



LoadGen User Manual

Revision History

Initial Release	28 October 2007
-----------------	-----------------

1 - General Concepts

1.0 - Important Note

Used inappropriately, programs in the WebTools suite can cause performance problems for servers and networks under analysis. The user is responsible for ensuring they only use these tools to collect information on sites for which they have the appropriate responsibility. The user is advised to examine the documentation thoroughly. The user's email address and name are appended to the User Agent string of all requests made by these tools to facilitate the rapid identification and correction of user errors or mis configurations.

1.1 - What is a Web Site Load Generator?

A web load generator is a program designed to stress test the performance of a web site. It does this by synthetically generating requests to the web site's servers that attempt to capture the important characteristics of the expected user generated requests, varying the intensity and duration of such request streams and measuring how the system performs. The goal of such testing is to identify bottlenecks in the site's implementation before they begin to impact the end user experience and further to develop an estimate of the maximum capacity of the site to handle user traffic.

1.2 - What is LoadGen?

LoadGen is a web site load generator developed by E-Insights, LLC for use by its clients. LoadGen includes features that facilitate the rapid development of synthetic traffic streams to test large complex dynamic web sites.

This document describes the use of LoadGen in an interactive environment. LoadGen can also be configured to perform load tests in batch mode – for example to perform a weekly check on a site as a quality control process. The interactive mode is convenient in the early stages of developing load test configurations, however once tests have been developed and de-bugged, the batch mode of operation is preferable.

LoadGen is intended for use by Web site owners, or their authorized representatives

as a tool to assist in enhancing the quality and performance of their sites.

1.3 – WebTools & Profiles

LoadGen is one of several related tools in the E-insights WebTools suite. The manner in which access is gained to these tools is based on the Java Network Launch Protocol (JNLP). Registered users on E-insights, upon logging in to the site using a browser, are provided with links to the tools they are allowed to use. Selecting one of these tools will pass a file with mime type x-application/jnlp back to the browser. If the browser is configured¹ to handle this type then the application will launch. The first time a user starts an E-insights application in this fashion on a specific computer, they will be asked if they wish to trust the code provided by E-insights – an affirmative answer is necessary in order to use the tools. Note that the use of JNLP also ensures that the user always gets access to the most current software – updating is automatic.

These tools also share a common form of authentication, access control and configuration. Normally, once a user has logged into the E-insights web site, then launching a tool as described in the previous paragraph requires no additional authentication. There are rare cases however when the tool itself will ask for authentication. The credentials used are identical to those used on the E-insights.com web site.

Upon launching a WebTools program, you may be required to choose a profile per the pictured dialog box.



Each WebTools-enabled account registered on E-Insights.com has access to one or more profiles. Each profile is attached to different a database in which data generated by WebTools programs is stored and from which said data may be read. A WebTools program launched under a given profile can only read from and write to the database attached to that profile. Otherwise, WebTools programs function identically between profiles; profiles are used solely for information management purposes. Since the specification of the database to use for a specific task is within the Profile, it is possible to have these databases anywhere, including inside corporate firewalls where they are not accessible to (amongst other people) E-insights staff.

¹ Many browsers are configured this way by default. If the browser is not so configured, the user will need to download and install the Java Runtime Environment appropriate to their computer from www.sun.com.

