# Table of contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>3</td>
</tr>
<tr>
<td>About the author</td>
<td>3</td>
</tr>
<tr>
<td><strong>Implementing a successful performance testing program</strong></td>
<td>3</td>
</tr>
<tr>
<td>Formulating a high-level test plan</td>
<td>4</td>
</tr>
<tr>
<td><strong>Tips for scripting</strong></td>
<td>5</td>
</tr>
<tr>
<td>Defining the directory structure and naming conventions</td>
<td>5</td>
</tr>
<tr>
<td>Recording scripts</td>
<td>5</td>
</tr>
<tr>
<td>Creating dynamic scripts through correlation and parameterization</td>
<td>5</td>
</tr>
<tr>
<td>Creating transactions</td>
<td>5</td>
</tr>
<tr>
<td>Parameterizing and converting itoa (C++) code</td>
<td>6</td>
</tr>
<tr>
<td>Date_Time manipulation</td>
<td>6</td>
</tr>
<tr>
<td>Get date seven days beyond current date</td>
<td>6</td>
</tr>
<tr>
<td>Polling for batch run status</td>
<td>7</td>
</tr>
<tr>
<td>Logging messages</td>
<td>7</td>
</tr>
<tr>
<td>Scripting checklist</td>
<td>7</td>
</tr>
<tr>
<td><strong>Tips for building scenarios</strong></td>
<td>8</td>
</tr>
<tr>
<td>Executing tests via HP LoadRunner Controller</td>
<td>8</td>
</tr>
<tr>
<td>Scheduling options</td>
<td>8</td>
</tr>
<tr>
<td><strong>Tips for data handling</strong></td>
<td>9</td>
</tr>
<tr>
<td>Using virtual tables</td>
<td>9</td>
</tr>
<tr>
<td>IP spoofing for client with same Netmask IP class and network</td>
<td>9</td>
</tr>
<tr>
<td>Parse string based on value</td>
<td>10</td>
</tr>
<tr>
<td>Randomize using switch and case</td>
<td>10</td>
</tr>
<tr>
<td><strong>Tips for test execution and scheduling</strong></td>
<td>11</td>
</tr>
<tr>
<td>HP LoadRunner timeout errors</td>
<td>11</td>
</tr>
<tr>
<td>Extended logging code</td>
<td>11</td>
</tr>
</tbody>
</table>
Implementing a successful performance testing program

Before you release any new application into production, you must perform extensive capacity and performance validation (CPV) testing. This guide is intended to help new users and seasoned professionals learn new ways to design and implement successful load testing initiatives using HP LoadRunner.

Performance testing processes can be divided into four phases: the initial performance testing request, preparation for testing, script development and execution, and test analysis.

The following sections of this document describe some of the steps and best practices to follow when designing and conducting an effective performance test program.

### Table 1.
The functional flow of a performance testing execution process, with suggested deliverables for each phase.

<table>
<thead>
<tr>
<th>Description</th>
<th>CPV request</th>
<th>Preparation</th>
<th>Development and execution</th>
<th>Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Meet with CPV lead to overview solution under test</td>
<td>Secure hardware</td>
<td>Create automated scripts</td>
<td>Analyze data provided by CPV for each test cycle</td>
</tr>
<tr>
<td></td>
<td>Conduct a planning session to outline requirements and resolve issues</td>
<td>Configure hardware/software</td>
<td>Debug automated scripts</td>
<td>Identify unacceptable performance</td>
</tr>
<tr>
<td></td>
<td>Reserve test slot with CPV</td>
<td>Data explosion</td>
<td>Run pre-determined load for each test cycle</td>
<td>Ensure accurate hardware capacity</td>
</tr>
<tr>
<td></td>
<td>Create IDs</td>
<td>Create business scenarios and test cases</td>
<td>Provide results after each test cycle</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Business owner</td>
<td>Business owner administrator(s)</td>
<td>Provide summary report</td>
<td></td>
</tr>
<tr>
<td>Owner</td>
<td>Business owner</td>
<td>CPV</td>
<td>Business owner CPV</td>
<td></td>
</tr>
<tr>
<td>Deliverables</td>
<td>Architecture review board approval</td>
<td>Performance requirements document (final)</td>
<td>Summary report outlining tuning changes, CPV issues, statistical data</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Performance requirements document (draft)</td>
<td>Support staff available</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Support staff established</td>
<td>Test environment configured and data validated</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Obtain automation tool, if not available</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Abstract: This guide provides tips and tricks for HP LoadRunner software configuration, scripting, and execution. It is a conglomerate of lessons learned by HP LoadRunner power user Opral Wisham, including unique code as well as code collected from other testers. This guide is intended to help testers just learning to use HP LoadRunner, as well as to provide new best practices for those who have used HP LoadRunner for many years.

