



A Framework for Performance Testing the SAP Portal

The Challenge

We load tested the SAP Portal for a worldwide defense contractor to provide single sign-on and centralized resource access for 80,000 employees. The portal infrastructure consisted of 28 servers, 3 firewalls, numerous switches and routers, 3 databases and 32 instances of the Portal Application Server. Our load injector farm consisted of 12 servers distributed across 6 geographic locations, and we maintained 40+ LoadRunner 8.1.4 scripts.

The project lasted 9 months and spanned multiple releases of the Portal, with frequent changes that impacted both performance and script integrity. Once a code change was implemented, we were expected to be ready to run a test “instantly”. As these changes often broke our scripts, we were forced to develop techniques to enhance script maintainability.

This Practical Solution illustrates some of the techniques we employed to develop maintainable and efficient LoadRunner scripts in this dynamic SAP Portal environment.

The Solution

We designed a scripting framework to facilitate re-recording as changes were implemented. The elements of the framework are:

1. **Script Template**: A template with multiple named Action sections to enable re-recording portions of the script in a specific Action section, so as not to require complete re-recording
2. **Recording Method**: We chose “URL Mode” recording, which enabled us to deal with frequent context changes and to isolate specific resources to pinpoint bottlenecks.
3. **Correlation Rules**: We created a custom correlation file with rules to automatically regenerate `web_reg_save_param` statements to correlate dynamic session variables
4. **Modularization**: We created a reusable Authentication Action that every script could use
5. **Execution Validation**: We used global text checks in lieu of individual page checks to eliminate validation check code maintenance.

These five elements, described in more detail below, enabled us to update scripts and turn out new tests usually within 4 to 6 hours of delivery of a new release.

Lessons Learned

- Time spent at the beginning to experiment with approaches and develop a solid design paid dividends throughout the project
- Use global text checks to capture the application errors to minimize text check code maintenance
- LoadRunner’s built-in SAP correlation rules are insufficient; time spent defining additional rules yields a recurring benefit
- By defining a framework, we were able to reduce script maintenance from 16 hours to 4-6 hours.



Technical Issues and Solutions

Issue: Dynamic application changes forcing re-records

Practical Solution: Design a Script Template

To make it easy to re-record small sections of a script instead of re-recording the entire workflow, we designed a template structured with named Action sections and with specific contents in the vuser_init and vuser_end sections. As LoadRunner supports re-recording and regenerating the script for a specific Action, this structure made maintenance much faster.

An example of our structure:

```
//      vuser_init
//      LoadSignOnPage
//      Authenticate
//      -----   BEGIN MAIN BODY
//      HOM_WC_init           #1
//      PortalServer_Global   #1a
//      MWS_MW_init           #2
//      PortalServer_USREGION #2a
//      MWS_MW_Resources      #3
//      MWS_MW_Overview       #4
//      MWS_MW_UReturn1       #5
//      MWS_CT_init           #6
//      MWS_CT_Docushare      #7
//      MWS_CT_UReturn        #8
//      MWS_CT_Search         #9
//      MWS_MW_UReturn        #10
//      MWS_MW_UReturn2       #11
//      -----   END MAIN BODY
//      vuser_end
```

Issue: Determining the “best” recording method for ease of maintenance

Practical Solution: URL mode

We experimented with all of the (LR 8.1.4) recording modes, from Click-N-Script to HTTP to URL Mode, and each setting within those modes.

The recording mode we chose may seem counter-intuitive: we picked URL Mode. As new parts of the Portal were implemented, the context from page to page often changed. Thus the lower-level/lower-context-less URL mode proved best. We could easily replace individual Action sections without losing context.

The second benefit of URL mode is that on each page some resources would be active in some tests, and other would not be active. We could choose individual resources to leave in or comment out for a given run.

A third benefit was in the diagnosis of performance bottlenecks: we could selectively comment out one resource at a time to identify which resource was causing the performance bottleneck. This is a little harder to do in context-sensitive mode, given the compound structure of the script function calls.



Issue: Many dynamic session values requiring manual correlation

Manually correlating multiple dynamic session identifiers is time-consuming, especially if done repetitiously after an Action re-record. The built-in SAP Portal correlation rules do not support all of the SAP dynamic session variables. In addition, the Portal implemented SiteMinder's Single Signon authentication, which itself introduces several dynamic session variables of its own. Moreover, the standard SAP rules often do not specify unique left borders but rather search for the nth occurrence (using "ORD=") which too often changes, nor does it use descriptive parameter naming. The combination of these issues yields a maintenance headache.