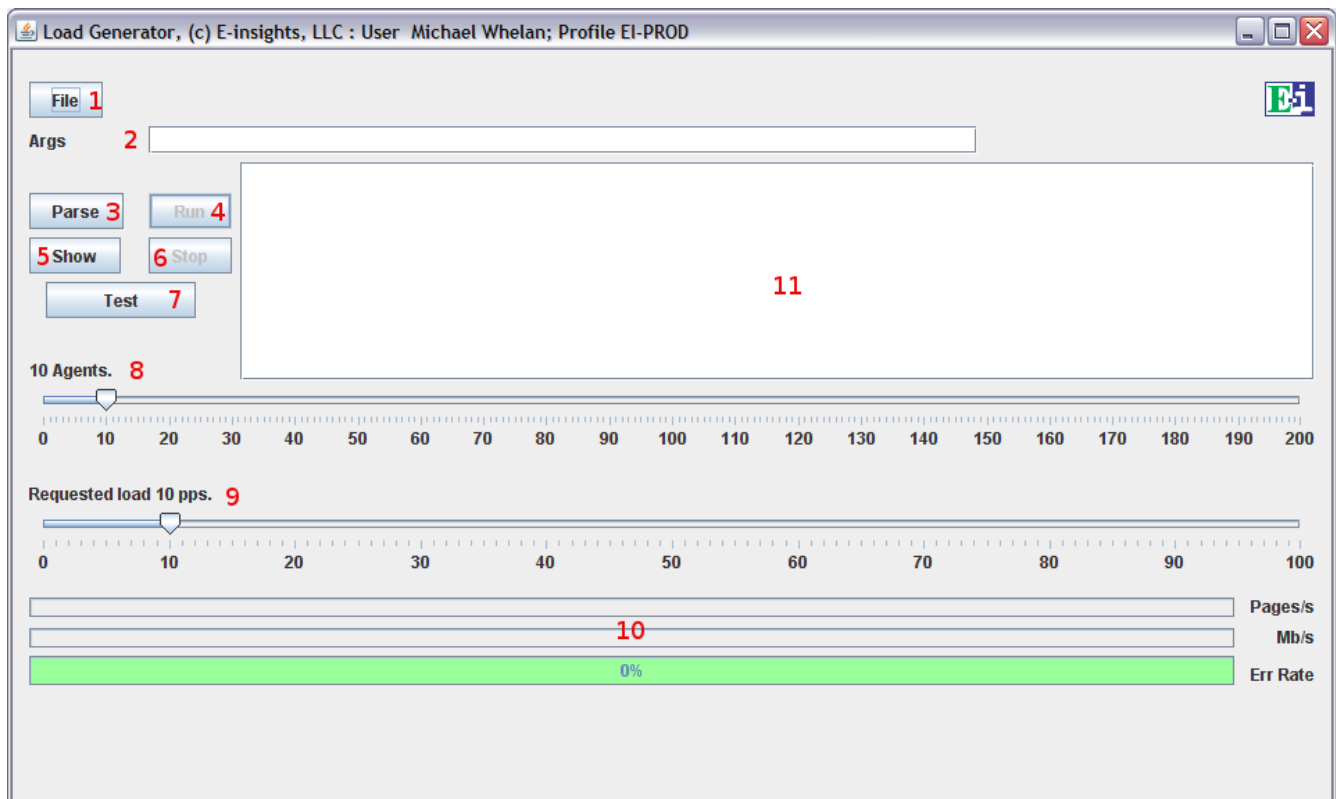
1.4 - What is a .lgx file?

The commands to control a load test are stored in a “.lgx” file. This file specifies one or more 'pages' and one or more 'sequences'. A 'page' describes an individual request that will be made to the web site being tested, whereas a sequence specifies a series of page requests as would be made by a specific type of visitor. Sequences can be weighted to simulate the expected relative frequency of different user 'types'; for example a heavy user versus a user coming in from a search engine. 'pages' can include the random generation of either URL arguments or data for POST operations. This allows more realistic testing of such features as searches and form submissions. More information on .lgx files and rules can be found below in section 3.

2 – Interfaces and Reports

The LoadGen User Interface

The LoadGen interface is shown in the image below. The username and profile in the title bar of the window are customized to the specific user using the tool and the profile they have selected to use.



1: The File Button. Pressing the File button will open a dialog box prompting the user to select the .lgx file containing the specifications for a load test.

2: The Arguments Box. Entering .lgx rules into the arguments box will cause them to be appended to the rules specified in the loaded .lgx file. Note that doing so has no effect on the .lgx file itself; additional arguments are specified for this load test only. Note also that where rules conflict, newer rules override older rules, so the arguments box can be used to fine-tune an .lgx file. Normally in an .lgx file commands are one per line. Since the argument box consists of a single line only, the contents are separated by spaces and the individual such elements interpreted as rules. This does mean that commands requiring spaces themselves (e.g. Lists) cannot be used in the arguments box.

