User Experience, not Metrics

by:

R. Scott Barber

Part 12: Conditional Navigation (Based on Return Values)

How often when you're surfing the Web do you decide what to do next based on the information presented on the page you're currently viewing? If you're like me, you do this often. When the sequence of actions you take (your user path) is based on results returned by a previous activity, you're performing what's known as conditional navigation. This article discusses when you might want to script conditional navigation as part of a performance testing effort and shows you how to do it. It expands on the concepts introduced in Part 3 and also applies some of the methods discussed in Part 11 to capture return variables.

This is the twelfth article in the "User Experience, Not Metrics" series, which focuses on correlating customer satisfaction with your Web site application's performance as experienced by users. This article is intended for intermediate to advanced Rational Suite® TestStudio® users. An ability to apply conditional logic in C and to do some manual handling of socket connections will be extremely useful. If you haven't done so already, it would be valuable to review Parts 3 and 11 before reading this article.

When Are Conditional Navigation Scripts Required?

Recently, I was asked the following question: "I'm testing a flight reservation system with production data (reservations are placed but immediately canceled by the system based on test user identification). My scripts work well except for one problem: if the search returns no flight available, the script tries to book it anyway. This causes significant errors that couldn't occur if the user were accessing the system manually. How do I simulate searching for a new flight based on the return data when I don't know whether the flight will be available or not when I write the script?"

The main reason we want to be able to model and script conditional navigation is to test scenarios like this one. In this case there's no way to lock the flight availability (production) database, so we can't test this scenario using a data-driven approach. If we don't dynamically handle the possibility of not finding an available flight, the best we can do when the script requests reservations for a full or nonexistent flight is to end the script when the request times out. This could have a significant impact on performance, since the script would be requesting something the server wasn't programmed to handle. Besides, that same request would be (virtually) impossible for a real user to submit.

What this example should make clear is that conditional navigation

scenarios are more often used to keep your scripts from inadvertently modeling something a user wouldn't or couldn't do than to model something a user would do.

Do you really need to go to the extra effort of modeling and testing conditional navigation scenarios? In many cases this level of scripting is unnecessary, particularly when the Web site user only has to decide which static page to select next, or when the options are covered sufficiently by scripts that model each relevant scenario independently. Furthermore, you might be able to eliminate the possibility of conditional navigation scenarios by carefully selecting or manipulating test data. In some cases this is the best approach, but it's important to remember that you're trying to model real users as accurately as possible, and make your choice of approach accordingly.

A data-driven approach has these advantages over scripting a conditional navigation scenario:

- Scripts are generally faster and easier to create.
- Results are generally easier to analyze.
- Exception cases can often be tested with separate scripts.
- You have more control over the exact distribution of tasks within the test as a whole.

On the other hand, a data-driven approach is at a disadvantage compared to conditional navigation scripting in these ways:

- Scripts are generally less accurate when compared to real user patterns.
- Performance results are potentially skewed by the elimination of activities that could trigger very poor performance.
- Extra scripts may need to be recorded and maintained.

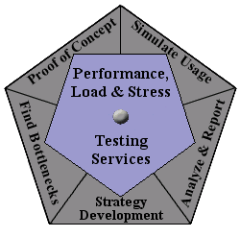
A static database is normally required for testing (often including a refresh after each test execution).

In Part 3, I showed you how to use what I referred to as smart scripts. What I was really talking about there was creating scripts to model randomness in user navigation, which I would call optional or variable navigation. What this article discusses is related conceptually, but here we actually want to make decisions based on interaction with the site, not just based on what random number we picked earlier in the script. Much like data-driven options to avoid conditional navigation, optional navigation may be all you need. As my father used to say, "Why do it the hard way if you can get the same results the easy way?"

There's no rule of thumb for which approach to use. The bottom line is that it's really up to you to determine when separate scripts are "good enough" and when conditional navigation scripts are required. Your particular testing needs may best be served by a data-driven approach or by optional (variable) navigation. Once you evaluate the risks and benefits of each of these approaches, it will almost always become clear which method is best for your particular project or application. Many times the data-driven approach is appropriate early in the testing lifecycle, and the conditional navigation approach becomes more appropriate as the application gets closer to production.

Getting Ready to Script Conditional Navigation

Creating scripts that include conditional navigation is probably the most complex topic we've addressed in this series. Since it's an extension of the smart scripting method discussed in Part 3 and uses some of



the same matching techniques described in Part 11, I'm going to assume that you've read, understand, and can apply the concepts presented in those articles. Specifically, you should know how to do the following:

- Identify the VU commands associated with a single Web page (or group of pages).
- Place a single Web page (or group of pages) inside a simple if block.
- Manually ensure that socket connections aren't stranded, whether or not the if block is executed, or record a separate (API) split script for each Web page (or group of pages) to be included in the if block.
- Write a block of code to isolate a variable from the _response file.

