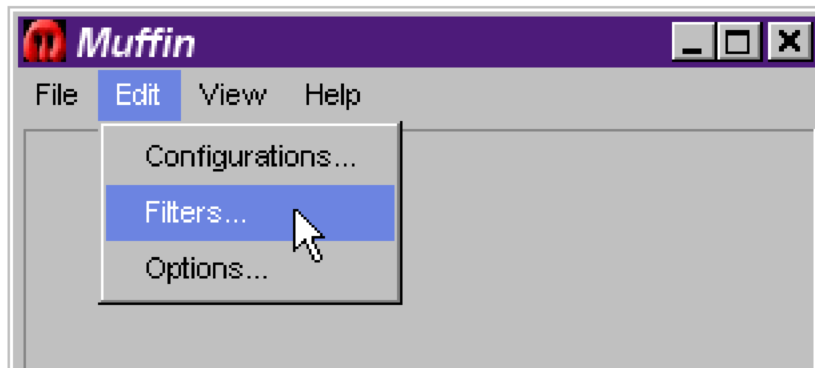
Installing Muffin (TODO)

Muffin is available from the Muffin website (http://muffin.doit.org/) and, at the time of writing, the current version is 0.9.3a.

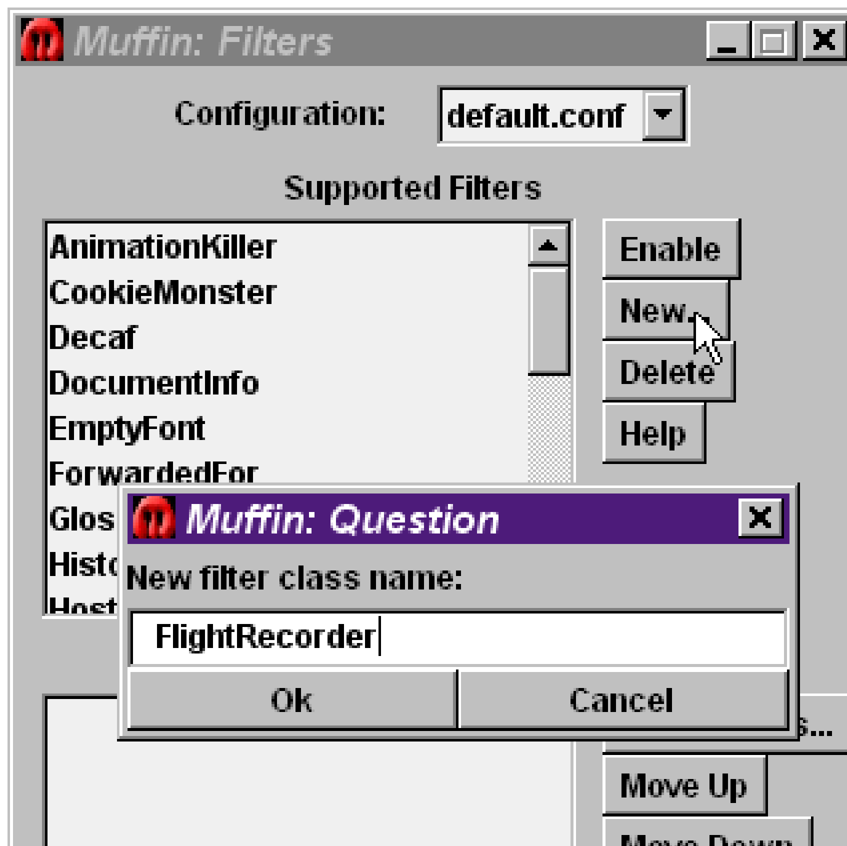
[...]