About the author: Opral Wisham is currently a senior performance and capacity specialist at Waste Management, Inc., the industry’s leading provider of comprehensive waste management services.
Formulating a high-level test plan

The performance testing team should begin by defining a high-level test plan that describes the timeline for all testing efforts, including what types of tests will be performed (e.g., online and infrastructure stress and load tests, batch performance tests, etc.). The plan should also indicate how the performance testing team will interact with the development, deployment and/or support groups within the enterprise. When possible, it helps to use the actual names and titles of resources when explaining the interaction.

The testing team should describe what the tests are intended to measure and/or report. It is important to clarify all terminology that may be at risk of being misunderstood (e.g., “Transaction’s Response Time” may refer to the time it takes to hit the “enter” button and get a result from the system, or it could indicate the time it takes to perform a function that requires some human intervention). Some metrics for web applications will have different definitions from traditional online and batch applications, like average hits/second, web server throughput, etc.

The following checklist can be used when defining the steps of a comprehensive performance testing process:

1. List all testing milestones and deliverables for every phase of the project, including the pilot, code freeze and production phases.
2. Create a comprehensive production physical architecture diagram that shows a very low level of detail, including how each component will be connected.
3. List all hardware and software requirements for the tests.
4. Determine all database data volume requirements.
5. Define all performance objectives.
6. Identify all performance benchmarks, if available.
7. Identify the total number of expected users.
8. Specify the minimum and maximum number of concurrent users.
9. Select five business scenarios (manual scripts), plus actions within each scenario that should be measured (transactions).
10. Determine the user load mix.
11. Define acceptable average response times per transaction.
12. Define the average hits/second or number of transactions within one hour.
13. Define throughput, if applicable.
14. Identify any concurrent batch processes that are running.
15. Identify all lead, technical and functional contact personnel.

The following sections will provide examples that will help the performance testing team streamline the processes of scripting, building scenarios, data handling, test execution and test scheduling.
Tips for scripting

Defining the directory structure and naming conventions

It is important to establish formal procedures for creating directory structures and establishing naming conventions. The following nomenclature can be used for creating a directory structure (folder names) for new projects:

<table>
<thead>
<tr>
<th>ProjectName</th>
<th>LR_Scripts</th>
<th>Scenarios</th>
<th>TestDoc</th>
<th>TestPlanAndReport</th>
<th>TestExecution</th>
<th>Recommendations</th>
<th>Reports</th>
<th>LR_Reports</th>
<th>Cycle1</th>
<th>Cycle2</th>
<th>HTML_Reports</th>
<th>Cycle1</th>
<th>Cycle2</th>
</tr>
</thead>
</table>

Project and script names should be descriptive and intuitive for new users of the system. For example:

- **ProjectName**: Should identify the project or application under test.
- **LR_Scripts**: Files should be the name of the saved HP LoadRunner scripts.
- **FunctionName_Type**: Type: Is func or rpt or qry
- **Scenarios**: Files should identify the project and number of Vusers simulated.
- **ProjectName_VuserCount**
- **TestDoc**: Directory should contain all test documentation.
- **TestPlanAndReport**: Test Plan, Test Cases, Workload, etc., Scripter Notes, Test Summary
- **TestExecution**: Execution Progress Log, Comparison Log
- **Recommendations**: Recommendations

Recording scripts

Whenever possible, the performance testing team should use HP LoadRunner’s automatic recording options. This method provides the most efficient way to record scripts for load testing most web-based applications.