If you're not comfortable with these tasks, please review the earlier articles and complete the "Now You Try It" exercises before continuing with this article.

Scripting the Flight Reservation Scenario

Now I'll take you step by step through the process of creating a conditional navigation script, using the flight reservation scenario discussed earlier as an example. Modeling that scenario yields the diagram in Figure 1.

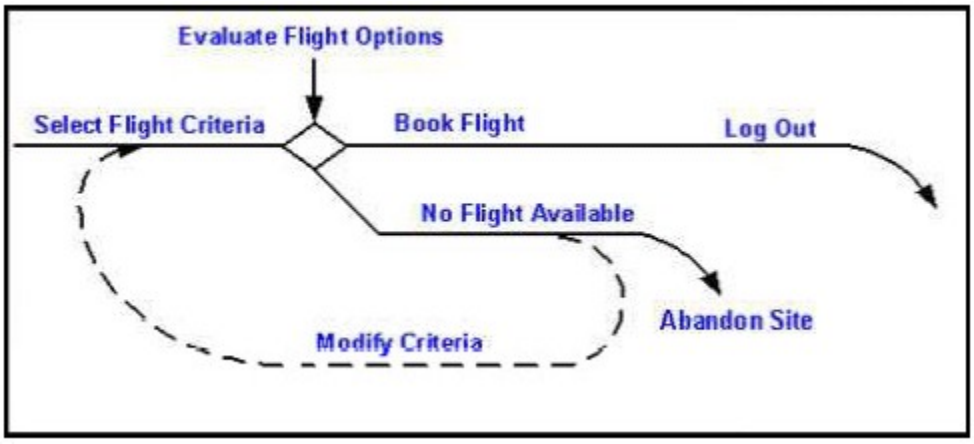


Figure 1: A model of our flight reservation scenario

Looking at Figure 1, you'll see that it follows the rules of user community modeling discussed in Part 3 and Part 4. The one difference is that this model contains a diamond that represents a decision point. Since we're not predetermining the distribution of tasks in this model, we use that decision point to indicate that tasks following "Select Flight Criteria" are determined by user evaluation of flight options.

Step 1: Record the Basic Scripts

There are two parts to recording the basic scripts for this model.

First, we'll record two straight-line scripts (as described in Part 3): one selecting criteria and booking a flight, the other selecting criteria, revising criteria, and submitting the new criteria. Both of these scripts need to be recorded with the "Display recorded rows" option on the Generator tab of the Session Record Options window set to All (see Figure 2).

