

Parameterization in OpenSTA

Author: Ranjit Shewale (jcrvs@hotmail.com)

Date: 14th April 2003

What is OpenSTA?

OpenSTA is a load testing tool used by performance test engineers. For more details please refer to the websites www.opensta.org and <http://portal.opensta.org>.

What is parameterization ?

Parameterization is a simple way of trapping the values in the DOM of the HTML page and using this values in appropriate places in the subsequent pages.

Why do I need parameterization?

The HTML pages in the Internet have changed from static nature to dynamic. A user navigating from one page to other needs to have some identification or transaction information. HTTP protocol is stateless protocol, so how does this information is passed from one page to subsequent pages? The answer is the web application uses cookies to store information or use session variables. A typical example of this could be a user buying books at www.amazon.com site needs to hold the information from the time he clicks buy this book button till he reaches the checkout and pay by credit card page.

How do I perform parameterization in OpenSTA?

OpenSTA like any other load test tool has a script modeler and provides a way to handle the parameterization.

Let us consider an example and follow few steps.

Suppose you want to load test a broadvision server and need to record a script for the same.

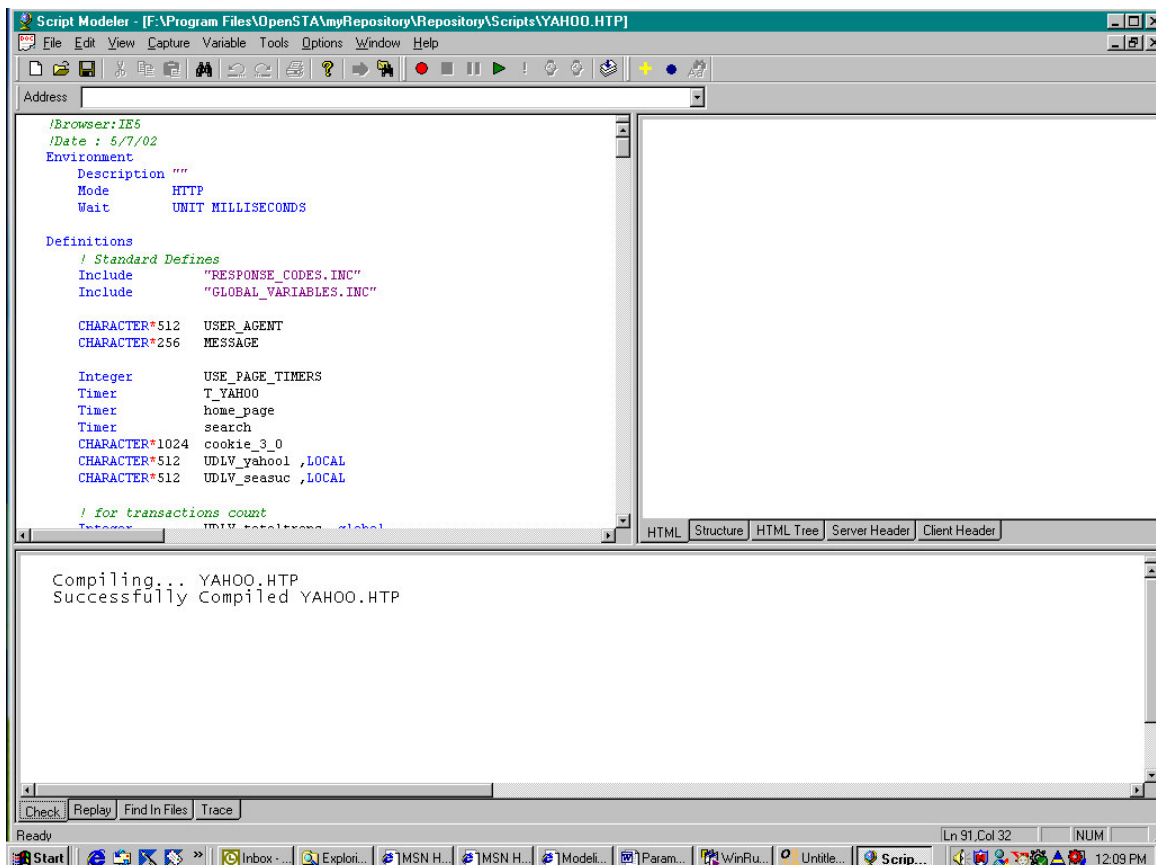
Steps 1: Analyze the web application

Broad vision web application, as soon as the home page is hit, creates a session identifier and engine identifier. This identifier needs to be passed by the browser to the subsequently pages being requested by it.