3: The Parse Button. Pressing the parse button will cause LoadGen to read the rules in the loaded .lgx file, as well as those entered via the arguments box. The finished rule set will be displayed in the central information window, and any errors it contains will be indicated. An error free parse is required before a load test can be initiated.

4: The Run Button. Pressing the Run button will initiate the load test. The Run button is grayed out until an acceptable instruction set has been parsed. Once a Run is started, the Run button is again grayed out while the Stop button becomes active.

5: The Show Button. Once an .lgx file has been parsed, clicking on the Show button will display a list of all defined pages with their associated elements and arguments.

6: The Stop Button. Pressing the stop button will stop a load test prematurely, that is, before the conditions specified by the rules have been met. A stopped test can not be resumed, but whatever information was obtained before the it was stopped will be stored in the appropriate database. The Stop button is grayed out unless a load test is actually in progress. The effect of pressing the Stop button is to stop new requests being generated and sent to the site under test. Requests that are in progress are allowed to complete and the results to that point are stored in the database. Hence, it can take a brief time for the 'Stop' to complete at which time button states will change again to allow a Parse or Run.

7: The Test Button. Once an set of instructions has successfully been parsed, the test may either be run or tested. Testing a set of instructions will cause every page specified to be loaded once. The results of the test can be saved. Errors encountered during the test will be displayed in the central window. The saved test results will show for every page loaded all headers and data received. The loaded page can be viewed to ensure that LoadGen is indeed sending the desired requests to the server. It is strongly recommended that a test be performed on all new scripts and the results thoroughly checked before an actual load test is run. The saved test files are stored in the same directory from which the .lgx file was loaded. If the testname was set to "abc", the test results are stored in the file named "indexLT_abc.html". In addition to this file, one other file per page named as LT_testname_pagename.html are saved. The index page contains all the header information and data associated with requests and responses, while the other pages contain the actual data received in response to the specific request as shown in the 'index' page.

8: The Agentcount Slider Control. This slider can be used to set the number of agents used during a load test. During parsing, if an agentcount directive is found the value of the slider is adjusted to reflect this. After parsing, but before starting a run the slider can be varied to change the agentcount value. Once a run has begun, the slider is removed from the interface and the number of agents is fixed for the duration of the run.

9: The Load Rate Slider Control. The Load Rate slider control is similar to the agentcount slider with the exception that it is not removed from the interface during a run and the load rate can be varied. This is only useful in early stage load testing where the goal is to quickly determine if the system being tested can sustain a particular load level by observing it in real-time. While the load can be varied, the database does not record when load values were changed so it is not possible to extract useful information from the statistics collected during a run.