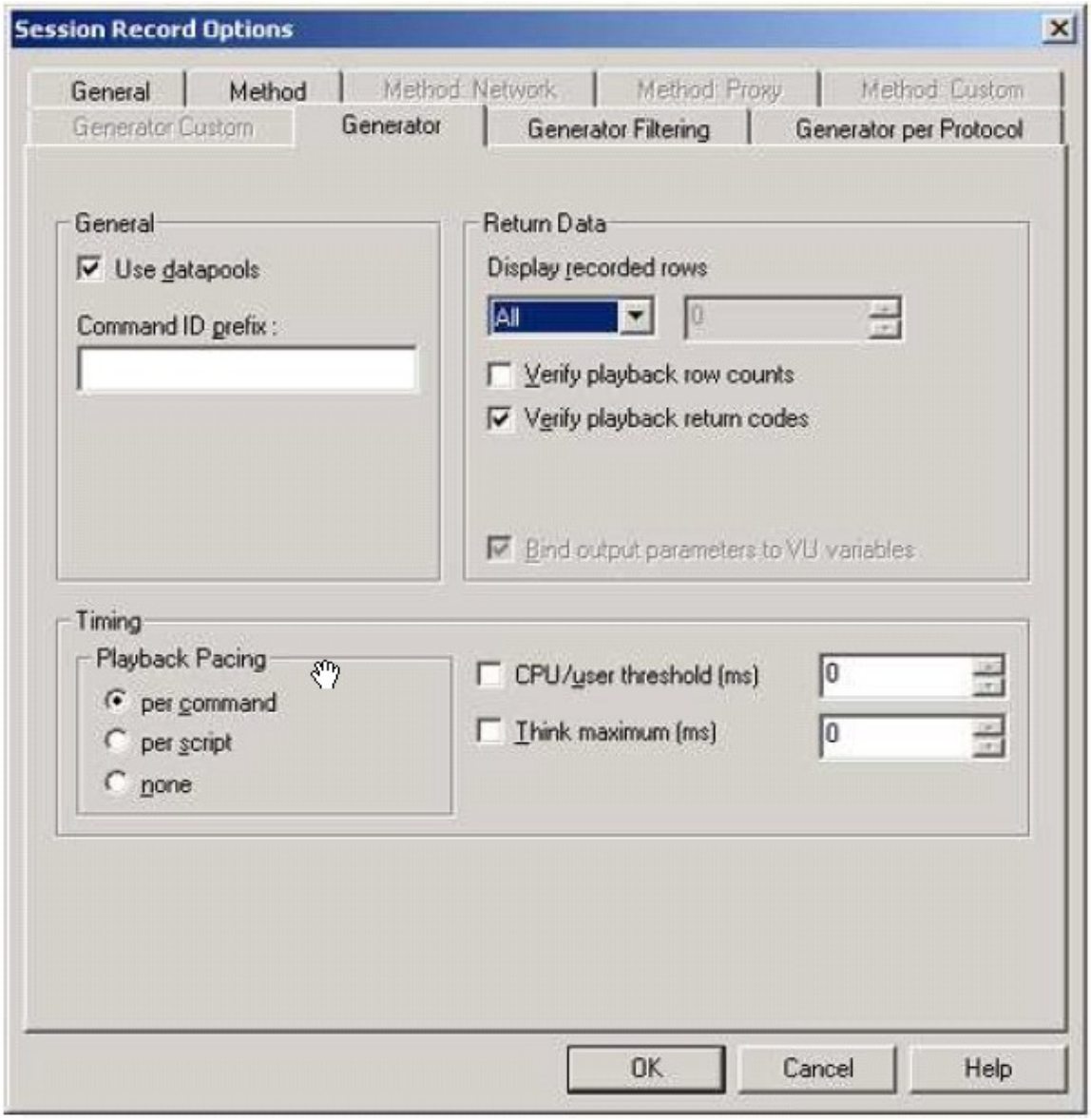
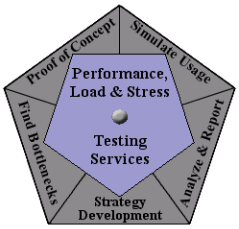


Figure 2: The "Display recorded rows" option set to All

Then we'll regenerate those scripts or record new scripts, this time with "Display recorded rows" set to None. If you record new scripts, I recommend recording three independent split scripts (meaning you'll stop and restart the recording between split scripts or use API recording): one to select criteria, one to book a flight, and one to return to the criteria selection page, modify your options, and submit your new options. We'll record three independent split scripts because although it's extra work, it makes handling socket connections easier.

We'll be using the straight-line scripts we first recorded to determine how to identify the conditions of our navigation and will then discard them. The split scripts we recorded will be the ones we use to create our final script.

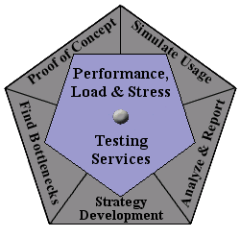


Step 2: Determine How to Identify the Possible Conditions

Now let's determine how to identify the possible conditions — in this case whether there's an available flight that meets the entered criteria or not — by analyzing the straight-line scripts we recorded in step 1.

