Sitemon Manual

Peter Pearson

June 2010

Contents

1 Introduction

Sitemon is an application designed to help monitor and test website availability, functionality and performance. It has the following features:

- Command-line interface for one-off tasks
- Built-in web server with a web interface for managing and displaying results of monitoring tasks
- Built-in database for storing results of monitoring tests
- Allows displaying header and body responses back from websites from the command line
- Regular testing of websites at certain intervals
- Allows scripts to be created, which are a series of requests to the same website, duplicating a user's journey (e.g. logging in to a website and then searching for and booking a flight)
- Allows multiple simultaneous script or individual requests to be performed at the same time, in order to perform load testing on a website, simulating multiple users accessing a website.

1.1 Technical Requirements and Specifications

Sitemon is written in C++ for high-performance and memory efficiency reasons. It is largely cross-platform (Windows support isn't perfect yet), and makes use of libCURL, an HTTP (and other) protocol library to perform fully RFC-compliant HTTP requests.

For basic usage, the Sitemon executable is the only file needed, but for more advanced usage, other files are required. Specificly, in order to run the built-in web server to control scheduled monitoring tasks, the /web/ directory must be specified within a configuration file.

Under most UNIX-like platforms, libCURL is installed by default and so Sitemon can fully function with no additional files other than the executable (and the web content directory mentioned above). However, under Windows, certain extra OpenSSL DLL files must exist within Sitemon's directory in order to perform encrypted HTTPS (SSL / TLS) requests.

2 Configuration

For certain functionality (scheduled monitoring, requiring an HTTP Proxy for access to the websites needing to be tested), Sitemon will need to be configured before it can be used.

Sitemon's configuration file is a standard XML document named `sm_config.xml` and should reside in the same directory as the Sitemon executable.

2.1 Configuration Parameters

Sitemon has both simple (single item) parameters, and complex (items with sub element items within the XML document).
Below are listed the simple parameters:

Parameter key	Description
web_content_path	The full path to the web content directory. Required for using the web interface.
mon_db_path	The full path to the database file for saving Monitoring configuration items and results.
lt_db_path	The full path to the database file for saving Load Testing results when requested.
web_server_port	The port number Sitemon should use for the built in web server.

TODO: Proxy settings.

3 Test Specifics

When Sitemon performs a test request, it does so in a clean state, whereby there is no cached information. The exception to this is when a Script request is run. In this case, Sitemon will remember information from previous steps in the Script - mainly DNS cache information, Cookies - and use them in following steps in the Script, in order to emulate how web browsers interact with web servers as accurately as possible.

The statistics of each request Sitemon makes are:

- The Time at which the request was made
- The Final URL arrived at
- HTTP Response Code
- Times taken for various parts of the request and response (see below for more details)
- Download size (the number of bytes downloaded)
- Content size (the size of the content downloaded) - this will only differ from the Download size if compression was used.

Sitemon records four different timing metrics for each request and its corresponding response. These times allow the analysis of the performance of a website to be done, in particular during load testing. The times are cumulative, and are each from when the request was initiated:

Time	Description
DNS Lookup time	The time it takes for the hostname specified within the requested URL to be resolved to an IP address which Sitemon can then connect to.
Connect time	The time taken for the remote web server or load balancer to accept the TCP socket connection.
Data Start time	The time it takes for the first byte of data to be received after the connection is established. This generally is the time taken for the application layer of the website in question to process the request (run code, do a database query, etc).
Total Time	the time taken for the entire request to complete, from DNS lookup until the last byte of information is received.

4 Basic command-line usage

There are four main areas of commands for Sitemon that can be run from the command line.

4.1 Basic single request

The first, and most basic is the single request for a specified URL.
At its most basic, this can be done with the following command:

```
./sitemon http://www.google.com/
```

This will simply perform a *GET* request for the relative URL resource requested at the hostname specified by the URL. Only the content of that item will be downloaded, but it will not be stored or saved in any way. No compressed content will be accepted either.

If the HTTP server responds to the request, the result of this command should display some statistics, including:

- Final URL arrived at (in case any redirect responses were returned)
- The HTTP response code
- Timing statistics for various stages of the request
- The HTML content and download sizes

An example output is displayed below:

```
Final URL: http://www.google.co.uk/
Respone code: 200

DNS Lookup: 0.05218 seconds.
Connection: 0.081362 seconds.
Data start: 0.149268 seconds.
Redirect count: 1.
Redirect time: 0.143107 seconds.
Total time: 0.385974 seconds.

HTML Content size: 9153
HTML Download size: 9153
```

There are several command-line options that can be specified (before any other options) which provide additional functionality to this request type (and all others):
`-dc` : makes Sitemon parse the response content (assuming it is HTML) and pull out linked Image and JavaScript files. These will be downloaded at the same time.

`-ac` : makes Sitemon's request to the server specify that it can accept *gzip* or *deflate* compressed content. It is up to the server as to whether content is returned compressed or not.