Creating dynamic scripts through correlation and parameterization

The performance testing team can use correlation to retrieve a value before it is needed in a preceding statement, enabling the server to dynamically generate the unknown value. The team can use parameterization when the values are known and they are not available prior to executing the statement.

Below is a parameterization example:

1. Parameterize, Source, Currency, Business Unit and Account, using a file. Select next row should be “same as Business Unit.” Business Unit Select next row should be “sequential” or “random,” depending on the duration or number of iterations.
2. Parameterize date as “date/time,” using %m/%d/%Y
3. Parameterize Amount as a random number from 1 to 10,000.
4. Parameterize Accounting Period as “date/time,” using %m
5. Parameterize the URL using a file. This will reduce the script changes required when the URL does change.

Correlation and parameterization are keys to creating a dynamic, data-driven test.

Creating transactions

The performance test team should then define the transactions to measure the performance of the server. Each transaction measures the time it takes for the server to respond to specified Virtual User (Vuser) requests.

To mark the start of a transaction, the testing team should click the “start transaction” button from the recording toolbar. Enter a transaction name in the transaction name box. Click “OK” to accept the transaction name. HP LoadRunner Virtual User Generator (VuGen) will automatically enter an lr_start_transaction statement into the script. Click the “end transaction” button to end the transaction. VuGen will automatically enter an lr_end_transaction statement into the script.
Performance testing typically creates transactions for the Log-in and Log-out and every place in the script where a “submit” appears. More transactions can be included, but all submits must have transactions to measure the response times.

Dynamic transactions names are names that change depending upon a parameterized value. If there are several reports that are run randomly, the team can define the transaction name dynamically using the code below:

```
Char rname[10];
sprintf( rname,
        "Run_Report_%s",
        lr_eval_string("{report}")
);
lr_output_message(rname);
lr_start_transaction(rname);
lr_end_transaction(rname,2);
```

Parameterizing and converting itoa (C++) code
If the team needs to increment a counter and use that counter as a parameter, it will need to convert the integer to a string by using sprintf. After conversion, the string needs to become a parameter. Therefore the string should be saved into a parameter using lr_save_string.

```
/* M:\Remedy\Test Scripts\Modify an Existing Remedy Help Desk Case */
/* insert before action */
char wid_str[6]; /* 0xFFFF = 65535 + NULL = 6 */
int i = 0; /* initialize I */
i = i + 2; /* increment I by 2 or whatever value */
sprintf( wid_str, "%d", i ); /* copy integer into string */
lr_save_string(wid_str, "wid_num"); /* save into parameter value */
```

Date_Time manipulation
The following statements can be inserted to manipulate the date time stamp. In this example, it was necessary for the date in the script to be one day after the current date. The parameter {Tomorrow} is saved where the current date appeared in the script. This parameter insertion results in tomorrow’s date being inserted where the current date would have appeared.

```
Action1()
{
    lr_save_datetime("Tomorrow is %m/%d/%Y", DATE_NOW +
    ONE_DAY, "Tomorrow ");
    //lr_output_message(lr_eval_string("{Tomorrow }");

    itemdata.....
    *
    *
    *
    "Name=PV_REQ_SR_WRK_DUE_DT$0", "Value={Today }",
    ENDITEM,
```

Get date seven days beyond current date
This code will add seven days to the current date. This is useful when a due date is required seven days from the current date:

```
lr_save_datetime("%m/%d/%Y", DATE_NOW + ONE_DAY +
    ONE_DAY + ONE_DAY + ONE_DAY + ONE_DAY + ONE_DAY +
    "curr_date");
```
Polling for batch run status