If we compare both scripts from the request generated by submitting the travel criteria through the first part of the server response, we can easily identify a unique difference in the returned pages based on whether a flight matching the criteria was available. Listing 1 shows this segment from the script where an available flight was found that matched the search criteria.

```
www_a_itn_net = http_request ["book_fu~003"] "www.itn.net:80",

HTTP_CONN_DIRECT,
"GET /cgi/air?stamp="

+ http_url_encode(datapool_value(DP1,
",itn/air/fakeairline&rt_ow="
+ http_url_encode(datapool_value(DP1,
"&airline="
+ http_url_encode(datapool_value(DP1,
"&persons="
+ http_url_encode(datapool_value(DP1,
"&air_avail="
+ http_url_encode(datapool_value(DP1,
"&depart="
+ http_url_encode(datapool_value(DP1,
"&dest.0="
+ http_url_encode(datapool_value(DP1,
"&mon_abbr.0="
+ http_url_encode(datapool_value(DP1,
"&date.0="
+ http_url_encode(datapool_value(DP1,
"&hour_ampm.0="
+ http_url_encode(datapool_value(DP1,
"&mon_abbr.1="
+ http_url_encode(datapool_value(DP1,
"&date.1="
+ http_url_encode(datapool_value(DP1,
"&hour_ampm.1="
+ http_url_encode(datapool_value(DP1,
"&best_itins="
+ http_url_encode(datapool_value(DP1,
"&return_to="
+ http_url_encode(datapool_value(DP1,
" HTTP/1.1\r\n"
"stamp"))+
"rt_ow"))+
"airline"))+
"persons"))+
"air_avail"))+
"depart"))+
"dest.0"))+
```




```
"mon_abbr.0"))+
"date.0"))+
"hour_ampm.0"))+
"mon_abbr.1"))+
"date.1"))+
"hour_ampm.1"))+
"best_itins"))+
"return_to"))+

"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, applicat"
"ion/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */"
"*\r\n"
"Referer: http://www.fakeairline.com/\r\n"
"Accept-Language: en-us\r\n"
"Accept-Encoding: gzip, deflate\r\n"
"User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)\r\n"
"Host: www.itn.net\r\n"
"Connection: Keep-Alive\r\n"
"\r\n";

set Server_connection = www_a_itn_net;

http_header_recv ["book_fu~004"] 200; /* OK */

http_nrecv ["book_fu~005"] 100 %% ; /* 38632 bytes (38632 total) */

#if 0
"HTTP/1.0 200 OK\r\n"
"Date: Wed, 13 Nov 2002 01:34:55 GMT\r\n"
"Server: Apache/1.3.26 (Unix) mod_ssl/2.8.10 OpenSSL/0.9.6e\r\n"
"Connection: close\r\n"
"Content-Type: text/html\r\n"
"\r\n"
"<html><head><title>Create itinerary: Select flights</title>\n"
"<link rev=\"made\" href=\"mailto:webmaster@itn.net\">\n"
"<!--\n"
...
#endif
```

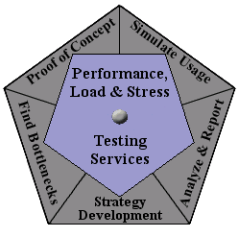
Listing 1: "Flight found" script segment

Notice that all of the criteria for the requested flight are handled in datapools, and that the response to that request is to return a page titled "Create itinerary: Select flights." Now look at Listing 2 to see how this segment differs in the script where a qualifying flight wasn't found.

```
www_a_itn_net = http_request ["modify_~003"] "www.itn.net:80",
HTTP_CONN_DIRECT,
"GET /cgi/air?stamp="

+ http_url_encode(datapool_value(DP1, "stamp"))+
",itn/air/fakeairline&rt_ow="
+ http_url_encode(datapool_value(DP1, "rt_ow"))+
"&airline="
+ http_url_encode(datapool_value(DP1, "airline"))+
```

```
"&persons="
+ http_url_encode(datapool_value(DP1, "persons"))+
"&air_avail="
+ http_url_encode(datapool_value(DP1, "air_avail"))+
"&depart="
+ http_url_encode(datapool_value(DP1, "depart"))+
"&dest.0="
+ http_url_encode(datapool_value(DP1, "dest.0"))+
"&mon_abbr.0="
+ http_url_encode(datapool_value(DP1, "mon_abbr.0"))+
"&date.0="
+ http_url_encode(datapool_value(DP1, "date.0"))+
"&hour_ampm.0="
+ http_url_encode(datapool_value(DP1, "hour_ampm.0"))+
"&mon_abbr.1="
+ http_url_encode(datapool_value(DP1, "mon_abbr.1"))+
"&date.1="
+ http_url_encode(datapool_value(DP1, "date.1"))+
"&hour_ampm.1="
+ http_url_encode(datapool_value(DP1, "hour_ampm.1"))+
"&best_itins="
+ http_url_encode(datapool_value(DP1, "best_itins"))+
"&return_to="
+ http_url_encode(datapool_value(DP1, "return_to"))+
" HTTP/1.1\r\n"
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, applicat"
"ion/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */"
"*\r\n"
"Referer: http://www.fakeairline.com/\r\n"
"Accept-Language: en-us\r\n"
"Accept-Encoding: gzip, deflate\r\n"
"User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)\r\n"
"Host: www.itn.net\r\n"
"Connection: Keep-Alive\r\n"
"\r\n";
set Server_connection = www_a_itn_net;

http_header_recv ["modify_~004"] 200; /* OK */

http_nrecv ["modify_~005"] 100 %% ; /* 25112 bytes (25112 total) */

#if 0
"HTTP/1.0 200 OK\r\n"
"Date: Wed, 13 Nov 2002 01:36:25 GMT\r\n"
"Server: Apache/1.3.26 (Unix) mod_ssl/2.8.10 OpenSSL/0.9.6e\r\n"
"Connection: close\r\n"
"Content-Type: text/html\r\n"
"\r\n"
"<html><head><title> No Qualifying Flights</title>\n"
"<link rev=\"made\" href=\"mailto:webmaster@itn.net\">\n"
"<!--\n"
...
#endif
```

Listing 2: "Flight not found" script segment



You'll notice that the two requests are virtually identical but the title of the returned document is quite different. This time the request returned a page titled "No Qualifying Flights." So we've determined a unique way to identify the conditions: by evaluating the document title in the _response file. We must now write the code to handle each condition accordingly.