4.2 Single Script Request

See the Scripts section (5) for details on Scripts.
To run a script request, the path to the script (either relative or full) should be specified in the following manner:

```
./sitemon script /usr/home/test_script.xml
```

Sitemon will then run the steps in the sequence found in the script, one-at-a-time.
If an error occurs on a step that is fatal (see error appendix), then the script will terminate at that point.

4.2.1 Debugging Scripts

Sitemon allows Scripts to be debugged in a limited fashion by allowing header requests and responses to be outputted to screen for problem steps, and by allowing body content of a problem step to be saved to a file (for example to work out why an Expected Phrase isn't being found.)
In order to run a script in debug mode and save the returned body content of the step with a problem to a specified file, use the following command:

```
./sitemon debug /usr/home/test_script.xml /usr/home/debug_output.html
```

This command will run the /usr/home/test_script.xml script, and output a problem step's HTML content to a file located at: /usr/home/debug_output.html.
It is also possible to embed debug settings within the Script file, and Sitemon will pull the settings out of the file without the need to specify the output file path on the command line. The following XML tag must be used:

```
<debug enabled="yes">
<debug_type>errors_only</debug_type>
<output_location type="file">E:\sm\_debug.txt</output_location>
<debug_items>both</debug_items>
<output_body_response>yes</output_body_response>
</debug>
```

4.3 Load Testing Scripts

Once a script has been written and works properly using the above commands, it can be used in a load-testing scenario in order to simulate multiple users accessing a website and using it. See the Load Testing section (6) for details.

4.4 Monitoring

Sitemon can be run in web daemon mode, where it will function as a web server on the local machine, allowing scheduled monitoring tests to be configured and the results to be viewed via a web interface in a web browser.
The HTML, CSS and Javascript content from the built-in web server are fully standards-compliant, and so will not display correctly in Internet Explorer 6, 7 or 8. It is recommended to use Firefox 3.5+, Google Chrome 4.0+ or Safari 4.0+.

Sitemon will store the results from these scheduled tests in the database file specified in the configuration file.

In order to run Sitemon in this mode, use the following command:

```
./sitemon -web
```

The Monitoring section (7) contains in-depth information on Sitemon's monitoring capabilities.

5 Scripts

Scripts allow a series of requests to be bundled together sequentially, in order to reproduce the journey a user might perform while using a website for a specific task (for example, logging in to a website, then searching for an item and adding it to the shopping basket). Scripts are stored as XML files consisting of a sequence of request items. Each request item - a step - can be configured to specify pretty much all aspects of the request, including:

- Request type (*GET* or *POST*)
- URL
- Parameters (which work for both GET and POST requests)
- Cookies
- A specific referrer used for that request
- Timeouts values for the request
- An Expected Phrase (Sitemon will check that the content returned by that step contains the specified phrase if it is specified)
- Pause time - If this is specified, Sitemon will pause for this duration after the step, in order to allow a more realistic test scenario, simulating a user reading the content of a page before continuing onto the next step.

5.1 Dynamic Parameters

Request HTTP parameters are by default hard-coded, in that they have a name and a value, and will always submit those values. Sitemon does however have support for Dynamic Parameters, which allow parameter values to be generated on demand when the script is run.

The main type of these is the Date Dynamic parameter, which allows date values to be formatted to strings in a specified format, with a date a specified number of days in the future. This allows the ability to have scripts that always submit valid dates as parameters for requests that require them, eliminating the need to constantly update and maintain the scripts with dates in the future.

A Dynamic Date parameter can be specified in a script XML file in a similar fashion to the normal hard-coded ones, however it has additional XML attributes and the content value of the tag is ignored:

```
<param name="date" type="date" format="%d-%b" days_forward="60"/>
```

The format specifiers for the Date Dynamic Parameter are the same as the C strftime function.

5.2 Using Scripts

When running Scripts, Sitemon will remember cookies returned from previous steps and send them back to the server in the requests for any following steps.

If a particular step of a Script fails, Sitemon will halt execution of that Script.

Scripts can also include sections concerning load testing settings and debug settings.

The most basic script could just have a single request in it. An example is the Google search for "finance" below:

```
<?xml version="1.0" encoding="UTF-8"?>
<script desc="Google Search for Finance">
 <request desc="Google GET request for Finance">
  <type>GET</type>
  <url>http://www.google.com/search</url>
  <params>
    <param name="rls">en</param>
    <param name="q">finance</param>
    <param name="ie">UTF</param>
    <param name="oe">UTF-8</param>
  </params>
  <pause>5</pause>
 </request>
</script>
```

The above script code contains a single request element, which is a GET request to the specified URL, has 4 parameters, and a pause time of 5 seconds.