This is the most important step. You need to understand what you need to parameterize and hence it is recommended to understand the application first.

Step 2: Record the flow using OpenSTA script modeler

Using the default options, use the red 'Record' button or 'Capture->Record' menu-item to launch the configured browser and navigate through the web application path to be load tested. When done click the black 'Stop' button or 'Capture->Stop' menu-item to stop recording. The script is generated for the flow by script modeler. View the script using the script modeler.



The screenshot shows the OpenSTA Script Modeler window. The top-left pane displays the script code for 'YAHOO.HTP'. The code includes environment settings, definitions, and character declarations. The bottom-right pane shows the compilation status: 'Compiling... YAHOO.HTP' and 'Successfully Compiled YAHOO.HTP'. The bottom status bar indicates 'Ln 91, Col 32' and 'NUM'.

```
!Browser:IE5
!Date : 5/7/02
Environment
  Description ""
  Mode HTTP
  Wait UNIT MILLISECONDS

Definitions
! Standard Defines
Include "RESPONSE_CODES.INC"
Include "GLOBAL_VARIABLES.INC"

CHARACTER*512 USER_AGENT
CHARACTER*256 MESSAGE

Integer USE_PAGE_TIMERS
Timer T_YAHOO
Timer home_page
Timer search
CHARACTER*1024 cookie_3_0
CHARACTER*512 UDLV_yahool ,LOCAL
CHARACTER*512 UDLV_seasuc ,LOCAL

! for transactions count

```

You can see the code in the top left hand split window.

Scroll down this script code window and find the place where the session id and the engine id is being passed to the GET or POST url. This is normally the second post. Why would that be the case? Simple, the first post requested a page from the broad vision web application and this resulted in creating a new session id and engine id on server and this is being passed in the response of the first request. The script modeler receives this request and stored it in the <<..\Repository\Captures\scriptname.all>> file.

Step 3: How do I trap this hidden variable in my script?

Re-run the script after recording using the 'Replay' button or the 'Capture->Replay' menu-item.

Click the address bar to select the first GET/POST and then click the button 'URL Details' (the yellow arrow button in the toolbar of script modeler). This results in the recorded web page being displayed in the top-right window of the script modeler. Now since the session id and engine are hidden, click the HTML tree tab of the window (or the query result pane). This shows the DOM structure of the HTML page that was received as a response. Search the DOM to find the property session id and select the value associated with this property. Right-click the value associated with the property session id and select the menu-item 'Address'. This action pops up a dialog prompting for a variable name. Enter variable name as UDLV_sessionid. Click OK.

This adds variable in the definitions section and a function call to 'Load response...' in the script under the first post.

```
LOAD RESPONSE_INFO BODY ON 1 &  
INTO UDLV_sessionid &  
,WITH  
"HTML(0)/BODY(1)/CENTER(0)/TABLE(2)/TBODY(0)/TR(0)/TD(0)/TABLE(1)/TR  
(0)/TD(1)/FONT(0)/A(6):" &  
"TEXT:(0)"
```

And now, we have the session id trapped in a variable.

Use the same steps to trap the engine id.

The HTML tree structure i.e. the DOM structure of the web pages displayed in the script modeler is one of the powerful features of the OpenSTA.

Steps 4: How to use the session id in subsequent pages?

Assuming you have followed the 'step 1' and analyzed the web application and how it uses the information, we continue further.