Step 3: Create a New Script That Includes Conditional Logic

We're going to start by working with the split script we recorded in step 1 to select flight criteria. We'll modify this script to include the information from the other two split scripts as well as code to evaluate the condition. I've attached commented versions of the split scripts — the "select criteria" script, the "book flight" script, and the "modify criteria" script — so you can follow along as you read the discussion below. I'd suggest that you focus on the concepts and the C scripting as you read, and then later review the scripts at your leisure to ensure that you can trace the sockets, datapool values, and emulation commands.

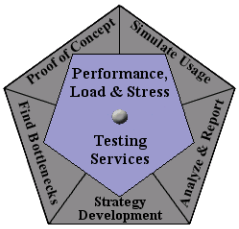
First we must declare the variables we'll need in order to evaluate the condition. Capturing the title of the Web page will be handled exactly the same way as capturing a session ID was in Part 11. Adding our variable declaration will result in the declaration section of the script shown in Listing 3.

```
#include <VU.h>

string str_title;
int int_start;
int int_end;

{
push Http_control = HTTP_PARTIAL_OK | HTTP_CACHE_OK | HTTP_REDIRECT_OK;
push Timeout_scale = 200; /* Set timeouts to 200% of maximum response time */
push Think_def = "LR";
Min_tmout = 120000; /* Set minimum Timeout_val to 2 minutes */

push Timeout_val = Min_tmout;
```

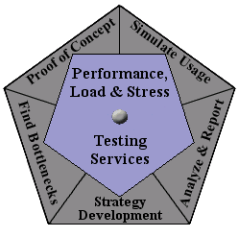
Listing 3: New script, variable declaration added to header

Now we'll start merging the three split scripts. First we'll include the criteria submission request (http_request) from Listings 1 and 2 that's common to both the "book flight" split script and the "modify criteria" split script, and the subsequent request (which is also identical between these two split scripts). These requests will be added after the last receive of the "select criteria" script but before the pop commands. Remember to also copy any datapool declarations that accompany this request. See Listing 4.