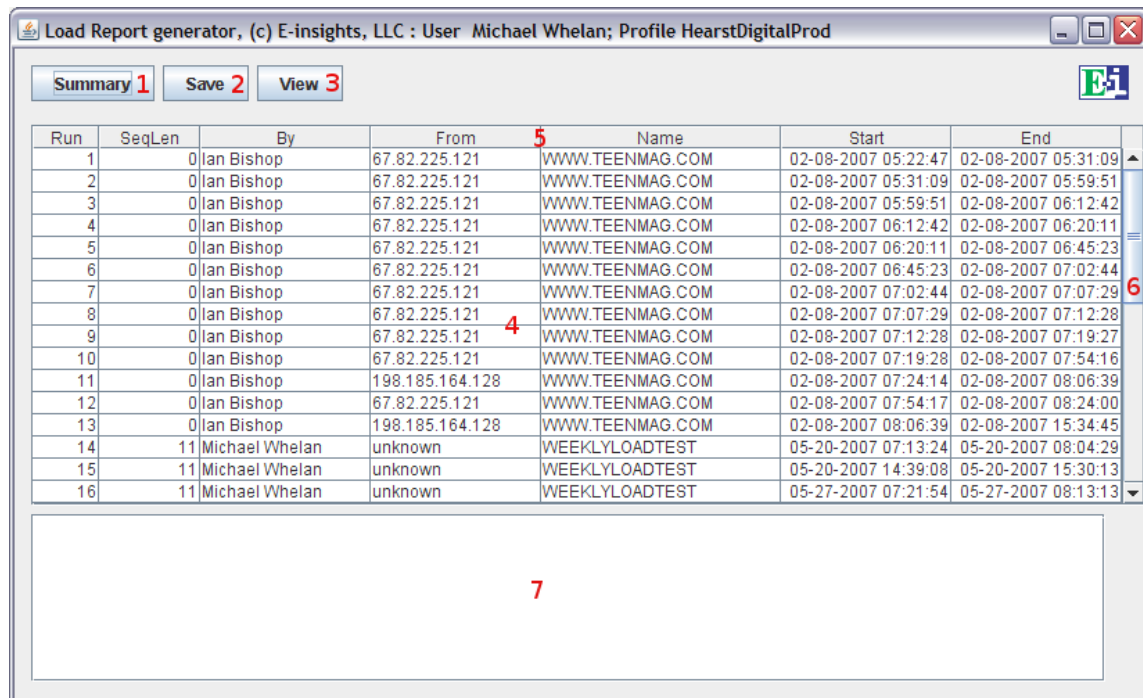
10: Progress Bars. The progress indicator displays information pertaining to a load test in progress, such as the number of pages per second, bandwidth and encountered error rate. The progress indicator is updated every 10 seconds – the cumulative and most recent values are shown in text in the center of each bar.

11: Message Window. Error messages, progress messages etc are displayed in this window.

2.2 - The Load Report Generator

2.1 The Report Generator Interface

The Load Report Generator is shown in the image below. Again, the username and profile in the titlebar are customize to the current user and their chosen profile. During a load test run, summary information (both performance and error related) is displayed. Detailed data on the performance of each test is captured and stored in a database. Subsequent to the execution of a test, to see the detailed data the Load Reporter tool can be used as described below.



1: The Summary Button. Pressing the save button will show a summary of the data associated with the test (row) selected in the main tabular window.

2: The Save Button. Pressing the save button will generate a detailed report of the selected test and open a dialog box to save the report.

3: The Show Button. Pressing the show button will generate a detailed report and launch a browser to view the report. The report will not be saved.

4: Test Listing Window This window displays information about prior tests. A specific test is selected by clicking inside its row. The data displayed in this window can be sorted by clicking on any of the table headers. Furthermore, in a case where there are more tests than can be shown in the space available, a scrollbar (indicated as 6 above) allows the data to be scrolled.

5: Test Window Header Bar As noted, clicking on a field in the header bar will cause the displayed table entries to be sorted according to that column. The significance of the column data is in order from left to right.

run Each data set is given a run identifier according to the order in which load tests were conducted.

seqLen or Sequence length indicates the number of distinct load levels used during the load test.

By The E-Insights.com username under which the load test was conducted.

From The IP address from which the load test was conducted if known.

Name per the `testname` rule..

Start: The time the load test started.

End: The time the load test ended.

6: Slider Control – In cases where there are more test results than can be displayed in the display area (4) a slider is used to allow scrolling through the results. The slider is only displayed if necessary.

7: Information Box: Displays occasional information messages.

2.3 - The Load Report

The Save and Show buttons on the load report generator each produce a load report in the form of an HTML file. Load reports are designed to be easily readable summaries of load test results. Each load report consists of a notes and summary section, followed with a section with per page details follows with two graphs of overall performance.

3 – LoadGen Scripts & .lgx Files

.lgx files may be produced in any standard text editor program. Each consists of a number of instructions, one instruction per line. Some particular instructions must be included in order for a load test to run. The order in which the instructions occur in a .lgx file is unimportant except that pages must be defined before they can be referenced in sequences and variables and arguments related to a specific page must be listed immediately after the page is declared.