This example was used in a script that required clicking the “refresh” button until the run status changed to “success.” This code provides an automatic refresh until the batch job has been completed. The next step requires the completion of the batch job.

```c
int x; // flag will be 0 or 9
char *temp, *temp2; // values to hold strings
Action1()
{
    temp2="Success"; //compare string 2
    //lr_message("temp2 = %s", temp2);
    ....
    // set x to 0 x is the success flag
    x=0;
    do {
        web_reg_save_param("RunStatus",
        "<td align='LEFT' class='PSLEVEL1GRIDODDROW'
        >x</td",
        "RB="n",
        "Ord=5",
        "Search=body",
        LAST);
        web_submit_data("PROCESSMONITOR.PROCESS
        MONITOR.GBL",
        "Action=http://crpu028a:8050/psc/fs84cpv/EMPLOYEE/ERP/c/
        PROCESSMONITOR.PROCESSMONITOR.GBL",
        "Method=POST",
        "RecContentType=text/html",
        "Referer=http://crpu028a:8050/psc/fs84cpv/EMPLOYEE/ERP/c/
        PROCESSMONITOR.PROCESSMONITOR.GBL?",
        "Page=PMN_PRCSLIST&Action=U&",
        "Snapshot=t17 .inf",
        "Mode=NORESOURCE",
        ITEMDATA,
        "Name=ICType", "Value=Panel", ENDITEM,
        "Name=ICElementNum", "Value=0", ENDITEM,
        "Name=ICStateNum", "Value=ICStateNum6", ENDITEM,
        "Name=ICAction", "Value=REFRESH_BTN", ENDITEM,
        "Name=ICXPos", "Value=0", ENDITEM,
        "Name=ICYPos", "Value=0", ENDITEM,
        "Name=ICFocus", "Value=" , ENDITEM,
        "Name=ICChanged", "Value=1", ENDITEM,
        "Name=ICFind", "Value=" , ENDITEM,
        "Name=PMN_FILTER_WS_OPRID", "Value=CPVID",
        ENDITEM,
        "Name=PMN_FILTER_PRCTYPE", "Value=" , ENDITEM,
        "Name=PMN_FILTER_FILTERVALUE", "Value=1",
        ENDITEM,
        "Name=PMN_FILTER_FILTERUNIT", "Value=" , ENDITEM,
        "Name=PMN_FILTER_SERVERNAME", "Value=PSUNIX",
        ENDITEM,
        "Name=PMN_FILTER_PRCSNAME", "Value=" , ENDITEM,
        "Name=PMN_DERIVED_PRCSINSTANCE",
        "Value=Process_Instance", ENDITEM,
        "Name=PMN_FILTER_RUNSTATUS", "Value=" , ENDITEM,
        ENDITEM);
        // Compare correlation value with character string
        temp = lr_eval_string(lr_eval_string("{RunStatus}"));
        //correlation value to variable
        //lr_message("temp = %s", temp);
        //compare string 1
        if(strcmp(temp,temp2)==0){
            // string compare success with
            //correlation value
            x=9; // set flag to indicate success
        }
    } while (x == 0); // do while flag not set
```

Logging messages

Vuser messages are sent to the Vuser log and output window by using one of the message functions. The lr_output_message is most commonly used by CPV. Log messages can be used to identify many things about Vusers during script execution, including:

1. Determining if a Vuser reaches a certain point during script execution.
2. Confirming that valid values are passed for parameters.
3. Confirming that passed values appear where they should in the database (by capturing the passed values and knowing what to search for).

Scripting checklist

HP LoadRunner script testing will occur after creation of each automated script. Scripts can be added to performance test bed when:

1. No scripting errors occur.
2. Looping successfully; replaces parameterization variables with various values.
3. Activity is noted on each system tier.
4. A minimum load of ten users is simulated.
5. Scripts are run on a subsequent day (dates correlated correctly).
Tips for building scenarios

Executing tests via HP LoadRunner Controller

The following steps can be used for performance test execution:

1. Add load generators and connect.
2. Add scenario groups: Add one or more scripts to the list of scenarios.
3. Create run-time settings: Pace users based on feedback from business analysts or users.
   a. Disable Logging
   b. Determine where to put think time in the script from business analysts or users. Think time can be randomized when a non-random value is in the script.
4. Simulate browser cache: This option instructs the Vuser to simulate a browser with a cache. A cache is used to keep local copies of frequently accessed documents and thereby reduces the time connected to the network. By default, cache simulation is enabled. If you disable this option, all Vusers emulate a browser with no cache available. Note: Unlike a regular browser cache, the cache assigned to a Vuser simulates storage of graphic files only. The cache does not store text or other page contents associated with the webpage. Every Vuser has its own cache—every Vuser must save and retrieve images from the cache. When the cache is disabled, HP LoadRunner still downloads each page image only once.
5. Simulate a new user for each iteration: Instructs VuGen to reset all HTTP contexts between iterations to their states at the end of the init section. This setting allows the Vuser to more accurately emulate a new user beginning a browsing session. It resets all cookies, closes all keep-alive connections, clears the cache, and resets the user names and passwords (enabled by default).
6. Schedule by scenario: As the number of users increases, there may be a need to adjust the ramp-up. Duration should be for one hour on most tests with the exception of stress tests.
7. Schedule by group: Allows testers to stagger the scenarios by group, meaning scenario B can start 10 minutes after scenario A. Ramp-up is how frequently a number of Vusers will log into the system per scenario. Duration is how long a particular scenario group will execute after ramp-up. Testers may manipulate duration to cease testing at a similar time.

Scheduling options

Schedule tests to run at later times. This can all be done in the HP LoadRunner controller as well. To set up a test to run at a later time, first set up the test as it should be run. For example, make the run-time setting adjustments on the design screen (just as it would be done with a test that wasn’t scheduled). Then select Scenario/Start Time from the menu bar. Enter the amount of time the test should be delayed (for example, if it is 5:00 p.m. now and the test should be run at 7:00 p.m., enter two hours for the delay).
Tips for data handling

Using virtual tables

Occasionally, scripts can be used to generate the data that will be required to run other scripts. In these situations, it may be necessary to use virtual tables to accomplish this task (which requires minimal altering to the original script). The HP Virtual Table provides the ability to send information from one script to a virtual table, in order to retrieve the information from the virtual table for the other script.

Example: Unzip the package under C:\Program Files and copy vtclient.dll, vtserver.dll and vts2.sh to the appropriate folders. Use regsvr32 and make sure the following files exist:

Regsvr32 "C:\WINNT\system32\msflxgrd.ocx"
Regsvr32 "C:\WINNT\system32\vtsctls.ocx"

Make sure that "C:\WINNT\system32\COMCTL32.DLL" exists.

Install the virtual table on the controller and the machine that will be used as the Virtual Table Server. Copy vtclient.dll and vtserver.dll to the lr/bin: folder, then locate the “include” folder (lr/include:) and insert vts2.h (all found in the product folder on the hard drive).

IP spoofing for client with same Netmask IP class and network

IP spoofing is used when a unique user IP address is required to identify unique data.

In Script:

CODE TO VERIFY IP SPOOF: lr_message("IP ADDRESS _ %s ", lr_get_vuser_ip());

Adding IP Addresses:

GO TO IP WIZARD …..FOLLOW INSTRUCTIONS AND ADD IP

Settings in the Controller:

• RUN-TIME SETTING—RUN AS “PROCESS” (IN GENERAL TAB)
• IN THE CONTROLLER TOOLBAR “SCENARIOS” TAB, ENABLE IP SPOOFER
• IN THE CONTROLLER TOOLBAR “TOOLS” TAB, CHECK “EXPERT MODE”

Verify that Wins address and DNS are completed with the correct IP addresses.

Note:
The IP SPOOFING ADD ROUTE BAT file needs to be run on the target server in order for IP SPOOFING to work.

Also, verify that all of the new IP addresses are on the list in “Network Neighborhood”/”Properties”/Protocol/Advanced.

Verify that Wins address and DNS are completed with the correct IP addresses.
Parse string based on value

Use the code below to parse a string and use the data in subsequent steps. The string “Wisham333445555” can be separated into multiple variables such as last name and ssn.

```c
char *string;
char *first_x;
// LASTNAME=Wisham333445555
string = lr_eval_string("{LASTNAME}"); // Save value in LASTNAME
// lr_output_message("LastName: %s", string);
first_x = (char *)strchr(string, '3'); // Find first occurrence of 3 and save values after
// lr_output_message("SSN: %s", first_x);
lr_save_var(first_x, 9, 0, "SSN"); // Save 9 characters after 3 encountered
// lr_output_message("SSN: %s", lr_eval_string("{SSN}")");
```