Installing The FlightRecorder filter

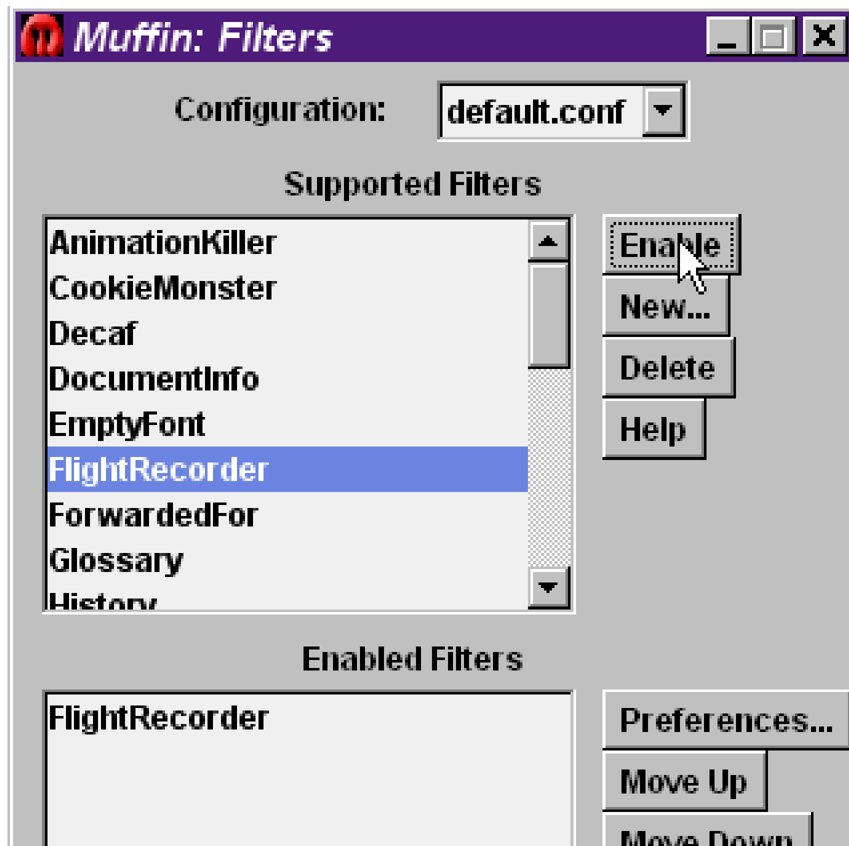
From Muffin’s main window, select Edit→Filters....



In the Filters dialogue that appears, select the New... button and enter the filter name, **`FlightRecorder`**.



Select Ok and the FlightRecorder filter should appear in the Supported Filters list. Select the filter and press the Enable button.



FlightRecorder should now appear in the Enabled Filters list.

Configuring your browser (TODO)

[...]

Appendix A. Configuration Script Syntax

Scripting Guide

> **Important:** This section does not describe how to program JavaScript or give detailed descriptions of it's syntax. You may be able to get by extending the simple examples with no knowledge of JavaScript, but you would do well to get yourself a book and learn to write more advanced scripts.

The scripts used by Flight Simulator are written in JavaScript. The environment these scripts run in differs from the environment available to scripts running on web pages, though the core set of JavaScript objects are be available (e.g. Math, Boolean).

In addition, FlightSimulator provides JavaScript objects which allow you to easily make requests to the webserver.

Example A-1. A simple script

```
// A simple example
agent.setHost("www.goodtechnology.com");
agent.get("/");
```

This script, when run, will request the GoodTech homepage. The first line starting with `//` is a comment. All text on a line after this symbol will be ignored. The second line instructs Flight Simulator that subsequent requests should be made to the server `www.goodtechnology.com`. The third line actualy tells Flight Simulator to request GoodTech's homepage.

Example A-2. Setting the referring page

```
// Referrer example
agent.setHost('www.goodtechnology.com');
agent.get('/');
agent.setHeader('Referer', 'http://www.goodtechnology.com/');
agent.get('/page2.jsp');
```

This example extends the first by requesting a second page and telling Flight Simulator to inform the server of the referring page. This information is sent by a web browser to tell the server what page is being linked from. Most of the time you won't need to bother with the referrer; only occasionaly do server-side components examine it.

The `agent.setHeader()` function allows you to send an arbitrary HTTP header with the next request. If you wanted to specify a header to be sent with every subsequent request, you can use the `agent.setPersistantHeader()` function.

Example A-3. Setting a persistant header

```
// Set the user-agent
agent.setPersistentHeader('User-Agent', 'Mozilla/4.0');
agent.setHost('www.goodtechnology.com');
agent.get('/');
agent.get('/anotherpage.jsp');
```

Here, we will send the `User-Agent` header with both requests to the server. This would be useful for testing systems that recognise the browser making a request and modify their content to suit.

Example A-4. Posting a form

```
// Set the user-agent
agent.setHost('www.goodtechnology.com');

var myForm = new Form();
myForm['firstname'] = 'David';
myForm['secondname'] = 'Holroyd';

agent.post('/register.jsp', myForm);
```

The form object is created and two form fields are defined. [...]