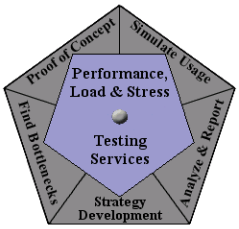
```
set Server_connection = fakeairlines_ssl_insightfirst_com;
http_header_recv ["select ~1.089"] 200; /* OK */
http_nrecv ["select ~1.090"] 100 %% ; /* 125 bytes */
http_disconnect(fakeairlines_ssl_insightfirst_com);

fakeairlines_ssl_insightfirst_com_1 = http_request ["book fl~2.001"]
"fakeairlinescollect.insightfirst.com:80",
HTTP_CONN_DIRECT,
"GET /data?if_nt_FF-Origin="
```




```
+ http_url_encode(datapool_value(DP1, "if_nt_FF-Origin"))+
"&if_nt_FF-Destination="
+ http_url_encode(datapool_value(DP1, "if_nt_FF-Destination"))+
"&if_nt_FF-Market="
+ http_url_encode(datapool_value(DP1, "if_nt_FF-Market"))+
"&if_nt_FF-Depart_Date=Dec/20/5%20am&if_nt_FFReturn_
Date=Dec/30/5%20am&if"
"_nt_FF-SearchBy="
+ http_url_encode(datapool_value(DP1, "if_nt_FF-SearchBy"))+
"&if_nt_FF-Passenger_Count="
+ http_url_encode(datapool_value(DP1, "if_nt_FF-Passenger_Count"))+
"&tax0_SiteID="
+ http_url_encode(datapool_value(DP1, "tax0_SiteID_1"))+
"&if_nt_URL=http%3A//www.fakeairlines.com/&if_pv="
""
+ http_url_encode(datapool_value(DP1, "if_pv"))+
" HTTP/1.1\r\n"
"Accept: */*\r\n"
"Referer: http://www.fakeairlines.com/"
"Accept-Language: en-us\r\n"
"Accept-Encoding: gzip, deflate\r\n"
"User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)\r\n"
"Host: fakeairlinescollect.insightfirst.com\r\n"
"Connection: Keep-Alive\r\n"
"Cookie: fakeairlines112202=4163617\r\n"
"\r\n";

SgenURI_001 = _reference_URI; /* Save "Referer:" string */

set Server_connection = fakeairlines_ssl_insightfirst_com_1;

http_header_recv ["book fl~2.002"] 200; /* OK */

http_nrecv ["book fl~2.003"] 100 %% ; /* 125 bytes */

http_disconnect(fakeairlines_ssl_insightfirst_com_1);
set Think_avg = 3195;

www_ual_com_3 = http_request ["book fl~2.004"] "www.fakeairlines.com:80",
HTTP_CONN_DIRECT,
"GET /homepage/homepagecontrol?action="

+ http_url_encode(datapool_value(DP1, "action"))+
"&rt_ow="
+ http_url_encode(datapool_value(DP1, "rt_ow"))+
"&airline="
+ http_url_encode(datapool_value(DP1, "airline"))+
"&air_avail="
+ http_url_encode(datapool_value(DP1, "air_avail"))+
"&depart="
+ http_url_encode(datapool_value(DP1, "depart"))+
"&dest.0="
+ http_url_encode(datapool_value(DP1, "dest.0"))+
"&mon_abbr.0="
+ http_url_encode(datapool_value(DP1, "mon_abbr.0"))+
```




```
"&date.0="
+ http_url_encode(datapool_value(DP1, "date.0"))+
"&hour_ampm.0="
+ http_url_encode(datapool_value(DP1, "hour_ampm.0"))+
"&mon_abbr.1="
+ http_url_encode(datapool_value(DP1, "mon_abbr.1"))+
"&date.1="
+ http_url_encode(datapool_value(DP1, "date.1"))+
"&hour_ampm.1="
+ http_url_encode(datapool_value(DP1, "hour_ampm.1"))+
"&lowestfares="
+ http_url_encode(datapool_value(DP1, "lowestfares"))+
"&persons="
+ http_url_encode(datapool_value(DP1, "persons"))+
" HTTP/1.1\r\n"
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, applicat"
"ion/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */"
"*\r\n"
"Referer: " + SgenURI_001 + "\r\n"
/* "Referer: http://www.fakeairlines.com/" */
"Accept-Language: en-us\r\n"
"Accept-Encoding: gzip, deflate\r\n"
"User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)\r\n"
"Host: www.fakeairlines.com\r\n"
"Connection: Keep-Alive\r\n"
"Cookie: ARPT=KIWVYOS64.95.89.82CKJYQ; ifSes=1\r\n"
"\r\n";
set Server_connection = www_ual_com_3;

http_header_recv ["book fl~2.005"] 302; /* Moved temporarily */

http_disconnect(www_ual_com_3);
set Think_avg = 100;

www_a_itn_net = http_request ["book fl~2.006"] "www.itn.net:80",
HTTP_CONN_DIRECT,
"GET /cgi/air?stamp="
+ http_url_encode(datapool_value(DP1, "stamp"))+
",itn/air/fakeairlines&rt_ow="
+ http_url_encode(datapool_value(DP1, "rt_ow"))+
"&airline="
+ http_url_encode(datapool_value(DP1, "airline"))+
"&persons="
+ http_url_encode(datapool_value(DP1, "persons"))+
"&air_avail="
+ http_url_encode(datapool_value(DP1, "air_avail"))+
"&depart="
+ http_url_encode(datapool_value(DP1, "depart"))+
"&dest.0="
+ http_url_encode(datapool_value(DP1, "dest.0"))+
"&mon_abbr.0="
+ http_url_encode(datapool_value(DP1, "mon_abbr.0"))+
"&date.0="
+ http_url_encode(datapool_value(DP1, "date.0"))+
"&hour_ampm.0="
```