In our case, the session id and the engine id is being used in the subsequent posts only in the URL. this can be found out by viewing the script. For other information to be trapped be sure to know how the application uses that information so as to parameterize it in proper posts in the script.

So, search for the recorded session id and engine id and replace it with the variable UDLV_sessionid, UDLV_engineid and use the SCL concatenation (+) operator.

So a script that had the original post earlier as

```
PRIMARY POST URI "http://myserver/cgi-
bin/immedium/broadway/scripts/bw_login.jsp HTTP/1.0" ON 3      &
HEADER DEFAULT_HEADERS                                     &
,WITH {"Accept: application/vnd.ms-powerpoint, application/vnd.ms-excel,
application/ms" &
"word, application/pdf, image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
&
"Referer: http://myserver/cgi-bin/immedium/broadway/scripts/bw_login.jsp",
&
"Accept-Language: hi",                                     &
"Content-Type: application/x-www-form-urlencoded",        &
"Content-Length: 165",                                    &
"Connection: Keep-Alive",                                 &
"Pragma: no-cache"}                                     &
,BODY
"BV_SessionID=@@@@2066901066.1042048820@@@@&BV_EngineID=
cccjadchfdhjelijcefecghkdgiidfhn.0" &
"UseBVCookieToBrowse=No&username=buffet&password=broadway&action=M
ember+Login"
```

will look after parameterization as

```
PRIMARY POST URI "http://myserver/cgi-
bin/immedium/broadway/scripts/bw_login.jsp HTTP/1.0" ON 3      &
HEADER DEFAULT_HEADERS                                     &
,WITH {"Accept: application/vnd.ms-powerpoint, application/vnd.ms-excel,
application/ms" &
"word, application/pdf, image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
&
"Referer: http://myserver/cgi-bin/immedium/broadway/scripts/bw_login.jsp",
&
"Accept-Language: hi",                                     &
"Content-Type: application/x-www-form-urlencoded",        &
"Content-Length: 165",                                    &
"Connection: Keep-Alive",                                 &
"Pragma: no-cache"}                                     &
,BODY "BV_SessionID="+ UDLV_bvsessionid +"&BV_EngineID="+
UDLV_bvengineid +"&" &
"UseBVCookieToBrowse=No&username=buffet&password=broadway&action=M
ember+Login"
```

What if I miss passing this information to few POSTS/GETS or parameterizing few posts?

Well, these posts will fail then and the response may be HTTP 404 error since the session id and engine id that were recorded while recording script are no longer valid for the subsequent runs of the script.

Steps 5: Is this all about parameterization?

No, the above applies for the parameterization of variable whose positions in the DOM remains constant. For parameterization values whose position is unknown, we have to use the function calls of the SCL language.

The 'steps 1 and 2' remain same. Now after you determine that the variable resides in response of the post1 in the script, then declare the necessary variables manually and call the following functions :-

1. Load response_info Body
2. Extract
3. Locate

A code snippet to trap the broad vision session id and engine id using SCL functions is as follows:-

```
Load Response_info Body on 1 Into UDLV_response
```

```
Set UDLV_offset =0
```

```
Set UDLV_temp = ~Extract(UDLV_offset, 1988, UDLV_response)
```

```
Set UDLV_offset = 13 + ~Locate("BV_SessionID=",UDLV_temp)
```

```
Set UDLV_offset2 = 12 + ~Locate("BV_EngineID=",UDLV_temp)
```

```
Set UDLV_bvsessionid = ~Extract(UDLV_offset, 29, UDLV_temp)
```

```
Set UDLV_bvengineid = ~Extract(UDLV_offset2, 33, UDLV_temp)
```

```
Log "BV session id :", UDLV_bvsessionid
```

```
Log "BV engine id :", UDLV_bvengineid
```

Follow the step 4 to use these dynamically trapped session id and engine id.

Conclusion

Load testing needs varying data so the database can have distributed information. Running recorded scripts in a test seldom help in load testing. Parameterization is needed for the script and this can be achieved in the above discussed way. OpenSTA has the capacity to handle static as well as dynamic variable parameterization.