Practical Solution: Custom correlation file

We defined rules and created custom correlation files to automatically handle the commonly repeating dynamic session values.

This is an export of the .cor file of the correlation rules established for the SiteMinder session variables:

```
<?xml version="1.0" ?>
<CorrelationSettings>
<Group Name="SiteMinder SSO" Enable="1" Icon="logo_default.bmp">
<Rule Name="Auth_Reason" LeftBoundText="&SMAUTHREASON=" LeftBoundType="1" LeftBoundInstance="0"
RightBoundText="&" RightBoundType="1" AltRightBoundText="" AltRightBoundType="1" Flags="12"
ParamPrefix="pSMAUTHREASON" Type="8" SaveOffset="0" SaveLen="-1" CallbackName="" CallbackDLLName=""
FormField="" ReplaceLB="" ReplaceRB="" />
<Rule Name="Method" LeftBoundText="&METHOD=" LeftBoundType="1" LeftBoundInstance="0" RightBoundText="&"
RightBoundType="1" AltRightBoundText="" AltRightBoundType="1" Flags="12" ParamPrefix="pMETHOD" Type="8"
SaveOffset="0" SaveLen="-1" CallbackName="" CallbackDLLName="" FormField="" ReplaceLB="" ReplaceRB="" />
<Rule Name="Agent_Name" LeftBoundText="&SMAGENTNAME=" LeftBoundType="1" LeftBoundInstance="0"
RightBoundText="&" RightBoundType="1" AltRightBoundText="" AltRightBoundType="1" Flags="12"
ParamPrefix="pSMAGENTNAMEenc" Type="8" SaveOffset="0" SaveLen="-1" CallbackName=""
CallbackDLLName="" FormField="" ReplaceLB="" ReplaceRB="" />
<Rule Name="Type" LeftBoundText="?TYPE=" LeftBoundType="1" LeftBoundInstance="0" RightBoundText="&"
RightBoundType="1" AltRightBoundText="" AltRightBoundType="1" Flags="8" ParamPrefix="pTYPE" Type="8"
SaveOffset="0" SaveLen="-1" CallbackName="" CallbackDLLName="" FormField="" ReplaceLB="" ReplaceRB="" />
<Rule Name="RealModelID" LeftBoundText="&REALMOID=" LeftBoundType="1" LeftBoundInstance="0"
RightBoundText="&" RightBoundType="1" AltRightBoundText="" AltRightBoundType="1" Flags="8"
ParamPrefix="pREALMOID" Type="8" SaveOffset="0" SaveLen="-1" CallbackName="" CallbackDLLName=""
FormField="" ReplaceLB="" ReplaceRB="" />
<Rule Name="Target" LeftBoundText="&TARGET=" LeftBoundType="1" LeftBoundInstance="0" RightBoundText="&"
RightBoundType="1" AltRightBoundText="" AltRightBoundType="1" Flags="8" ParamPrefix="pTARGETenc"
Type="8" SaveOffset="0" SaveLen="-1" CallbackName="" CallbackDLLName="" FormField="" ReplaceLB=""
ReplaceRB="" />
</Group>
</CorrelationSettings>
```

(To use these, copy/paste into a notepad file, set the file extension to .cor, and import them in Vugen/Recording Options/Correlation).



Issue: Authentication occurs in all scripts

Authentication is an action that needs to be in every script, and with 40+ scripts to maintain, we needed a different way of handling this.

Practical Solution: Modularization

As authentication was very stable from release to release, it is a perfect candidate for turning it into a reusable component. We created an Authentication Action, recorded the logon session, incorporated the dynamic sessions code, applied a descriptive naming standard and added a comment section describing the session variables. We then created a common data table of the users, and then imported this Action into all our scripts.