A script example with Load test settings specified within it is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<load_test>
 <type>profile</type>
 <segments>
  <seg threads="1">1</seg>
  <seg threads="5">2</seg>
  <seg threads="2">1</seg>
 </segments>
</load_test>
<script>
 <request desc="Google GET search">
  <type>GET</type>
  <url>http://www.google.com/search</url>
  <params>
   <param name="rls">en</param>
   <param name="q">finance</param>
   <param name="ie">UTF</param>
   <param name="oe">UTF-8</param>
  </params>
  <pause>5</pause>
 </request>
</script>
```

The load test settings above are for a profile test, with 3 segments: 1 concurrent user for 1 minute, 5 concurrent users for 2 minutes and then 2 concurrent users for the final minute.

6 Load Testing

Load Testing involves running a specified number of identical (future enhancements to Sitemon may allow more dynamic requests) and independent requests to be run against a website concurrently, in order to simulate a large number of users using a website in a realistic scenario.

Either single requests or Script requests can be run.

Load Testing options can to a limited extent be specified within the command-line arguments that Sitemon is run with when specifying a Script, or they can be embedded in a Script document and Sitemon will automatically pull out these parameters when running the specified Script.
Sitemon has two different types of load test it can run, Hit Testing and Profile Testing.

6.1 Hit Test

Hit testing is the most basic type, and simply involves an instant creation of a duplicate number of simultaneous requests. Basically, a specified number of threads are created, and each thread is fully self-contained and independent, and performs the specified script test a certain number of times (by default only once).

6.2 Profile Test

Profile testing is a more realistic time-based type of testing, allowing variations in the number of simulated users performing the script over a specified time period. For example, a profile could be set up to initially start testing with 10 simulated users for 5 minutes, then ramp up to 40 users for the next 10 minutes, and then ramp down to 5 users for the final 3 minutes. Sitemon will keep the specified number of threads active for each profile segment needed, with the request being continually repeated.
There are two styles of Profile testing which govern how any increase or decrease in the number of requests over time is made:
Step style means that if a profile specified going from 20 users at 2 minutes to 50 users at 4 minutes, the increase will be instant.
Smooth style means that Sitemon will gradually increase the number of requests being performed in the previous profile segment, so that the change happens more gradually.

6.3 Results

Results can be saved to a .csv file or to a database file. If a Load Test run is configured to save results to a .csv file, the results are only viewable after the entire test run has finished. If results are configured to be saved to a database file, then they are saved in batches at regular intervals throughout the test run.

6.4 Example commands

Hit testing:
Run 30 concurrent threads of requests to a single URL and save results to output_results.csv in current directory:

```
./sitemon lt-hit http://news.bbc.co.uk/ 30 output_results.csv
```

Run 30 concurrent threads of requests to a single URL, accepting compressed content and downloading linked images and Javascript files, and save results to output_results.csv in current directory:

```
./sitemon -ac -dc lt-hit http://news.bbc.co.uk/ 30 output_results.csv
```

Run 50 concurrent threads of script /Users/peter/scripts/google_search.xml and save results to output_results.csv in current directory:

```
./sitemon lt-hit /Users/peter/scripts/google_search.xml 50 output_results.csv
```

Run Hit Load Test of script /Users/peter/scripts/google_search.xml, pulling load testing config parameters from the script file, and save results to output_results.csv in current directory:

```
./sitemon lt-hit /Users/peter/scripts/google_search.xml output_results.csv
```

Profile testing:
Run 30 concurrent threads of requests to a single URL for 5 minutes and save results to output_results.csv in current directory:

```
./sitemon lt-profile http://news.bbc.co.uk/ 30 5 output_results.csv
```

Run 30 concurrent threads of requests to a single URL for 5 minutes and save results to output_results.csv in current directory, accepting compressed content back from the web server:

```
./sitemon -ac lt-profile http://news.bbc.co.uk/ 30 5 output_results.csv
```

Run Profile Load Test of script /Users/peter/scripts/google_search.xml, pulling load testing profile parameters from the script file, and save results to output_results.csv in current directory:

```
./sitemon lt-profile /Users/peter/scripts/google_search.xml output_results.csv
```

7 Monitoring

When run in Web Daemon mode, Sitemon runs a built-in webserver and scheduler and can perform both single requests and script tests against sites at selectable intervals, 24/7. Single and script tests need to be set up via the web interface. (In the future it should be able to import and export XML scripts to the monitoring part of the application.) Results data is stored in the monitoring database file specified in the Sitemon configuration file.

Results currently need to be viewed via the web interface.

8 Appendixes

A Error Codes

If Sitemon cannot DNS lookup the hostname, connect to or receive data from the server for a request, it returns an error.

0 (OK)	Received a response from the server, and if an Expected Phrase was specified, it was found.
1	The main request succeeded, but there were component errors.
-1	Couldn't resolve host
-2	Couldn't connect to host
-3	Timed out getting data from host
-4	Couldn't receive data from host
-5	Expected Phrase not found in content