Randomize using switch and case

Use the switch command to vary transactions. If you are testing multiple reports with significantly different web URLs but all other steps are the same, the user can use the switch statement.

```c
int num;
num = ((rand() % 4));

switch (num)
{
 case 0:
 lr_think_time(300);
 lr_start_transaction("FS_QRY_06_ZAM_ACQ_VERIFY_SEARCH");
 : ;
 break;
 case 1:
 lr_think_time(300);
 lr_start_transaction("FS_QRY_06_ZLT_UDP_VERIFY_SEARCH");
 : ;
 break;
 default:
 lr_output_message("Random Number selected was not between 0 and 1");
}
```
Tips for test execution and scheduling

HP LoadRunner timeout errors

Use the following steps to prevent HP LoadRunner timeout errors if you need less than 1,000 seconds for a function such as save data to complete:

1. From the Controller go to Run-time Settings -> Browser Emulations tab -> Advanced
2. Change default timeout settings (should be three) to be a number less than or equal to 1000
3. Click OK
4. Go to Tools -> Options -> Timeout tab
5. Change Load Generator and VU timeout parameters to be a number less than or equal to 1000
6. Click OK

Use the following steps to prevent HP LoadRunner timeout errors if you need greater than 1,000 seconds for a function such as upload to complete:

1. Modify LoadRunner\template\qtweb\default.cfg.
   Insert the following:
   ```
   [web]
   ReceiveTimeout=10000
   ```
   Change:
   ```
   [LIST]
   1=T_COMBOBOX;Web;EnableIPCache;DNS caching;Save a host’s IP address to a cache to conserve time in subsequent calls to the host.;1;Yes;<Yes;No>
   2=T_COMBOBOX;Web;HTTPVer;HTTP version;Indicates the version of HTTP used by your application. Select version 1.0 or 1.1.;1;1;1;1;1;<1;0;1;1;1>
   3=T_COMBOBOX;Web;KeepAlive;Keep-Alive HTTP connections;Allow persistent HTTP connections to enable multiple requests to be sent over the same TCP connection.;1;Yes;<Yes;No>
   4=T_EDIT_NUM;Web;MaxConnections;Concurrent connections limit;Sets the maximum number of simultaneous keep-alive connections that a Vuser can maintain during script execution;1;4;<0;10>
   5=T_EDIT_NUM;Web;ConnectTimeout;Connect timeout (seconds);The maximum amount of time in seconds, that a Vuser waits to connect to a server, before issuing an error.;1;120;<0;10000>
   6=T_EDIT_NUM;Web;ReceiveTimeout;Receive timeout (seconds);The maximum amount of time in seconds, that a Vuser waits to receive information from a server, before issuing an error.;1;120;<0;10000>
   7=T_EDIT_NUM;Web;NetBufSize;Network buffer size;Sets the maximum number of the network buffer that may be used by the Vuser. The default is 12288 bytes.;1;12288;<0;100000>
   8=T_COMBOBOX;Web;CacheAlwaysCheckForNewerPages;Cache-Emulate always check for newer pages.;1;Yes;<Yes;No>
   9=T_EDIT_NUM;Web;PageDownloadTimeout;Page download timeout(seconds);1;120;<0;32000>
   ```
3. Use steps 1-6 above to change the Controller GUI parameters.

Extended logging code

This code turns tracing on to replay a specify part of a script in extended mode without replaying the entire script in extended mode. This reduces the playback time substantially.

```c
// TURN FULL TRACE ON
lr_set_debug_message(LR_MSG_CLASS_EXTENDED_LOG | LR_MSG_CLASS_FULL_TRACE | LR_MSG_CLASS_RESULT_DATA | LR_MSG_CLASS_PARAMETERS, LR_MSG_ON);

web_reg_save_param("EMPLID", "LB/IC='PSEDITBOX_DISPONLY' >", "RB/IC=<", "Ord=4", "Search=body", LAST);

// TURN FULL TRACE OFF
lr_set_debug_message(LR_MSG_CLASS_EXTENDED_LOG | LR_MSG_CLASS_FULL_TRACE | LR_MSG_CLASS_RESULT_DATA | LR_MSG_CLASS_PARAMETERS, LR_MSG_OFF);
```