Instructions in a .lgx file are one per line. If the first non space character on a line is '#' the line is considered a comment and discarded. For convenience, long commands can be split across several lines by ending all but the last line with the '\' character. Hence

```
this is a \  
long line
```

is equivalent to

```
this is a long line
```

What follows is a list of acceptable instructions:

```
maxpages
testname*
page*
    element
    argument
    field
sequence*
file
useragent
auth
ramp
agentcount
runtime
```

* indicates that at least one instruction of this type is required by LoadGen.

A more detailed account follows:

maxpages

syntax: maxpages=[integer]

The maximum total number of pages that will be requested before a load is terminated. The default is 500.

Example: maxpages=750

LoadGen will load no more than 750 pages. Fewer than 750 pages will be loaded if fewer than 750 pages are allowed before the test terminates because of specified time constraints.

Setting maxpages to zero means that the number of pages tested plays no role in terminating a test (i.e. it is unlimited). It is recommended that maxpages only be set to unlimited after the load test script has been thoroughly tested and confidence gained that the test duration and specified levels do not cause any unacceptable problems.

testname

syntax: testname=[string]

The Name Parameter under which the data produced by the load test will be stored in the LoadGen database. There is no default value for testname; a value must be supplied by the user.

Example: testname="TestMySite"

LoadGen will store the load test data in the appropriate database under the name TestMySite.

file

syntax: `file=[.lgx filename]`

Includes the rules set forth in the indicated .lgx file.

Example: `file="commonargs.lgx"`

LoadGen will insert in-line the rules listed in commonargs.lgx. Trying to parse mutually- or self-referencing .lgx files is a bad idea.

useragent

syntax: `useragent=[string]`

Specifies the agent name LoadGen will include with requests. The agent name specifies browser, operating system, et cetera, and is sometimes relevant to the content a server will provide.

Example: `useragent="xxx"`

auth

syntax: `auth=[string:string]`

Specifies the username and password LoadGen will supply during the course of a load test. There is no default value for auth. LoadGen currently supports HTTP Basic Authentication to a single site only.

Example: `auth="gyges:briareus"`

LoadGen will use "gyges" as a username and "briareus" as a password in basic authentication.

page

syntax: `page=string URL`

This declares a paged named as the first string with the URL specified as the second argument. The page may later be referenced in sequences by using its name. The URL can include argument names (see below) enclosed between two '\$' characters. The argument references will be replaced with the current value of the arguments each time the page is requested.

Examples

`page=FrontPage http://www.somesite.com`

`page=PersonalizedFrontPage http://www.somesite.com?iam=\$myname\$`

This second form would substitute the current value for the argument 'myname' for the string '\$myname\$' every time the page is requested.

element

syntax: element=URL

The elements are associated with the last page declared. During testing, after a page is loaded, elements associated with it, if any, are also loaded. The time to load such elements are added to the page load time. Elements would typically specify images, but can specify anything. Note that while SiteCrawl will parse retrieved HTML documents and determine what 'elements' to load automatically, LoadGen does not attempt to do this. In fact, LoadGen simply records the amount of data it receives in response to its requests along with the elapsed time and transaction related status information and stores this. While it does 'read' the response data from the server (to properly emulate user behavior) it does not examine it in any way. This approach allows a single LoadGen instance to generate a much higher level of load against a server than would be possible if it were parsing each HTML file received since such parsing is a computationally expensive process.

argument

syntax: argument=string argument_declaration

Argument declarations are one of

`oneof list`

Choose at random one element from the list (of space separated strings).

`nmof int,int list`

Choose at random between the lower limit specified by the first integer and the higher limit specified by the second integer elements from the list of items (blank separated). The value is the concatenation of the chosen items. Note that 'bnmof' and '_nmof' behave similarly except that in the 'bnmof' case the concatenated items are separated with a space character whereas in the '_nmof' case the items are separated by a '+' character. This latter case may be needed when the arguments are being substituted into the URL of the associated page as opposed to being POST arguments.