API Reference

The following are the standard JavaScript objects which will be available in your scripts: `Array`, `String`, `Boolean`, `Number`, `Date`, `Math`, `Call`, `With`, `RegExp`, `Script`

In addition, Flight Simulator provides the following objects:

Object `Agent`

The agent object allows you to control the 'user agent' – the software that makes requests to the webserver. The Agent object is implicitly created and can be referenced by the name `agent` in your session script.

Object Properties

`get(object)`

Causes the agent to make an HTTP GET request for the named object. Any headers set with `setHeader(name, value)` will be cleared after this call.

`post(object, form)`

Causes the agent to make an HTTP POST request to the named object. The data in the `Form` object, `form`, will be passed to the server.

`setCookie(name, value, path)`

Sets the named cookie to the given value. This cookie will only be send by the agent when requesting objects in the given path. The expiry time of the cookie is unknown, so it will last for the duration of this session.

`setCookie(name, value, path, expires)`

Sets the named cookie to the given value. This cookie will only be sent by the agent when requesting objects in the given path. The cookie will be deleted from the agent after the given expiry time.

`setHeader(name, value)`

Sets the named header to the given value for the next request the agent makes. If you want all subsequent requests to send the given header, use `setPersistentHeader(name, value)`.

`setPersistentHeader(name, value)`

Sets the named header to the given value for the next, and all subsequent requests. To stop the header being sent, make a call to `removePersistentHeader(name)`.

`removePersistentHeader(name)`

Stop the named header (as set by `setPersistentHeader(name, value)`) from being sent with any further requests.

`setHost(url)`

Set the host to which all subsequent requests for objects will be made. Each host has a seperate set of cookies – cookies set agains one host will not be sent in requests to another.

`setIfModifiedSince(when)`

Set the `If-Modified-Since` header to the given timestamp.

`setIfModifiedSince(when, contentLength)`

Set the `If-Modified-Since` header to the given timestamp and length.

`setIfModifiedSinceObject(name)`

Set the `If-Modified-Since` header to reflect the `Last-Modified` (and possibly `Content-Length`) property of the named object in the agent's cache (`name` should be the the full URL of the object you are about to request).

Object `Form`

Represents the data in an HTML form, that may be sent in a POST request. The following code creates a Form and sets the values of some fields:

```
myForm = new Form();
myForm['title'] = 'New title';
myForm['body'] = 'This is the body text';
// a name *can* have multiple values (e.g. checkboxes),
myForm['options'][0] = '223';
myForm['options'][1] = '272';
myForm['options'][2] = '273';
agent.post('/cgi-bin/script.pl', myForm)
```

See `Agent.post(object, form)`.

NB JavaScript experts: In order to implement the 'multi-valued properties' in the simple form shown above (the `options` property), Form objects do a little magic: accessing a (thus far) undefined property causes it to become defined as an array. This should be of no consiquence, but is mentioned to explain why otherwise illigal usage works in this case.

Object Properties

`encoding`

You may optionaly specify the encoding in which the data is to be sent. At present, the default, and only supported option is

```
myForm.encoding = 'application/x-www-form-urlencoded';
```

in the future, '`multipart/form-data`' may be supported for uploading files.

Appendix B. Future Plans / Bugs / Todo

Bugs

- Cookies do not expire (Flight Recorder)
- Flight Recorder guesses the path for 'client side' cookies (and if multiple cookies have the same name, they are al assigned the *same* path, and we only remember one of them). Maybe add JS interface so these cookies are set before every request they are sent with?
- Real browsers are multithreaded but Flight Recorder serialises all requests and they are run from a single thread
- `Cookie:` before `Host:` header will cause Flight Recorder problems (never seen it happen though)

Ideas for the Future

- Capture POST data in Flight Recorder (Muffin doesn't seem to support this)
- Suport `multipart/form-data` encoding for file uploads
- More configurability
- Support authentication (Flight Recorder and Flight Simulator)
- return response objects in JavaScript:

```
request = agent.get('/stockprice.jsp');
if (request.status != 200) print('ALERT stockprice.jsp is down!');
```

- Provide access to entity data in JavaScript:

```
request = agent.get('/stockprice.jsp');
if (request.status == 200) {
    var document = request.readData();
    if (!new RegExp("stockprice is currently:").match(document)) {
        print('ALERT stockprice.jsp is down!');
    }
}
```

- Provide access DOM / AST in JavaScript?

- Investigate using Jigsaw (http://www.w3c.org/jigsaw/) for HTTP client API (and possibly proxy framework)
- Internationalisation?