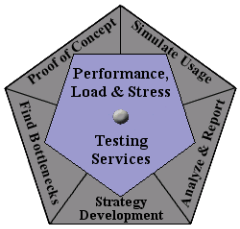



```
+ http_url_encode(datapool_value(DP1, "hour_ampm.0"))+
"&mon_abbr.1="
+ http_url_encode(datapool_value(DP1, "mon_abbr.1"))+
"&date.1="
+ http_url_encode(datapool_value(DP1, "date.1"))+
"&hour_ampm.1="
+ http_url_encode(datapool_value(DP1, "hour_ampm.1"))+
"&best_itins="
+ http_url_encode(datapool_value(DP1, "best_itins"))+
"&return_to="
+ http_url_encode(datapool_value(DP1, "return_to"))+
" HTTP/1.1\r\n"
"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, applicat"
"ion/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */"
"*\r\n"
"Referer: " + SgenURI_001 + "\r\n"
/* "Referer: http://www.fakeairlines.com/" */
"Accept-Language: en-us\r\n"
"Accept-Encoding: gzip, deflate\r\n"
"User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)\r\n"
"Host: www.itn.net\r\n"
"Connection: Keep-Alive\r\n"
"\r\n";
{string SgenURI_008; }

SgenURI_008 = _reference_URI; /* Save "Referer:" string */

set Server_connection = www_a_itn_net;

http_header_recv ["book fl~2.007"] 200; /* OK */

http_nrecv ["book fl~2.008"] 100 %% ; /* 35650 bytes (35650 total) */

http_disconnect(www_a_itn_net);

pop [Think_def, Think_avg, Timeout_val, Timeout_scale];

pop Http_control

DATAPOOL_CONFIG "select" OVERRIDE DP_NOWRAP DP_SEQUENTIAL DP_SHARED

...
```

Listing 4: New script, criteria submission request added

Now, between the last two lines we copied we'll add our code to capture the title string. Then immediately above the pop commands we'll add our conditional logic. See Listing 5.

```
http_nrecv ["book fl~2.008"] 100 %% ; /* 35650 bytes (35650 total) */

int_start = strstr(_response, "<title>");
int_end = strstr(_response, "</title>");
str_title = substr(_response, int_start + 7, int_end -(int_start + 7));

http_disconnect(www_a_itn_net);
```




```
while (str_title != "Create itinerary: Select flights")
{

}

pop [Think_def, Think_avg, Timeout_val, Timeout_scale];

pop Http_control
```

Listing 5: New script, conditional logic added

The first three new lines simply capture the title of the returned HTML document in the string str_title using the same method discussed in Part 11. The new block after http_disconnect is a while loop that says in effect, "While the title of the HTML document is not 'Create itinerary: select flights,' execute the code between the braces."

This gets us close to what we want to do. What we haven't done yet is to allow for users to abandon the site or give up. As is, the code simulates a scenario where users continue changing criteria until they find a flight. So let's add another piece of code to simulate users abandoning the site. We'll have the program select a random number between 1 and 3; every time a 3 comes up, the program will simulate a user abandoning the application and will write a message to that effect to the log file. See Listing 6.

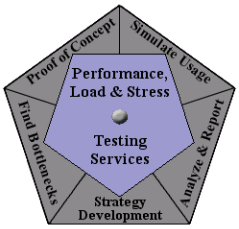
```
int_start = strstr(_response, "<title>");
int_end = strstr(_response, "</title>");
str_title = substr(_response, int_start + 7, int_end -(int_start + 7));
http_disconnect(www_a_itn_net);

while (str_title != "Create itinerary: Select flights") {

if (uniform(1,3) == 3){
user_exit(0,"User abandoned application");
}

}

pop [Think_def, Think_avg, Timeout_val, Timeout_scale];
pop Http_control
```

Listing 6: New script, abandonment logic added

The new code says in effect, "If the randomly generated number is 3, terminate this virtual user normally and write the message 'User abandoned application' to the log file."

The last thing we need to do before copying and pasting the rest of the emulation commands into the script is to add the three lines of code to reevaluate the HTML title after the new criteria have been submitted. See Listing 7.

```
int_start = strstr(_response, "<title>");
int_end = strstr(_response, "</title>");
str_title = substr(_response, int_start + 7, int_end -(int_start + 7));
http_disconnect(www_a_itn_net);
while (str_title != "Create itinerary: Select flights") {
```




```
if (uniform(1,3) == 3){
user_exit(0,"User abandoned application");
}

int_start = strstr(_response, "<title>");
int_end = strstr(_response, "</title>");
str_title = substr(_response, int_start + 7, int_end -(int_start + 7));
}

pop [Think_def, Think_avg, Timeout_val, Timeout_scale];
pop Http_control
```