Examples:

```
nmof 2,4 one two three four
could generate any of the following:
onetwothreefour
oneone
threetwofour
while
_nmof 1,3 one two three four
could generate
two
three+four
four+one+one
```

Note that the same element may be chosen several times.

prev string

Choose the previously generated value used for the argument named by the string. In this case a random value is not re-generated for the referenced argument. This allows a single randomly generated value to be used in several locations within a page request.

field

syntax: field=string=string

A POST argument named from the first string will be created and submitted with the page request. The value of the post argument is created from the second string by substituting for any argument names found in the string the values for those arguments. Argument names are indicated in the second string by enclosing the argument name between two '\$' characters. For example

```
argument size=oneof 1 2 3 4 5
field=thesize=size is $size$
```

Would result in the post argument “thesize” being sent with a value something like “size is 3”.

sequence

syntax: sequence=string, integer {string}+

The first string specifies the name of the sequence to be used for reporting purposes. The integer following the name (separated with a comma) represents a weight which is applied in sequence selection; more on this momentarily. The trailing list (there must be at least one) of strings specifies names of pages. It is an error to specify a page that has not been defined.

With regard to weights, each time an agent becomes free (either initially or after completing an earlier sequence) a random choice amongst the defined sequences is made to select the next sequence to be used by that agent. The random process is based on the weights normalized by the sum of the weights. The probability that a sequence A will be chosen is (weight of A)/(sum of all weights) .

Example:

```
sequence heavy,90 pagea , pageb , pagec , paged
sequence light,10 pagea
```

Will result in 90% of agent choices of sequence being the 'heavy' sequence in which the agent will in order request the four specified pages. The other 10% of agent choices will be the 'light' sequence which is a single page.

agentcount

syntax: agentcount=integer

Specifies the number of agents to use during the run. The default is value is 1. During an interactive session, the agentcount slider can also be used to set the number of

agents.

ramp

syntax: ramp=integer;real;real;real

That is an integer followed by three real numbers separated by the semi-colon character. The integer specifies the duration in minutes of each level of the load ramp. In order to have sufficient data to provide useful results this value cannot specify less than five minutes. The succeeding three arguments specify the initial load rate in pages per second, the increment by which this is increased at each ramp stage, and the terminal load level. For example

ramp=7,1.5,0.5,4

would start with a cumulative² page request rate of 1.5 pages per second. This level would be maintained for 7 minutes at which the rate would increase to 2.0 pages per second which would again be maintained for seven minutes. This pattern would continue until a load rate of 4.0 had been maintained for 7 minutes at which point the run would terminate. In an actual run, a one minute period with the load set at the initial level precedes this sequence and a one minute period with the load at the terminal level succeeds it. These bracketing one minute periods help to minimize the effect startup and shutdown have the on the measured data.

In interactive mode a ramp command need not be specified since the rate can be set with the slider control. In a batch mode a ramp command must be specified.

The ramp specifies the cumulative page load rate to be generated. This rate is divided evenly over the agents. Hence if a ramp specifies “ramp=5,10.0;1.0;20.0” and there were 10 agents “agentcount=10” then at the first stage of the ramp (10.0 pages per second) each agent would attempt to generate a load of 1.0 pages per second. The agent does this by calculating the average inter page request time necessary to generate this load – in this case 1.0 seconds between page load is necessary. As an agent initiates a page request it notes the time. Once the page request has completed and the status information is recorded, the time is noted again. If the elapsed time is greater or equal to the necessary to the inter page request time the next request is generated immediately, otherwise the agent will pause until sufficient time has elapsed before initiating the next request. The logis is in fact somewhat more complex and will allow an agent to issues several page requests quickly to make up time in case a single earlier request took longer that the inter page request time. Nonetheless, in a case where the average page load time is longer than the inter page request time the specific agent set size will not be able to achieve the desired load. In such a case the agentcount should be increased.

runtime

syntax: runtime=integer{:integer}


Specifying the runtime in minutes or in minutes and seconds (as is 10:30) . If a ramp is specified then the runtime determined from the ramp specification is used and issuing a runtime command will result in a warning.