Our Authenticate Action code:

```
Authenticate()
{
  lr_start_transaction("Authenticate");
  web_reg_save_param    ("pSMAGENTNAMEenc", "LB=hidden name=smagentname value=\"", "RB=\"", LAST);
  web_reg_save_param    ("pTARGETnenc", "LB=hidden name=target value=\"", "RB=\"", LAST);
  web_reg_save_param    ("pPOSTPRESERVATIONDATA", "LB=hidden name=postpreservationdata value=\"", "RB=\"",
    LAST);
  web_reg_save_param    ("pSMENC", "LB=HIDDEN NAME=\"SMENC\" VALUE=\"", "RB=\"", LAST);
  web_reg_find          ("Text=Password", LAST);
  web_url("Authenticate1",
    "URL=https://webauth.ext.portal.com/siteminderagent/forms/login.fcc"
    "?TYPE={pTYPE}"
    "&REALMOID={pREALMOID}"
    "&GUID={pGUID}"
    "&SMAUTHREASON={pSMAUTHREASON}"
    "&METHOD={pMETHOD}"
    "&SMAGENTNAME={pSMAGENTNAMEenc}"
    "&TARGET={pTARGETenc}",
    "RecContentType=text/html",
    "Resource=0",
    "Referer={pProtocol}://{pHost}{pPort}{pDir}",
    "Mode=HTML",
    EXTRARES,
    "URL=/images/sso.gif",ENDITEM,
    "URL=/images/space.gif",ENDITEM,
    "URL=/images/logo.gif",ENDITEM,
    "URL=/images/contact.gif",ENDITEM,
    "URL=/siteminderagent/forms/images/siteminder_logo.jpg", ENDITEM, LAST);

  web_submit_data      ("Authenticate2",
    "Action=https://webauth.ext.portal.com/siteminderagent/forms/login.fcc"
    "?TYPE={pTYPE}"
    "&REALMOID={pREALMOID}"
    "&GUID={pGUID}"
    "&SMAUTHREASON={pSMAUTHREASON}"
    "&METHOD={pMETHOD}"
    "&SMAGENTNAME={pSMAGENTNAMEenc}"
    "&TARGET={pTARGETenc}",
    "Method=POST",
    "RecContentType=text/html",
    "Referer=",
    "Mode=HTTP",
    ITEMDATA,
    "Name=SMENC", "Value={pSMENC}", ENDITEM,
    "Name=USER", "Value=saptest{pSeqNum}", ENDITEM,
    "Name=PASSWORD", "Value=s4p{pSeqNum}", ENDITEM,
    "Name=target", "Value={pTARGETnenc}", ENDITEM,
```



```
"Name=smauthreason", "Value={pSMAUTHREASON}", ENDITEM,  
"Name=smagentname", "Value={pSMAGENTNAMEenc}", ENDITEM,  
"Name=postpreservationdata", "Value={pPOSTPRESERVATIONDATA}", ENDITEM,  
"Name=login_uid", "Value=", ENDITEM,  
LAST);  
  
lr_end_transaction ("Authenticate", LR_PASS);  
  
// reset maxredirection back up to a default level  
web_set_option("MaxRedirectionDepth", "10", LAST);  
  
return 0;  
}
```

Issue: Execution Validation

Thorough transaction validation done by identifying unique text on each page takes a lot of time to debug and re-record.

Practical Solution: Global text check

We utilized the `web_global_verification` function to confirm that the application displayed each page properly. A global validation was inserted in the initialization of the script, which searches throughout the script execution for that error. The SAP Portal lends itself well to this approach because it throws the same set of errors when a page is not properly delivered. Once the possible error statements are cataloged, these can be checked globally.

An example of execution validation code in our `user_init` section:

```
// SAMPLE ERROR STRING: "Runtime Error"  
web_global_verification ("Text/IC=Portal Runtime Error", "Fail=Found", "Search=NORESOURCE", LAST);  
//  
// SAMPLE ERROR STRING: "An exception occurred while processing a request for :"  
web_global_verification ("Text/IC=occurred while processing", "Fail=Found", "Search=NORESOURCE", LAST);  
//  
// SAMPLE ERROR STRING: "ERROR: The requested URL could not be retrieved"  
// </TITLE>  
// </HEAD><BODY>  
// <H1>ERROR</H1>  
// <H2>The requested URL could not be retrieved</H2>  
web_global_verification ("Text/IC=The requested URL could not be retrieved", "Fail=Found", "Search=NORESOURCE",  
LAST);  
  
//=====
```

ABOUT Mentora www.mentora.com

We test, host and manage business-critical applications. We specialize in enterprise application performance testing, including SAP, Oracle, PeopleSoft, SunGard EXP, and Trizetto/Facets.

On the hosting side, we are pioneers in the new generation of managed application hosting providers, combining support of Linux and the newest technologies, system performance testing and a complete menu of managed services for networks, servers, operating systems, databases, and storage, with our exclusive, personalized *named-engineer-on-call* response and *Application Availability SLA*.