Listing 7: New script, code to reevaluate the HTML title added

Finally, we're going to copy the remaining emulation commands from the split scripts and paste them into their correct places in the new script, as shown in Listing 8 and in the final modified script. First, we copy the remaining emulation commands from the "modify criteria" split script (from the end of the emulation commands we've already copied to create Listing 4 to the last emulation command in the script, which is the last line before the pop commands) and paste them after the end of the if block but before the code to reevaluate the HTML title. Next we copy the emulation commands from the "book flight" split script and paste them below the while block but before the pop commands. Once again, remember to ensure that the datapool value declarations are also copied and that all of the socket connects and disconnects are matched.

```
int_start = strstr(_response, "<title>");
int_end = strstr(_response, "</title>");
str_title = substr(_response, int_start + 7, int_end -(int_start + 7));
http_disconnect(www_a_itn_net);
while (str_title != "Create itinerary: Select flights") {
if (uniform(1,3) == 3){
user_exit(0,"User abandoned application");
}
/* Remaining "modify criteria" code goes here. */

int_start = strstr(_response, "<title>");
int_end = strstr(_response, "</title>");
str_title = substr(_response, int_start + 7, int_end -(int_start + 7));
}

/* Remaining "book flight" code goes here. */

pop [Think_def, Think_avg, Timeout_val, Timeout_scale];

pop Http_control
```

Listing 8: New script, with remaining code

Let's summarize what we've done conceptually. Starting with the straight-line script for selecting an available flight and booking it, all we really did was (1) add code to evaluate the response to validate that a suitable flight is available, and (2) insert the emulation commands to modify those criteria if the requested flight isn't available.

It's important to keep the simplicity of the concept in the front of your mind as you're modifying the scripts. Otherwise, it's easy to get caught up in the code and make a real mess of your scripts (which I often do). This is definitely something you'll want to do on your own a few times before you feel truly comfortable doing it on a project with a deadline. After you've done it successfully a few times, it will become second nature. Don't be discouraged the first time you compile your script and see the message "too many errors." Most of the time it's just a simple mistake like a semicolon or brace out of place that confuses the rest of the script.

Now You Try It

As always, I recommend that you try this method on your own. I think the flight reservation scenario is a very good one for a first exercise on conditional navigation. I checked out several airline reservation sites and found that every one had different HTML titles based on whether qualifying flights were found. I suggest you walk through the exercise in this article, step by step. Once you've mastered that, I suggest trying a different scenario altogether, like searching for a book on a bookstore site and then building a datapool of books that may or not be available on the site.

Summing It Up

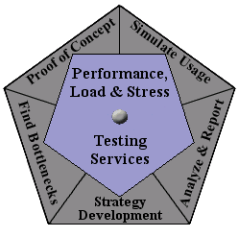
Adding the ability to program conditional navigation into your performance scripts will probably save you from having to look at your client one day and say "There's no way I can script that!" Conditional navigation isn't something that you'll use often, but when you need it, there's generally no other way to accurately simulate users of the application under test.

Acknowledgments

- The original version of this article was written on commission for IBM Rational and can be found on the IBM DeveloperWorks web site

About the Author

Scott Barber is the CTO of PerfTestPlus (www.PerfTestPlus.com) and Co-Founder of the Workshop on Performance and Reliability (WOPR – www.performance-workshop.org). Scott's particular specialties are testing and analyzing performance for complex systems, developing customized testing methodologies, testing embedded systems, testing biometric identification and security systems, group facilitation and authoring instructional or educational materials. In recognition of his standing as a thought leading performance tester, Scott was invited to be a monthly columnist for Software Test and Performance Magazine in addition to his regular contributions to this and other top software testing print and on-line publications, is regularly invited to participate in industry advancing professional workshops and to present at a wide variety of software development and testing venues. His presentations are well received by industry and academic conferences, college classes, local user groups and individual corporations. Scott is active in his personal mission of improving the state of performance testing across the industry by collaborating with other industry authors, thought leaders and expert practitioners as well as volunteering his time to establish and grow industry organizations. His tireless dedication to the advancement of software testing in general and specifically performance



testing is often referred to as a hobby in addition to a job due to the enjoyment he gains from his efforts.

About PerfTestPlus

PerfTestPlus was founded on the concept of making software testing industry expertise and thought-leadership available to organizations, large and small, who want to push their testing beyond "state-of-the-practice" to "state-of-the-art." Our founders are dedicated to delivering expert level software-testing-related services in a manner that is both ethical and cost-effective. PerfTestPlus enables individual experts to deliver expert-level services to clients who value true expertise. Rather than trying to find individuals to fit some pre-determined expertise or service offering, PerfTestPlus builds its services around the expertise of its employees. What this means to you is that when you hire an analyst, trainer, mentor or consultant through PerfTestPlus, what you get is someone who is passionate about what you have hired them to do, someone who considers that task to be their specialty, someone who is willing to stake their personal reputation on the quality of their work - not just the reputation of a distant and "faceless" company.