² That is the load rate specified is not a load rate per agent but the cumulative load rate to be generated by all agents.

[To be added browserhitrate, akamaihitrate, akamize, maxpageloadtime, baseref]

Creating A LoadGen Configuration File

The image below with a text input box (containing "Pinter") two selection boxes and a submit button (labelled SEARCH in the above example) is typical of a basic search interface in a web environment. The HTML code generating this example is shown immediately below the image.



```
<FORM METHOD=GET ACTION=runQuery.cgi>
Information relating to: <INPUT TYPE=TEXT NAME="terms" WIDTH=20>
in
<SELECT NAME="SOURCE">
<OPTION VALUE=0>Books <OPTION VALUE=1>Magazines <OPTION VALUE=2>Newspapers
</SELECT>
<SELECT NAME="YEAR">
<OPTION VALUE=2002>2002 <OPTION VALUE=2001>2001 <OPTION VALUE=2000>2000
</SELECT>
<INPUT TYPE=HIDDEN NAME=feeder VALUE="LOC">
<INPUT TYPE=SUBMIT NAME=SEARCH VALUE=SEARCH VALUE=FREE>
</FORM>
```

"Pinter" was typed by the user into the text input field and selections were made in both selection boxes to generate prior to the image being captured. At this point clicking on the SEARCH button would cause the browser to initiate an HTTP GET transaction with the URL

```
runQuery.cgi?terms=Pinter&SOURCE=1&YEAR=2001&feeder=LOC&SEARCH=FREE
```

Note that the name and any associated value of the "SUBMIT" button are also sent, as is the 'hidden field' .

In order to include this page in a LoadGen test the .lgy would look like:

```
page=Search
http://server.to.test.com/runQuery.cgi?terms=\$t\$&SOURCE=\$s\$
&YEAR=$y$&feeder=$f$&SEARCH=FREE
argument t one of Pinter Strauss Shelly Rogers Smyth
argument s one of 0 1 2
argument y one of 2000 2001 2002
argument f one of LOC NA NOAA
```

During a load test, each time the page 'Search' is referenced, appropriate arguments are generated and substituted into the URL prior to making the server request.

If instead of being coded to use a GET operation, the above form had been coded to use a POST operation, the HTML would look identical to that shown above with the

exception that the term “ACTION=GET” would be replaced with “ACTION=POST” . The .lgx code would now look like

```
page=Search http://server.to.test.com/runQuery.cgi
argument t oneof Pinter Strauss Shelly Rogers Smyth
argument s oneof 0 1 2
argument y oneof 2000 2001 2002
argument f one of LOC NA NOAA
field=terms=$t$
field=SOURCE=$s$
field=YEAR=$y$
field=feeder=$f$
field=SEARCH=FREE
```

Note that in the above the “GET” arguments (coded in the text after the '?' character in the URL have been removed, and a collection of fields have been added. While the information being sent to the server is the same, the manner in which it is sent is significantly different. Note that web sites sometimes use elements of GET and POST together, where a POST form has some GET arguments encoded into the URL.

LoadGen will use a GET transaction (as specified in a FORM with the ACTION=GET command) whenever there are no fields specified for a page and will use a POST transaction whenever there is at least one field defined.

Arguments & Argument Ranges

The overall form of the .lgx entry can be determined from examining the HTML of the referring page (the request type and variable names). Determining valid ranges for the arguments requires an understanding of the serving process, or absent that may be gained through examining many site pages and inferring reasonable variable ranges. While the ideal situation in load testing would be to replicate the end user usage patterns this is generally not feasible; it is not known in advance, and may vary over time. For example, search terms may be influenced by external events, or by contextual or editorial changes on a site (“Search our library for new ...”).

While accurately simulating end user behavior is unattainable, it is nevertheless important to provide a sufficiently rich (in the sense of variable) load during testing to avoid simple caching mechanisms performing in a much more effective fashion than could be expected in reality. This is true of both search like operations (where one is performing read-only operations against a data base) and form submission like operations (where data may be written to a data base).

Since the goal of load testing is to build confidence that a system will behave satisfactorily in an environment that is neither fully defined no static, having the load tests push the boundaries of expected behavior (for example a higher percentage of search requests, or longer search terms than anticipated) is useful in that it may identify areas where problems could arise if user behavior varies even a small amount from that anticipated. Such a situation should either result in modifying the system to accommodate a wider range of behavior, or to re-validate the behavioral assumptions.

Appendix – Examples I

```
# set maxpages to unlimited and runtime to 5 minutes
maxpages=0
runtime=5:00
# declare the some pages along with their associated images
page= Front http://www.e-insights.com/index.php
element=http://www.e-insights.com/images/ei-logoB.gif
element=http://www.e-insights.com/images/dashboard.png
page=CMGR http://www.e-insights.com/indexcmgr.php
element=http://www.e-insights.com/images/dashboard.png
page=FinMod http://www.e-insights.com/indexfinmod.php
element=http://www.e-insights.com/images/dashboard.png
page=MUBS http://www.e-insights.com/indexmubs.php
element=http://www.e-insights.com/images/dashboard.png
page=DOCS http://www.e-insights.com/indexdocs.php
element=http://www.e-insights.com/images/dashboard.png
# a query with random data both in the URL and as POST arguments
page=ReplyToPost http://www.e-insights.com/showArgs.php?a=$getarg$&b=$lastget$
argument=getarg oneof get1 get2 get3
argument=lastget prev getarg
argument=sentence bnmof 10,30 hi there this is a strange thing here i dont know why i am
saying this
field=s1=$sentence$
field=s2=$sentence$
# declare one sequence only and a testname
sequence=test1,1 Front CMGR FinMod MUBS DOCS ReplyToPost
testname=YoYo
# note agentcount has not been specified so will be the default
```

Appendix – Example II

```
# Example II: Multiple Sequences
# load the standard stuff for this site
file=setup
# now define all the pages
page=FrontPage http://stage.teenmag.com/
file=elements/FrontPage
page=AllAbout http://stage.teenmag.com/all-about-you/
file=elements/AllAbout
page=GamesGadgets http://stage.teenmag.com/games-gadgets/
file=elements/GamesGadgets
page=Advice http://stage.teenmag.com/advice/
file=elements/Advice

page=CelebFinder http://stage.teenmag.com/celeb-stuff/celeb-finder/
file=elements/CelebFinder
page=CelebFinderX http://stage.teenmag.com/celeb-stuff/celeb-finder/$letter$/
file=elements/CelebFinderX
argument=letter oneof a b c d e f g h i j k l m n o p q r s t u v w xyz
# Search Page
page=Search http://stage.teenmag.com/search/fast_search?term=$term&x=1&type=$type$
# note term is padded with + between the fields since its in the URL
argument=term _nmof 1,3 esteem confused whyme righteous hottie cuticle exhausted stupid
argument=type oneof title by_type relevancy by_section publication_date
# Some Article Pages
page=AdviceArticle http://stage.teenmag.com/advice/$aid$

... several additional page declarations deleted for clarity

# now sequences think of weight as % of type – note line continuations
sequence=searchengineiA,20 FrontPage
sequence=searchengineiB,20 DailyBuzzArticle
sequence=light,25 FrontPage CelebFinder CelebFinderX SeenInTeenArticle
sequence=normal,20 FrontPage CelebFinderX Search Submit SeenInTeenArticle \
DailyBuzzArticle
sequence=heavy,10 FrontPage Advice GamesGadgets Search AllAbout \
Search CelebFinderX CelebFinder Submit Search Submit
sequence=obsessed,5 FrontPage AdviceArticle Advice GamesGadgets Search AllAbout \
Search CelebFinderX DailyBuzzArticle CelebFinder Submit Search Submit \
SeenInTeenArticle Search Submit SeenInTeenArticle Submit DailyBuzzArticle Search
```